Univerzita Hradec Králové
Fakulta informatiky a managementu
katedra informačních technologií

DISERTAČNÍ PRÁCE

Návrh Edge Computing platformy umožňující efektivní provoz globálně distribuovaných aplikací s nízkou latencí

Autor: Ing. Jakub Pavlík, MSc.

Studijní obor: Informační a znalostní management

Školitel: doc. Ing. Vladimír Soběslav, Ph.D.

Hradec Králové 2020

Prohlášení

Prohlašuji, že jsem disertační práci vypracoval samostatně a s použitím uvedené literatury.

V Praze dne 30. března 2020 Ing. Jakub Pavlík, MSc.

Poděkování

Rád bych poděkoval všem, kteří mě podporovali při psaní této práce, zejména své manželce Hance a dceři Stellušce.
Dále bych rád poděkoval svému školiteli doc. Ing. Vladimíru Soběslavovi, Ph.D., a vedoucímu katedry informačních technologií doc. Ing. Vladimíru Burešovi, Ph.D., MBA, za konzultace a vzájemnou spolupráci při řešení projektů a psaní publikací. Současně bych rád poděkoval kolegovi Ing. Aleši Komárkovi za vzájemnou podporu během doktorského studia.
V neposlední řadě bych rád tuto práci a celé své doktorské studium věnoval mému zesnulému dědovi Prof. Ing. Ivanu Pavlíkovi, CSc.

Anotace

V současné době můžeme sledovat rapidní rozvoj internetu věcí společně s 5G sítěmi, které umožňují připojit stále více zařízení k internetu. Díky tomuto trendu vstupujeme do období po Cloud Computingu, ve kterém koncová zařízení generují velké množství dat a je nutné vytvořit aplikace, která tato data zpracují a umožní jejich transformaci na informace, jež podpoří různé oblasti, jako jsou například podpora rozhodování, umělá inteligence apod. Některé z těchto aplikací vyžadují krátkou dobu odezvy, jiné zase obsahují soukromá data a některé jen produkují velké množství dat. Cloud Computing nedokáže takové aplikace obsloužit, a proto vznikla nová doména nazvaná Edge Computing. Její cílem je přesunutí funkcí Cloud Computingu blíže ke koncovým zařízením na tzv. okraj sítě a snaha vyřešit problémy s dobou odezvy, limitovanou kapacitou baterie, propustnost, bezpečnost a soukromí. Edge Computing problematika je ovšem velmi mladá a objevuje se v ní spousta neobjasněných problémů jako například nedostatečné standardy, optimalizace, centrální správa edge aplikací apod.

Existence těchto problémů vedla k hlavnímu cíli disertační práce, kterým je návrh vlastní Edge Computing platformy, jež umožní efektivně provozovat globálně distribuované aplikace vyžadující nízkou latenci pomocí standardizovaného aplikačního rozhraní pro jejich orchestraci. První část disertační práce obsahuje úvod do problematiky Edge Computingu, vymezuje její základní terminologii a aktuální příležitosti či problémy publikované v relevantních vědeckých pramenech. Současně také porovnává existující řešení společností Cisco, Microsoft a AWS s identifikovanými problémy.

Následující část práce se zabývá samotným návrhem vlastní Edge Computing platformy z důrazem na řešení identifikovaných problémů nízké doby odezvy, multifunkčnosti, multi-tenance a standardního API pro orchestraci edge aplikací. Poslední část práce je zaměřena na zhodnocení přínosů navržené platformy. Ty jsou zkoumány z technologického pohledu a z pohledu přínosu pro podnikový management z hlediska softwarové výkonnosti. Součástí přínosu pro podnikový management jsou také výstupy reálné případové studie nasazení platformy do restauračního řetězce ve Spojených státech amerických, a to díky působení autora práce na pozici jednoho z hlavních architektů v nadnárodních společnostech Mirantis a Volterra. Dosažené výsledky jsou tak podpořeny nejen publikační činností autora, ale také jejich aplikací do praxe.

Klíčová slova

Edge Computing, Kubernetes, Virtualizace

Annotation

Nowadays we can watch the rapid development of Internet of Things along with 5G networks that allow more and more devices to connect to the Internet. As a result of this trend, we are entering the era after Cloud Computing, in which terminal devices generate large amounts of data and it is necessary to create applications that process this data and enable their transformation into information that supports various activities such as decission making or domains related to artificial intelligence etc. Some of those applications require a short response time, others contain private data, and some only produce large amounts of data. Cloud Computing is unable to serve such applications, and a new area called Edge Computing has been created. The goal is to move Cloud Computing closer to end-to-end devices on the edge of networks and address low response time, limited battery capacity, throughput, security and privacy. However Edge Computing is very young and there are many unresolved issues, such as insufficient standards, optimization, central edge application management, etc.

The existence of these problems has led to the main goal of the dissertation, which is the design of its own Edge Computing platform, which will enable to effectively operate globally distributed applications requiring low latency using a standardized application interface for their orchestration. The first part of the thesis contains an introduction to Edge Computing, defines basic terminology and current opportunities or problems in this area based on research papers. It also compares existing vendor solutions from Cisco, Microsoft and AWS with the problems found.

The following part of the thesis deals with the design of the Edge Computing platform itself with emphasis on solving current problems of low response time, multifunctionality, multi-tenancy and standard orchestration API. The last part is focused on the evaluation of the benefits of the proposed platform. These are examined from the technological perspective and the contribution to company management in terms of performance software delivery. Outcomes of this contribution are measured by a real case study of platform deployment into american restaurant chain. This real contribution comes due to the author's work as one of the leading architects in the international companies Mirantis and Volterra. Therefore the achieved results are justified not only by the author's publication activity, but also by their application in practice.

Key words

Edge Computing, Kubernetes, Virtualization

Obsah

1 Úvod

Nyní se nacházíme na prahu další průmyslové revoluce, která je vedená rozmachem internetu a digitalizace, která také mimo jiné pracuje s informacemi. Kvalitní informační management je bez podpory informačních technologií v současné informační společnosti v podstatě nemožný. Velkou oblastí, která se v posledních letech šíří jako lavina, je internet věcí (Internet of Things – IoT). Internet věcí využívá technologie, které přinesla tato průmyslová revoluce a jejich spojením vzniklo zcela nové odvětví. Jedna z prvních zmínek, která se dá alespoň vzdáleně považovat za IoT, se datuje už do roku 1984. Na Tokijské univerzitě se tehdy mluvilo o konceptu „počítače všude". Koncept, z něhož IoT vychází, se však stává populárním díky radiofrekvenční identifikaci (Radio-frequency identification - RFID), kde jsou věci představovány jako jedinečně identifikovatelné objekty, které mají virtuální reprezentaci. Od té doby však uplynulo mnoho let, než se dalo mluvit o internetu věcí, a hlavním důvodem, který umožnil jeho nástup, je právě rozmach internetu (Honbo, 2013).

Díky tomu, že se IoT stává stále rozšířenějším, vstupujeme do éry po Cloud Computingu, ve které koncová zařízení generují velké množství dat a je nutné vytvořit aplikace, které tato data zpracují a umožní jejich transformaci na informace, jež mohou být využity pro různé oblasti, jako je podpora rozhodování, umělá inteligence apod. (Ashton, 2019). Společnost IDC předpověděla, že do roku 2025 se k internetu připojí přibližně 41,6 miliard IoT zařízení (IDC, 2019), která budou generovat 79,4 zetttabajtů dat. Některé z provozovaných aplikací budou vyžadovat velmi krátkou dobu odezvy, některé budou obsahovat soukromá data a některé budou naopak produkovat velké množství dat. K těmto aplikacím patří oblasti jako rozšířená realita, průmyslový IoT, Connected Vehicles atd. Cloud Computing nedokáže takové aplikace podporovat, a proto má být řešením nová doména nazvaná Edge Computing, jinými slovy - přesun funkcí z Cloud Computingu blíže ke koncovým zařízením na tzv. edge sítě. Ty budou schopny řešit problémy s dobou odezvy, limitovanou kapacitou baterie mobilních zařízení, propustnost, bezpečnost a soukromí.

Na základě výše uvedeného je disertační práce zaměřena na analýzu a návrh Edge Computing platformy adresující současné problémy a výzvy publikované v relevantních vědeckých pramenech. Tato práce je výsledkem 4 let vědeckého působení autora na pozicích hlavního architekta v oblasti Cloud Computingu a později Edge Compu-

tingu v několika nadnárodních společnostech, podpořená publikační a projektovou činností dostupnou v přehledu publikovaných prací disertanta.

Disertační práce je rozdělena do šesti kapitol včetně úvodu a závěru. V kapitole 2 jsou představeny hlavní a dílčí cíle této disertační práce. Kapitola 3 se zabývá shrnutím aktuálního stavu problematiky Edge Computingu, ve které se díky analýze současných trendů snaží najít správný směr výzkumu. Nejprve je vysvětlena motivace k Edge Computingu společně s rešerší dané problematiky. Následně dochází k představení 3 standardů MEC, Cloudlet a Fog Computing. V podkapitole 3.3 je provedena detailní rešerše v oblasti vědeckých pramenů týkajících se problémů a příležitostí, tak aby bylo možné identifikovat prostor pro disertační výzkum. Tyto problémy jsou potvrzeny přehledem a porovnáním existujících řešení od výrobců jako Cisco, Microsoft atd. Na tomto základě je v závěrečném shrnutí této teoretické kapitoly vysvětlena potřeba nového návrhu Edge Computing platformy, která vnímá jako největší problém dostupných řešení model s pouze třemi vrstvami.

Vlastní přínos práce je pak rozdělen do dvou kapitol. Kapitola 4 obsahuje samotný návrh vlastní Consumer Edge Computing (CEC) platformy. Nejprve přiřazuje nevyřešené problémy a příležitosti k dílčím cílům. Následně postupně rozebírá a popisuje architekturu CEC ve vztahu k nim místo obecného popisu komponent. Předposlední kapitola 5 je zaměřena na zhodnocení přínosů navržené platformy. Ty jsou zkoumány z technologického pohledu a z hlediska přínosu pro podnikový management, tedy z hlediska softwarové výkonnosti. Součástí je i případová studie reálného nasazení do restauračního řetězce ve Spojených státech amerických, u které autor působil jako hlavní architekt. Podkapitola 5.4 pak shrnuje přínosy a diskuzi výsledků. Poslední kapitola 6 je závěrečným shrnutím celé práce.

2 Cíle disertační práce

Jak již bylo řečeno v úvodu této práce, díky rozšíření internetu věcí postupně vstupujeme do éry po Cloud Computingu, ve které koncová zařízení generují velké množství dat a je nutné vytvořit aplikace, které tato data zpracují a umožní jejich transformaci na informace. Tento trend přináší nové výzvy především pro společnosti, které musí tyto aplikace provozovat a spravovat ve stovkách či tisících poboček (restaurace, maloobchodní řetězce, chytrá města apod.). Edge Computing by měl být odpovědí na tyto nové výzvy a problémy, avšak mnoho z nich není stále vyřešeno, čímž se otevřel prostor pro výzkum prezentovaný v této disertační práci.

Hlavním cílem disertační práce je návrh kompletní Edge Computing platformy, která umožní efektivně provozovat globálně distribuované aplikace vyžadující nízkou latenci pomocí standardizovaného aplikačního rozhraní pro jejich orchestraci. To umožní výrazně zefektivnit provoz aplikací v tisících lokalitách a přinést inovaci v doručování tohoto softwaru ve společnostech. Tento cíl vychází z nastíněných problémů z kapitoly 3, kde jsou jednotlivé problémy vysvětleny s odkazem na patřičné publikace z dané oblasti a přehledem existujících řešení.

Naplnění hlavního cíle vyžaduje splnění dílčích cílů, které formují obsah práce a zároveň otevírají některé otázky pro další výzkum.

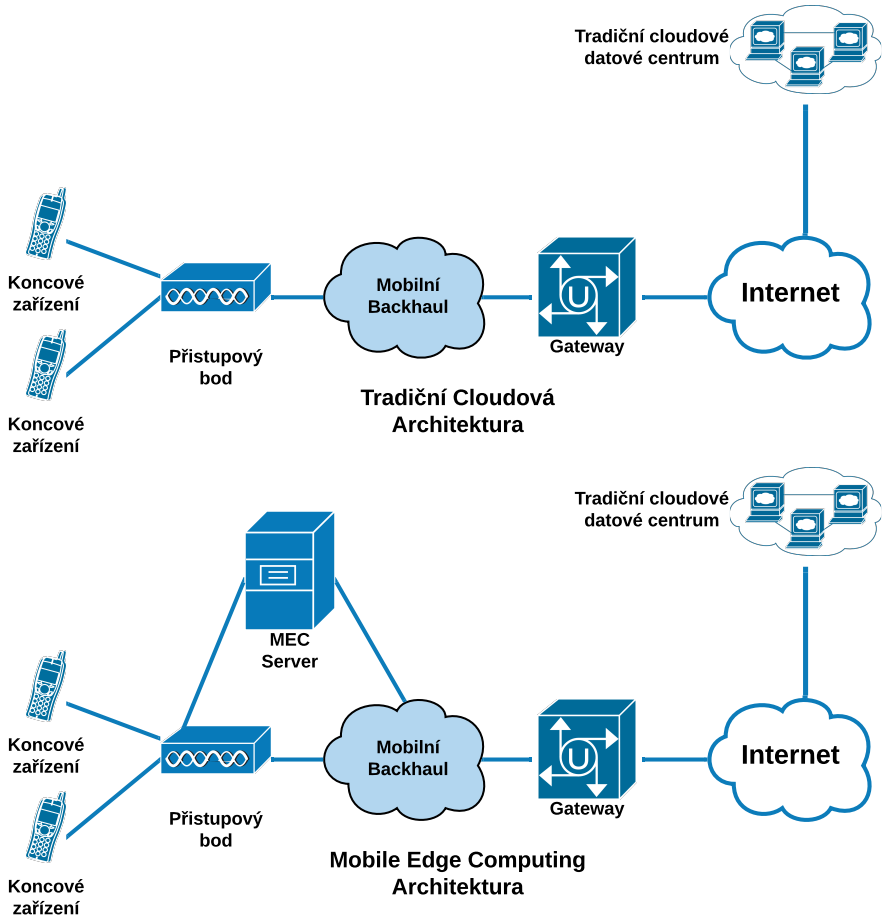
1. Analyzovat a prozkoumat problematiku Edge Computingu.
2. Návrh řešení dílčích problémů Edge Computingu, které zahrnují následující body:
   - Optimalizace doby odezvy a nestabilita internetového připojení.
   - Multi-tenance v edge řešení a přechod od modelu klient-server k modelu multifunkčního edge.
   - Standardizované API pro orchestrace edge aplikací, jež existují ve veřejných cloudech a datových centrech.
   - Stanovení přínosu nové Edge Computing platformy pro podnikový management z hlediska výkonnosti doručování softwaru.
3. Návrh samotné Edge Computing platformy včetně jednotlivých modulů a její ověření z pohledu informačního managementu, tedy technického a podnikového přínosu v rámci případové studie nebo laboratorního měření.

Disertační práce obsahuje řadu cizích výrazů a anglicismů, které jsou velmi těžko přeložitelné. Tyto termíny jsou vysvětleny převážně v poznámkách pod čarou anebo v seznamu zkratek tak, aby usnadnili čtení samotné práce.

3 Analýza současného stavu

V rámci analýzy současného stavu v oblasti Edge Computingu je tato kapitola rozdělena do čtyř částí. První z nich je průzkum aktuálního stavu problematiky na poli výzkumu. Zde se práce zabývá motivací k Edge Computingu. Poté navazuje kapitola 3.2 rozebírající současné standardy v oblasti výzkumu. V této části jsou detailněji představeny 3 standardy na poli Edge Computingu. V kapitole 3.3 jsou potom rozebrány výzvy a příležitosti Edge Computingu, kde je hlavním cílem nastínit směřování výzkumu disertační práce. Následně jsou v předposlední kapitole 3.4 představena a porovnána existující dostupná řešení pro Edge Computing, aby bylo možné potvrdit vymezené problémy a příležitosti. V kapitole 3.5 je provedeno shrnutí této oblasti a nastínění potřeby pro vytvoření nové Edge Computing platformy.

Revoluce mobilních služeb a Cloud Computingu (CC) v poslední době přitahovala značnou pozornost ze strany průmyslu i akademické obce. CC poskytuje zpracování a uchovávání dat spíše na úrovni centrálního cloudu než v rámci koncových zařízení uživatelů. Tento způsob práce s daty se stává problematickým především pro aplikace, které jsou citlivé na dobu odezvy. Příkladem mohou být aplikace z oblasti rozšířené reality nebo connected cars (Ashton, 2019), které jsou podrobněji rozvedeny v následující kapitole. K překonání těchto omezení byl zaveden Edge Computing (EdgeC), jehož úkolem je zpracování, ukládání a síťové optimalizace na tzv. okraji sítí (pevných nebo mobilních). Okrajem sítě jsou myšleny například rádiové přístupové sítě (RAN) nebo koncová místa uživatelů jako například pobočky maloobchodu. Cesta EdgeC začala především v důsledku vzestupu internetu věcí (IoT), kde dochází ke generování velkého množství nezpracovaných dat a ta mohou mít různé požadavky. Právě k překonání těchto výzev přichází různé skupiny průmyslu a vědeckých týmů s několika návrhy v rámci stejného paradigmatu zvaného EdgeC (Ullah, Ahmed, & Kim, 2018). Nicméně CC a EdgeC nejsou nezávislé a mají silnou korelaci (Amadeo, Campolo, Molinaro, & Ruggeri, 2018). Obrázek 3.1 zachycuje základní rozdíl mezi tradiční cloudovou architekturou a Mobile Edge Computing architekturou.



Obrázek 3.1: Porovnání Cloud a Mobile Edge Computing architektury, převzato z (Nunna & Ganesan, 2017)

Jak již bylo řečeno, myšlenka (EdgeC) pochází z (CC), který se dále rozvinul směrem k mobilnímu Cloud Computingu (MCC). Ten nabízí cloudové zdroje na tzv. okraji sítě se snahou o co nejnižší latenci a vysokou propustnost (Shahzadi, Iqbal, Dagiuklas, & Qayyum, 2017). Mobile Edge Computing (MEC) si v posledních letech získal pozornost mnoha vědců, díky čemuž můžeme najít výstupy týkající se: standardizace klíčových rozhraní pro MEC (ETSI, 2020), design aplikací vyžadující velmi nízkou dobu odezvy (Patel, Naughton, Chan, Sprecher, & Abeta, 2014), (ETSI, 2016) nebo modelování velkokapacitních datových center s mikro- datovými centry na edge sítích (Bahl, 2015). Velkým problémem poslední doby se stávají aplikace náročné na výpočetní výkon, protože spotřebovávají značné množství energie.

Pokroky okolo CC umožnily poskytovat infrastrukturu, platformu a software jako

službu koncovým uživatelům z libovolného počítače s pevným, nebo bezdrátovým připojením k internetu. EdgeC může tyto služby rozšířit i na mobilní zařízení. Vzhledem k tomu, že již dnes existují miliardy mobilních zákazníků po celém světě, EdgeC má potenciál výrazně zasáhnout bezdrátový průmysl v naší společnosti. Provoz populárních mobilních aplikací (např. streamování, rozšířená realita, online hry atd.) závisí především na bezdrátových sítích (např. WiFi, 3G, 4G, 5G atd.), které přenášejí data mezi cloudem a mobilními zařízeními. Ve srovnání s pevnými sítěmi mají bezdrátové sítě omezenou šířku pásma. Navíc při růstu počtu mobilních zařízení může být šířka pásma, která je k dispozici pro každé zařízení, výrazně snížena. Společně se sníženou propustností také roste latence sítě, která výrazně ovlivňuje dobu odezvy uživatelských aplikací. Hlavními cíli je tedy představit scénáře spojené s Edge Computingem, popsat nejnovější pokroky a diskutovat o budoucích výzvách ve výzkumu těchto technologií.

3.1 Motivace k Edge Computingu

Jak již bylo zmíněno, s příchodem Cloud Computingu došlo v posledních 3 letech ke konsolidaci výpočetního výkonu a k efektivnějšímu zpracování dat na základě většího výkonu a flexibility jejich správy. Nicméně koncová zařízení vytváří více dat a síť se tak stává stále přetíženější. Dalším faktorem je také změna směru toku dat. Zatímco poslední dekádu koncoví uživatelé více přijímali než odesílali, nyní je tomu u některých koncových zařízení naopak (Shi & Dustdar, 2016).

Například se můžeme podívat na kameru uvnitř autonomního auta, která zachycuje obrovské množství videozáznamu. Ten musí být zpracován v reálném čase, aby provedl správné rozhodnutí při řízení. Pokud by zařízení posílalo všechna data do cloudu ke zpracování, doba odezvy by byla příliš dlouhá. Navíc větší množství aut v jedné lokalitě by výrazně snížilo propustnost a spolehlivost sítě. Zpracování dat přímo v krajním bodě sítě by zajistilo kratší dobu odezvy, efektivnější zpracování a snížení zátěže sítě. Nedávné práce na mikro datacenterech (malé modulární datové centrum určené k optimalizaci výkonu síťových zařízení prostřednictvím cloudu) (Greenberg, Hamilton, Maltz, & Patel, 2008) a tzv. Cloudlet (malé cloudové datové centrum umístěné přímo na síťovém edge sloužící k práci s mobilními zařízeními) (Satyanarayanan, Bahl, Caceres, & Davies, 2011) tyto problémy velmi dobře představily.

Jako reakce na tyto problémy je tedy počátečním cílem EdgeC přizpůsobit CC mobilnímu prostředí tak, aby mohla být data zpracovávána kdykoli a kdekoli mimo mobilní zařízení (Vaquero & Rodero-Merino, 2014), (K.-C. Li, Li, & Shih, 2019). Některé z nejdůležitějších problémů souvisejících s EdgeC zahrnují (1) latenci, (2) sníženou propustnost sítě či (3) mobilitu uživatelů. Navzdory pokrokům v oblasti chytrých telefonů mají telefony stále velmi omezené možnosti zpracování a životnost baterie. Tyto

limity jsou stále viditelnější s rostoucí poptávkou po energeticky náročných aplikacích, jako jsou video streaming nebo 3D hry. V (Tran, Hajisami, Pandey, & Pompili, 2017) byl popsán MEC jako nově vznikající paradigma, které poskytuje prostředky pro výpočet, ukládání a vytváření sítí přímo uvnitř edge mobilní sítě RAN. Přípravy na nasazení sítí 5G vyvolaly rozhovory o problémech, které je třeba vyřešit, aby nedošlo k výraznému snížení kvality těchto aplikací, jež vyžadují nízkou latenci a data v reálném čase, aby dokázaly efektivně využít všechny dostupné funkce (Hassan, Yau, & Wu, 2019). Výzkum provedený (Wang, Han, Du, & Rodrigues, 2015) zjistil, že stávající cloudová infrastruktura nemůže vyřešit tento problém. V (Ahmed & Rehmani, 2017) vysvětlili, že klíčový problém, jenž se EdgeC snaží vyřešit, je (1) snížit propustnost a (2) latenci tak, aby se zlepšila kvalita Quality of Experience (QoE). Ta se měří uživatelskou zkušeností s danou aplikací. Toho by bylo dosaženo právě umístěním cloudové infrastruktury blíže ke koncovému uživateli. V (Clinch, Harkes, Friday, Davies, & Satyanarayanan, 2012) se zase ukázalo, že umístění Cloudlets blíže k uživateli zlepšuje běh aplikací vyžadujících nízkou latenci.

Obecně můžeme říci, že trend budování cloudové infrastruktury na edge sítích se očekává i v dalších letech. Podle (Tran et al., 2017) jsou největší výzvy a otevřené otázky spojené s MEC - interoperabilita dat, správa zdrojů, jejich orchestrace, zabezpečení a automatické zjišťování dat. Přibližme si nyní scénáře, ve kterých může být MEC velkým přínosem pro zlepšení výkonnosti (ETSI, 2016).

3.1.1 Rozšířená realita

Rozšířená realita (AR) spojuje pohled na reálný svět a počítačem generované senzorické vstupy, jako jsou grafika, GPS data, zvuk a video (Bachhuber, Martinez, Pries, Eger, & Steinbach, 2019). AR umožňuje uživateli vidět skutečný svět s virtuálními objekty, které se vrství na sebe společně s reálným světem. Tyto informace o okolním skutečném světě uživatele se stávají interaktivními a digitálně manipulovatelnými. EdgeC může být zde použit pro generování a vykreslování. Požadované zpracování lze provádět na serveru EdgeC namísto serveru v cloudu, a to právě kvůli požadavkům na vysokou rychlost zpracování a nízké doby odezvy.

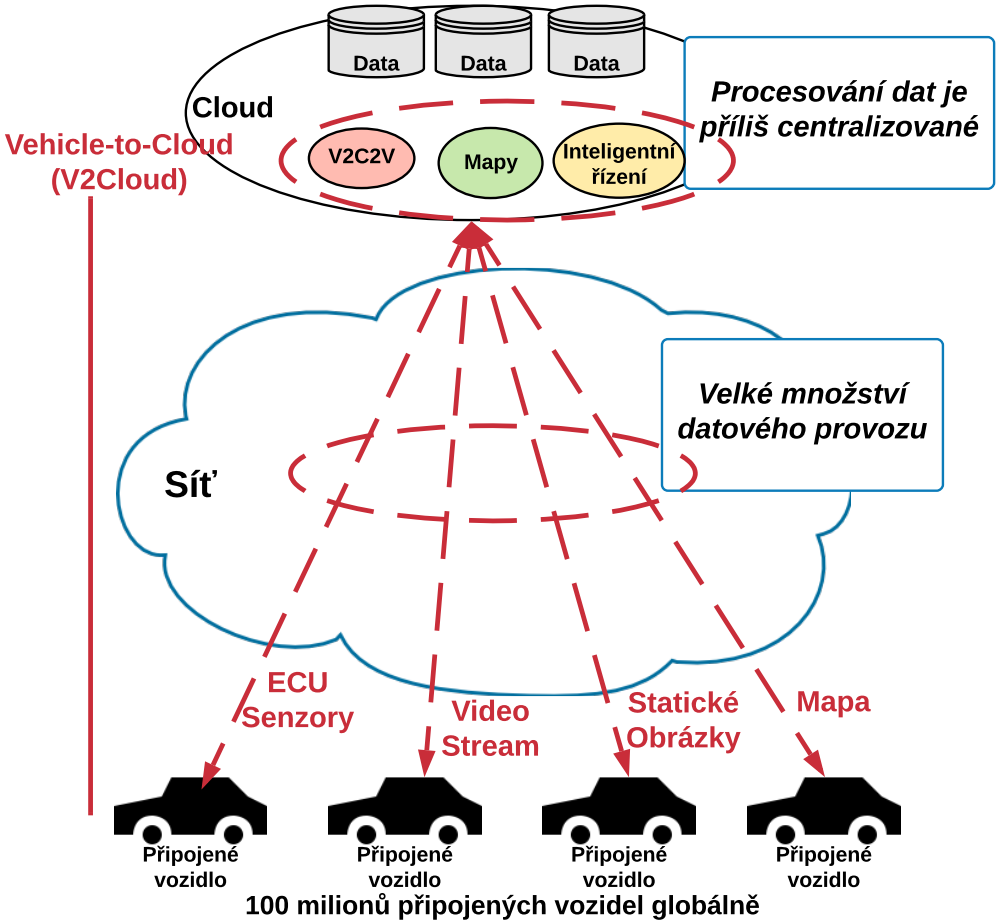
3.1.2 Connected Cars

Nedávno se zvýšil počet připojených vozidel (Connected Cars/Vehicles) podporujících komunikaci Vehicle-to-everything (V2X), kde se např. informují vozidla o podmínkách silničního provozu (predikce trasy díky analýze obrazu a videa), nebo existují různé varovné aplikace týkající se bezpečnostních informací, které by mohly ovlivnit vozidlo. Navíc rozmísťování tzv. Roadside Units (RSU) je zaměreno na zvýšení efek-

tivnosti a pohodlí V2X aplikací (Zhuang, 2017). V2X RSU je komunikačním rozhraním mezi silniční infrastrukturou a vozidly. Přístroje jsou přímo připojeny k semaforu nebo rychlostním radarům a vysílají relevantní informace prostřednictvím bezdrátové technologie 5.9GHz.

Vzhledem k tomu, že počet připojených vozidel se zvyšuje a případy použití se rozrůstají, objem dat se bude dále zvyšovat společně s potřebou minimalizovat latenci a optimalizovat kvalitu služby QoE. EdgeC může být velmi užitečný právě pro přesun V2X aplikací, dat a služeb z cloudu přímo na síťové edge (např. RSU), což by pomohlo přiblížit zpracování aplikačních dat a analýzy i přímo k vozidlům. Tento posun by umožnil zrychlit komunikaci aplikací mezi vozidly (K. Zhang, Mao, Leng, He, & Zhang, 2017). Mobile EdgeC aplikace by mohly běžet jako vysoce distribuované RSU podporující V2X komunikaci a pomáhat tak posílat užitečné informace vozidlům v okolí bez zpoždění. Tato komunikace by pomohla řidičům včas reagovat na události silničního provozu, aby se předešlo nehodám a zlepšila se bezpečnost na silnici.

Obrázek 3.2 zachycuje problémy použití existujících technologií pro Connected Cars podle (*AECC: General Principle and Vision*, 2018). Jejich předpověď počítá se zpracováním 20GB měsíčně pro každé vozidlo, což je při například třech milionech připojených vozidel okolo 60 petabajtů dat, která přitečou do cloudu každý měsíc. Při předpokladu, že maximální transakční zpracování v cloudu je 10 GB za vteřinu, pouze jejich zpracování by trvalo 70 dní. Do roku 2025 má být připojeno 100 milionů vozidel globálně, což ukazuje na jasné mezery v současném model V2Cloud a nutný přechod do více distribuovaného modelu, jakým je EdgeC.



Obrázek 3.2: Problémy nasazení existujících technologií, převzato z (*AECC: General Principle and Vision*, 2018)

3.1.3 Internet of Things (Internet věcí)

IoT je síť, která spojuje fyzická zařízení, senzory, vozidla a každodenní elektronické objekty zabudované do softwarů, ovladačů a senzorů, které sbírají a vyměňují data. Nicméně IoT zahrnuje také připojení a sítě mezi dopravními, komunitními a společenskými infrastrukturními službami (Gubbi, Buyya, Marusic, & Palaniswami, 2013). Například společnost IDC odhaduje, že do roku 2025 budou IoT zařízení produkovat 79,4 zettabajtů dat (IDC, 2019). IoT patří k nejnovějším technologiím a je stejně důležitý jako internet. Jedná se o síť, která spojuje všechny věci s internetem tak, aby si mohla jednotlivá zařízení vyměňovat informace skrz definované protokoly pomocí identifikace, lokalizace, monitorování a správy zařízení (S. Chen, Xu, Liu, Hu, & Wang, 2014). Jinými slovy, internet už není vázán pouze na stolní počítače, ale jde do světa „jiných"

věcí (Gubbi et al., 2013). Obrovské množství dat generovaných tímto procesem by bylo nejlepší ukládat do cloudu, který sníží náklady a komplexitu týkající se správy hardwaru, avšak jeho kapacita není nekonečná (Botta, de Donato, Persico, & Pescape, 2014). Podobně jako v oblasti týkající se připojených vozidel i zde je potřeba dosahovat nízké doby odezvy. IoT zařízení jsou připojena různými způsoby jako 3G, 4G, 5G, Wi-Fi nebo pomocí rádiové technologie (Corcoran & Datta, 2016). Obecně jsou zprávy těchto zařízení malé, šifrované a používají různé formáty protokolů (např. MQTT, CoAP atd.). Vzniká zde potřeba agregovat komunikaci v určitých místech, kde by bylo možné tyto různé protokoly spravovat a dále distribuovat jejich zprávy nebo provádět analýzy ze získaných dat IoT aplikací (Dastjerdi & Buyya, 2016). EdgeC se proto nabízí jako ideální řešení těchto problémů.

3.1.4 Optimalizace médií v Edge Computingu

Distribuované edge cloudy byly navrženy a vyvinuty pro podporu mediálních služeb v heterogenních bezdrátových a konvergovaných sítích. EdgeC podporuje uživatelské aplikace, které se potřebují vyrovnat s problémy, jako jsou mobilita uživatelů nebo omezené síťové zdroje. EdgeC také pomáhá vytvářet správu pro aplikace s médii, která potřebují doručit obsah uživatelům na základě inteligentního rozpoznání dostupné komunikační platformy a schopností koncového zařízení. Tento typ aplikací tedy vyžaduje funkce, jako je přizpůsobení se danému kontextu zařízení klienta pomocí renderování a ukládání do mezipaměti.

Cílem tohoto scénáře je optimalizovat kvalitu služby QoE pro videoaplikace přes rádiovou síť. Toho lze dosáhnout odhadnutím propustnosti z rádiové linky rozhraní nebo z informací získaných rádiovou analýzou. EdgeC může být použit ke zkvalitnění QoE pro uživatele tím, že přizpůsobí video kódování na úrovni aplikace tak, aby odpovídalo odhadované kapacitě rádiové linky (S. Chen, Wang, & Pedram, 2013). Edge cloud by v tomto případě mohl být adaptován síťovými operátory buď na přístupových sítích (např. bezdrátové přístupové body), nebo na agregačních bodech, které jsou propojeny s hlavními sítěmi prostřednictvím páteřní sítě. Hlavní problém je v tomto případě uživatelská mobilita, která ovlivní poskytování služeb v případě, že je uživatel připojen na AP, kde provozovatel nemá pokrytí. V takovém případě může operátor využít dostupné výpočetní zdroje třetích stran.

3.2 Edge Computing standardy

V této části jsou zpracovány dostupné standardy v oblasti Edge Computingu, jež jsou rozděleny do tří kategorií dle přístupu na MEC, Cloudlet a Open Fog Consortium.

3.2.1 MEC

Organizace ETSI (The European Telecommunications Standards Institute) představila Mobile Edge Computing (MEC) ve specifikaci (ETSI, 2020), která má za cíl přiblížit výpočetní zdroje k rádiovým sítím 4G a 5G. Tímto způsobem se snaží doručit funkcionality Cloud Computingu a IT služeb přímo na okraj sítě. Toto prostředí je charakterizováno velmi nízkou latencí a vysokou propustností, které jsou důležité pro provozované aplikace. Od roku 2017 je průmyslová skupina ETSI MEC přejmenována z „Mobile Edge Computing" na „Multi-Access Edge Computing", aby lépe odrážela rostoucí zájem o MEC nejen skupinou mobilních operátorů. MEC má za cíl snížit tlak na sítě přesunem zdrojů z cloudu do mobilního edge (Beck, Werner, Feld, & Schimper, 2014). V rámci (Beck & Maier, 2014a) představují návrh plně virtualizové MEC infrastruktury. Dalším příkladem je návrh rozložení výpočetních algoritmů, kterým se zabývají v (X. Chen, Jiao, Li, & Fu, 2016). Vytváření a nasazení EdgeC služeb se potýká s problémy okolo škálovatelnosti, vysoké dostupnosti, odolnosti proti chybám a robustnosti. Jako jedno z řešení těchto problémů byl představen koncept SEcS (Scalable Edge Computing Services), detailně vysvětlený v (Grieco, Malandrino, & Scarano, 2006).

Multi-access MEC architektura je navržena tak, aby řešila problémy s latencí a šířkou pásma pro aplikace, jako jsou lokalizace videa analytických služeb, internetová komunikace (IoT), rozšířená realita, optimalizovaná distribuce a ukládání obsahu médií do mezipaměti. Sem patří také mnoho dalších scénářů z oblasti chytrých měst, zdravotnictví, chytrého zemědělství apod. Podívejme se na některé relevantní výzkumy z prostředí MEC. Například v (Dey, Mukherjee, Paul, & Pal, 2013) navrhli schéma, které se zabývá nepředvídatelností dostupnosti výpočetních zdrojů v EdgeC pomocí úloh distribuovaných na nevyužitých serverech. V rámci (Desertot, Escoffier, Lalanda, & Donsez, 2006) zase vysvětlují, že autonomní výpočetní techniky jsou základním prvkem pro dynamické řízení edge zařízení. Novou architekturu ke snížení latence v prostředí MEC představují v (Zhu, Luo, Wang, & Li, 2011). Jakým způsobem provést migraci běžící aplikace z VM nebo kontejneru z cloudu přímo do edge, se zase zabývají v (Machen, Wang, Leung, Ko, & Salonidis, 2016). Nový koncept WiCloud založený na SDN/NFV má podle (H. Li, Shou, Hu, & Guo, 2016) řešit poskytování edge sítí. V (Ahmed, Gani, Sookhak, Hamid, & Xia, 2015) se zabývají koncepty a srovnávacími studiemi týkajícími se mobilních aplikací v MCC. Detailní porovnání a analýza rámců pro tvorbu aplikací v MCC je shrnuta například v (Ahmed, Gani, Khurram Khan, Buyya, & Khan, 2015). Jak je vidět, MEC je velmi aktuální téma výzkumu.

3.2.2 Cloudlets

Termín „cloudlet" byl vytvořen vědcem na univerzitě Carnegie Mellon, kde byl jeho prototyp vyvinut jako součást výzkumného projektu (Elijah, 2017). Cloudlety jsou navrženy tak, aby podporovaly aplikace pro mobilní zařízení, která jsou náročná na výpočetní zdroje jako např. aplikace pro rozšířenou realitu, cloudové hry, Wearables kognitivní systémy typu Google Glass, Apple Siri, Google Now a mnoho dalších. Prakticky sem patří jakákoli aplikace, která vyžaduje částečné zpracování náročných úkolů přímo na mobilním zařízení, aby bylo dosaženo požadované kvality a dobré uživatelské zkušenosti. Hlavním důvodem je snížení latence při komunikaci a větší výkonnost při provádění náročných operací. Hlavní motivace pro vytvoření Cloudlet pochází od internetové komunity, která se zabývá nastavením limitů pro výpočetní zdroje na mobilních zařízeních.

Cloudlet představuje střední vrstvu v třívrstvé hierarchii (3-tier), jako je například mobilní zařízení, cloudlet a cloud. Cloudlet lze považovat za lokální datové centrum v krabičce, umožňující poskytovat cloudové služby s vysokým výkonem a obsloužit více uživatelů zároveň. Navíc se zabývá problémy s velkou latencí WAN sítí, nižší propustností a přidruženými problémy, které se odráží v nákladech (K. A. Khan, Wang, Luo, Wang, & Grecos, 2014). Cloudlet vytvořil díky zájmu klíčových průmyslových hráčů (např. Intel, Vodafone, Cisco) open source iniciativu Open Edge Computing (OEC). Ta definovala open source API rozhraní jako rozšíření technologie OpenStack a propagovala tak tento přístup. Hlavním cílem bylo zapojit širší odvětví IT a telekomunikací, aby bylo možné synchronizovat práce mezi organizacemi jako ETSI MEC a OPNFV. Pilotní projekt Elijah na Univerzitě Carnegie Mellon byl rozšířen na OpenStack++. Jeho cílem je poskytnout cloudletovou knihovnu založenou na modifikovaném QEMU s integrací do platformy OpenStack (*Open Edge Computing*, 2018). Další zajímavou prací je mesh cloudová architektura navržená v (K. A. Khan et al., 2014), která se skládá z Cloudlet, internetového cloudu a bezdrátových mesh sítí. Dalším příkladem je experimentální koncept navržený v (K. Khan, Wang, & Grecos, 2012), kde se snažili implementovat privátní Cloudlet a bezdrátovou mesh síť. Ta je zde schopna automaticky navazovat síťovou konektivitu mezi více zařízeními s funkcí automatické obnovy při jakémkoli výpadku.

Podívejme se teď na různé vědecké práce týkající se oblasti Cloudlet. Cloudlet architektura představená v (Verbelen, Simoens, Turck, & Dhoedt, 2012) spravuje aplikace na úrovni komponent, jež jsou distribuovány napříč dynamickými cloudlety, což dovoluje uživateli připojit se, nebo opustit kterýkoli z nich. V práci (Jararweh, Tawalbeh, Ababneh, & Aldosari, 2013) se zase zabývají cloudletem založeným na MCC tak, aby snížili latenci a spotřebu energie koncového zařízení. Pro MCC (Whaiduzzaman, Gani, & Naveed, 2014) je zase navržen koncept zlepšení výkonnosti nazvaný Performance Enhan-

cement Framework for Cloudlets (PEFC). Jakým způsobem spravovat centralizovanou cloudlet architekturu je vysvětleno v publikaci (Routaib, Badidi, Elmachkour, Sabir, & ElKoutbi, 2014). Závislost výkonnosti cloudletu na mobilitě uživatelů zase zkoumali v pramenu (Y. Li & Wang, 2013). Další systém založený na cloudletu je navržen v (Whaiduzzaman et al., 2014), kde se zaměřují na zlepšení výkonu v mobilním cloud computingu. Cloudlet je v tomto případě instalován společně s přístupovým bodem AP, což umožňuje přímý přístup mobilních zařízení. Tato mobilní zařízení se připojují k bezdrátovému cloudu skrze Wifi (Jararweh et al., 2013). Pro minimalizaci zpoždění a spotřeby energie mobilních uživatelů je navržena cloudlet infrastruktura v (Jararweh et al., 2016). V (Niyato, Wang, Joo, Han, & Kim, 2015) zase zavedli přístup k řízení energie pro mobilní a kapesní zařízení. Vědci a námořní sbor pracují společně na koncepci tzv. „taktického cloudletu", aby dokázali zavést distribuovanou koncepci cloud computingu při scénáři vzdáleného válečného bojiště přímo na nepřátelském území. Tento model by byl užitečný např. během válečné mise nebo při obnově po havárii, kde se požadavky na komunikaci rychle mění a vyžadují vyšší výpočetní výkon (OECI, 2017). Taktické cloudlety jsou navrženy také v rámci (Lewis, Echeverria, Simanta, Bradshaw, & Root, 2014) k podpoře cyber foraging, kde musí být náročné výpočetní úlohy prováděny mimo koncová zařízení. Posledním zajímavým zdrojem je (Liu, Lee, & Zheng, 2016), jenž řeší strategii přidělování více zdrojů mezi cloudlet a mobilní zařízení, což zvyšuje kvalitu poskytované služby (QoS).

3.2.3 Open Fog Consorcium

Fog Computing je koncept, který představila společnost Cisco v roce 2011, aby splnila požadavky různých odvětví jako například IoT, internetu všeho (IoE) nebo Internet of Me (IoM). Klasické paradigma cloud computingu může jen sotva zajistit nízkou dobu odezvy, podporu mobility a lokalizaci klientů. Pro řešení těchto problémů byl vytvořen Fog Computing, který zlepšuje kvalitu služeb (QoS) pro aplikace vyžadující právě nízkou latenci, lokalizaci, streamování v reálném čase apod. (Stojmenovic & Wen, 2014). Hlavní motivací je zmírnit nevýhody cloud computingu, kam patří právě vysoká latence a nedostatečná propustnost na sítích WAN. Fog Computing zavádí decentralizovanou výpočetní infrastrukturu tak, aby byly výpočetní zdroje a aplikační služby distribuované napříč lokalitami co nejefektivněji. Hlavní důraz Open Fog Consorcia je kladen na definování horizontální architektury na systémové úrovni, která dokáže distribuovat služby a výpočetní zdroje, úložiště a sítě kontinuálně kdekoli mezi cloudem a „věcmi" (angl. Things z IoT) (Bonomi & Milito, 2012). Tímto dokáže dostat data blíže k uživateli, čímž snižuje latenci a zlepšuje kvalitu QoS (T. H. Luan et al., 2015). Umožňuje také přidat informace o lokalizaci, kontextu a mobilitě koncového zařízení (Stojmenovic & Wen, 2014). Používá se také technika Decoy Information na detekci

škodlivých útoků, které jinak nemohou být řešeny pomocí tradičních bezpečnostních opatření. Příkladem může být útok zevnitř zasíláním falešných dat, která se v případě standardního Cloud Computingu tváří jako skutečná. Pomocí Decoy Information může být tedy bezpečnost vynucena na úrovni Fog Computingu namísto cloudu (Suryawanshi & Mandlik, 2015). Fog také poskytuje vysokou kvalitu streamingu přes přístupové body a proxy k mobilním uzlům včetně pohybujících se vozidel (Aazam & Huh, 2014). To je především vhodné pro ty typy aplikací, které vyžadují nějakou formu předpovědi nebo nízké doby odezvy, jako jsou videokonference a hry (Bonomi, Milito, Natarajan, & Zhu, 2014). Fog architektura je podrobněji vysvětlena v (T. Luan, Gao, Li, Xiang, & Sun, 2015) a případně fog založený na rádiových přístupových sítích (F-RAN) zase v (Peng, Yan, Zhang, & Wang, 2015).

Ve fog computingu se cloudové zdroje jako výpočetní zdroje nebo zdroje úložiště přesouvají do okraje sítě, kde se i samotné síťové prvky mohou stát virtualizovanou infrastrukturou (Vaquero & Rodero-Merino, 2014) a služby mohou být provozovány přímo z koncových zařízení jako např. set-top boxy nebo AP (Stojmenovic & Wen, 2014). Navíc mnoho heterogenních decentralizovaných zařízení komunikuje a spolupracuje navzájem, čímž dokážou zpracovávat úlohy a ukládat data pomocí sítě bez nutnosti zásahu třetích stran (Vaquero & Rodero-Merino, 2014). Fog computing tedy zajišťuje virtualizovanou platformu, která poskytuje výpočetní, síťové služby a služby úložiště mezi cloudovými datovými centry a koncovými zařízeními (K.-C. Li et al., 2019). Podporuje tak více služeb a aplikací, kde je vyžadována nízká latence. Fog edge servery mají dostatečný výpočetní výkon pro usnadnění zpracování uživatelských úloh z koncových zařízení. Stejný koncept provádění výpočtů přímo na edge se používá uvnitř cloudu ke snížení doby odezvy mezi více zařízeními.

Konsorcium ETSI-MEC bylo vytvořeno kvůli sjednocení IT cloudu a telekomunikačního průmyslu skrze MEC standard, který poskytuje IT a cloud computing služby v rádiových sítích RAN prostřednictvím mobilního orchestračního API (ETSI, 2020).

3.2.4 Shrnutí dostupných standardů

Jak je vidět, fog computing, MEC a cloudlet se v moha vlastnostech překrývají a existujte zde vzájemná synergie. Podívejme se na následující shrnutí jejich překrývajících se částí podle (Borcoci, 2018):

- Lokalita a přístup
  - Všechny tři jsou geo-distribuované a obvykle umístěné mezi koncovým zařízením a hlavním datovým centrem,
  - cloudlet a dokonce některé fog computing nody mohou běžet v terminálových zařízeních,

  - obvykle jsou umístěny u AP, agregačních bodů, směrovačů, přepínačů nebo síťových bran,
  - obecně všechny počítají s bezdrátovým přístupem, avšak mohou být připojeny i klasicky pevnou linkou,
  - všechny tři se snaží snížit dobu odezvy.
- Všechny tři se snaží o multi-tenanci[1] aplikací v edge použitím virtualizace vycházející z IaaS cloudové platformy.
- Všechny tři typicky bývají rozšířením cloud computingu.
- IoT je hlavní hnací silou všech tří přístupů.
  - Především služby vyžadující distribuované úložiště a výkon.
- Mají společnou mobilitu koncových zařízení.
- Všechny tři mají povědomí o kontextu aplikací.

Následující tabulka 3.1 zobrazuje, v čem se liší jednotlivé přístupy z obecného hlediska.

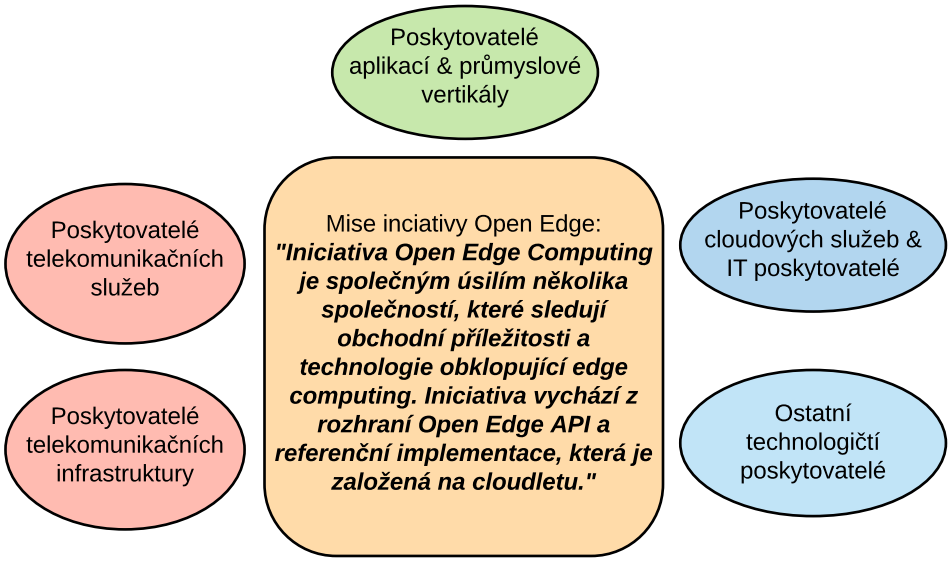
Položka	**Fog Computing**	**MEC**	**Cloudlet**
Použití virtuálního IaaS	ano	ano	ano
Podpora Multi-tenance	ano	ano	ano
Umístěné mezi DC a koncovým zařízením	ano	ano	ano, může běžet i v autě, vlaku, letadle.
Rozšíření cloudu	ano	ano, ale nemusí být	ano
Hnací síla	IoT	Nízká latence, kontext zařízení	Tactile Internet
Používaný s	Bezdrátový přístupem	Bezdrátovým přístupem	Bez/drátový přístupem
Zaměřeno na online analytiku	ano	ne	ne, ale může být rozšířeno
N vrstvená architektura	3 vrstvy	2 nebo 3 vrstvy	3 vrstvy

Tabulka 3.1: Porovnání přístupů Edge Computingu. Převzato z (Schuster & Ramchandran, 2016)

Intel, Huawei, Vodafone a CMU založily iniciativu Open Edge Computing (OEC), která byla vytvořena za účelem propagace cloudletu jako technologie budoucnosti. V

[1] Z anglického Multitenancy týkající se softwarové architektury, ve které jedna instance softwaru běžící na jednom serveru obsluhuje více skupin uživatelů (tenant) (Krebs, Momm, & Kounev, 2012).

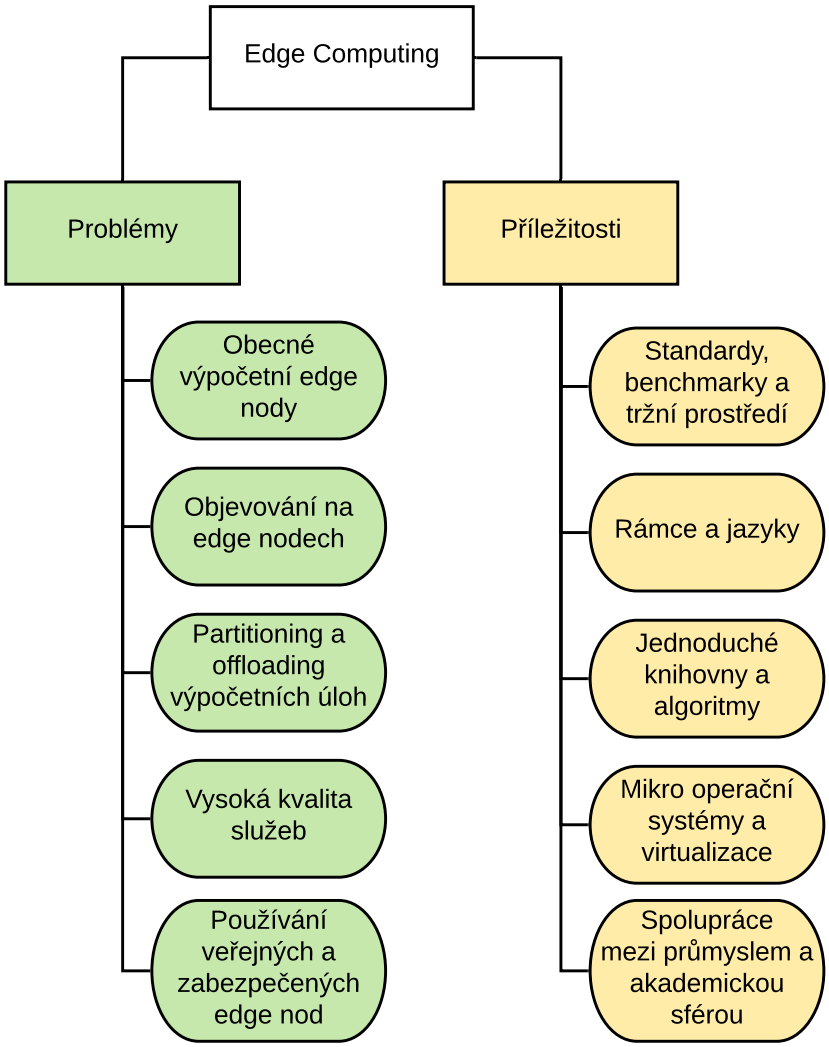
roce 2015 se připojili i další vědci právě díky přesahu charakteristik z MEC. Dnes se snaží synchronizovat práci s dalšími iniciativami jako ETSI ISG MEC nebo OPNFV. OEC tedy sdružuje všechny tři přístupy a snaží se definovat referenční platformu pro Edge Computing jako takový. Na obrázku 3.3 je vidět mise této iniciativy spojit IT, cloud computing i telekomunikační průmysl a vytvořit jednotnou platformu (*Open Edge Computing*, 2018).



Obrázek 3.3: Mise Open Edge Computingu, převzato z (*Open Edge Computing*, 2018)

3.3 Problémy a příležitosti Edge Computingu

V této části jsou představeny některé výzvy neboli problémy a příležitosti Edge Computingu zvolené na základě relevantních vědeckých pramenů představených dále v textu. Obrázek 3.4 shrnuje tyto výzvy a příležitosti, které jsou následně podrobněji rozebrány.



Obrázek 3.4: Grafické znázornění výzvev a příležitostí Edge Computingu, částečně inspirováno (Varghese, Wang, et al., 2016)

3.3.1 Problémy

Edge Computing je stále ještě v plenkách a rámec nebo standardy, které by ho přesně definovaly, stále ještě nejsou k dispozici. Ty budou muset splňovat požadavky pro aplikace zpracovávající dotazy v reálném čase přímo na edge sítě. Současné cloudové standardy, jakými jsou Amazon Web Service (*Amazon Web Services (AWS) - Cloud Computing Services*, 2019), Microsoft Azure *Microsoft Azure Cloud Computing Platform & Services* (2019) nebo Google App Engine (*App Engine – Build Scalable Web & Mobile Backends*

in Any Language, 2019) dokáží provozovat aplikace náročné na zpracování dat, avšak jejich okamžité zpracování a provoz přímo na okraji sítě je stále otevřenou oblastí výzkumu (W. Li, Lu, & Chen, 2015) (Hromic et al., 2015). Navíc bude potřeba správně definovat požadavky pro správu aplikací v edge lokalitách. Těm se odborně říká deployment strategie, které řeší, kde a jak rozmístit aplikační instance, případně politiky pro jejich připojení v edge lokalitách, dále také jak řešit různé typy edge zařízení a zohlednit všechny tyto informace při nasazení jednotlivých aplikací. Pro vytvoření takového rámce předpokládají například v (Varghese, Wang, et al., 2016), že bude nutno vyřešit následujících pět výzkumných výzev v oblasti hardwaru, middlewaru a softwaru.

Obecné výpočetní edge nody

V teorii může být edge computing aktivní na několika bodech, které se nacházejí mezi koncovým zařízením a cloudem jako například přístupové body AP, brány, základny, dopravní agregační body, směrovače, přepínače apod. Například základny, které obsahují Digital Signal Processors (DSPs), jsou přizpůsobené pro určitý typ aplikací. V praxi tedy nemusí být tyto základny vhodné pro zpracování analytických aplikací jednoduše proto, že DSP na to nejsou připraveny. Řada komerčních výrobců podnikla první kroky k realizaci edge computingu právě skrze softwarové řešení. Například softwarové řešení od Nokie (*Multi-access Edge Computing* (*MEC*), 2018) pro mobile edge computing (MEC) má za cíl umožnit běh edge computingu na základnách. Podobně řešení od společnosti Cisco nazvané IOx nabízí takové prostředí pro svoje integrované směrovače (Systems, 2019). Tato řešení jsou hardwarově specifická, a proto nemusí být vhodná v heterogenním prostředí. Jednou z výzev v oblasti softwaru bude tedy vyvinout řešení, které dokáže být nezávislé na typu HW a jeho prostředí.

Existuje výzkum, který se zabývá aktualizací zdrojů na edge nodech tak, aby bylo možné podporovat obecné výpočetní operace. Například bezdrátový domácí směrovač může být vylepšen, aby podporoval běh dalších typů aplikací (Meurisch et al., 2015). Intel Smart Cell Platform (*Network Edge Smart Cell Solutions Revolutionize Mobile Services*, 2018) používá virtualizaci pro umožnění běhu dalších typů aplikací. Výměna specifických DSP procesorů se srovnatelnými univerzálními procesory CPU dokáže nabídnout slušnou alternativu pro tento problém. Překážkou ovšem zůstávají potenciálně vysoké investice.

Objevování na edge nodech

Objevování zdrojů a služeb v distribuovaném výpočetním prostředí je oblast, která je relativně dobře prozkoumána. Existuje spousta technik integrovaných uvnitř monitorovacích řešení (Povedano-Molina, Lopez-Vega, Lopez-Soler, Corradi, & Foschini,

2013) (Chaves, Uriarte, & Westphall, 2011) (Montes, Sanchez, Memishi, Pérez, & Antoniu, 2013) nebo zprostředkovatelů služeb (Garg, Versteeg, & Buyya, 2013) (Ferrer et al., 2012) (Yin & Kosar, 2011). Techniky, jako je benchmarking, podporují rozhodovací metody dle dostupných zdrojů pro optimální přidělení úkolů, což výrazně zlepšuje celkový výkon.

Nicméně procházení síťových edge nodů vyžaduje tzv. mechanismus objevování (discovery mechanism), který dokáže najít a poskládat tyto zdroje do decentralizovaného cloudu. Tyto mechanismy nemohou být provedeny manuálně kvůli velkému množství zařízení. Kromě toho budou muset zajistit funkčnost napříč heterogenními zařízeními z různých generací, které rozhodně nebyly navrženy pro tento druh výpočetních úloh. Metody benchmarkingu budou muset být výrazně rychlejší při oznamování dostupnosti zdrojů. Tyto mechanismy musí umožnit bezproblémovou integraci serverů do výpočetního procesu na různých úrovních, aniž by zvýšily latenci nebo poškodily uživatelskou zkušenost. Tyto mechanismy se také budou muset spolehlivě a proaktivně vypořádat s chybami nebo výpadky jednotlivých edge nodů. Podobným problémem se zabývali například v rámci (Zavodovski, Mohan, & Kangasharju, 2018), kde představili vlastní metodu nazvanou eDisco. Ta funguje na principu DNS a snaží se využít existující protokoly tak, aby nebylo nutné nijak modifikovat již nasazenou infrastrukturu.

Partitioning a Offloading výpočetních úloh

Vývoj distribuovaných výpočetních prostředí vedl k vytvoření mnoha technik, které mají za cíl snadnější rozdělení výpočetních úloh, které lze provádět na více geografických územích zároveň (W. Chen & Deelman, 2012) (Tang et al., 2015). Například ve výzkumech (Beck & Maier, 2014b) (Simoens, Herzeele, Vandeputte, & Vermoesen, 2015) rozdělují tento proces do různých výpočetních míst. Rozdělení těchto úloh je obvykle vyjádřeno explicitně programovacím jazykem nebo nástrojem pro správu.

Nicméně využití edge nodů k provozu výpočetních operací představuje výzvu nejen pro efektivní rozložení úloh, nýbrž také pro jejich automatizovaný výběr, aniž by bylo nutné explicitně definovat konkrétní lokalitu. Výpočty by mělo být možné provádět hierarchicky podle úrovní (nejprve datové centrum, pak edge nody, nebo nejprve edge nody a potom datové centrum) nebo distribucí napříč více edge nody současně. V podstatě zde vzniká potřeba rozvíjet plánovače[2], které dokáží distribuovat výpočetní úlohy napříč edge lokalitami a jejich servery.

[2] Z angličtiny scheduler

Vysoká kvalita služeb QoS a zkušeností (QoE)

Kvalita poskytovaná edge nody by měla být zachycena pomocí QoS a kvalita, která je poskytovaná koncovému uživateli skrze tzv. QoE. Jedním z principů, které budou muset být dodrženy na edge zařízení, je, aby nedocházelo k jejich přetížení náročnými aplikacemi (Baset, 2012) (Bui, 2015). Úkolem je zajistit, aby edge nody dosáhly vysoké propustnosti a spolehlivě poskytovaly cílové služby. Uživatel totiž očekává dodržení jejich SLA bez ohledu na to, zda byl edge node přetížen, či nikoliv. Když je například centrála přetížena, může to mít vliv i na službu poskytovanou koncovým zařízením, která je k ní připojena. Proto je nutná důkladná znalost časových špiček vytížení edge lokalit, aby mohly být výpočetní úlohy flexibilně naplánovány a distribuovány. Role definovaného rámce pro správu zde bude významná, avšak vyvolává otázky spojené s monitorováním a plánováním na úrovni infrastruktury a její samotnou aplikací.

Používání veřejných a zabezpečených edge nodů

Hardwarové prostředky, které jsou ve vlastnictví datových center, superpočítačových center a soukromých organizací využívajících virtualizaci, mohou být transformovány tak, aby poskytovaly dostupné výpočetní zdroje v edge computingu pro běh aplikací třetích stran. Má to však vlastní rizika, která jsou spojená s tímto modelem pay-as-you-go (placení za skutečně využité zdroje), a musí být detailně specifikována ve smlouvách o dostupnosti SLA (Varghese, Subba, Thai, & Barker, 2016).

Je-li však třeba použít alternativní zařízení, jako jsou přepínače, směrovače a základny pro veřejně přístupné edge nody, bude třeba řešit řadu dalších problémů. Za prvé bude nutné vyjasnit riziko spojené s veřejnými a soukromými organizacemi, které vlastní tato zařízení, a také s těmi, kdo tato zařízení bude používat. Za druhé, původní účel zařízení nemůže být nijak omezen v případě jeho zapojení do edge computingu. Ukázkovým příkladem je směrovač, který řídí internetovou komunikaci, tudíž nemůže dojít k degradaci jeho hlavní služby. Za třetí, multi-tenantnost na edge nodech bude možná pouze tehdy, pokud bude zajištěna striktní bezpečnost a izolace. Například virtuální kontejnery mohou být jednou ze zvažovaných technologiích, avšak je nutné ověřit jejich bezpečnost (Varghese, Akgun, Miguel, Thai, & Barker, 2014). Za čtvrté musí být uživateli edge nodu zaručena minimální úroveň služby SLA a jako poslední je třeba vzít v úvahu náklady na údržbu, vytížení nebo přenos dat atd., aby bylo možné vytvořit patřičné cenové modely a vytvořit tak smysluplný prodejní model.

3.3.2 Příležitosti

Navzdory výzvám a problémům, které vznikají při realizaci edge computingu, existuje také mnoho příležitostí pro akademický výzkum a rozvoj. Podívejme se na pět takových

příležitostí podle (Varghese, Wang, et al., 2016).

Standardy, Benchmarky a tržní prostředí

Edge computing může být skutečně poskytován stejnou formou jako veřejný cloud, pokud budou splněny a vydefinovány všechny vztahy a rizika s ním spojená. Jak již bylo zmíněno v předchozí kapitole, existuje několik iniciativ na standardizaci jako např. Institutes of Standards and Technology (NIST), IEEE (IEEE Standards Association), International Standards Organisation (ISO), Cloud Standards Customer Council (CSCC) a International Telecommunication Union (ITU).Tyto standardy však budou muset vzít v potaz všechny zúčastněné strany, jako jsou veřejné a soukromé organizace, které vlastní edge nody. Bude nutné vydefinovat sociální, právní a etické aspekty využívání těchto edge nodů, což určitě není snadný úkol, a bude to vyžadovat nemalé investice ze strany veřejných a soukromých organizací.

Standardy mohou být implementovány pouze tehdy, pokud jsou stanoveny metriky pro definici spolehlivosti. Příkladem mohou být tzv. Benchmark iniciativy pro cloud zahrnuté v iniciativě Standard Performance Evaluation Corporation (SPEC) a také v několika vědeckých pracích (Cooper, Silberstein, Tam, Ramakrishnan, & Sears, 2010) (Deelman et al., 2014) nebo (Baset, Silva, & Wakou, 2017). V dynamickém prostředí, jako je cloud, představují tyto srovnávací metriky nemalou výzvu. Jejich současný stav ještě není úplně dotvořený a je zapotřebí dalšího výzkumu, aby bylo možné poskytnout komplexní benchmarkové sady. Benchmarking na edge nodech bude proto ještě náročnější, ale otevírá se tím zajímavá možnost pro výzkum.

Použití edge nodů bude značně perspektivní ve chvíli, kdy budou všechny tyto vztahy a rizika vyjasněny. Podobně jako na trhu s cloudem bude možné nabízet různé konfigurace edge nodů v modelu pay-as-you-go. Nicméně předtím bude nutné provést výzkum okolo tvorby cen a jejich SLA. Zároveň bude nutné se podívat na způsob měření doručování softwaru v prostředí EdgeC, aby bylo možné efektivním způsobem vyvíjet a provozovat nové aplikace.

Rámce a jazyky

Existuje mnoho možností, jak spouštět aplikace v prostředí cloudu. Kromě populárních programovacích jazyků existuje široká škála služeb pro nasazení aplikací v cloudu. Tyto nástroje a rámce pro distribuované prostředí jsou velmi dobře popsány například v (Kartakis & Mccann, 2014).

Nicméně s rozšířením služeb přímo na edge nodách bude potřeba tyto nástroje a rámce rozvíjet. Případy užití Edge Computingu se také budou velmi pravděpodobně dost lišit. Programovací model běžící na edge nodech bude muset podporovat paralelní

zpracování úloh a dat současně na několika úrovních hardwaru. Jazyk, který bude podporovat tento programovací model, bude muset vzít v úvahu heterogenní hardware a kapacitu dostupných zdrojů. Pokud budou edge zařízení specifická podle jednotlivých výrobců, bude se s tím muset programovací jazyk vypořádat. Můžeme se tak dostat do mnohem složitější situace než u stávajících modelů v cloudu. Jeden z příkladů vznikajících rámců je představen v (Yang, Wu, & Wang, 2019), kde se právě snaží o inteligentní sdílení zdrojů v heterogenním prostředí cloudu a edge zařízení.

Jednoduché knihovny a algoritmy

Edge nody nebudou schopné podporovat stejný software jako velké servery z důvodu hardwarových omezení. Například malá mobilní základna s Intel T3K Concurent Dual-Mode system-on-chip (SoC) má 4 jádra ARM CPU procesoru a omezenou paměť, která není dostatečná pro běh komplexních aplikací třeba pro zpracování dat jako Apache Spark[3]. Ten totiž sám o sobě vyžaduje minimálně 8 jader CPU a 8 GB paměti, aby byl vůbec schopen nastartovat. Analytika provozovaná na edge zařízeních vyžaduje lehké algoritmy, které mohou provádět přiměřené úlohy týkající se strojového učení nebo zpracování dat (Santos, Tilly, Chandramouli, & Goldstein, 2013) (Xu, Wang, & Chen, 2015). Dobrým příkladem je Apache Quarks[4] jako lehká knihovna, která může být použita na zařízeních s malým rozsahem, jakým je chytrý telefon. Quarks však podporuje pouze základní zpracování dat, jako jsou filtrování nebo agregace dat, která nejsou dostatečná pro pokročilé analytické úlohy. Knihovny strojového učení, které spotřebovávají méně paměti a diskového prostoru, bude nutné vytvořit pro běh v prostředí edge computingu. TensorFlow[5] může být dobrým příkladem rámce, který podporuje algoritmy pro strojové učení a zároveň také heterogenní distribuované systémy. Ovšem jeho potencionální využití v edge computingu bude nutné ještě prozkoumat.

Mikro operační systémy a virtualizace

Výzkum a vývoj mikro operačních systémů nebo mikro kernelů může být značným přínosem pro řešení problémů týkajících se nasazení aplikací napříč heterogenními edge nody. Vzhledem k tomu, že tyto edge nemají stejné výpočetní zdroje jako standardní server v datovém centru, je nutné spotřebovávat méně zdrojů pro běh virtuální infrastruktury. Je také nutné velmi rychle startovat aplikace a důkladně je izolovat (Andrus, Dall, Hof, Laadan, & Nieh, 2011). Existuje předběžný výzkum, který naznačuje, že mobilní kontejnery dokážou poskytovat podobný výkon jako nativní hardware (Schwarzkopf, Murray, & Hand, 2012).

[3] https://spark.apache.org/docs/latest/
[4] https://quarks-edge.github.io/
[5] https://www.tensorflow.org/

Z tohoto důvodu je možné použít například technologii kontejnerové virtualizace Docker, která se stala velmi populární i ve světě cloudu. Její dostatečnou výkonnost v prostředí IoT zařízení prokázali v (Morabito, 2017). Zhodnocením použití Docker v prostředí EdgeC se zabývali i v rámci (Ismail et al., 2015). Nicméně je potřeba provést více testů i z oblasti bezpečnosti kontejnerů, aby mohly být uznány jako vhodný mechanismus pro nasazení aplikací do edge.

Spolupráce mezi průmyslem a akademickou sférou

Edge computing nabízí jedinečnou příležitost pro akademickou sféru, aby mohla svou výzkumnou činnost zaměřit v širokém měřítku na aplikované distribuované výpočty v prostředí cloudu a mobile computingu. Zároveň není snadné v akademickém prostředí testovat škálovatelnost edge computingu, jelikož akademická sféra nemá k dispozici rozsáhlé komerční nebo vládní infrastruktury. Z toho důvodu nemá akademická sféra ty správné možnosti validace výzkumu a výsledky mohou být zkreslené (Schwarzkopf et al., 2012). Díky tomu, že autor této práce působí v nadnárodní společnosti zabývající se edge computingem, bylo možné provést některá z ověření v této práci na reálném nasazení v komerčním sektoru.

Výzkum v oblasti edge computingu může být řízen otevřeným konsorciem průmyslových partnerů, jako jsou mobilní operátoři, vývojáři a cloudoví poskytovatelé stejně jako zainteresovaní akademičtí partneři. Z výsledků poté mohou těžit všechny strany (Barker, Varghese, Ward, & Sommerville, 2014).

3.4 Dostupná řešení pro Edge Computing

Tato kapitola zkoumá dostupná řešení na trhu pro uvedené problémy a příležitosti představené v předchozích kapitolách. Důležité je zmínit, že výzkum a rešerše dostupných řešení pro edge computing platformu byly provedeny během roku 2019, kdy neexistovala řešení jako například Google Anthos[6] a Microsoft Acr[7]. Obě tato řešení se objevila marketingově až ke konci roku 2019, a tudíž nemohla být součástí této práce. Hlavním podkladem pro výběr existujících řešení se stal průzkum trhu provedený společností Grand View Research (*Edge Computing Market Size, Share & Trends Analysis Report By Component* (*Services, Hardware, Software, Edge-Managed Platforms*), *By End Use* (*Transportation & Logistics, Retail, Datacenters, Wearables*), *And Segment Forecasts, 2019 - 2025*, 2019), podle něhož patří k lídrům trhu společnosti AWS, Cisco, Intel a Microsoft.

[6] https://cloud.google.com/blog/topics/hybrid-cloud/new-platform-for-managing-applications-in-todays-multi-cloud-world

[7] https://azure.microsoft.com/en-us/blog/azure-services-now-run-anywhere-with-new-hybrid-capabilities-announcing-azure-arc/

Za hlavní trend v oblasti řešení pro edge computing jsou považována IoT a Big data případy použití, a to především jejich propojení skrz cloudlet nebo fog computing zařízení s centrálním cloudem. Z průzkumu je patrné, že většina dostupných řešení těchto výrobců je orientována na IoT platformy. Kromě IoT řešení pak také zaznívají platformy orientující se na mobilní operátory v souvislosti s příchodem 5G, jehož expanze se očekává především v roce 2020 a později.

Na základě tohoto reportu jsme provedli analýzu dostupných řešení pro edge computing s IoT zaměřením uvedených výrobců a identifikovali 4 platformy:

- Cisco IoT platforma,
- Microsoft Azure IoT Hub,
- Intel IoT platforma,
- AWS IoT platforma.

Tyto platformy jsou stručně představeny v následujících kapitolách společně s jejich výhodami a nevýhodami. Následně kapitola 3.4.5 zobrazuje přehled jejich porovnání ve vztahu k této práci.

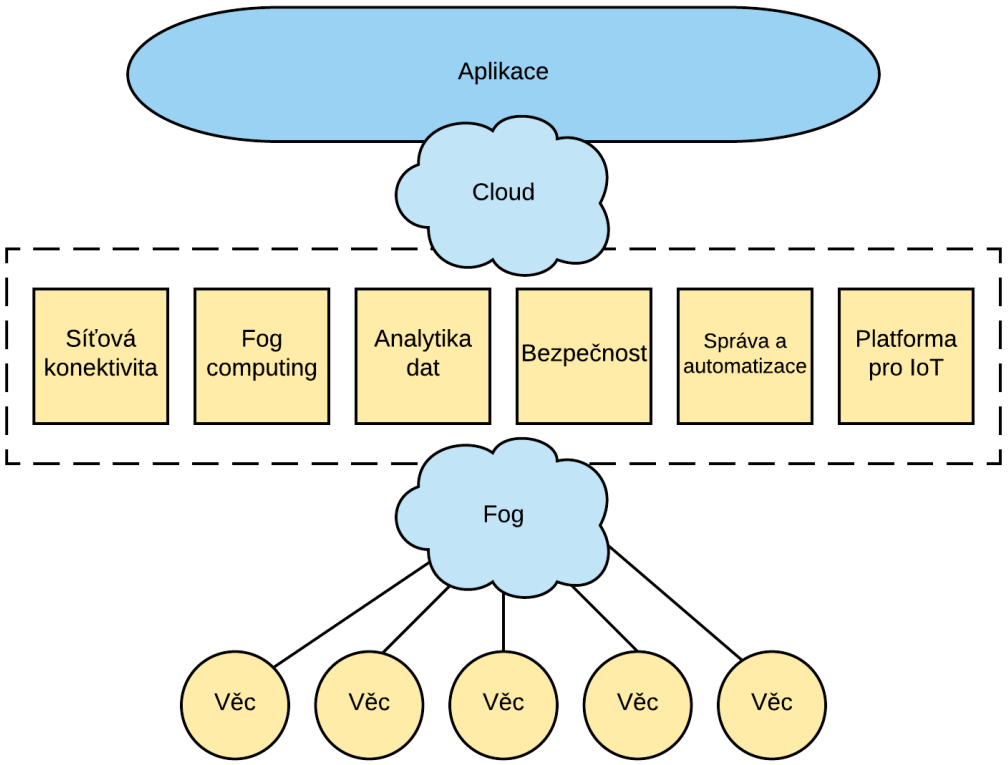
3.4.1 Cisco IoT platforma

Platforma Cisco IoT poskytuje technologii a software pro nasazení a provoz IoT aplikací. Jedná se o komplexní řešení postavené na fog computing standardu, které cílí na různá průmyslová odvětví od výroby, veřejných služeb až po dopravu či těžký průmysl. Jedná se o řešení provozované v režimu poskytovatele nikoli SaaS, jako je tomu u AWS nebo Microsoft Azure. To znamená, že si provozovatel kupuje licenci na software a provozuje jak řídicí část, tak koncová zařízení.

Systém se skládá z šesti komponent zobrazených na obrázku 3.5:

- Síťová konektivita - poskytuje širokou škálu síťových prvků a způsobu pro propojení cloud a fog computing serveru.
- Fog Computing - referenční implementace zmíněná v kapitole 3.2.3, jelikož Cisco je zakladatelem tohoto přístupu.
- Datová analytika - poskytuje analytické nástroje pro sledování sítě a jejich provozu. Současně umožňuje integrace s nejpoužívanějšími IoT protokoly jako MQTT, ZigBee apod.
- Bezpečnost - poskytuje kombinaci fyzické a virtuální kybernetické bezpečnosti.

- Správa a automatizace - poskytuje nástroje na centrální správu síťových prvků. Cisco produkty jsou primárně orientované na správu sítí.
- IoT platforma - vývojové nástroje a způsoby pro integraci IoT protokolů, jakými jsou MQTT, ZigBee, Bluetooth, apod.



Obrázek 3.5: Architektura Cisco IoT system, převzato z (*The Cisco IoT System*, 2018)

Výhody:

- Řešení není omezené pouze na vztah klient - server, takže je možné provozovat serverovou část aplikace a obsloužit částečně klienty přímo v edge. Díky tomu je možné tolerovat například výpadky internetového připojení nebo snížit objem dat přenášených do centrálního cloudu. Edge zařízení mohou komunikovat přímo mezi sebou a nevyžadují přenos dat skrz cloud.
- Fog node je multifunkční, jelikož se jedná prakticky o standardní hypervisor. Lze na něm provozovat kontejnerovou i serverovou virtualizaci na rozdíl od většiny ostatních IoT platforem. Zároveň je možné izolovat jednotlivé aplikace a docílit tak multi-tenantního řešení.

Nevýhody Cisco IoT:

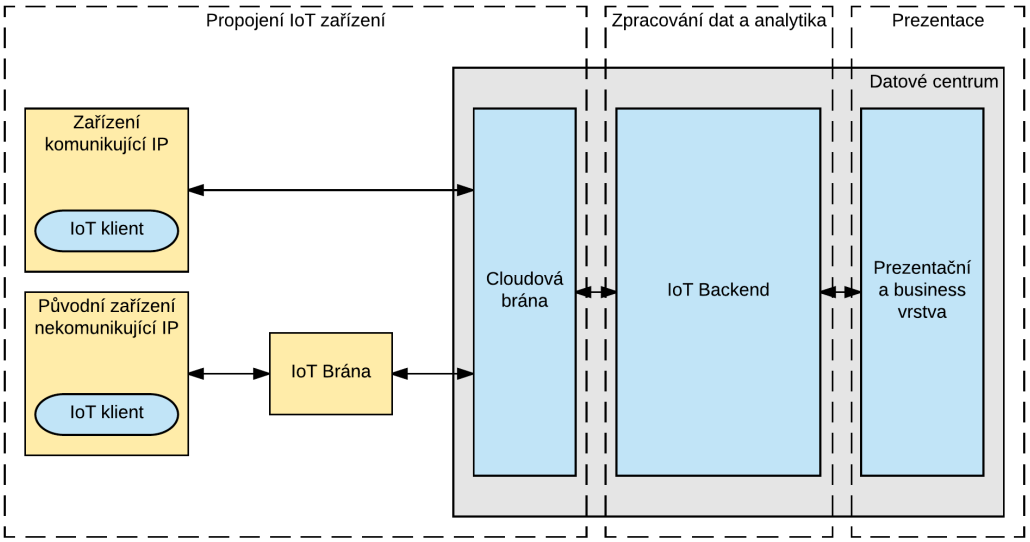
- Jedná se primárně o síťové řešení, kdy Cisco dodává kompletní připojení pro servery, a je nutné vybudovat identickou infrastrukturu jako v datovém centru.
- Pro orchestraci aplikací přímo v edge je potřeba zakoupení externího produktu Cisco Fog Director, který dokáže spouštět VM nebo kontejnery skrz své proprietární API. Na to navazuje chybějící interoperabilita aplikací mezi cloudem a edge.
- HW náročnost, kdy se používají klasické servery x86 a infrastruktura vhodná pouze pro datová centra. Není vždy možné vybudovat spolehlivé chlazení a prostory pro provoz takového HW v edge lokalitách, jako jsou například restaurace.
- Složitá instalace 5 různých nezávislých produktů společnosti Cisco, která není stavěna na provoz v režimu SaaS a vyžaduje odbornou instalaci každé lokality.

3.4.2 Microsoft Azure IoT Hub

Microsoft Azure IoT Hub poskytuje bezpečnou a spolehlivou komunikaci mezi IoT aplikacemi a zařízeními, které spravuje. Z pohledu Edge Computingu se jedná o kategorii Cloudlet. Řešení Poskytuje IoT backend hostovaný v Azure cloudu formou SaaS, ke kterému se připojí prakticky jakékoli zařízení. Azure cílí především na chytrý průmysl a zpracování dat v cloudu. Nicméně nabízí možnosti i pro jednotlivé skupiny vývojářů, kteří chtějí sbírat a analyzovat data ze svých senzorů. Podporuje celou řadu průmyslových protokolů, jakými jsou CoAP, OPC, Bluetooth, ZigBee nebo MQTT.

Architektura na obrázku 3.6 zobrazuje 3 její hlavní komponenty:

- Propojení IoT zařízení s cloudovou bránou lze dvěma způsoby. První, nastavuje zařízení komunikující po IP a posílá zprávy přímo bráně skrz internet. Druhou možností je instalace IoT brány, která sbírá zprávy a komunikuje s lokálními zařízeními a poté odesílá data a přijímá požadavky zpět od brány skrz internet.
- Zpracování dat a analytika - Azure poskytuje velmi dobré možnosti pro zpracování IoT dat a jejich analytiku v rámci svého veřejného cloudu.
- Prezentační část zodpovědná za integraci IoT prostředí do obchodních procesů společnosti díky integraci s interními IT systémy, jako jsou CRM nebo ERP.



Obrázek 3.6: Architektura Microsoft Azure IoT platformy, převzato z (*Azure IoT Suite*, 2018)

Výhody Azure IoT:

- HW nezávislé softwarové řešení oproti například Cisco IoT. Je možné provozovat software na libovolném x86 nebo ARM zařízení s podporou OS linux.
- Nevyžaduje nákup síťové infrastruktury a podporuje HW zařízení s malou kapacitou zdrojů paměti nebo procesoru.
- Jednoduchá instalace a provoz skrz SaaS řešení veřejného cloudu umožňuje použití individuálním vývojářům nebo velkým společnostem.
- Široce dostupný ekosystém IoT aplikací s možností vzdálené orchestrace.

Nevýhody Azure IoT:

- Komunikace mezi IoT bránou a cloudem pouze na protokolech AMQP 1.0, MQTT 3.1.1 a HTTP. Nelze tedy provádět libovolnou TCP nebo UDP komunikaci mezi centrálním cloudem a službami v něm běžících.
- Proprietární orchestrační API umožňující pouštět Docker kontejnery bez možnosti izolace sítě mezi jednotlivými kontejnery nebo s pokročilým nastavením persistentního úložiště.
- Vztah IoT brány s cloudovou bránou je v režimu klient-server, tedy řešení není multi-tenantní na úrovni IoT brány. Pro oddělení aplikací je nutné přidávat IoT brány namísto sdílení výpočetních zdrojů jedné z nich.

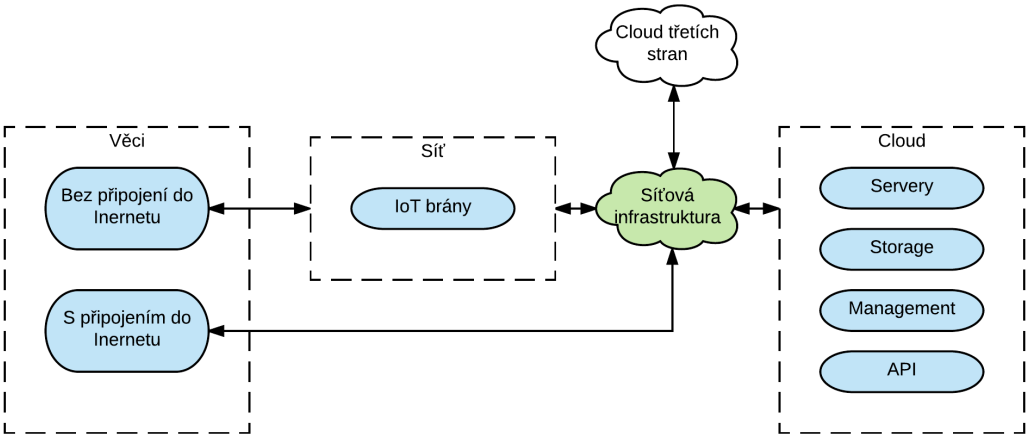
- IoT brány nemohou komunikovat mezi sebou, ale prakticky jen posílají metrická data do a přijímají řídicí zprávy z centrálního cloudu.

3.4.3 Intel IoT platforma

Intel® IoT platforma je referenčním modelem zahrnujícím řadu produktů od společnosti Intel, které mají poskytnout bezpečnou a spolehlivou komunikaci mezi IoT zařízením a cloudem. Zapadá tak také do kategorie cloudlet řešení. Nejedná se ovšem o konkrétní softwarovou implementaci. Jejím cílem je poskytnout řešení pro odvětví jako například automobilový průmysl, zdravotnictví nebo chytrá města.

Ve své podstatě se jedná o velmi podobou architekturu, jako je Azure IoT. Obrázek 3.7 se skládá ze čtyř částí:

- Věci neboli klienti - ti mohou být bez připojením, nebo s připojením k internetu. Komunikují tak na přímo s cloudem nebo skrz IoT bránu.
- IoT brána - slouží pro sběr a připojení senzorů či klientů, dále je komunikace přes protokoly HTTP, MQTT, CaAP a XMPP s centrálním cloudem.
- Cloud - cloud zpracovává data z IoT bran a současně také spravuje a registruje všechny IoT brány a zařízení.



Obrázek 3.7: Architektura Intel IoT platformy, převzato z (*IoT Security and Scalability on Intel® IoT Platform*, 2018)

Výhody Intel IoT:

- HW nezávislé softwarové řešení oproti například Cisco IoT. Je možné provozovat software na libovolném x86 zařízení s podporou OS linux.

- Nevyžaduje nákup síťové infrastruktury a podporuje HW zařízení s malou kapacitou zdrojů paměti nebo procesoru.
- Nediktuje konkrétního výrobce a dává na výběr ze svých partnerů.

Nevýhody Intel IoT:

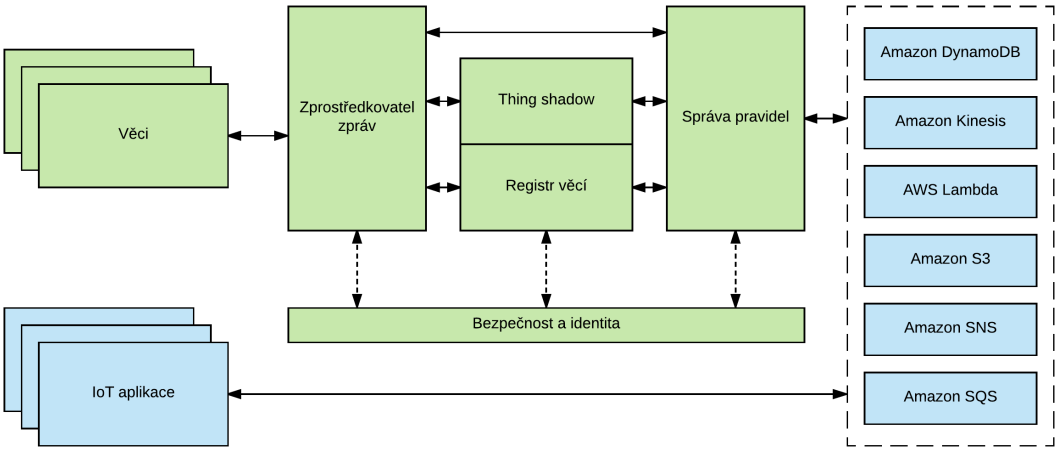
- Jedná se pouze o referenční model, nikoli o konkrétní implementaci softwaru. Při implementaci je nutné se obrátit na partnery společnosti Intel, kteří řešení poskládají na míru.
- Komunikace mezi IoT bránou a cloudem pouze na protokolech HTTP, MQTT, CaAP a XMPP. Opět tedy nelze provádět libovolnou TCP nebo UDP komunikaci mezi centrálním cloudem a edge lokalitou.
- Orchestrační API nebo způsob správy aplikací není konkrétně stanoven, takže pro správu aplikací je nutné použít jiné externí nástroje.
- Vztah IoT brány s cloudovou bránou je klient - server, tedy řešení není multitenantní na úrovni IoT brány, podobně jako u Azure IoT.
- Stejně tak IoT brány nemohou komunikovat mezi sebou. Prakticky jen posílají metrická data a přijímají řídicí zprávy z centrálního cloudu.

3.4.4 AWS IoT platforma

AWS IoT služba je provozovaná společností Amazon jako SaaS řešení stejně, jako je tomu u Azure. Umožňuje oboustrannou komunikaci mezi cloudovými aplikacemi a zařízeními připojenými k internetu. Běžné IoT aplikace buď shromažďují a zpracovávají data ze zařízení, nebo umožňují uživatelům jejich dálkové ovládání. Základním rozdílem oproti předcházejícím řešením je přímé napojení IoT aplikací nebo věcí na AWS cloud. AWS neposkytuje vlastní formu brány, jako je tomu u Azure. Z pohledu definice edge computingu se tak vlastně ani nejedná o žádnou z kategorií Cloudlet, MEC nebo Fog. Tuto vrstvu si musí uživatel řešit sám. Obrázek 3.8 zobrazuje architekturu IoT řešení:

- Věci nebo klienti se mohou připojovat na přímo do cloudu pomocí AWS IoT Device SDK knihovny. Současně lze napojit i libovolné IoT aplikace pomocí AWS klasických SDK na integraci s cloudovými službami. V tomto případě není téměř rozdíl mezi klasickými aplikacemi v cloudu a IoT, pokud pomineme, že se jedná o IoT aplikaci.

- Zprostředkovatel zpráv poskytuje možnosti příjímání a zasílání zpráv mezi zařízeními a cloudovými aplikacemi. Je zde možné použít buď MQTT, MQTT přes WebSocket, nebo HTTP REST API.
- Registr věcí a Thing shadow je zodpovědný za správu stavu jednotlivých zařízení. Současně poskytuje mechanismus centrální správy nastavení zařízení.
- Bezpečnost a identita vystavuje každému zařízení X.509 certifikát, jímž se zařízení autentizuje a autorizuje. Každé zařízení v registru má přiřazený certifikát a používá ho k šifrované komunikaci s cloudovou částí.



Obrázek 3.8: Architektura AWS IoT platformy, převzato z (*AWS IoT Documentation*, 2018)

Výhody AWS IoT:

- HW nezávislé softwarové řešení oproti například Cisco IoT. Je možné provozovat software na libovolném x86 nebo ARM zařízení s podporou OS linux.
- Robustní škálovatelná architektura na sběr metrických dat a centrální registr zasílání zpráv, která dokáže spravovat miliony klientů nebo věcí.
- Pokročilý systém správy identity jednotlivých klientů a jejich centrální registr. Tato identita může být využita pro přístup k dalším službám jako je distribuovaná databáze nebo objektové úložiště.

Nevýhody AWS IoT:

- Neexistuje možnost spouštět nebo orchestrovat aplikace (VM, kontejnery) z centrálního místa. Musí to být řešeno externím nástrojem.

- Není dostupné řešení pro bránu. Pouze AWS SDK na integraci aplikací pro sběr nebo ovládání s AWS cloudovou bránou. Edge v podobě IoT brány si musí vytvořit uživatel sám, takže jednoduše řečeno AWS neřeší problém edge computingu z pohledu jeho managementu. Nabízí pouze API pro sběr a ovládání milionů klientů.
- Komunikace mezi klienty a cloudem pouze na protokolech HTTP, MQTT a WebSocket. Opět tedy nelze provádět libovolnou TCP nebo UDP komunikaci mezi cloudem a edge lokalitou.

3.4.5 Porovnání dostupných řešení

Nyní se podívejme na porovnání jednotlivých řešení z pohledu vybraných 7 vlastností. Použili jsme 4 z nich na základě rešerše z kapitoly 3.3 a 3 jsme doplnili z vlastního výzkumu open source IoT platformy (Pavlik & Sobeslav, 2018) tak, aby souvisely s dílčími cíli disertační práce. Díky tomu jsme zjistili, zda nejsou již některé z nich vyřešeny, a náš další výzkum dává smysl. Tabulka 3.2 reflektuje tyto výsledky porovnání, které jsou níže postupně rozebrány.

Orchestrace aplikací v Edge Computingu sleduje možnost spravovat aplikace běžící v edge zařízeních z centrálního místa jednotným způsobem. Zároveň nás také zajímalo, zda je využíván nějaký standard pro rozhraní orchestrace, který se používá v rámci existujících cloudových řešení. Toto kritérium přímo koresponduje s dílčím cílem této práce. Z výstupu je patrné, že Intel IoT platforma a AWS vůbec tuto funkcionalitu nenabízí, jelikož řeší pouze napojení aplikací a jejich komunikaci s centrálním cloudem. Jakým způsobem je konkrétní aplikace spravována nebo spouštěna nechávají kompletně na koncovém uživateli. U Azure IoT je možné spouštět aplikace z jejich předdefinovaných šablon nebo si vytvořit vlastní pomocí docker kontejneru, který lze spustit uvnitř IoT brány. Nicméně se jedná o proprietární rozhraní s velmi základními možnostmi konfigurace. Není například možné nijak oddělit síťovou komunikaci mezi kontejnery nebo využívat perzistentní úložiště. Cisco IoT platforma je jediná, které umožňuje provozovat i standardní virtuální servery s pomocí VMware nebo KVM virtualizace, avšak je nutné pořídit jejich zvlášť licencovaný software Cisco Fog Director jako proprietární nástroj na správu VM a kontejnerů v Edge Computingu.

Mikro operační systémy a virtualizace v edge computingu částečně navazují na předchozí bod. Cisco IoT podporuje jak kontejnerovou, tak serverovou virtualizaci. Azure IoT Hub podporuje ve své bráně pouze docker kontejnery a má značné limity v jejich síťové izolaci nebo konfiguraci persistentního úložiště. Zbývající dvě platformy tento problém vůbec neřeší.

V rámci **Obecných výpočetních edge nodů** jsme sledovali závislost na konkrétním

hardwaru a také složitost instalace. Cisco IoT platforma je z velké části pouze o prodeji jejich síťového hardwaru. Pro nasazení řešení je nutné nakoupit směrovače, přepínače a kompletní síťovou architekturu od Cisco. V rámci serverů je možnost volby u výrobců, kteří jsou jejich partnery, a jako operační systém v nich běží buď VMware hypervisor nebo linuxová distribuce Ubuntu od společnosti Canonical. Při nasazení tohoto řešení tak prakticky není rozdíl mezi instalací standardního datového centra nebo edge lokality z hlediska robustnosti a složitosti. Rozhodně není toto řešení vhodné pro malé nasazení v podobě jednoho zařízení s 6GB RAM. Hned za Cisco IoT řadíme z pohledu závislosti a složitosti Intel IoT platformu, která vyžaduje pouze jejich procesorovou architekturu. Azure IoT Hub je flexibilní a dokáže běžet jak na ARM, tak x86 architektuře a nemá žádné striktní požadavky na konkrétního výrobce. AWS IoT poskytuje pouze integrační SDK pro komunikaci IoT aplikací s jejich cloudovými službami, a proto tento problém vůbec neřeší.

Veřejné zabezpečené nody a s tím související multi-tenance jsou také jedním z dílčích cílů této práce. Podívali jsme se, jestli je možné izolovat jednotlivé aplikace uvnitř edge a jaké způsoby řešení se nabízí. AWS IoT podobně jako předchozí dvě vlastnosti vůbec tento problém neřeší a nechávají ho na uživateli. Cisco IoT umožňuje jako jediná provést izolaci díky standardní serverové virtualizaci i síťovým prvkům. „Částečně" je v tabulce 3.2 uvedeno z důvodu nutnosti explicitní konfigurace namísto výchozího chování. Podobně jako v datovém centru si musí provozovatel platformy vyřešit izolaci aplikací sám prostřednictvím kombinace síťových firewall pravidel, VLAN a virtualizace. Neexistuje zde předdefinovaná koncepce izolovaného projektu, který by automaticky žil ve svém odděleném virtuálním světě od edge zařízení až po cloud.

Způsob distribuce řešení bylo další sledované kritérium, které nás zajímalo. Azure IoT i AWS IoT fungují v klasickém režimu veřejného cloudu, tedy Software-as-a-Service. Zákazník pouze registruje brány, věci nebo aplikace do centrálního cloudu, kde spravuje jejich konfiguraci nebo sbírá metrická data. Způsob použití je tak velmi jednoduchý a téměř kdokoli včetně individuálního IoT vývojáře může připojit svého klienta nebo bránu ke cloudu v rámci několika minut. Intel IoT udává pouze referenční architekturu s odkazem na partnery výrobce. Jedná se tedy o softwarovou distribuci podobně jako u Cisco IoT. V tomto případě musí existovat provozovatel IoT platformy, u kterého běží a je provozováno centrální řízení celého řešení. Bariéra vstupu a použití je proto velmi vysoká a řešení si mohou dovolit jen velké výrobní firmy a korporace. Variabilita konfigurace je také velmi vysoká a každá instalace platformy je svým způsobem unikátní ve výběru HW, komponent pro řízení (jednotlivé licencované produkty).

Poslední dvě kritéria **Režim klient-server** a **podpora komunikačních protokolů s cloudem** spolu velmi úzce souvisí. Klient-server model je myšlen ve vztahu mezi IoT bránou/Cloudlet/Fog zařízení a centrálním cloudem. IoT aplikace jsou v tomto pří-

padě klienti, kteří jsou připojeni k nebo volají na server API v cloudu. Azure IoT, Intel IoT i AWS IoT zapadají do tohoto modelu, jelikož poskytují pouze API rozhraní na sběr metrických dat, distribuované úložiště nebo systémy na zasílání zpráv. Tito klienti přistupují tak na server skrz protokoly AMQP, MQTT nebo HTTP. Není však možné provést IP-IP komunikaci mezi aplikací v IoT bráně a virtuálním serverem ve veřejném cloudu, jelikož ani jedno z těchto tří řešení neposkytuje žádnou formu například VPN připojení. Cisco IoT je jediné z nich, které umožňuje propojit edge lokality s cloudem a běžet serverové aplikace přímo v edge namísto centrálního cloudu. Právě díky přesunu serverových aplikací do edge je například možné snížit množství přenesených dat do centrálního cloudu a provádět pokročilejší práci s daty přímo v místě jejich sběru. Dále je možné eliminovat ztrátu dat způsobenou výpadkem připojení mezi edgem a centrálním cloudem apod.

Vlastnost	Cisco IoT	Azure IoT	Intel IoT	AWS IoT
Orchestrace aplikací v edge zařízeních	ano, proprietární	ano, proprietární	ne	ne
Obecné výpočetní edge nody	částečně, závislost na Cisco HW	ano	částečně, závislost na Intel architektuře	neřeší
Mikro operační systémy a virtualizace	ano	ano, pouze kontejnery	neřeší	neřeší
Veřejné zabezpečené nody / Multi-tenance	ano, částečně	ne	ne	neřeší
Způsob distribuce platformy	distribuce software	SaaS	distribuce software	SaaS
Režim klient-server	ne	ano	ano	ano
Podpora protokolů komunikace s cloudem	libovolný síťový protokol	AMQP, MQTT a HTTP	HTTP, MQTT a CaAP, XMPP	MQTT, Websocket HTTP

Tabulka 3.2: Porovnání dostupných edge computing řešení, zdroj: vlastní tvorba porovnání na základě uvedených zdrojů

3.5 Potřeba návrhu nové edge computing platformy

Z výše uvedeného lze usoudit, že edge computing přináší spoustu výhod oproti tradičnímu cloud computingu, avšak existuje také mnoho nezodpovězených otázek a nejasností. Podívejme se na shrnutí současného stavu v několika bodech.

- Nejvíce vědeckých publikací se soustředí na řešení okolo 5G a telco ISP (MEC) a neexistují téměř žádné případové studie pro využití v obchodních řetězcích,

chytrých domácnostech, restauracích atd., ve kterých rapidně roste potřeba digitalizace a využití služeb strojového učení a umělé inteligence.

  - Inovace a rapidní doručovaní změn (Bello, Mosquera, & Rogers, 2018). Vzniká zde potřeba centrální správy všech klientských lokalit tzv. fleet management a dynamické doručování softwarových změn. Edge se tímto způsobem dostává do stejného problému jaký mají velké společnosti při vývoji a provozu tradičních aplikací v datových centrech nebo cloudu (Humble, Kim, & Forsgren, 2018). Právě proto musí získat stejné nástroje jaké běžně používají při provozu aplikací v cloudu.

- Z porovnání existujících komerčních řešení pro Edge Computing v Kapitole 3.4 vyplývá, že většina z nich není navržena pro plnohodnotnou správu a orchestraci aplikací přímo v edge zařízeních. Tato řešení jsou navržena primárně pro sběr metrických dat z koncových klientů do cloudu nebo jejich vzdálené ovládání skrz systémy zasílání zpráv. Klienti nebo IoT brány vyžadují přímé 100% připojení k internetu a nepodporují běh izolovaných aplikací uvnitř stejného zařízení. Multitenance je tak dosahováno přidáváním těchto zařízení namísto sdílení HW zdrojů. Jedinou možností pro vyřešení těchto problémů je použití Fog Computingu jako například u Cisco IoT, které ovšem prakticky staví plnohodnotné datové centrum nebo privátní cloud v edge lokalitě a rozhodně nemůže běžet na jednom serveru s kapacitou 4CPU a 8GB RAM. Tím se značně snižuje flexibilita použití například v prostředích, jakými jsou automobily, restaurace apod.

- Tradiční edge computing architektura obsahuje 3 vrstvy *Cloud Computing - Edge* (*Cloudlet, MEC*) *- mobilní zařízení/klienti*. Tento model má několik zásadních limitů:

  - Různé vytížení edge systémů. Může dojít k dočasnému přetížení některých lokalit z důvodu zpracovávání velkého počtu koncových zařízení. Výsledkem je, že některé typy provozů, a to i ty, které ke zpracování vyžadují minimální fyzické a jiné zdroje, které jsou citlivé na latenci, nebudou zpracovány včas, čímž se snižuje výkon a případně QoS.
  - Některé lokality mohou být příliš daleko i od umístěného edge. Vzniká potřeba umístění edge zařízení ke koncovému uživateli a jeho vzdálenou správu.
  - Některé lokality mají nestabilní internetové připojení a vzniká potřeba provozovat všechny služby i v režimu offline.
  - Tato edge zařízení musí podporovat multi-tenanci a sdílení HW zdrojů pro různé oddělené aplikace tak, aby nebylo nutné s každou aplikací přidávat jednoúčelové HW zařízení.

Závěrem této kapitoly si uveďme konkrétní příklad obchodního řetězce s tisíci pobočkami. Klasický MEC je provozován z rádiostanic, ne přímo z poboček. Fog computing nemůže být použit, jelikož není možné z kapacitních důvodů provozovat HW datové centrum přímo v pobočkách. Každá pobočka má různé poskytovatele připojení k internetu. Ten nezaručuje 100% dostupnost internetového připojení, jehož je zde zapotřebí. Dále tyto pobočky nebo obchody nejsou ve vztahu klient - server, ale vyžadují plně funkční a multi-tenantní HW edge zařízení. V neposlední řadě aplikace běžící v pobočkách musí být spravovány a organizovány centrálně. Víceméně vyvstává potřeba rozšířit architekturu o další 4. vrstvu - *cloud computing - edge* (*cloudlet, MEC*) - ***Consumer Edge*** - *mobilní zařízení/klienti*.

Abychom mohli vyřešit všechny uvedené problémy edge computingu, navrhneme novou architekturu se čtyřmi úrovněmi, nazvanou Consumer Edge Computing (CEC). Tato nová architektura umožní těmto pobočkám nebo domácnostem provozovat místní HW nenáročné datové centrum nazvané consumer edge, ve kterém lze zpracovat požadavky citlivé na latenci nebo provozovat plnohodnotné serverové aplikace. Kromě toho bude možné do těchto datových center integrovat služby třetích stran a rozšíří se tím nabídka i způsob využití. Zároveň navrhneme standardní API pro orchestraci a doručování dynamických aktualizací softwaru, čímž umožníme vyšší výkonnost v doručovaní softwaru ve společnosti provozující aplikace v edge lokalitách. Díky tomu bude možné přenést cloudové aplikace a způsoby jejich provozu do světa edge computingu bez nutnosti výrazných změn. Detailně jsou tyto potřeby včetně analýzy řešených problémů a cílů technického návrhu popsány v následující kapitole 4, resp. její podkapitole 4.1.

4 Návrh Edge Computing platformy

Tato kapitola je zaměřena na představení vlastního výzkumu, vlastní vytvořené platformy Consumer Edge Computing. Navazuje tak na problémy a závěry zmíněné v kapitole 3.

Cílem MEC nebo cloudletu bylo přinést funkce cloud computingu co nejblíže k uživateli a vyřešit problémy jako mobilitu, dobu odezvy, procesování výkonu a podobně. I přesto nedokázaly tyto koncepty zcela vyřešit všechny problémy edge computingu (Varghese, Wang, et al., 2016). Edge umístěné u rádiových ústředen sice dokázaly snížit dobu odezvy a poskytly prostor pro vzdálené procesování výkonu, avšak ve špičkách provozu mají problémy s přetížením zdrojů a není možné je efektivně škálovat. Díky tomu dochází znovu k pomalejší době odezvy a klesá kvalita poskytovaných služeb jednotlivým uživatelům. Podobný problém nastává i v případě, kdy uživatelé jsou příliš daleko od edge anebo je nestabilní internetové připojení. Restaurace nebo dobíjecí stanice pro elektromobily nemají často garantované internetové připojení, a je proto nutné provozovat tyto služby přímo na místě, dokonce i v režimu offline. Dalším problémem je jednoúčelovost dnešních edge zařízení, kterou jsme našli u téměř všech existujících řešení v kapitole 3.4.5. Dostupná řešení fungují ve vztahu klient-server či pouze sbírají data z IoT zařízení a není možné provozovat plně multi-tenantní prostředí jako u cloud computingu. Z této analýzy vyplývá, že prakticky není možné provozovat tzv. real-time aplikace jako rozšířená realita, connected cars nebo tactile internet[1]. K překonání těchto nedostatků jsme navrhli novou architekturu nazvanou Consumer Edge Computing (CEC). Naše architektura usnadní provoz real-time aplikací, škálování, multi-tenancy a problémy s připojením skrz umístění edge zařízení přímo do koncového místa.

V kapitole 4.1 jsou nejprve představeny čtyři základní cíle a pilíře navrženého consumer edge computingu, které jsou mapovány na problémy a příležitosti z analýzy současného stavu. Následně je v kapitole 4.2 představena obecná čtyřvrstvá architektura včetně vymezení pojmů a jejich propojení. V dalších čtyřech kapitolách jsou pak postupně rozebrány odpovědi na vydefinované cíle, jako jsou optimalizace doby ode-

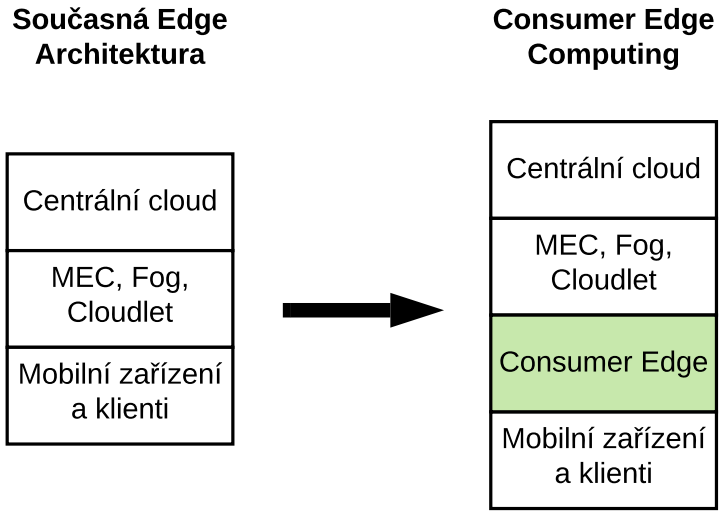
[1] Termín Tactile Internet obecně označuje komunikační síť, která je schopna poskytovat informace o řízení, dotyku a ovládání v reálném čase prostřednictvím dostatečně spolehlivé, pohotové a inteligentní konektivity. Vznikla především v souvislosti s příchodem 5G sítí.(Aijaz & Sooriyabandara, 2019)

zvy 4.3, mutlifunkční edge 4.4, standardní API pro orchestraci 4.5 a bezpečnost v CEC platformě 4.6.

4.1 Cíle a základní pilíře Consumer Edge Computing

4.1.1 Co je Consumer Edge Computing?

Consumer Edge Computing (CEC) je nová koncepce navržená autorem této práce na základě více než pětiletého výzkumu a působení v oblasti cloud computingu a edge computingu v několika nadnárodních společnostech. Jedná se o umístění malého HW, nenáročného datového centra přímo ke koncovému uživateli (např. domácnost, pobočka apod.). Jeho cílem je snížit zátěž, nestabilitu připojení, multifunkčnost edge zařízení a celkové zkvalitnění provozu real-time aplikací. CEC se snaží vyřešit zmíněné problémy (kapitola 3.5), ke kterým dochází u klasických MEC a cloudlet zařízení, která jsou provozována poskytovateli edge platforem nejčastěji telco operátorem nebo ISP. Architektura CEC zahrnuje 4 úrovně (*cloud computing - edge* (*cloudlet, MEC*) *- consumer edge - mobilní zařízení/klienti*), na rozdíl od klasické tříúrovňové architektury, jak zachycuje obrázek 4.1. Termín „Consumer" v consumer edge computingu značí především, že vlastnictví zařízení a místo provozu je u cílového uživatele nikoli u poskytovatele platformy. Jedná se o místa, jakými jsou pobočky firem, restaurace, nákupní centra, nemocnice apod. S tímto konceptem nemusí klienti posílat dotazy do nejbližšího datového centra nebo provozovat běžné serverové aplikace vzdáleně například v cloudu. CEC umožňuje, aby aplikace a služby běžely přímo v daných lokalitách, a data nebo dotazy do centrálního cloudu by se měly posílat jen v případě nezbytné nutnosti nebo při synchronizaci dat. Nicméně v případě nedostupnosti CEC zařízení je možné automaticky spustit danou výpočetní úlohu nebo službu v nejbližším MEC, fog nebo cloudlet zařízení a minimalizovat tak jakýkoli výpadek. Odstraňuje tak již zmíněný model klient - server nebo přímou závislost na připojení k internetu, které jsou vidět u dnešních dostupných řešení pro edge computing.



Obrázek 4.1: Grafické znázornění rozdílu mezi MEC/Cloudlet a CEC architekturou, zdroj: vlastní tvorba

4.1.2 Vymezení cílů CEC

K vytvoření tohoto CEC konceptu jsme si stanovili několik základních cílů, které musí být vyřešeny, aby měl náš výzkum smysl a platforma mohla být provozována v širším spektru. Ty jsou popsány v následujících několika odstavcích.

Optimalizace doby odezvy a běh aplikací co nejblíže ke klientům s vysokou dostupností a kvalitou QoS je naprosto jednoznačný cíl, kterého chceme s CEC dosáhnout. Umožněním běhu aplikací přímo v CEC chceme dosáhnout ultra krátké doby odezvy a tím i odstranění problému s přetížením MEC a cloudlet v náročných špičkách provozu. Současně je cílem poskytovat služby v případě výpadku z nejbližšího bodu nebo umožnit aplikacím komunikaci napříč jednotlivými lokalitami. Aby toto bylo možné, musíme se detailně podívat na trendy vývoje aplikací za poslední dekádu. S evolucí cloud computingu došlo ke značné změně, jakou dochází k vývoji produktů a softwaru. Moderní aplikace jsou navrženy podle cloud native konceptu a většinou se skládají z velkého množství mikroslužeb (Balalaie, Heydarnoori, & Jamshidi, 2016). Ty mohou být implementovány v různých programovacích jazycích, mohou být také ve vlastnictví různých vývojových skupin a mohou mít tisíce instancí konstantně měnící své stavy díky dynamické distribuci. V tak dynamickém prostředí se stává jejich provoz a hledání chyb velmi náročným úkolem z důvodu komplexity závislostí (Zhou et al., 2018). Představíme-li si, že přechodem do edge computingu dochází ještě k umocnění

počtu lokalit a dalších závislostí na připojení, dostáváme se do ještě složitější situace. Z toho důvodu byl vytvořen koncept Service Mesh, jenž má zmírnit tuto situaci zavedením vyhrazené vrstvy infrastruktury nad mikroslužbami, aniž by došlo k jakýmkoli změnám v jejich implementaci. Integrace Service Mesh konceptu je tedy nezbytná pro účely naplnění vysoké dostupnosti a kvality.

Podle výzkumu, který běžel čtyři roky (Humble et al., 2018), bylo prokázáno, že společnosti, které aplikují koncept DevOps do vývoje a běhu softwaru, dosahují větší akcelerace a inovace, což přímo ovlivňuje růst společnosti. Detailněji je tato problematika rozvedena v kapitole 5.3. Tento koncept nepřímo souvisí i s nástroji využívanými pro doručování softwaru do produkce. Z tohoto důvodu bylo naším cílem zakomponovat **Standardní API pro Orchestrace** v CEC zařízeních a poskytnout tak stejnou funkcionalitu, na kterou jsou dnes společnosti zvyklé při využívání služeb cloud computingu a která není dostupná u většiny dostupných řešení. Současně jsme chtěli použít již existující nástroje a koncepty pro objevování služeb (discovering), což je jeden z problémů definovaných v kapitole 3.3.1. Díky tomu bude možné přenést existující aplikace do prostředí edge computingu bez výrazných změn.

Množství bezpečnostních hrozeb narůstá ve všech technologických oblastech a výjimkou není ani edge computing, kde z nedávných výzkumů vyplynulo, že největší jeho bezpečnostní hrozby jsou distribuované denial of service útoky, side-channel útoky, malware injection útoky a útoky vedené na autentizaci a autorizaci (Xiao et al., 2019). Dalším cílem bylo tedy navrhnout CEC v souladu s **konceptem Zero Trust**. Ten byl poprvé definován (Kindervag, 2010) a na rozdíl od konvenčního modelu má jako svůj princip *„nikdy nedůvěřovat, vždy ověřovat“*. Ten se aplikuje jak na interní, tak i externí sítě a komunikaci. Tato zásada je základem pro snížení rizika útoků nejen vnějších, ale i vnitřních. Zároveň tak může vyřešit část problémů s používáním veřejných a zabezpečených edge zařízení definovaných v kapitole 3.3.1 společně s problémem multitenance a izolace aplikací uvnitř edge zařízení.

Dalším cílem bylo vytvořit **Multifunkční edge zařízení** tak, aby bylo možné provozovat různé aplikace spravované různými subjekty na stejném CEC zařízení. V rámci našeho výzkumu jsme se soustředili především na softwarovou část, nikoli hardware. A proto bylo důležité zahrnout do cílů běh mikroslužeb a segmentaci operačních systémů (kapitola 3.3.2). Zároveň bylo cílem vytvořit koncept obecného výpočetního edge nodu (kapitola 3.3.1). Současně by kontejnerová virtualizace měla umožnit oddělení výpočetních zdrojů a poskytnout multi-tenancy definovanou v kapitole 3.3.1. S tím souvisí i přechod od zmiňovaného modelu klient -server na plnohodnotný provoz virtualizovaných aplikací v edge zařízeních.

Cíl	Problémy & Výzvy
Optimalizace doby odezvy	Vysoká kvalita služeb QoS
Standardní API pro Orchestrace	Standardy, Objevování na edge nodech
Koncept Zero Trust	Veřejné zabezpečené nody
Multifunkční edge zařízení	Obecné výpočetní edge nody, Mikro operační systémy a virtualizace

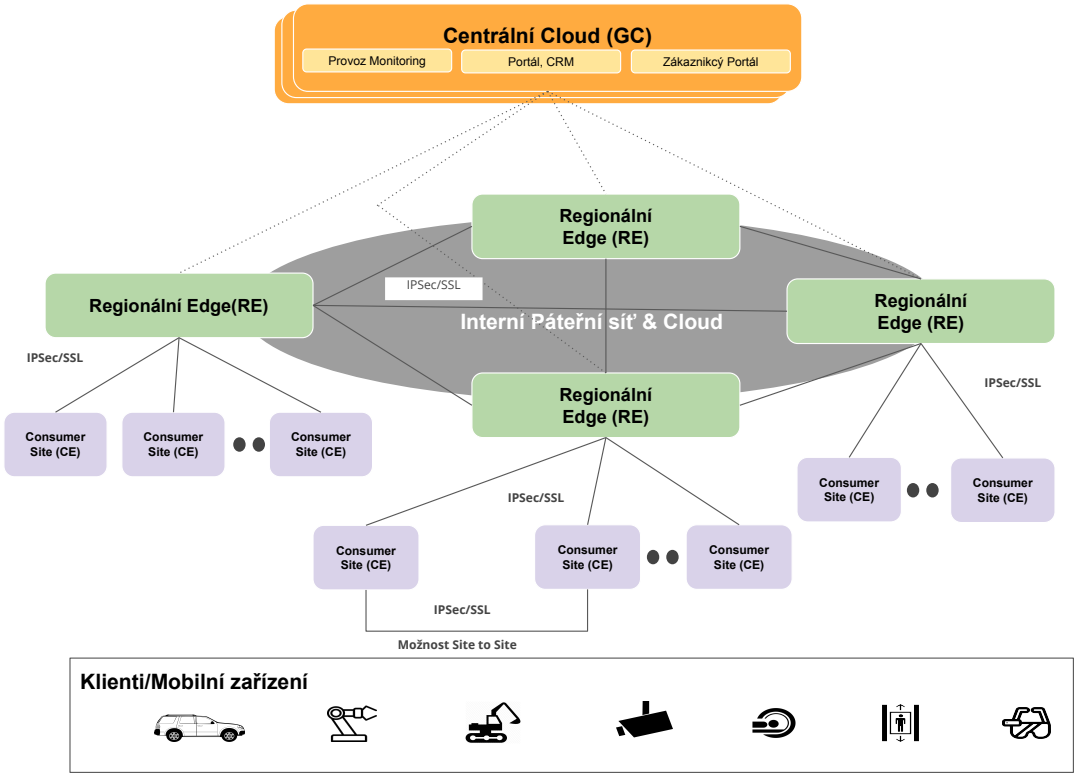
Tabulka 4.1: Vazba cílů CEC na problémy a výzvy edge computingu 3, zdroj: vlastní tvorba

Tabulka 4.1 zobrazuje, jakým způsobem jsou namapovány naše vydefinované cíle na problémy a příležitosti představené v analýze současného stavu Edge Computingu 3.

4.2 CEC architektura

Jak již bylo zmíněno, CEC architektura je výsledkem několikaleté autorovy práce v nadnárodní společnosti, kde působí na pozici jednoho z hlavních architektů. Jeho hlavním přínosem je navržení dílčích komponent platformy a částečná implementace některých z nich. V tuto chvíli je provozována u několika zákazníků z oblasti automobilového průmyslu a maloobchodu, z nichž výstupy jednoho jsou uvedeny v rámci případové studie v kapitole 5.3.2.

Před samotným vysvětlením CEC architektury je nutné objasnit pojem site, který je převzatý z angličtiny. Ten reprezentuje fyzickou nebo virtuální lokalitu s jedním nebo mnoha zařízeními formujícími jednu logickou výpočetní entitu. Site tedy může běžet ve veřejném cloudu jako virtuální server, ve fyzickém datovém centru anebo přímo v koncových lokalitách, jakými jsou například továrny, obchody, restaurace, dobíjecí stanice pro elektromobily apod. Obrázek 4.2 zobrazuje CEC architekturu se čtyřmi vrstvami. Centrální cloud (GC) je hlavní řídicí prvek celé architektury a může běžet ve standardním veřejném, nebo privátním cloudu. Jeho hlavní komponentou je uživatelský portál a standardní API. Tyto komponenty poskytují multi-tenantní řešení pro řízení a správu infrastruktury, aplikací, služeb i vizualizací. Je to centrální rozhraní pro SaaS řešení a správu. Centrální cloud je připojen a komunikuje se dvěma typy site - regionální edge a consumer edge.



Obrázek 4.2: CEC Architektura se čtyřmi vrstvami, zdroj: vlastní tvorba

Regionální edge (RE) je v tomto návrhu na podobné úrovni jako fog node, MEC nebo cloudlet zařízení. Jedná se buď o PoP, což jsou lokality tvořící páteřní sítě poskytovatelů internetu, anebo běží u rádiových ústředen blíže ke koncovým zařízením. RE současně slouží také pro propojení ostatních edge zařízení a formují interní páteřní síť CEC platformy. Všechny RE site (lokality) jsou propojeny skrze IPSec nebo SSL VPN do síťové topologie full mesh, ve které je každý uzel spojen se všemi ostatními uzly. RE také slouží jako rozhraní pro vystavení zákaznických služeb na veřejném internetu a umožňuje provoz koncových aplikací. Díky tomu je možné CEC platformu využívat jak pro použití standardní MEC topologie se třemi vrstvami, tak jako rozšířenou o consumer edge vrstvu.

Consumer edge site (CE) je zařízení, virtuální server nebo skupina nodů běžící u koncového uživatele nebo zákazníka edge platformy. Jedná se o jakýsi vztyčený bod a již zmíněnou čtvrtou rozšířenou vrstvu této CEC platformy. Jeho role je přenést výpočetní výkon úložiště a síťovou konektivitu přímo ke klientům a snížit tak fyzickou vzdálenost a závislost na připojení k regional edge. CE site navazuje dynamicky IPSec/SSL VPN připojení ke dvěma nejbližším RE. Teoreticky je také možné nasadit několik CE site a zformovat full mesh topologii s přímým připojením bez nutnosti využití

RE. Nicméně v tom případě je nutný přímý síťový přístup mezi CE site, aby bylo možné navázat VPN tunely.

Poslední vrstvou jsou stejně jako u běžných edge architektur klienti a mobilní zařízení. To mohou být roboti, stroje, různá IoT zařízení či mobilní telefony komunikující se službami běžícími buď v CE nebo RE v závislosti na dané lokalitě a připojení. Díky čtyřvrstvé architektuře je možné poskytovat tyto služby z CE zařízeních a v případě jejich výpadku přepnout na záložní aplikaci v nejbližším RE. Stejně tak je možné vynechat pro některé případy využití CE site a konzumovat služby přímo z RE site. V určitých případech je možné umístit CE site přímo do koncových zařízení, čímž klienti jsou vlastně aplikace běžící uvnitř CE site.

V následujících čtyřech kapitolách jsou rozebrány jednotlivé stavební bloky CEC platformy ve vztahu k vydefinovaným cílům práce namísto strohého popisu komponent architektury. Jedná se o optimalizaci doby odezvy, mutlifunkční edge, standardní API pro orchestraci a bezpečnost v CEC platformě. Z ohledem na rozsah práce není bohužel možné obsáhnout detailně všechny části platformy.

4.3 Optimalizace doby odezvy a Service Mesh

Optimalizace doby odezvy má dva základní aspekty, a to fyzikální a softwarový. Je možné provést sebelepší optimalizace na úrovní softwaru a hardwaru, avšak pokud budou dané subjekty fyzicky velmi vzdálené, doba odezvy bude dlouhá. Je tedy nutné podívat se na oba aspekty, kde fyzická vzdálenost je řešena právě rozšířením CEC architektury o čtvrtou vrstvu běžících site přímo u koncových klientů. Logicky lze předpokládat, že pokud přesuneme jakoukoli technologii blíže ke klientovi, jeho doba odezvy bude nižší, tudíž nedává smysl se věnovat fyzickému pohledu. Pro softwarovou optimalizaci je potřeba vyřešit otázky bezpečného a optimálního připojení distribuovaných aplikací běžících mezi koncovými body. Existují tři typy síťové komunikace - aplikace s aplikací, uživatel/stroj s aplikací a aplikace s veřejným internetem. Jelikož je naším cílem přenést funkce cloud computingu do prostředí edge, tak jsme detailně mapovali síťové funkce veřejných cloudů (*Azure Networking Concepts*, 2018), (*AWS VPC Networking*, 2018) a Service Mesh (W. Li, Lemieux, Gao, Zhao, & Han, 2019), (Indrasiri & Siriwardena, 2018). Následně jsme vytvořili vlastní obecný návrh síťových požadavků pro CEC site, který je promítnutý do obrázku 4.3 a skládá se z komponent uvedených níže.

- Síťové směrování - základní funkce u dnešních síťových prvků je směrování a přepínání paketů.
  - IP přepínání a směrování - v tomto případě je nutné směrovat pakety mezi

interními a externími sítěmi.

  - VPN koncentrátor pro IPsec nebo SSL VPN mezi ostatními edge lokalitami. Řešení terminuje VPN tunely a směruje síťový provoz do jednotlivých virtuálních sítí na základně tunelového profilu. VPN klienti mohou být ostatní site (lokality), nebo standardní síťové boxy podporující IPSec a SSL VPN protokoly

- Síťová ochrana - nezbytná nutnost pro regionální sítě, které jsou vystaveny do veřejného internetu, kde velmi často dochází k síťovým útokům. Dále i pro CE site umístěné na veřejných místech je jejich síťová ochrana nezbytná.
  - DDoS mitigation - je sada technik a nástrojů, které brání, nebo zmírňují dopad útoků na prvky připojené k internetu. DDoS je jedním z nejvýznamnějších a nejdůležitějších útoků dnešního kybernetického světa (Mahjabin, Xiao, Sun, & Jiang, 2017).

- Síťová mikrosegmentace
  - Izolované sítě pro fyzické i virtuální rozhraní - všechna rozhraní musí umět izolovat síťový provoz, aby bylo možné oddělit aplikace na síťové vrstvě. Tato izolace je prováděna například pomocí protokolu MPLSoverUPD nebo VXLAN.
  - Firewall politiky - síťové TCP/IP pravidla a klasické stavové firewally. Jejich pravidla mohou být konfigurována standardním ACL formátem.

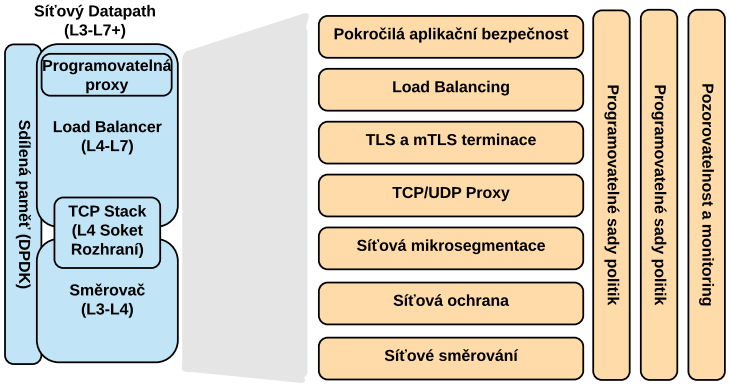
- TCP/UDP proxy - jedna z nejdůležitějších funkcí pro kontrolu a sledování aplikačního síťového provozu. Proxy je server, který vstupuje mezi klienta a cílový server. Klient navazuje spojení s TCP/UDP proxy serverem, který následně navazuje spojení s cílovým serverem.

- TLS a mTLS terminace - tato funkcionalita by měla být součástí TCP/UDP proxy, kdy je možné komunikovat bezpečně a autentizovaně pomocí mTLS, kde dochází k obousměrné autentizaci komunikace klienta i serveru. TLS protokol při výchozím stavu prověřuje pouze identitu serveru přistupujícímu ke klientovi, a proto musí být autentizace prováděna na aplikační úrovni.

- Load Balancing - vyvažování je někdy oddělené od proxy, avšak v posledních letech bývá její součásti. Vyvažování se provádí skrz protokoly HTTP, TCP a UDP včetně sledování dostupnosti koncových bodů (tzv. health-check).

- Pokročilá aplikační bezpečnost

– WAFS je aplikační firewall určený pro HTTP aplikace. Funguje na principu aplikování různých pravidel na HTTP konverzaci. Obecně tato pravidla dokáží zachytit běžné útoky, jako jsou cross-site scripting (XSS) nebo SQL Injection.
– Detekce anomálií založená na principu strojového učení ze sběru logů a metrik ze síťového provozu. Lze vytvořit model umělé inteligence a následně předcházet nebo blokovat různé druhy útoků.



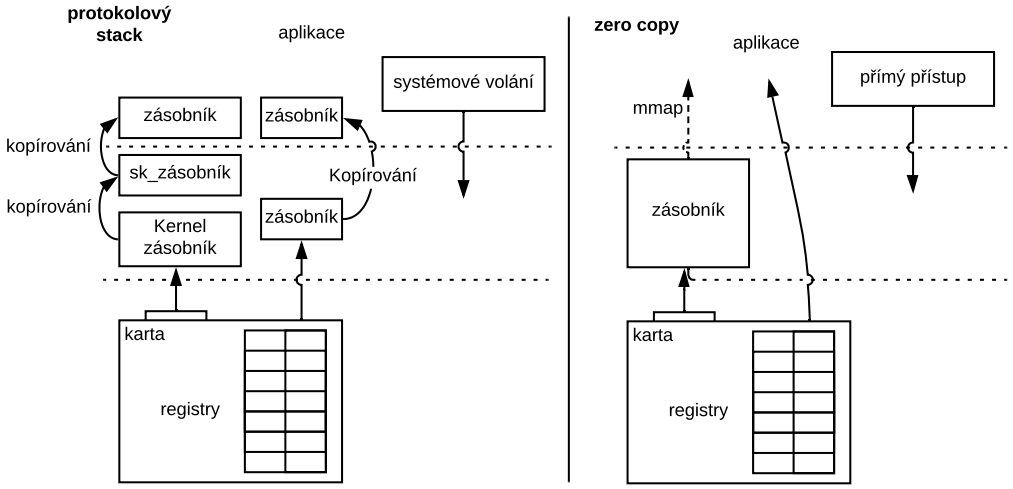
Obrázek 4.3: Obecné zobrazení požadavků na síťové funkce v edge lokalitě, zdroj: vlastní tvorba

V prostředí veřejných cloudů jsou tyto požadavky a funkce splněny různými virtuálními síťovými komponentami v závislosti na konkrétním poskytovateli (*Azure Networking Concepts*, 2018). Ve světě edge computingu jako například u vybraného Cisco IoT řešení v kapitole 3.4.1 je to kombinace někdy až 3 HW zařízení dohromady (směrovač, firewall, fog server), což je prakticky stejná infrastruktura jako v datovém centru kladoucím si nároky na chlazení, prostor apod. Ostatní analyzované IoT platformy neřeší žádné ze síťových funkcí a přenáší tento problém na provozovatele. Abychom docílili uvedených funkcí a současně umožnili běh na jednom HW zařízení, musí každé z nich sloužit jako plně integrované síťové řešení L3-L7 vrstev ISO/OSI modelu. Z těchto důvodů jsme se rozhodli navrhnout vlastní Edge gateway, dále EGW, která je popsána v následující kapitole.

4.3.1 Návrh edge gateway

Jako základ pro EGW síťového řešení je použit Intel DPDK, což je sada knihoven a síťových ovladačů pro rychlé zpracování paketů. Jedná se o rámec přímo propojující

aplikaci se síťovým hardwarovým rozhraním pro zpracovávání síťového přenosu, minimalizování přerušení OS, systémových volání a přepínání kontextu. DPDK využívá stávající technologie Intel procesorů, jakými jsou SIMD instrukce, Huge-pages a mnoho paměťových kanálů včetně ukládání do mezipaměti, což umožňuje výrazné zrychlení (Zhu, Li, Luo, Xu, & Zhang, 2018). Tento přístup využívá koncept Zero-copy, který se snaží snížit počet kopírování paketu při jeho cestě z fyzické sběrnice do aplikace. Rozdíl mezi tradičním a Zero-copy je zachycen na obrázku 4.4. Při tradiční cestě paketu dochází v levé části obrázku při jeho přijetí nejprve k přerušení, paket je následně zkopírován ze zásobníku síťové karty do zásobníku fronty kernelu, poté je odeslán skrz kernel a znovu zkopírován do user space, kde je následně zpracován. Tento datový tok obsahuje několik akcí kopírování paketů, které značně snižují výkonnost. Zero-copy mapuje kernel space (OS) přímo do user space (aplikace) a základní ovladač tak přenáší data přímo do user space přes DMA, čímž eliminuje kopírování mezi kernelem a user space jako v případě tradičního toku. V několika výzkumech bylo prokázáno, že DPDK zvládne přenést až 8krát více paketů za vteřinu, než je to u klasického datového toku skrz kernel space (Padit, Prasad, Bian, & Kwatra, 2018).



Obrázek 4.4: Porovnání tradičního zpracování paketů a Zero-copy, převzato z (Bi & Wang, 2016)

Z výše uvedených důvodů je Zero-copy a jeho implementace v podobě Intel DPDK značně výkonnější, a proto se stala základním kamenem při snaze o snížení doby odezvy a vysoké propustnosti na softwarové úrovni, ať už v našich RE, nebo CE site. Navržená EGW je založena z velké části na existujících open source projektech, které jsme rozšířili o požadované funkce.

Výběr komponenty pro datový provoz

Jako první bylo nutné vybrat a zvolit část pro datový přenos 3. a 4. vrstvy. Při zkoumání existujících řešení jsme narazili na dva nejpoužívanější projekty v open source komunitě softwarově definovaných sítí Open vSwitch a OpenContrail vRouter. Oba typy řešení byly navrženy primárně pro rozvoj telekomunikačních privátních cloudů jako například OpenStack. Open vSwitch je softwarový vícevrstvý virtuální přepínač uvolněný pod licencí Apache 2.0. Byl navržen, aby umožnil masivní škálovaní a síťovou automatizaci (*Open vSwitch Documentation*, 2018). Jeho cílem je být rozumnou alternativou ke komerčním virtuálním síťovým prvkům od společností VMware nebo Cisco. Open vSwitch byl původně navržen jako virtuální přepínač pro softwarově definované řešení Nicira (koupeno společností VMware) namísto klasických linux bridge používaných v KVM linuxové virtualizaci. Jeho úskalí bylo, že do verze 1.X nebylo možné pakety směrovat přímo v Open vSwitch a musel se tak síťový provoz směrovat skrz linuxové kernel namespace. To značně komplikovalo celou architekturu a zpomalovalo síťový provoz. V dnešní době už podporuje širokou škálu protokolů jako NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP apod.

OpenContrail vRouter, dnes již přejmenovaný na Tungsten Fabric vRouter je také pod open source licencí Apache 2.0 a pod záštitou Linux Foundation (*Tungsten Fabric Architecture*, 2018). Původně vznikl jako náhrada za Linux bridge a IP Tables stack pro SDN řešení OpenContrail (koupeno společností Juniper 2012). vRouter implementuje funkci síťového data plane, která umožňuje asociovat virtuální rozhraní s VRF. VRF je virtual routing a forwarding technologie, která umožňuje běžet více oddělených instancím směrovací tabulky v rámci jednoho směrovače. Díky tomu může více logických nebo fyzických rozhraní patřit k různým VRF a nesdílí tak jejich cestu pouze v rámci stejné VRF. Zde je hlavní rozdíl oproti Open vSwitch, který byl navržen primárně pro 2. vrstvu ISO/OSI modelu podobně jako Linux bridge. vRouter, jak již název napovídá, byl od začátku navržen pro směrování na 3. vrstvě ISO/OSI modelu.

Z hlediska nízké doby odezvy a propustnosti nás nejvíce zajímalo, jak si obě řešení vedou ve svém porovnání. Při testování jsme proto nejvíce sledovali síťovou propustnost, která udává, jaká je maximální rychlost přenosu dat přes danou cestu. Ta je zpravidla měřena pomocí přenesených bitů za vteřinu, avšak nezbytnými metrikami jsou také množství přenesených paketů za vteřinu (PPS) nebo latence. Právě PPS je hlavní ukazatel, který se snaží Intel DPDK optimalizovat, jelikož přenesené bity jsou víceméně podobné (Padit et al., 2018). Tabulka 4.2 ukazuje přehled výsledků naměřených testů s 64bajtovými UDP pakety, kde je patrné, že obě řešení dosahují podobných výsledků. Naše testování probíhalo na třech identických serverech s použitím jedné 10Gbits síťové karty 1GB hugepages a 2 jádrech procesoru. Testy byly provedeny pomocí DPDK Test Suite (McNamara, 2018).

Typ	PPS	% z maxima linky
vRouter DPKD	8.50 Mpps	85 %
OpenVSwitch DPDK	8.30 Mpps	83 %

Tabulka 4.2: Výsledky testování propustnosti 64bajtových UDP paketů mezi Open vSwitch a vRouter DPDK, zdroj: vlastní tvorba

Nakonec byl zvolen OpenContrail vRouter z důvodu mírně lepších výsledků, ale také jelikož se stal nejpoužívanějším SDN řešením pro privátní cloudy a běží na něm společnosti jako americký operátor AT&T nebo Saudi Telecom Company (*Tungsten Fabric Architecture*, 2018). Díky těmto případům použití je mimo jiné jisté, že spousta problémů s výkonem a stabilitou je odladěna na těchto rozsáhlých instalacích.

Výběr proxy řešení

Jako další klíčovou komponentu pro integraci bylo nutné navrhnout komponentu pro komunikaci v horních vrstvách ISO/OSI modelu, a to zejména na sedmé vrstvě L7. Důvodem je především podpora distribuovaných aplikací a mikroslužeb, které používají protokoly jako jsou HTTP/2, gRPC, Kafka, MongoDB apod. Tyto protokoly jsou postaveny na typické transportní vrstvě, jako je TCP. Správa a monitoring L7 se tak stává nezbytnou součástí dnešních aplikací. Začali jsme vyhodnocením různých funkcí tří proxy serverů NGINX, HAProxy a Envoy vybraných podle (Saeid & Ali Yahiya, 2018) a (R. Li, 2018).

HAProxy je velmi spolehlivá, rychlá a léty prověřená technologie. Její první verze byla uvolněna již v roce 2006, tedy v době, kdy se internet choval poněkud jinak, než je tomu dnes. Komunita okolo ní není v současné době příliš velká a reaguje velmi pomalu na požadavky nových funkcionalit. Například její verze 1.5 přidala podporu pro SSL po téměř 4 letech čekání. Její největší nevýhodou během provozu a testování se ukázalo dynamické znovu načítání konfigurace bez nutnosti restartu, protože restart znamená ukončení všech aktivních síťových spojení a jejich znovunavázání. To by v praxi znamenalo síťový výpadek na aplikační úrovni při každé změně konfigurace. Nicméně HAProxy byla schopna tento problém vyřešit, avšak až ve verzi 1.8 během roku 2018 (Tarreau, 2017).

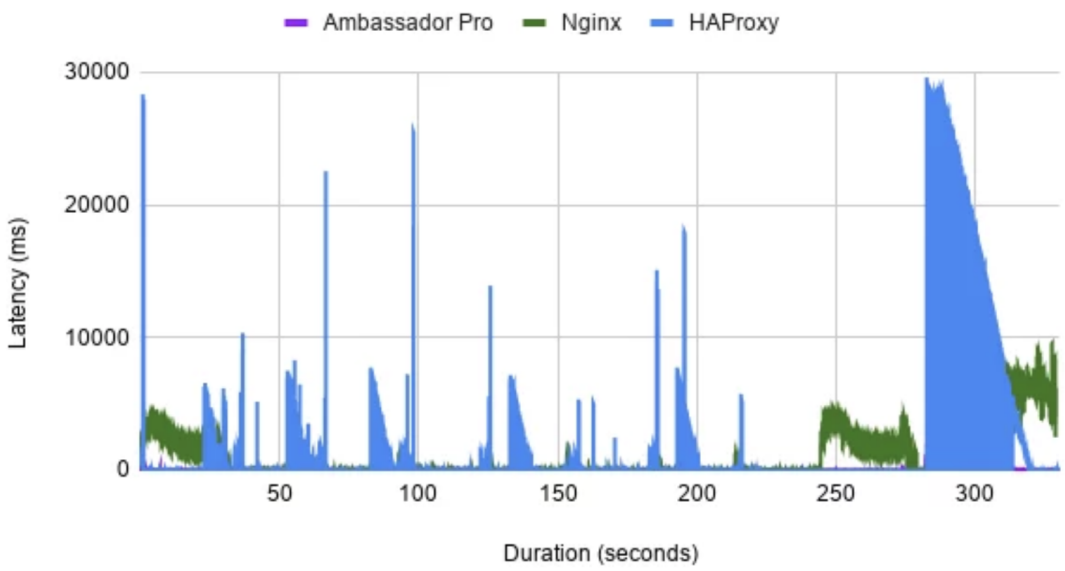
NGINX je vysoce výkonný webový server, který podporuje dynamické znovunačtení konfigurace již dlouhou dobu. Původně byl navržen jako webový server a teprve postupně konvergoval k více tradičnímu proxy užití s příchodem nových funkcí. V základu má dvě varianty - komerční NGINX Plus a NGINX open source. Podle (Garrett, 2016) rozšiřuje komerční verze tu volnou o funkci load balanceru a aplikačního controlleru. Na základě našeho testování a průzkumu před návrhem architektury toto

prakticky znamenalo, že NGINX nemůžeme použít, jelikož jeho otevřená verze měla limitace například ve sledování síťového provozu nebo health check (sledování dostupnosti). Tyto limity byly prokázány také v (Lynch & Matthews, 2017), kde musely zkombinovat HAProxy a NGINX dohromady, aby bylo možné vytvořit globální vysoce dostupný load balancer pro interní i externí vyvažování. Zatímco NGINX má tedy oproti HAProxy rychleji reagující a rozsáhlou komunitu uživatelů, zamknutí některých funkcí se stalo překážkou v jeho využití v rámci našeho návrhu.

Poslední analyzovanou proxy se stal **Envoy**, který je z nich nejnovější a je používaný ve společnostech Google, Lyft, Apple a dalších (*Envoy Proxy*, 2018). Byl primárně navržen pro běh v mikroslužbách s podporou dynamického načítání konfigurací nazývaných hot restart, sledování síťového provozu, spolehlivost a pokročilý load balancing. Envoy také nabízí dynamické API pro jeho konfigurace. Tradiční proxy používají statické konfigurační soubory, zatímco Envoy podporuje jak statickou konfiguraci, tak dynamickou přes gRPC/protobuf APIs. Toto značně zjednodušuje jeho správu v distribuované a rozsáhlé architektuře, jakou edge computing bezpochyby je. Tento nový přístup u Envoy je unikátní současně také díky rozdílnosti komunity okolo tohoto projektu. Zatímco HAProxy a Nginx jsou primárně vlastněné jednou společností, Envoy byl vyvinut společností Lyft, která nemá jako hlavní předmět podnikání prodej distribuovaného load balanceru. Namísto toho prostě potřebovala vyřešit svůj problém s provozem svého řešení. V neposlední řadě je Envoy součástí CNCF Foundation[2], která je součástí Linux Foundation[3] zaštiťující tento open source projekt.

V neposlední řadě jsme nalezli několik provedených měření (R. Li, 2018), abychom měli jistotu, že Envoy v porovnání s NGINX a HAProxy obstojí. V opačném případě bychom nemohli naplnit náš cíl pro nízkou dobu odezvy a velkou propustnost platformy. Ve stručnosti byl měřen počet dotazů za vteřinu na všech třech řešeních, během něhož byly dynamicky přidávány a odebírány koncové body. Obrázek 4.5 ukazuje výsledky testování, z nichž je patrné, že Envoy dopadl nejlépe. Výkyvy v grafu značí právě velké množství přidávaných koncových bodů během generování požadavků na proxy.

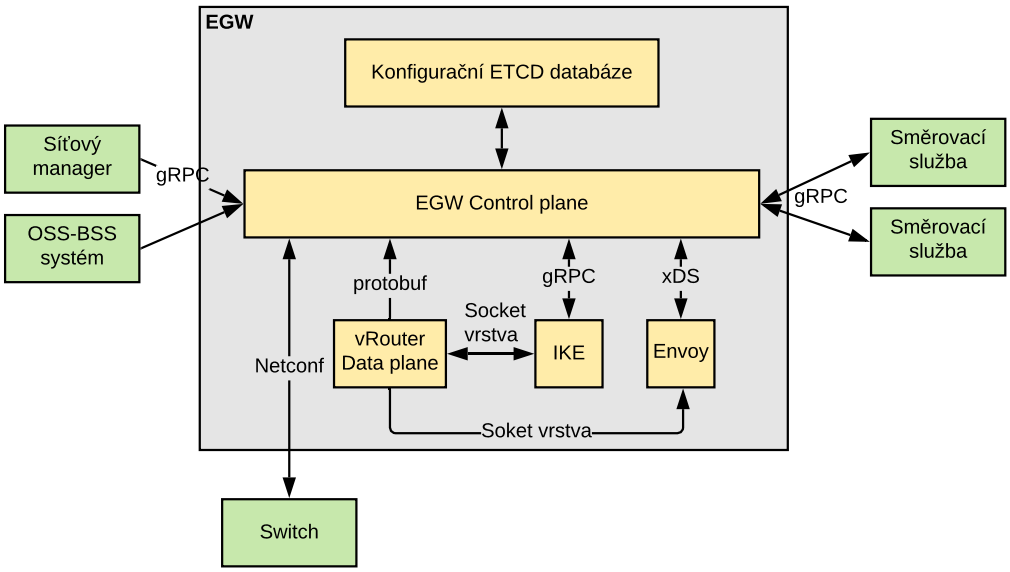
[2]https://www.cncf.io/
[3]https://www.linuxfoundation.org/



Obrázek 4.5: Porovnání doby odezvy při 1000 dotazech za vteřinu RPS, převzato z (R. Li, 2018)

Architektura EGW

Na základě rozboru klíčových komponent pro síťový provoz jsme navrhli architekturu pro novou Edge gateway, která je zachycena na obrázku 4.6. EGW je součástí každého zařízení nebo virtuálního serveru v edge site. Její samotná implementace byla provedena týmem vývojářů specializujících se na low level programování. Autor této práce provedl návrh, výběr a testovaní jednotlivých komponent.



Obrázek 4.6: Architektura navržené EGW, vlastní tvorba

Hlavou celého řešení je EGW control plane, který má 3 základní role:

- Control plane pro jednotlivou instanci - v této roli je CP zodpovědný za správu a běh na každé instanci a dává řídicí instrukce:
  - EGW Data plane postavený na Tungsten Fabric (vRouter),
  - ADC - Application Delivery Controller konfigurující Envoy proxy,
  - IKE - konfigurace IPSec tunelů postavená na technologii Strongswan[4].
- Control plane pro cluster několika instancí - pouze jedna instance v clusteru může být hlavní neboli lídr. Tato funkcionalita je zodpovědná za mechanismus volby lídra v rámci clusteru. Zvolený EGW-CP poskytuje virtuální IP adresy VIP a slouží také jako brána pro příchozí provoz. Poslední funkcí je schopnost konfigurovat externí síťové prvky, jako jsou například přepínače skrz protokol Netconf.
- API server - Tato funkcionalita je odpovědná za poskytnutí standardního aplikačního rozhraní pro konfigurace EGW. Všechny instance CP přijímají konfigurační změny na jejich API.

Persistentní konfigurace EGW-CP ukládá do ETCD databáze, která je navržena jako distribuované spolehlivé key-value úložiště pro přístup z mnoha míst zároveň. Je také

[4]https://www.strongswan.org/

součástí CNCF Foundation podobně jako Envoy proxy a stala se v posledních letech velmi populární, a to především díky popularitě Kubernetes. Více o něm v kapitole 4.4. ETCD není použita pouze jako úložiště konfigurace EGW-CP, ale také na vytváření dynamických zámků pro volbu lídra v clusteru. Každá instance v EGW clusteru se snaží vytvořit klíč s hodnotou ve stejné cestě. První, které se to povede, se stává lídrem, a tedy cluster CP.

EGW Data plane je modul na procesování paketů v EGW založený na Tungsten Fabric DPDK vRouter. Pakety přicházející z fyzických nebo virtuálních rozhraní. Nejprve dorazí na Data Plane, který vlastní síťové rozhraní při startu a je zodpovědný za všechny I/O operace. Jeho základní funkcionality jsou:

- Směrování paketů
- Podpora pro více VRF
- Terminace TCP spojení
- Bezpečnostní funkcionality IP firewall pravidel
- Rate Limiting
- VPN terminace
- NAT Gateway

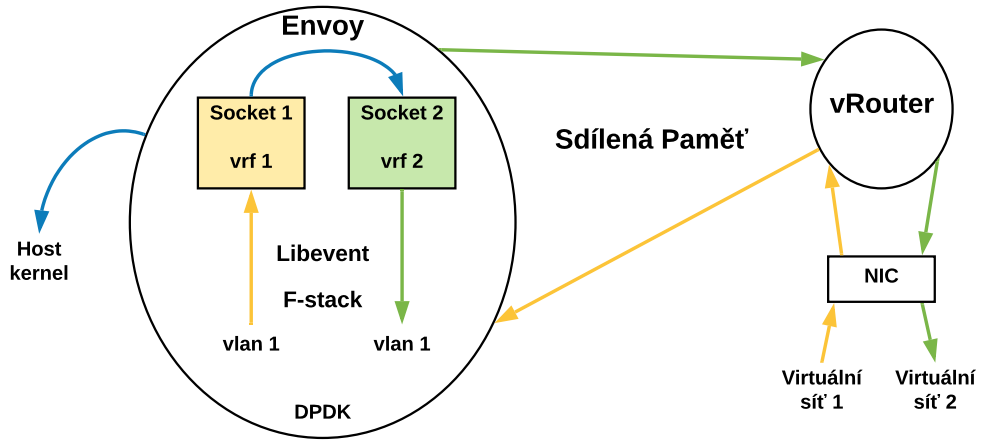
EGW Data plane používá DPDK framework současně s F-Stack TCP/IP řešením pro terminaci TCP/UDP síťového provozu. F-Stack je open source síťový framework pro optimalizaci výkonu (*F-Stack*, 2019). F-Stack framework původně vznikl jako řešení na odpověď DDoS útoků směřovaných na veřejné DNS servery. Primárním problémem bylo právě kopírování a vysoké přerušení při průchodu linuxovým jádrem. Integrací DNS server aplikace do DPDK bylo možné zpracovávat 11 milionů QPS na jednom 10GE síťovém portu. Dnes má F-Stack různé integrace, například s databází Redis nebo NGINX proxy.

F-Stack je tedy hlavním integračním bodem mezi EGW Data plane založeném na Tungsten Fabric vRouter a Envoy proxy, která je součástí tzv. ADC neboli Application Delivery Controller. Envoy díky integraci s F-Stack tak běží jako DPDK aplikace na vRouter. ADC je naším pojmem, kterým označujeme právě rozšíření standardní envoy proxy a je zodpovědná za:

- Vyvažování HTTP, TCP a UDP připojení včetně kontroly dostupnosti (health check)
- SSL terminace

- Podporu pro běh vlastního Javascript kódu

EGW-CP konfiguruje Envoy skrz náš ADC manager napsaný v jazyce Golang a jeho úkolem je být xDS serverem, ke kterému je připojen Envoy xDS klient. xDS je protokol používaný v Envoy na jeho konfiguraci a dynamické změny. Je založen na gRPC stream, což je metoda, kdy klient poslouchá server a dostává všechny změny okamžitě. Obrázek 4.7 zobrazuje námi navržený tok paketů v rámci vRouteru a Envoy z jedné do druhé virtuální sítě. Paket přijde na fyzické rozhraní, odkud ho DPDK vRouter v rámci své VLAN (oranžová virtuální síť 1) sítě odešle do F-Stack, kde ho následně Libevent zpracuje skrz kontinuální volání F-Stack. Libevent je existující knihovna používaná v Envoy pro notifikace mezi událostmi. Následně se paket dostane do virtuální směrovací tabulky soketu v Envoy (žluté vrf 1), ten ho předá do druhého soketu, a to vše v rámci sdílené paměti. Celé flow se děje přímo v user space, neprochází kernelem a díky tomu je možné zamezit duplicitní kopírování paketů a vysoký počtu systémových přerušení. Tato implementace by měla dokázat zvednout počet požadavků za vteřinu až 5krát, což bude ověřeno v rámci kapitoly 5. Celkem bylo nutné implementovat několik změn v Envoy, aby mohl být plně integrován do DPKD jako například podpora pro více VRF v rámci Envoy, integrace Libevent volání do F-Stack pro navazování síťových spojení nebo přidání podpory pro virtio rozhraní do F-Stack konfigurace. Samotná implementace změn kódu nebyla provedena autorem této práce, pouze její část návrhu architektury, jak již bylo zmíněno v předchozím textu.



Obrázek 4.7: Posílání paketů v integraci Envoy integraci do DPDK, zdroj: vlastní tvorba

Poslední komponenta v celém řešení je nazvaná IKE a je zodpovědná za udržování šifrovacích parametrů pro IPSec a spojení mezi EGW instancemi. IKE poslouchá na gRPC kanále a přijímá požadavky od EGW-CP, aby se naučila všechny IPSec tunely.

Jakmile je tunel vytvořený, IKE nakonfiguruje síťová rozhraní v EGW-DP s příslušnými IPSec parametry. IKE také komunikuje s EGW-DP skrz sokety exportované EGW-DP. IPSec tunel může být vytvořen ze dvou důvodů:

- Site-to-Site tunely mezi jednotlivými RE site,
- VPN koncentrátor pro správu VPN tunelů z koncových CEC site.

4.3.2 EGW proxy jako základ pro Service Mesh

Jak již bylo zmíněno v úvodu, doba latence je jen jedním ze síťových problémů v edge computingu. Současně je potřeba také řešit problém dostupnosti a spolehlivosti komunikujících koncových bodů, jelikož aplikační entity jsou dnes napsané v různých programovacích jazycích. Tento problém otevírá v posledních letech popularitu nového konceptu nazvaného Service Mesh (Indrasiri & Siriwardena, 2018). Tu lze definovat jako konfigurovatelnou vrstvu infrastruktury s nízkou latencí, navrženou pro zpracování velkého objemu síťové meziprocesorové komunikace mezi službami aplikační infrastruktury pomocí API. Service Mesh zajišťuje, že komunikace mezi kontejnerovými a často nepersistentními službami aplikační infrastruktury je rychlá, spolehlivá a bezpečná. Jejím cílem je poskytnout kritické funkce, včetně objevování služeb, vyvažování zátěže, šifrování, viditelnosti, autentizace a autorizace atd. (W. Li et al., 2019). Tyto důvody nás vedly k myšlence integrovat CEC platformu právě do konceptu Service Mesh a vyřešit tak problémy s komunikací rozsáhlých distribuovaných aplikací napříč tisíci lokalitami nebo mezi cloud computing a edge computing mikroslužbami. Klíčová komponenta navržené EGW je Envoy, která je shodou okolností používána pro integrace do různých řešení Service Mesh jako například Istio[5] propagované společností Google.

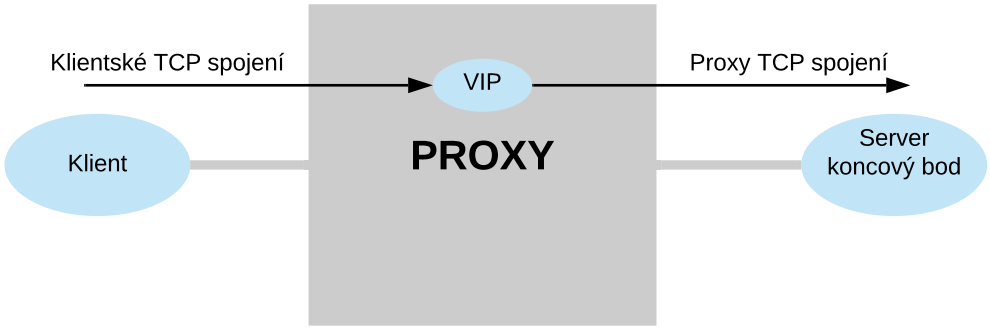
Základem konceptu Service Mesh je využití proxy a load balancer stojící před jednotlivými aplikačními instancemi tak, aby bylo možné kontrolovat kompletní síťový provoz mezi jednotlivými službami a to jak uvnitř, tak směrem do internetu. V případě navržené CEC platformy je jako load balancer použita proxy v podobě EGW, která je definována jako entita ukončující příchozí TCP nebo UDP připojení a inicializující nové spojení dále (Sysel & Doležal, 2014). Server je označován jako koncový bod a je obvykle kolekcí nebo sadou koncových bodů, které poskytují určitou službu. Klienti i servery mohou být: uživatelé, stroje, aplikace nebo mikroslužby. Jak již bylo zmíněno, existuje spousta důvodů pro nasazení proxy mezi klientem a serverem:

- Propojení klienta se serverem skrz izolované prostředí,
- Vyvažování a směrování na 7. vrstvě aplikační úrovně, která se využívá například u API Gateway,

[5] `https://istio.io/`

- Bezpečnost, jako je například URL filtrování, aplikační firewall, aplikační politiky, TLS offloading atd.,
- Síťová a aplikační spolehlivost,
- Content caching neboli ukládání do mezipaměti je mechanismem optimalizace výkonu, ve kterém jsou data doručována z nejbližších serverů pro optimální výkon aplikace,
- Optimalizace TCP spojení.

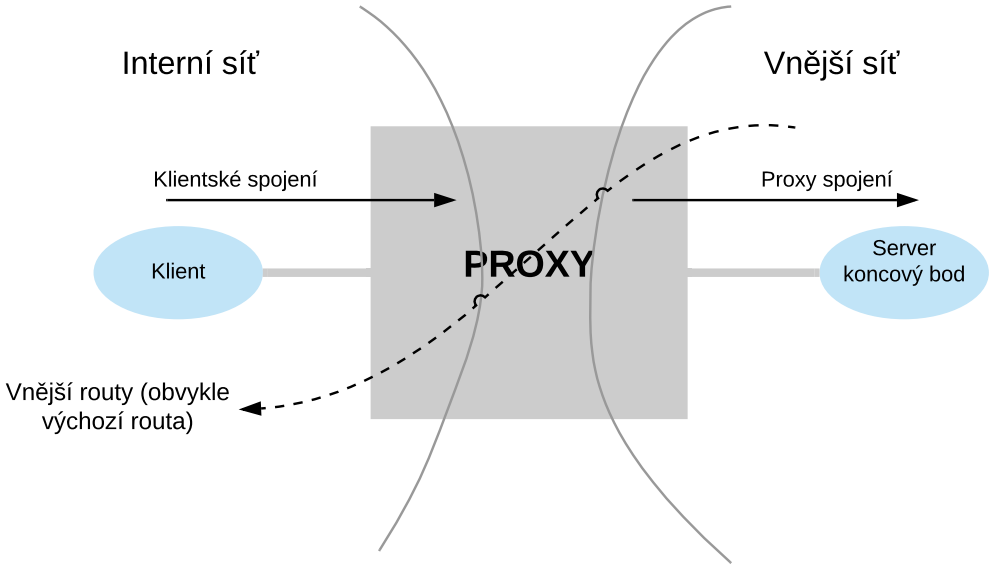
Jak je zobrazeno na obrázku 4.8, proxy reprezentuje virtuální IP (VIP) serveru pro přístup klienta. Klient musí nejprve zjistit VIP, což se obvykle dělá pomocí DNS, kdy klient provede požadavek na překlad DNS jména na IP. Existují i další možnosti zjišťování služeb, kterých mohou klienti využít - například orchestrační nástroj Kubernetes nebo Hashicorp Consul. Proxy control plane musí zajistit, aby klient dokázal zjistit tuto IP adresu serveru, čehož je dosaženo zveřejněním VIP právě její propagací do servisního registru nebo DNS. Současně musí proxy podobným způsobem zjistit IP adresy koncových bodů vázajících se na server, kde se používají stejné metody pro zjišťování z DNS, registru služeb atd.



Obrázek 4.8: Funkce proxy v rámci CEC platformy, zdroj: vlastní tvorba

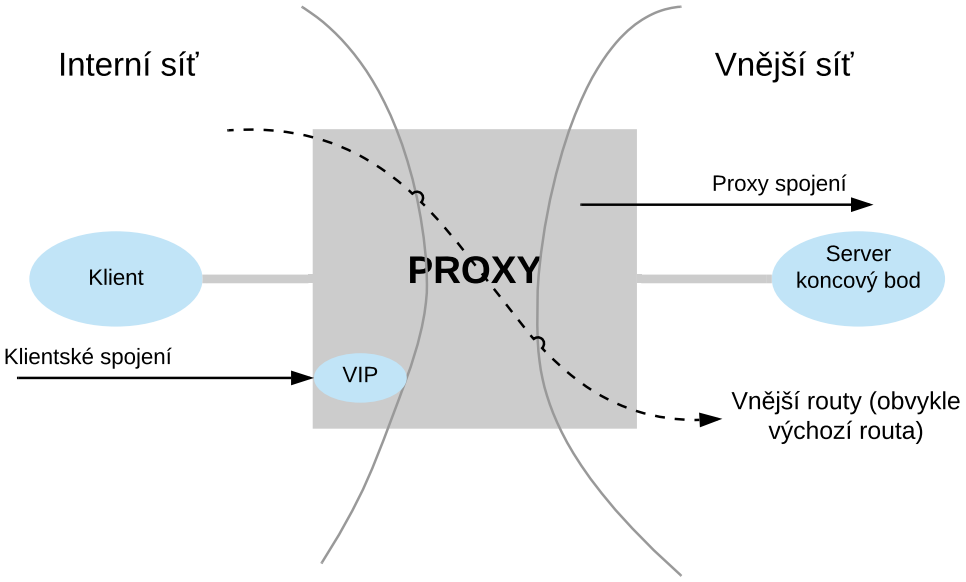
Navržená CEC platforma poskytuje dva standardní módy proxy - forward proxy a reverse proxy (Cloudflare, 2019). **Forward proxy** se obvykle konfiguruje pro propojení vnitřní privátní sítě s vnější veřejnou sítí. Zde neexistuje žádná VIP a síťový provoz je obvykle směrován pomocí výchozí brány do vnější sítě. Uživatel může nakonfigurovat směrování mezi oběma sítěmi pomocí síťového konektoru. DNS se používá k nalezení IP adres serveru. Jednou z vlastností forward proxy je to, že IP adresa serveru/koncových bodů je stejná v interní i vnější sítí. Díky tomu může uživatel řešit problémy, jako

jsou implementace filtrování adres URL pro provoz HTTP, provádět kontrolu TLS a filtrovat na základě jmen DNS atd. Forward proxy na obrázku 4.9 si lze částečně představit jako mód domácího Wifi zařízení, kdy veškerý provoz směrem na internet prochází skrze toto zařízení.



Obrázek 4.9: EGW Forward Proxy na CEC platformě, zdroj: vlastní tvorba

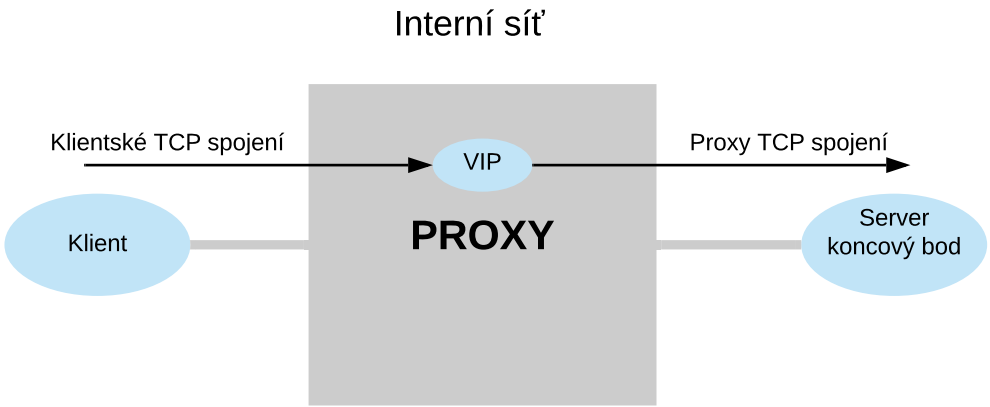
Druhým způsobem použití je **reverse proxy**, a to v případě, kdy je server ve vnitřní síti a klienti na něj přistupují z vnější sítě. Server tak není dosažitelný z venku, pokud před ním není nasazena proxy. Pro zajištění připojení k serveru je proxy nakonfigurována jako „reverzní" a server dostane přidělenou VIP z vnější sítě. To umožňuje uživateli řešit problémy, jako je řízení příchozího provozu a implementovat bezpečnostní kontroly, aby se zabránilo útokům na server, které hrozí při jeho přímém vystavení. Obrázek 4.10 zobrazuje tento typ konfigurace.



Obrázek 4.10: EGW Reverse Proxy na CEC platformě, zdroj: vlastní tvorba

4.11

Na obrázku 4.11 je zobrazena také konfigurace, kdy se nerozlišuje mezi vnitřní a vnější sítí. Tudíž může být reverse proxy využita i pro interní řízení toku jako vyvažování nebo omezování přístupu. Tento způsob použití je přesně typickým použitím v rámci konceptu Service Mesh.



Obrázek 4.11: EGW Reverse Proxy v rámci stejné sítě na CEC platformě, zdroj: vlastní tvorba

V CEC platformě je tedy load balander a reverzní proxy stejná věc konfigurovaná tak, aby měla více koncových bodů pro každý server. Distribuuje zatížení příchozích požadavků klientů na základě různých algoritmů, například round robin, hashovacího kruhu atd. Vybírá pouze aktivní koncové body na základě explicitně nebo implicitně (dedukce založená na době odezvy, míře chybovosti atd.) definovaných sond pro kontrolu stavu. To umožňuje uživateli vyřešit několik problémů:

- Nastavit směrovací pravidla pro kontrolu toku aplikačního provozu,
- Autentizace a autorizace,
- Zjednodušení konfigurace vlastností služeb, jako jsou timeout, opakované požadavky, circuit-breaker [6] atd.,
- Nastavit rozdělení provozu na koncové body na základě testování nových verzí, například Canary[7] nebo A/B upgrade strategie,
- Implementovat bezpečnostní kontroly a aplikační politiky předcházející různým útokům,
- Poskytnout komplexní možnosti pozorování a trasování při řešení problémů.

Reverse proxy je v rámci CEC platfromy a dále v práci také nazývána a konfigurována jako virtual-host, který musí být přiřazen k VIP. Ten obsahuje také list doménových názvů, které se k dané VIP vážou. Vytvořili jsme čtyři typy reverse proxy podporované skrz virtual-host.

- TCP proxy - mód, při kterém je příchozí TCP připojení odesláno na vybraný odchozí server. Každá TCP proxy vyžaduje kombinaci VIP a TCP portu.
- HTTP proxy - pokud příchozí připojení TCP používá protokol HTTP, lze virtual-host nakonfigurovat tak, aby analyzoval hlavičku HTTP. Na základě hlavičky HOST v HTTP protokolu a požadované adresy URL může být provoz nasměrován na jinou sadu koncových bodů. Výběr těchto koncových bodů a adres URL se nazývá aplikační směrování. Daný virtual-host může být identifikován na základě host hlavičky a musí obsahovat doménu, která je v něm nakonfigurována. Kombinace portu a VIP tedy může být poté sdílena mezi více virtual-host, které jsou také nakonfigurovány jako HTTP proxy.

[6] Cirtuit-breaker je koncept pocházející z elektřiny a jeho cílem je zamezit kaskádovému šíření problému například v případě výpadku jedné služby.

[7] Canary je technika, jak snížit riziko zavedení nové verze softwaru v produkci pomalým zaváděním změny do malé podskupiny uživatelů, namísto změny všem uživatelům najednou.

- HTTPS proxy - pokud je příchozí připojení TCP šifrováno pomocí protokolu HTTPS, lze virtual-host nakonfigurovat pomocí parametrů TLS, které lze použít k dešifrování paketů a funkce nazývané HTTPS proxy. Všechny funkce směrování a řízení provozu jsou stejné jako u HTTP proxy.
- TCP Proxy s SNI - pokud je příchozí tok TLS, ale virtuální hostitel nemá tyto TLS parametry k dešifrování paketu, i přesto může virtual-host použít informace SNI v paketu, aby získal informace o cílených doménových názvech. Po přiřazení patřičného virtual-host se příchozí datový provoz TCP odešle na vybraný koncový bod serveru. Tomuto módu se říká TCP proxy s SNI. VIP a port lze sdílet mezi HTTPS a TCP proxy s SNI.

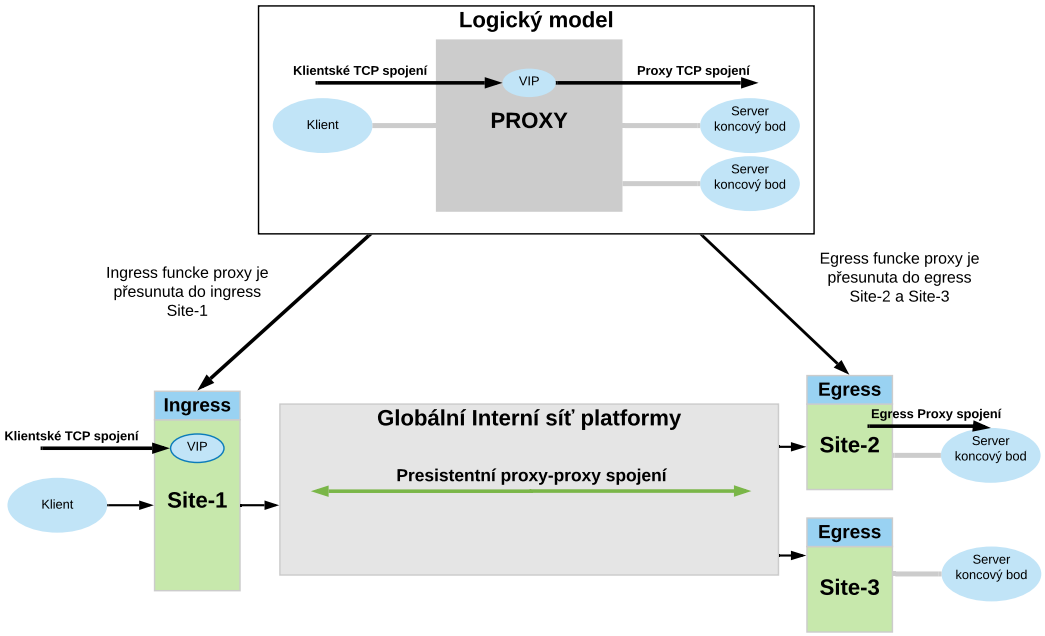
4.3.3 Distribuovaná aplikační brána

Aplikace v Service Mesh se tedy obvykle skládají z mnoha služeb, které jsou reprezentovány více než jedním virtual-host. Z toho důvodu dává smysl seskupit tyto virtual-host, aby bylo možné lépe porozumět interakci služeb a pozorovat ji jako celek. Sběrem metrik z různých služeb a jejich interakcí klientů a serverů je možné vytvořit graf služeb. Kromě toho je také možné skrz přístupové protokoly a strojové učení identifikovat koncové body API napříč celým servisním grafem sítě. Pro každý z těchto koncových bodů API je následně možné aplikovat například umělou inteligenci a statistické algoritmy pro detekci anomálií, distribuci pravděpodobnosti apod. Spojením všech těchto informací z proxy dostáváme koncept, který v rámci CEC platformy nazýváme distribuovanou aplikační bránou. Jejím úkolem je řešit dvě věci:

1. Bezpečnost a připojení - použitím CEC platformy je možné vytvořit privátní a bezpečné propojení mezi lokalitami a postavit tak distribuovanou aplikační proxy.
2. Globálně distribuované aplikační směrování a kontrola - protože je server proxy distribuován a stav všech koncových bodů je k dispozici na více site, lze nyní rozhodovat o směrování a řízení provozu na základě skutečného stavu jednotlivých koncových bodů na všech site, a nikoli pouze na koncových bodech proxy. To je zvlášť užitečné pro webové služby s náročným provozem.

Obrázek 4.12 zachycuje příklad distribuované proxy napříč dvěma typům edge site - ingress a egress. O sadě vstupních (ingress) site se rozhoduje pomocí nastavení propagační politiky. V nejběžnějším případě je VIP vybrána automaticky jako IP adresa rozhraní site. Vždy musí být definován protokol a port, na kterém je služba k dispozici. Pokud má site více sítí, musí být zvolena i síť pro VIP. Výsledkem je propsání VIP do registru služeb (DNS nebo Kubernetes service). Na základě tohoto propsání dokáže

klient zjistit VIP a navázat spojení do jednoho z uzlů v ingress site. V tomto případě se klient připojuje na VIP site-1. V závislosti na strategii vyvažování zátěže a politikách nastavených pro virtual-host cluster vybere ingress site proxy cílovou egress site proxy. Příkladem politiky může být výběr nejbližšího místa připojení nebo podle doby odezvy. Následně se zahájí spojení s výstupním serverem a jeho koncovým bodem. V případě HTTP a HTTPS proxy je použito existující a trvalé síťové spojení mezi ingress a egress, aby bylo dosaženo většího výkonu.



Obrázek 4.12: Distribuovaná aplikační brána s použitím proxy v CEC platformě, zdroj: vlastní tvorba

O sadě výstupních (egress) site se rozhoduje podle konfigurace koncových bodů. Koncový bod může specifikovat site a metodu zjišťování k nalezení koncových bodů. Jakmile jsou objeveny všechny koncové body, egress proxy server začne automaticky provádět kontroly dostupnosti. V případě, že je koncový bod dostupný, začne propagovat přístupové informace o svých koncových bodech do všech ostatních site v rámci CEC platformy.

Díky tomuto návrhu je možné přenést funkce Service Mesh, používané především ve veřejném cloudu (Google, 2019), do prostředí edge computingu a spravovat tak tisíce lokalit jako jednu distribuovanou aplikační bránu. Při pohledu na dostupné řešení IoT platforem z kapitoly 3.4.5 žádná z nich se nezabývá problémem distribuovaných aplikací a jejich komunikací a ani je neřeší.

Nedílnou součástí je i detailní sledování síťového provozu, bohužel to není možné obsáhnout v rámci rozsahu této práce.

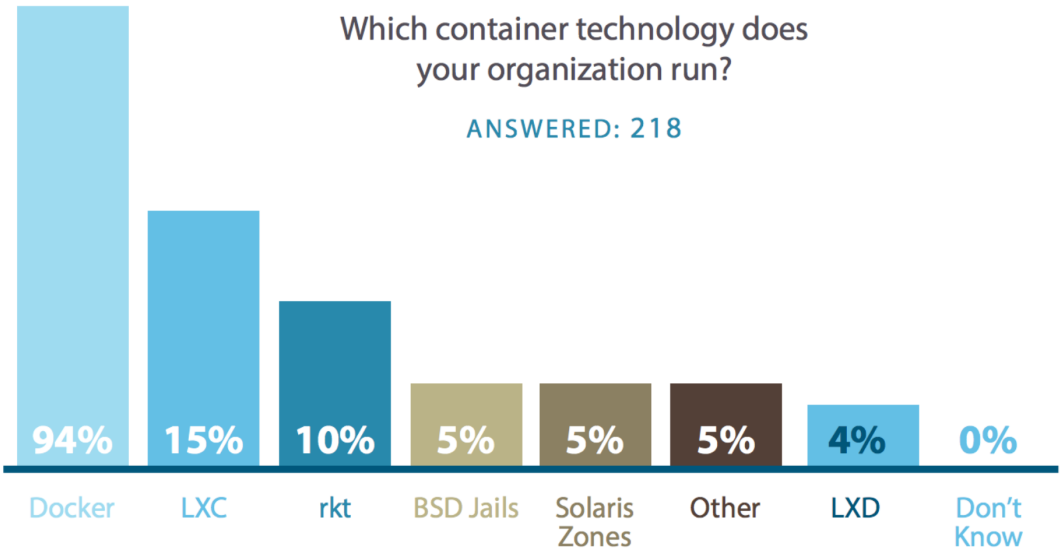
4.4 Multifunkční edge

Jak již bylo zmíněno v kapitole 3.3.2, edge computing otevírá výzvu využití obecných výpočetních nodů a současně využití virtualizace a mikroslužeb v těchto nodech či zařízeních. Na základě toho by bylo možné z jednoúčelových klientských zařízení komunikujících na servery v cloudu utvořit multifunkční edge, který dokáže běžet oddělené aplikace s různými vlastníky a přístupy. K naplnění multifunkčnosti je nutné vyřešit i multi-tenancy (Odun-Ayo, Misra, Abayomi-Alli, & Ajayi, 2017). Multi-tenance je architektonický vzor, ve kterém je jediná instance softwaru spuštěna na infrastruktuře poskytovatele služeb a více tenantů přistupuje ke stejné instanci (Bezemer & Zaidman, 2010). V případě edge zařízení se tedy jedná o fyzické sdílení zdrojů různými uživateli. V této části popisu CEC řešení se zaměříme právě na návrh CEC site z pohledu logického rozdělení. Tato izolace se provádí skrze virtualizaci. Jejím hlavním cílem je zefektivnit využití hardwarových zdrojů. V dnešním cloud computing prostředí existují dva hlavní typy virtualizace - kontejnerová a virtuální servery tzv. VM (Yadav, Garg, & Mehra, 2019).

Virtuální servery (VM) jsou centrální částí cloud computingu sloužící k poskytnutí infrastrukturní vrstvy celých operačních systémů. VM poskytuje uživateli kompletní operační systém, na kterém může uživatel pracovat s různými aplikačními programy. Současně všechny jeho periferní zařízení jsou také emulovány pomocí virtualizace. Kontejnerová virtualizace poskytuje podobný koncept, avšak je mnohem méně náročná na spotřebu zdrojů. Kontejnery totiž sdílejí stejné jádro hostitele, tedy fyzického stroje, a z toho důvodu jsou mnohem rychlejší a méně náročnější (Ali Babar & Ramsey, 2017). Původně vznikly jako jakýsi způsob doručování aplikačních balíků, jednoduše přenosný mezi prostředími. Z analýzy současného stavu edge computingu vyplývá, že poslední vrstva edge site jsou spíše zařízení s nízkou kapacitou zdrojů. Například některé IoT zařízení poskytují maximálně 8GB paměti RAM. Současně také některé z existujících řešení jako Cisco IoT nebo Azure IoT již využívají kontejnerové virtualizace, a proto jsme se zaměřili na kontejnerovou izolaci uvnitř CEC platformy.

4.4.1 Výběr řešení pro kontejnerovou izolaci

Podle průzkumu zachyceném na obrázku 4.13 existují tři nejpoužívanější kontejnerové izolace: Rkt, Docker a LXD. Během výběru řešení jsme je zanalyzovali a porovnali jejich vhodnost pro CEC řešení.



Obrázek 4.13: Nejpoužívanější kontejnerové technologie, převzato z (Nanobox, 2017)

Docker kontejnerová virtualizace je považována za nejvíce vyzrálou z výše zmíněných (Ali Babar & Ramsey, 2017). Jeho systém je založený na image, který definuje obsah a prostředí instance kontejneru. Jeho hlavní součástí je docker engine, což je systémový démon běžící v hostitelském systému pod účtem administrátora. Všechny operace jsou prováděny právě tímto démonem a to včetně správy kontejnerů, jejich image i vytváření nových image. Správa kontejnerů v Dockeru zahrnuje vytváření, spouštění, zastavování, mazání a pozastavení kontejnerů. Koncepčně to znamená, že celý tento životní cyklus kontejneru musí projít skrze jednoho démona, který se tím stává potenciálním bodem selhání celého řešení.

Rkt je relativně nový kontejnerový systém vyvinutý společností CoreOS jako alternativa k Docker. Od začátku jeho návrhu byl důraz kladen na bezpečnost. Rozdíl oproti Docker je především v tom, že je implementován podle appc specifikace pro aplikační kontejnery, na rozdíl od open container specifikace. Jeho největším rozdílem oproti ostatním je také to, že nevyžaduje žádného centrálního démona, přes kterého musí jít všechny řídicí operace. Namísto toho je každý kontejner v Rkt oddělený systémový proces. Tento rozdíl je využíván především pro snadnou integraci do init systémů v Linuxu (např. Systemd a upstart), čímž jim umožňuje řídit kompletní životní cyklus Rkt kontejneru. U Docker je použit jeden z init systémů s jeho enginem a tím pádem musí provádět všechny operace skrze něj. Díky tomu je možné dosáhnout větší bezpečnosti. Jelikož není nutné dávat plná administrátorská práva démonovi, je možné uplatnit různé přístupy jednotlivým Rkt kontejnerům podobně jako standardním procesům v OS.

LXD kontejnerový systém umožňuje jednoduché vytváření LXD kontejnerů vyvinuté společností Canonical. Skládá se ze dvou komponent - systémový démon a klient pro příkazový řádek. Systémový démon poskytuje RESTful API, které může být použito jak lokálně, tak vzdáleně ke správě kontejnerů. Jeden z hlavních rozdílů mezi implementacemi LXD a Rkt/Docker je v použitém souborovém systému. Zatímco Rkt používá OverlayFS a Docker aufs, jedná se v obou případech o tzv. union-based souborový systém. To znamená, že tento souborový systém běží na existujícím systému hostitele, takže dochází ke spojení adresářů mezi kontejnery a jejich hostitelem. Zatímco LXD využívá souborový systém běžící v user space (FUSE), jehož cílem je poskytnout kontejnerům podobnou emulaci jako u tradičních VM (LXC, 2019). LXCFS používané u LXD tak poskytuje neprivilegovaný souborový systém pro kontejnery, který poskytuje cgroupfs kompatibilní pohled na neprivilegované kontejnery. To znamená, že dokáže poskytnout emulaci adresářů jako /proc a /sys/fs/cgroup pro neprivilegované kontejnery. Díky tomu je možné běžet LXD kontejnery bez nutnosti privilegovaného přístupu k hostovi, jako je tomu u Dockeru nebo Rkt.

Z našeho zkoumání vyplynulo, že Docker je nejvyspělejší technologií používanou na nasazování aplikací v dnešním prostředí i přesto, že bezpečnost nebyl její primární cíl. Mezi jeho největší problémy patří již zmiňovaná správa přes centrální engine a síťování. Jelikož v CEC platformě jsme navrhli vlastní EGW síťové řešení, tak se nás tento problém netýká. CEC platforma tedy implementuje Docker jako výchozí kontejnerovou virtualizaci.

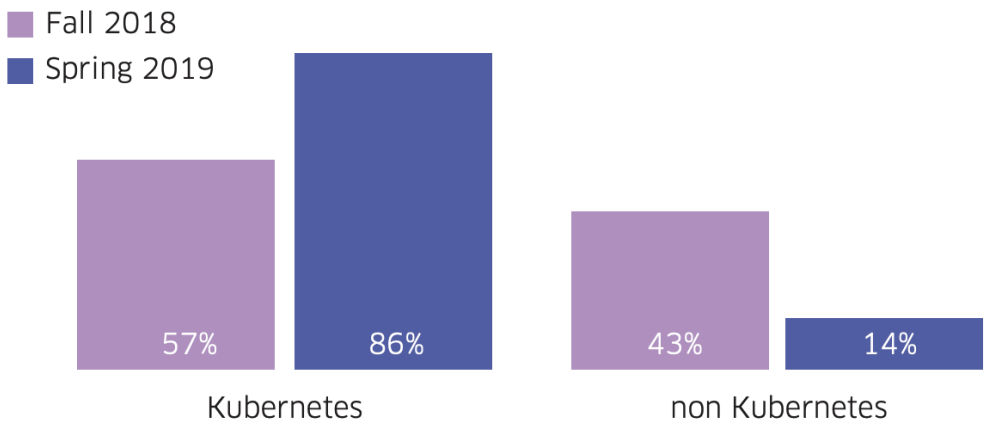
4.4.2 Výběr orchestrátoru kontejnerů

S příchodem kontejnerů přichází i provoz mikroslužeb. Architektura mikroslužeb rozděluje aplikace na menší samo funkční bloky, z nichž každý je reprezentován jedním kontejnerem. Rozsáhlé distribuované aplikace tak mohou dosahovat desítek funkčních bloků, které se škálují na desítky instancí. V součtu tak může taková distribuovaná architektura obsahovat stovky i tisíce kontejnerů. S tím přichází i problém jak spravovat tyto kontejnery a zaručit vždy běh jejich požadovaného počtu. Zde přichází na řadu tzv. orchestrátory kontejnerů provádějící úkony, jako je spouštění požadovaného kontejnerového image na správném serveru nebo distribuce v závislosti na zatížení zdrojů apod. Dále se starají o přesouvání kontejnerů v případě výpadků serverů nebo zajišťují jejich správnou síťovou konfiguraci. Podle (Casalicchio, 2019) můžeme klíčové požadavky na správu kontejnerů shrnout do následujících bodů:

- Spouštění a nasazení kontejnerů
- Redundance a dostupnost kontejnerů

- Dynamické přidávání nebo ubírání kontejnerů podle jejich stavu vytížení
- Přesun kontejnerů z hostitele, který je přetížený nebo má výpadek na jiného hostitele
- Alokace zdrojů mezi jednotlivými kontejnery
- Externí přístup na služby poskytované kontejnerem z vnějšího světa
- Load balancing a automatické objevování služeb
- Sledování stavu kontejneru v podobě health check
- Správa konfigurací pro kontejnery

V době výzkumu byly na trhu dostupné dvě open source řešení na kontejnerovou orchestraci Docker Swarm a Kubernetes. Teprve během roku 2019 vyšlo jasně najevo, že Kubernetes je víceméně standardem v kontejnerové orchestraci, jak je vidět z nárůstu počtu uživatelů za 6 měsíců na obrázku 4.14. I přesto je dobré se podívat na jejich porovnání.



Obrázek 4.14: Nárůst počtu uživatelů Kubernetes v otázce používaného kontejnerového orchestrátoru, převzato z (Bouchard, 2019)

Docker Swarm je nativní orchestrace obsažená v Docker pro správu a management několika Docker engine hostů. Díky tomu není nutné spravovat individuální Docker hosty. Ve vztahu k edge computing je jeho největší výhodou právě nativní běh přímo v Dockeru bez nutnosti instalace a správy dalších komponent. Díky tomu je možné snížit nároky na CPU a paměť RAM pro management edge site. Zároveň je použito stejné Docker API jako v případě samostatné kontejnerové virtualizace a tím výrazně

snižuje křivku učení pro uživatele. Naopak podle (Casalicchio, 2019) je nevýhodou jeho robustnost a stabilita především ve větší škále desítek serverů.

Kubernetes je open source projekt pro kontejnerovou orchestraci inspirovaný interním projektem společnosti Google nazvaný Borg (Awada, 2018). Kubernetes se používá na orchestraci kontejnerů, které reprezentují instance aplikací. Kubernetes reprezentuje tradiční klient-server architekturu, kde tzv. master node má globální přehled o clusteru a je zodpovědný za rozhodování o alokaci zdrojů a přiřazení jednotlivých kontejnerů. Uživatel komunikuje s masterem skrz REST API, webové rozhraní nebo klienta v příkazové řádce CLI. Master node komunikuje s tzv. worker servery, které hostují kontejnerové aplikace. Hlavním důvodem pro volbu Kubernetes jako orchestračního řešení pro CEC platformu bylo v roce 2018 především nezávislost na kontejnerovém systému. I přes výběr Docker řešení bylo cílem navrhnout modulární systém, který umožní nahradit konkrétní kontejnerový systém v případě potřeb. Výsledky komplexního testování jsme publikovali v (Mercl & Pavlik, 2019a) a jsou vidět v tabulce 4.3.

	Compose	**Swarm**	**Fleet**	**Mesos**	**Kubernetes**
Podpora kontejnerového runtime	Docker	Docker	Docker, rkt	Docker, rkt	Docker, rkt, cri-i, Windows kontejnery
Škálování	Ano	Ano	Ano	Ano	Ano
Převzetí služeb při selhání	Ne	Ano	Ano	Ano	Ano
Vysoká dostupnost	Ne	Ano	Ano	Ano	Ano
Síťové pluginy	Ne	Ano	Ne	Ano	Ano

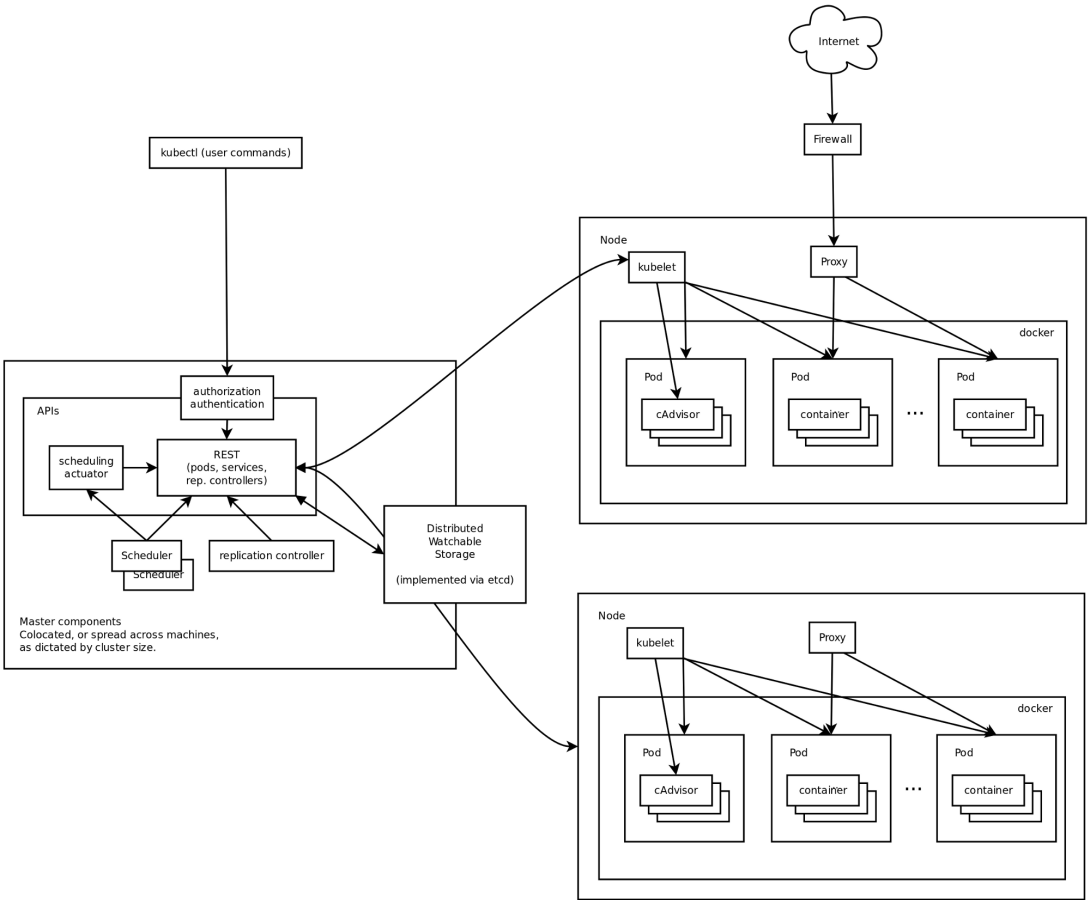
Tabulka 4.3: Funkční porovnání kontejnerových orchestrátorů, zdroj: vlastní tvorba

Před samotným vysvětlením role Kubernetes v Edge Computingu je nutné vymezit jeho základní terminologii:

- Kontejner - kontejnery jsou zabalené jednotky nesoucí spustitelný aplikační kód s jeho závislostmi, konfigurací a knihovnami.
- Pod - pody reprezentují jednotky nasazení v Kubernetes ekosystému, které obsahují jeden nebo více kontejnerů na stejném hostu. Tyto skupiny kontejnerů mohou běžet společně a sdílet zdroje.
- Node - reprezentuje jeden server v clusteru běžící Kubernetes aplikace. Node může být fyzický nebo virtuální server.
- Cluster - několik nodů spojených dohromady formující společnou výpočetní skupinu se zdroji, které jsou sdíleny běžícími aplikacemi.

- Perzistentní volume - jelikož mohou být kontejnery dynamicky přesouvány mezi nody, ztrácí tak datovou perzistenci. Z toho důvodu řeší perzistentní volume tento problém pro aplikace, které potřebují ukládat svá data.

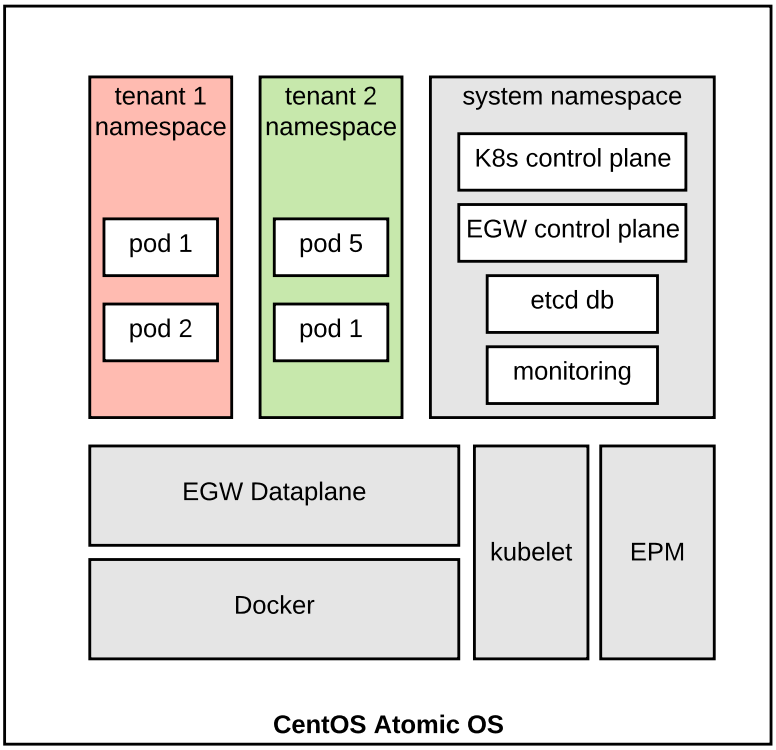
Obrázek 4.15 zachycuje architekturu Kubernetes.



Obrázek 4.15: Kubernetes architektura, převzato z (Sayfan, 2018)

4.4.3 Architektura multifunkčního edge zařízení

Obecná navržená architektura multifunkčního edge zařízení v rámci CEC platformy je zobrazena na obrázku 4.16. Skládá se z několika vrstev, jako jsou operační systém, kontejnerový systém, síťový management, úložiště pro persistentní data, kontejnerová orchestrace a další aplikace nutné pro edge správu jako monitoring, sběr logů apod. Vysvětleme nyní jednotlivé části.



Obrázek 4.16: Obecný pohled na navrženou architekturu multifunkčního edge node, zdroj: vlastní tvorba

Jako operační systém lze použít téměř libovolnou linuxovou distribuci. Nicméně navržené řešení bylo vyvíjeno a testováno na platformě CentOS Atomic, a to z důvodu způsobu správy aktualizací. Klasické linuxové distribuce používají systém repozitářů s balíky jednotlivých aplikací nebo knihoven pro OS. V praxi tak bývá obtížné zajistit konzistentní aktualizace napříč celou infrastrukturou, jelikož různé systémy mohou mít nainstalované různé balíky. CentOS Atomic byl vyvinut jako kontejnerový OS, jehož cílem je minimalizovat množství softwaru a současně znemožnit instalaci pomocí balíků, a to zákazem zápisu do systémových cest. Operační systém je tak rozdělen na dva svazky A a B, ze kterých lze nastartovat. Aktualizace se tedy provádí zapsáním vždy na druhý neaktivní svazek, který je nahrán až po restartu OS. Tento přístup umožňuje provádět okamžité obnovení předchozích verzí a zajišťuje konzistenci ve verzích a napříč všemi servery v našem případě edge site. Jedinou nevýhodou je, že i při aktualizaci menší záplaty do systému je nutné provést restart celého OS (*Project Atomic*, 2019). Nicméně snaha je omezit OS komponenty na minimum a všechny další nástroje a služby doručit jako kontejner tak, aby byly všechny aktualizace konzistentní.

Kontejnerová virtualizace a orchestrace je založena na již zmíněných komponentách Docker a Kubernetes. Docker je součástí CentOS Atomic OS, takže není nutné provádět žádnou speciální úpravu. Kubernetes je nutné nainstalovat a nakonfigurovat při spouštění nového edge zařízení, jelikož je nutné vygenerovat certifikáty a specifické konfigurace daného prostředí, jako jsou například IP adresy, DNS servery apod. Z toho důvodu bylo nutné napsat speciálního démona EPM neboli edge platform manager, který běží jako systemd proces uvnitř každého node a poslouchá na API. Jeho rolí je provádět následující akce:

- Správa hosta - jedná se o úkony jako nastavovat a provádět upgrade OS, spravovat uživatelské přístupy do OS nebo nastavovat velikost a počet Hugepages pro EGW DPDK dataplane.
- Kubernetes správa - přijetí certifikátů, instalace, upgrade a konfigurace etcd databáze, instalace jednoho nebo více nodového Kubernetes clusteru.
- Správa kontrolních aplikací - většina řídicích služeb pro EGW, cluster monitoring, sběr logů nebo interní DNS server běžící také uvnitř Kubernetes v systémovém namespace (obrázek 4.16). EPM doručuje jejich aktualizace a změny konfigurací přes lokální Kubernetes API.
- Konfigurace služeb přes API - EGW přijímá konfigurace pro volbu vzdálené site IPSec připojení přes své API. EPM proto provádí tyto lokální konfigurace.
- Reset do továrního nastavení - tato funkce umožňuje obnovit zařízení do původního nastavení. Cílem je umožnit vzdálenou správu a v případě kritických poškození zařízení zresetovat bez nutnosti fyzického přístupu.

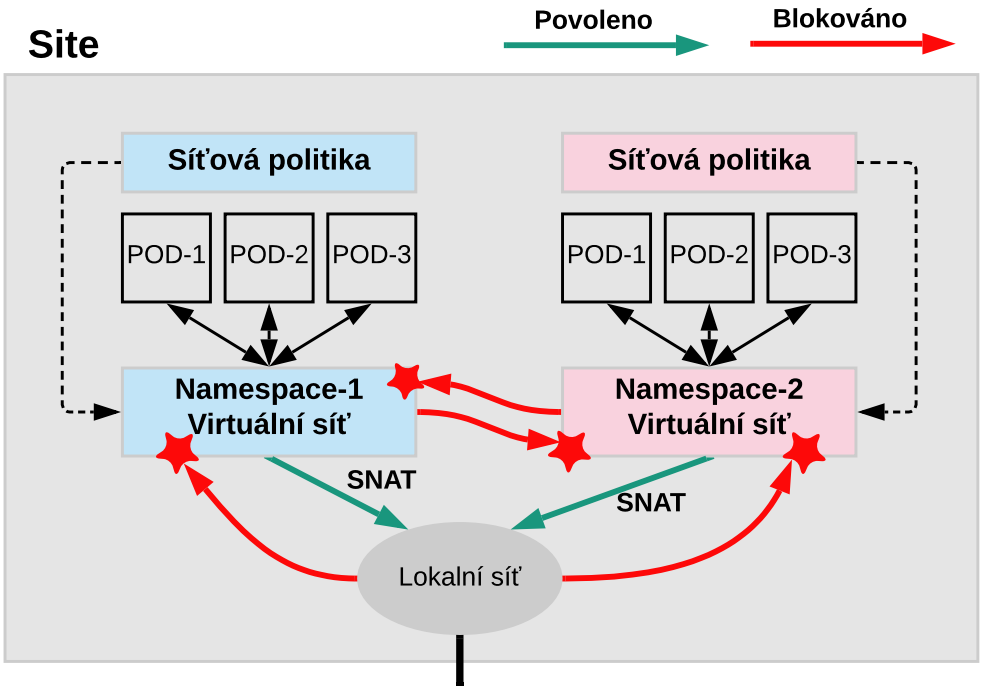
Jak je tedy vidět na obrázku 4.16, mimo Kubernetes běží pouze Docker engine, kubelet a EPM. Zbytek řídicích služeb včetně Kubernetes API, controller-manager nebo EGW control plane běží uvnitř systémového namespace podobně jako aplikace jednotlivých tenantů. Jediným rozdílem je typ síťování. System namespace kontejnery běží v módu host networking, který připojuje rozhraní kontejnerů přímo do jejich hostitele. Díky tomu se tyto kontejnery tváří, jako by byly procesy přímo uvnitř OS. Tento přístup značně zjednodušuje správu řídicích služeb. Ostatní komponenty už tvoří aplikace jednotlivých tenantů separovaných pomocí Kubernetes namespaces. Jejich separace na síťové úrovni je provedena na úrovni EGW, která je popsána v kapitole 4.4.3.

Jako úložiště pro multifunkční edge je dočasně navržen lokální hostPath systém, který mapuje části hostova souborového systému přímo do jednotlivých kontejnerů. Problém nastává při vícenodové instalaci, kde může kontejner být dynamicky přesunut na jiný node a tím přichází o svá perzistentní data. Abychom tento problém vyřešili, provedli jsme výzkum (Mercl & Pavlik, 2019b) pro výběr typu úložiště, kde

jsme podrobně prověřili dostupná řešení Ceph, PortWorx, GlusterFS a OpenEBS. Ačkoli nejlepších výsledků dosahovalo komerční řešení PortWorx, rozhodli jsme kvůli otevřenosti CEC platformy navrhnout druhé nejlepší podle výsledků, a to softwarové úložiště Ceph. Nicméně jsme dospěli k závěru, že pro malá edge zařízení není tento přístup vhodný. Otázka úložiště pro vícenodové edge site s malou kapacitou zdrojů, tak zůstává stále otevřená a budeme se jí zabývat v budoucím rozvoji.

Síťová komunikace Podů

Jelikož je navržená EGW vlastním síťovým řešení CEC platformy, bylo nutné napsat integraci pro Kubernetes. Ta je provedena skrz implementaci CNI pluginu. CNI je specifikace a sada knihoven pro psaní síťových pluginů na konfigurace síťových rozhraní u linuxových kontejnerů (Sayfan, 2018). Diagram 4.17 zobrazuje komunikaci mezi Pody uvnitř Kubernetes. Při vytvoření Podu dojde také k vytvoření síťového rozhraní v rámci namespace. Kubernetes namespace tvoří jednu virtuální síť v EGW. Každá síť je kompletně izolována pomocí enkapsulace MPLSoverUDP. Každý namespace tak uvnitř používá stejný IPv4 rozsah. Obrázek 4.17 ukazuje povolenou a blokovanou komunikaci mezi Kubernetes pody.



Obrázek 4.17: Síťová komunikace mezi pody v rámci edge site, zdroj: vlastní tvorba

Tato situace vede k tomu, že:

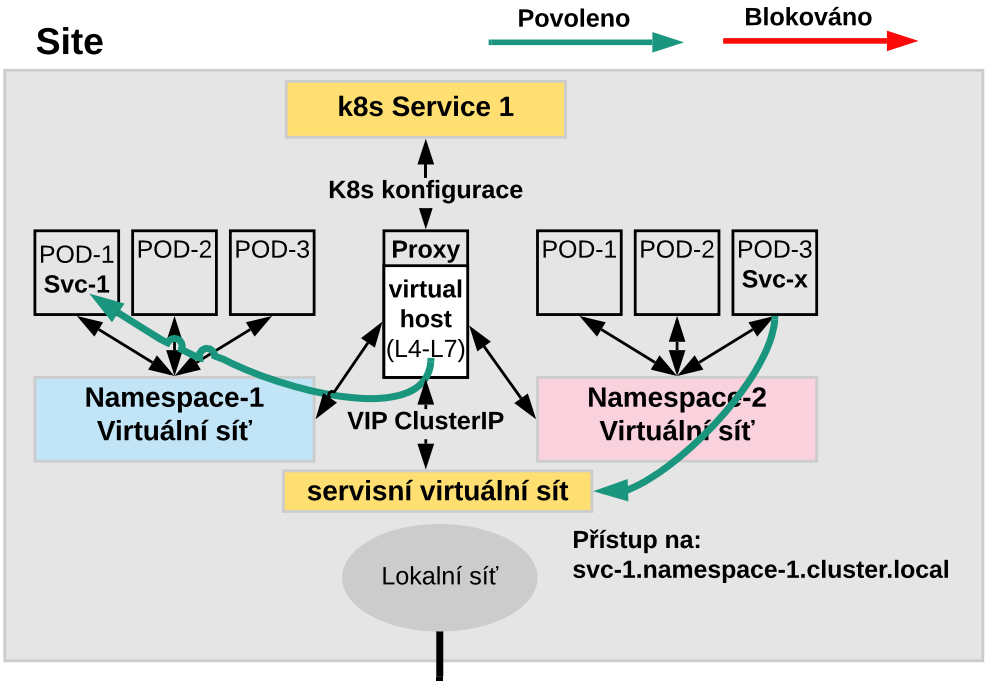
- Aplikační pody v rámci jednoho namespace mohou komunikovat mezi sebou,
- Aplikační pody nemohou komunikovat mezi namespace,
- Aplikační pody mohou komunikovat s lokální fyzickou sítí pomocí SNAT.

Síťové politiky mohou být konfigurovány v rámci namespace a mohou tak segmentovat síťový provoz i uvnitř. Pokud tedy chce komunikovat aplikace v namespace-1 s jinou aplikací v namespace-2, jedinou možností je vytvořit Kubernetes service objekt.

Síťová komunikace Service

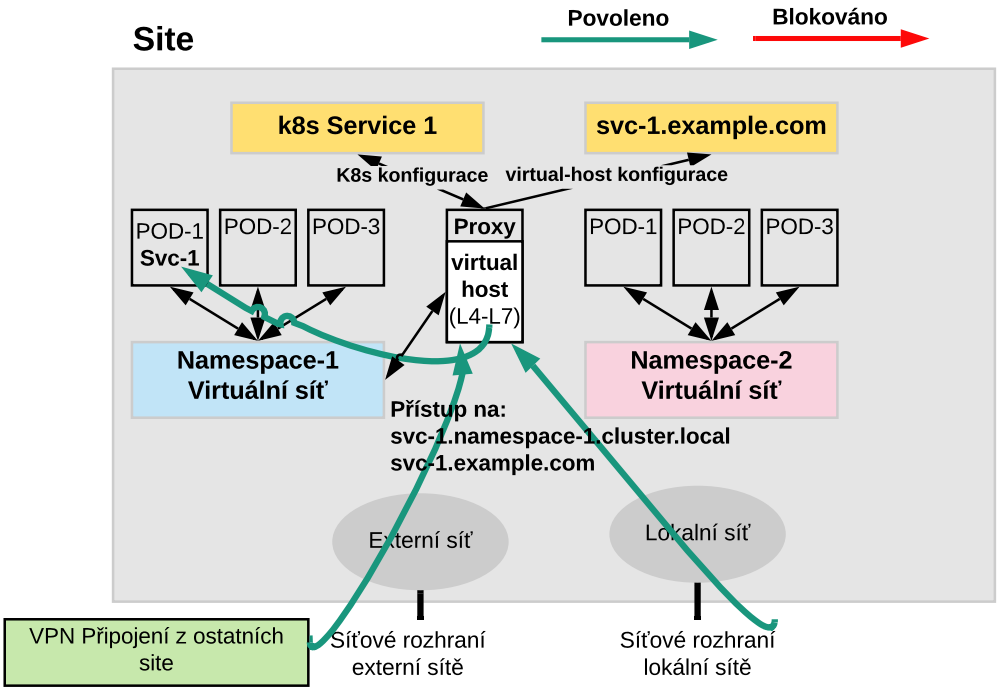
Service jsou mechanismus pro vytvoření proxy na komunikaci mezi aplikacemi. V souladu s výchozí implementací v Kubernetes jsou Services plně přístupné v rámci jednoho clusteru, v našem případě edge site. Diagram 4.18 zachycuje komunikaci Podu z namespace-2 na Pod v namespace-1 skrz Kubernetes Service, která je reprezentována

virtuální IP adresou a virtual-host v EGW Envoy proxy. Podobně jako u Pod síťové komunikace je možné omezit výchozí komunikaci skrz síťové politiky.



Obrázek 4.18: Síťová komunikace mezi Kubernetes službami v rámci edge site, zdroj: vlastní tvorba

Přístup z vnějšku k site může být jednak z lokální sítě nebo skrz IPSec VPN. V obou případech se vytváří objekt virtual-host, který skrz propagační politiku říká, odkud má být služba dostupná. Diagram 4.19 tak zachycuje oba typy přístupů z vnějšku na Kubernetes služby.



Obrázek 4.19: Síťová komunikace mezi Kubernetes službami v rámci edge site, zdroj: vlastní tvorba

Při návrhu implementace CNI plugingu a oddělených jednotlivých Kubernetes namespaces jsme se inspirovali existující implementací v SDN řešení Tungsten Fabric, odkud jsme přijali vRouter data plane. Výhodou je především kompletní izolace jednotlivých namespaces díky použité encapsulaci MPLSoverUDP, jelikož ostatní pluginy pro Kubernetes jako například Calico vyžadují pro izolaci explicitní konfiguraci skrz síťové politiky[8]. Jakým způsobem jsou aplikace distribuovány napříč edge site nebo jak je jejich komunikace řízena, je popsáno v kapitole 4.5.

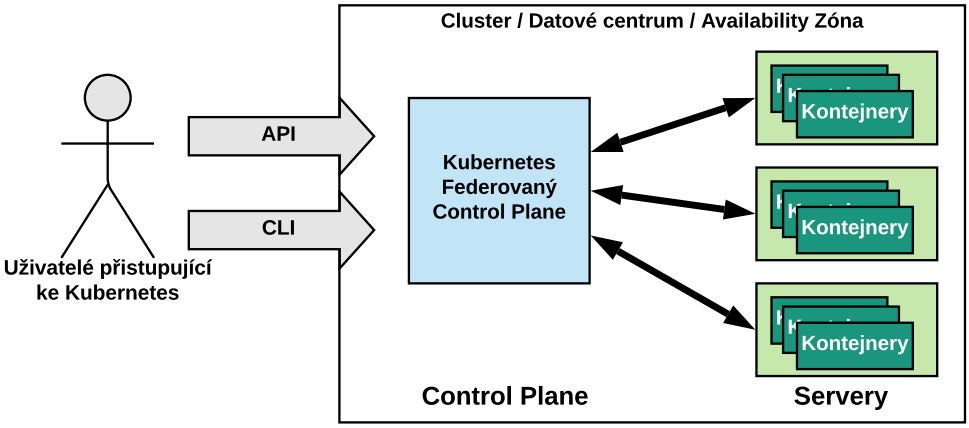
4.5 Standardní API pro orchestraci

Nejprve je nutné se podívat na to, co je standardním API pro orchestraci ve světě cloud computingu. Jak bylo zmíněno v kapitole 4.4 Kubernetes a jeho API je standard pro orchestraci kontejnerů a během posledních let byla služba správy Kubernetes poskytnuta největšími veřejnými cloudy, jako jsou Amazon, Google a Microsoft Azure. V oblasti privátních datových center a cloudů zase společnost VMware vydala svoji verzi

[8]Ukázka Calico síťové politiky https://docs.projectcalico.org/v3.9/reference/resources/networkpolicy

Kubernetes[9]. V těchto prostředích společnosti většinou spouští jednotky velkých, nebo malých Kubernetes clusterů pro svoje aplikace. K nasazení těchto aplikací pak používají různá řešení typu Helm, Spinnaker nebo Jenkins. V případě edge computingu se však jedná o potenciální stovky až tisíce site, z nichž každá je v případě CEC platformy separátní Kubernetes cluster. Znamená to tedy, že tyto nástroje by musely spravovat přístupové údaje a mít neustálý přímý síťový přístup na všechny z nich. I když CEC platforma vytváří IPSec site-to-site VPN připojení mezi všemi lokalitami, tak tento přístup není z hlediska stability a bezpečnosti vhodný.

Z těchto důvodu bylo nutné se podívat na způsoby distribuce Kubernetes objektů pro multi cluster řešení. Existují dva způsoby, jakými toho lze docílit. Prvním z nich je oficiální způsob správy více clusterů **Kubernetes federace**. Clusterová federace je koncepčně velmi jednoduchá, spojuje se více clusterů Kubernetes a považují se za jeden logický celek. Existuje federační control plane, který poskytuje klientům jednotný pohled na celý systém. Obrázek 4.20 zachycuje architekturu clusterové federace, která se skládá z federačního API serveru a controlleru, jež spolu spolupracují a předávají požadavky dílčím clusterům v daných lokalitách.



Obrázek 4.20: Architektura Kubernetes federace, převzato (Sayfan, 2018)

Jeho druhá funkce kromě distribuce a řízení stavu objektů v dílčích clusterech je objevování napříč těmito clustery. Každý Kubernetes cluster při vytvoření kontejnerů a služeb alokuje veřejný přístupový bod, který je následně dynamicky přidán do externího DNS. V případě výpadku clusteru, serveru nebo kontejnerů je na něm opět dy-

[9]VMware Tanzu https://blogs.vmware.com/cloudnative/2019/08/26/vmware-completes-approach-to-modern-applications/

namicky tento záznam upraven tak, aby klient přistupující na federovanou službu byl vždy obsloužen.

Na první pohled se zdá, že tento koncept je přesně to, co bylo nutné i v rámci edge computing platformy. Distribuovat objekty a získávat informace o jejich stavu zpět. Nicméně tento přístup byl v době našeho výzkumu relativně nový a ne příliš stabilní. Zároveň hlavním způsobem použití federace bylo geografické vyvažování síťového provozu pro distribuované aplikace. Typicky se jednalo o webovou aplikaci, kterou je potřeba provozovat ve více regionech jako například Asii, Evropě a USA. Vytvořilo se federační API a následně se rozdistribuoval kontejner s aplikací do všech lokalit. Potom dynamické objevování získalo veřejnou IP adresu z každého Kubernetes clusteru a vytvořilo DNS záznam v externím DNS, jenž prováděl DNS vyvažování na základě lokality (Sayfan, 2018). V případě edge computingu je však tento směr opačný, jelikož klient chce přistupovat na nejbližší edge site zařízení s běžící službou, a nelze tedy spravovat centrální veřejné DNS s tisíci záznamy. V mnoha případech ani neexistuje veřejná IP adresa v edge site, a tudíž není možné toto objevování skrz veřejné DNS vůbec použít. V neposlední řadě měl federovaný control plane při našem testování problémy s řešením vysoké dostupnosti, a tudíž ani jeho použití pro část distribuce objektů nedávalo smysl.

Druhá dostupná možnost bylo použití **Continuous Delivery nástroje** s klasickým přímým přístupem na Kubernetes API. Jedním z nejznámějších nástrojů této skupiny je open source řešení Spinnaker vyvinuté společností Netflix (*Cloud Native Continuous Deliver*, 2019). Netflix využívá tento koncept při správě svých serverových farem v Amazon cloudu, kde obhospodařuje několik regionů. Při detailním zkoumání jsme zjistili, že je nutné ukládat všechny přístupy ke všem clusterům na centrálním místě a mít přímý přístup na API. To v praxi znamená, že musí být vždy přímé síťové spojení mezi Spinnaker a tisíci lokalitami. Separace uživatelů a tenantů by bylo nutné provádět na úrovni Spinnakeru, jelikož musí mít přístup s administrativními právy do všech Kubernetes clusterů. Komplexita správy clusterů se tak přesouvá do jednoho centrálního místa, které musí být schopné škálovat, a stává se tak potenciálním problémem v případě výpadku. Bylo nutné tedy vytvořit seznam požadavků pro edge computing.

Námi definované požadavky na orchestrační API a správu edge site se dají shrnout do několika bodů:

- Poskytnout API založené na Kubernetes tak, aby změny nutné pro přechod ze standardního cloudu na CEC platformu byly minimální,
- Správa aplikací stovek až tisíců site pomocí centrálního API,
- Různé verze aplikací mohou být nasazeny v různých site, a proto je nutné podporovat možnost testovacích site apod.,

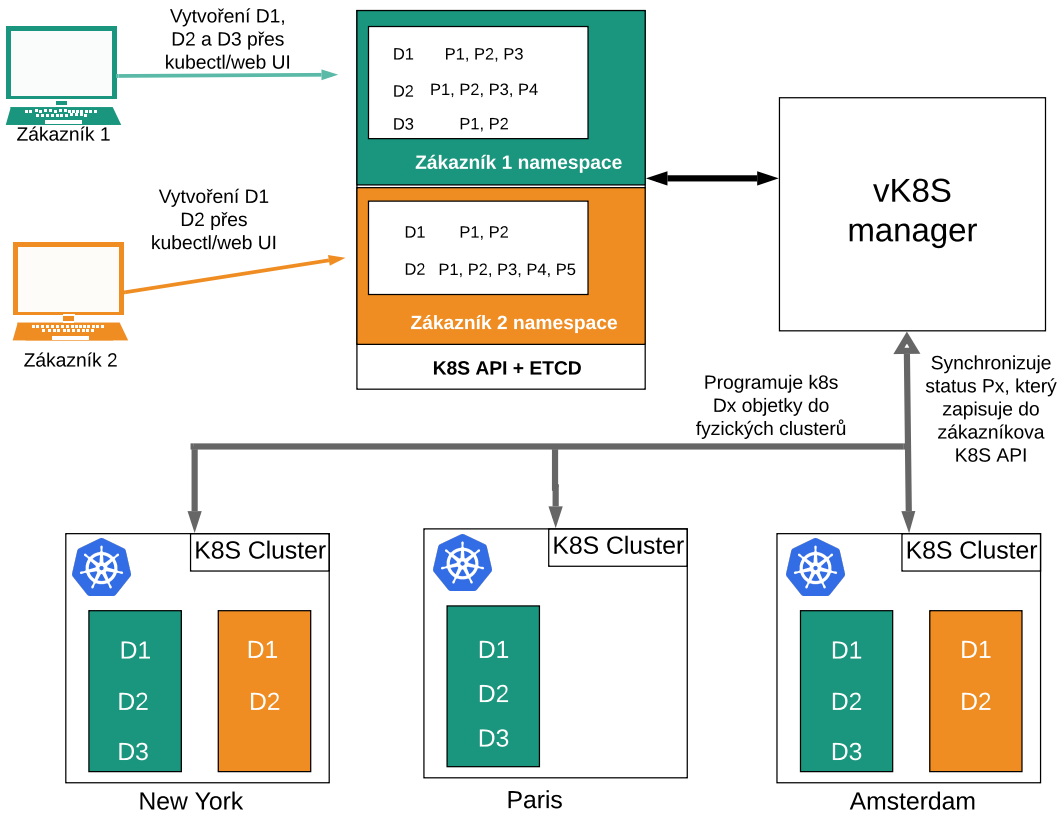
- Distribuovaná architektura nezávislá na jedné centrální komponentě, jako je tomu v případě uvedeného Spinnaker nástroje.

Vzhledem k výše uvedeným existujícím řešením a jejich limitům jsme vytvořili a implementovali vlastní koncept virtuálního Kubernetes (vK8s), který je popsán v kapitole 4.5.1.

4.5.1 Návrh virtuálního Kubernetes

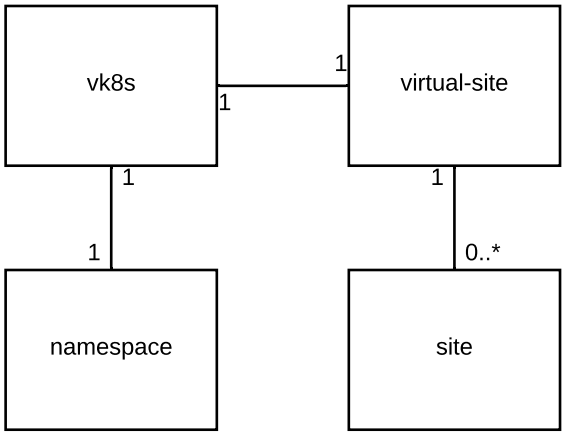
Koncept virtual Kubernetes neboli vK8s je Kubernetes kompatibilní API pro centralizovanou orchestraci aplikací napříč flotilou site (CE i RE). Toto API je pouze kompatibilní, protože nepodporuje všechny typy objektů Kubernetes. Pro vybrané objekty však implementuje plnou podporu. vK8s architektura se skládá z několika částí, které distribuují a spravují objekty ve fyzických lokalitách.

Koncový uživatel pracuje s dedikovaným Kubernetes API serverem, který je spuštěn při vytvoření nového vk8s objektu v controlleru. Díky tomu se rozhraní a celková práce neliší od klasického skutečného Kubernetes API. Hlavním rozdílem je, že Kubernetes controller-manager a scheduler je nahrazen jedním démonem nazvaným vk8s manager, který emuluje jejich akce. vK8s manager poslouchá na Kubernetes API a při vytvoření nového deploymentu aplikace ho následně distribuuje do skutečných fyzických Kubernetes clusterů běžících v edge site, kde se spustí jednotlivé aplikační pody. Poté začne poslouchat na API těchto skutečných clusterů, a status nasazených podů aplikací zapisuje zpět do dedikovaného Kubernetes API koncového uživatele. Obrázek 4.21 zachycuje architekturu vk8s, kde dva koncoví uživatelé vytvářejí své deploymenty Dx ve vlastních Kubernetes API serverech, avšak výsledkem jsou běžící aplikační pody Px ve stejných fyzických clusterech oddělené pomocí namespace.



Obrázek 4.21: Architektura vk8s manager, zdroj: vlastní tvorba

Volba a rozhodování o distribuci objektů do konkrétních site se provádí pomocí objektů virtuálních site, které tvoří abstrakci nad konkrétními fyzickými clustery. Při stovkách či tisících není dostatečně flexibilní vybírat konkrétní site a z toho důvodu uživatel vytváří referenci mezi vK8s a virtual-site objektem. UML diagram 4.22 nastiňuje vazby mezi třídami, kde namespace může mít právě jednu vk8s třídu, která má vazbu na virtual-site. Virtual-site pomocí funkce volby nálepek má vazbu na žádnou, nebo více konkrétních site.



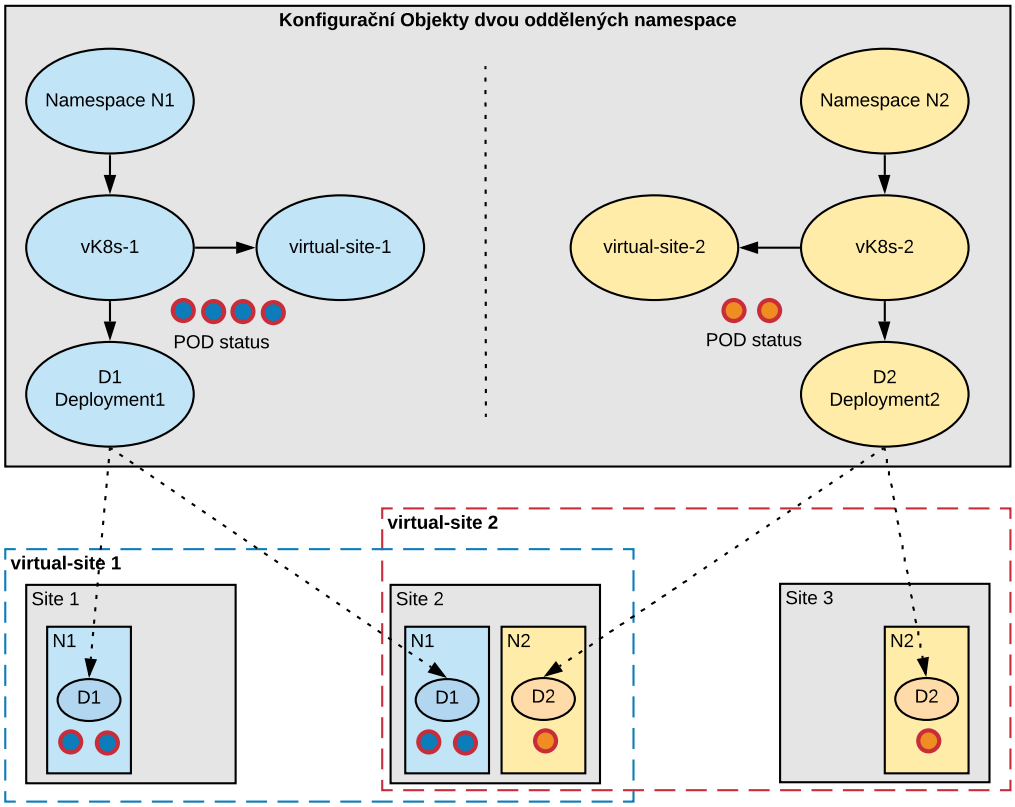
Obrázek 4.22: Diagram tříd pro vK8s manager, zdroj: vlastní tvorba

Navržené vK8s podporuje následující typy Kubernetes objektů:

- Deployment - typ řídicího objektu pro bezstavové aplikace, jako jsou webové servery apod.
- Statefulset - typ řídicího objektu pro stavové aplikace, jako jsou databáze apod.
- Job - je typ objektu pro jednorázové aplikace, jako jsou běh skriptu, defragmentace databáze apod.
- Service - typ objektu pro komunikaci aplikací mezi sebou.
- ConfigMap - aplikace musejí být konfigurovány v závislosti na typu prostředí, a proto tento typ objektu umožňuje připojit ke kontejneru libovolnou konfiguraci skrze namapovaný soubor nebo enviromentální proměnnou.
- Secret - některé aplikace vyžadují přístup k senzitivním informacím, jako jsou certifikáty, tokeny atd. Namísto vkládání těchto informací do kódu aplikace nebo viditelné konfigurace, je možné využít typ objektu Secret.
- PVC - stavové aplikace jako je databáze potřebují ukládat svá data perzistentně. PVC umožňuje vytvořit tento perzistentní svazek.

Mezi nepodporované objekty patří přímé vytváření podů, jelikož ty jsou dynamicky vytvářeny skrze deployment objekty. Dále nejsou podporovány CRD objekty, které slouží k vlastnímu rozšíření Kubernetes API, což by bylo velmi složité podporovat pomocí vK8s API.

Obrázek 4.23 zobrazuje logický pohled na vk8s deployment, kde existují dva namespace N1 a N2. Každý má svůj vk8s objekt vK8s-1 a vK8s-2. Tyto objekty mají referenci na své virtual-site, které podle své volby nálepky vybere skutečné fyzické Kubernetes clustery v site 1, site 2 a site 3. V tomto případě virtual-site-1 adresuje site-1 a site-2. Virtual-site-2 potom adresuje site-2 a site-3. Uvnitř těchto site jsou dynamicky vytvářeny identické namespace na základě toho, do jaké virtual-site patří. V momentu vytvoření Kubernetes deployment objektu D1 nebo D2 dochází k replikaci těchto deployment objektů do patřičných site. Kubernetes podle počtu replik nastavených v deploymentu vytváří pody s aplikačními kontejnery (znázorněno pomocí barevných teček), které jsou následně propsány zpět do vK8s jako jejich status. Tento příklad má nastavené dvě repliky pro D1 a jednu repliku pro D2.



Obrázek 4.23: Logický pohled na nasazení do vK8s, zdroj: vlastní tvorba

Jeden z počátečních prototypů vK8s managera jsme vytvořili a publikovali v rámci diplomové práce (Cach, 2019) vedené autorem této práce. Celé řešení bylo potom vyvinuto do první fáze autorem práce společně s dalšími spolupracovníky v rámci jeho pracovního působení.

4.5.2 Příklad nasazení aplikace pomocí vK8s

Podívejme se na jednoduchý příklad vytvoření aplikace productpage se dvěma replikami v Paříži, New Yorku a Amsterdamu z pohledu koncového uživatele. Nejprve si ukažme definici standardního Kubernetes deploymentu 4.1. Počet replik je nastaven na dvě, což ve standardním Kubernetes nastartuje stejný počet podů, v tomto případě kontejnerů.

Kód 4.1: Ukázka výpisu kubectl s výpisem jména site a node

```
# cat productpage.yml
metadata:
  name: productpage-v1
spec:
  replicas: 2 # Počet replik je nastaven na dvě
  selector:
    matchLabels:
      app: productpage
      version: v1
  template:
    metadata:
      labels:
        app: productpage
        version: v1
    spec:
      containers:
        - name: productpage
          image:
             istio/examples-bookinfo-productpage-v1:1.8.0
```

Při aplikaci 4.1 je patrné, že běžících replik je 6. Je to z důvodu distribuce definice aplikace do 3 lokalit, kde v každé vzniknou právě dvě repliky.

Kód 4.2: Ukázka výpisu kubectl deployment s počtem běžících podů

```
# kubectl get deployment
NAME        READY  UP-TO-DATE AVAILABLE
productpage 6/6    6          6
```

Celkový počet včetně zobrazení site je vidět na ukázce 4.3. vK8s manager, v tomto případě přepisuje jméno Kubernetes node a přidává k němu prefix se jménem edge.

Kód 4.3: Ukázka výpisu kubectl s výpisem jména site a node

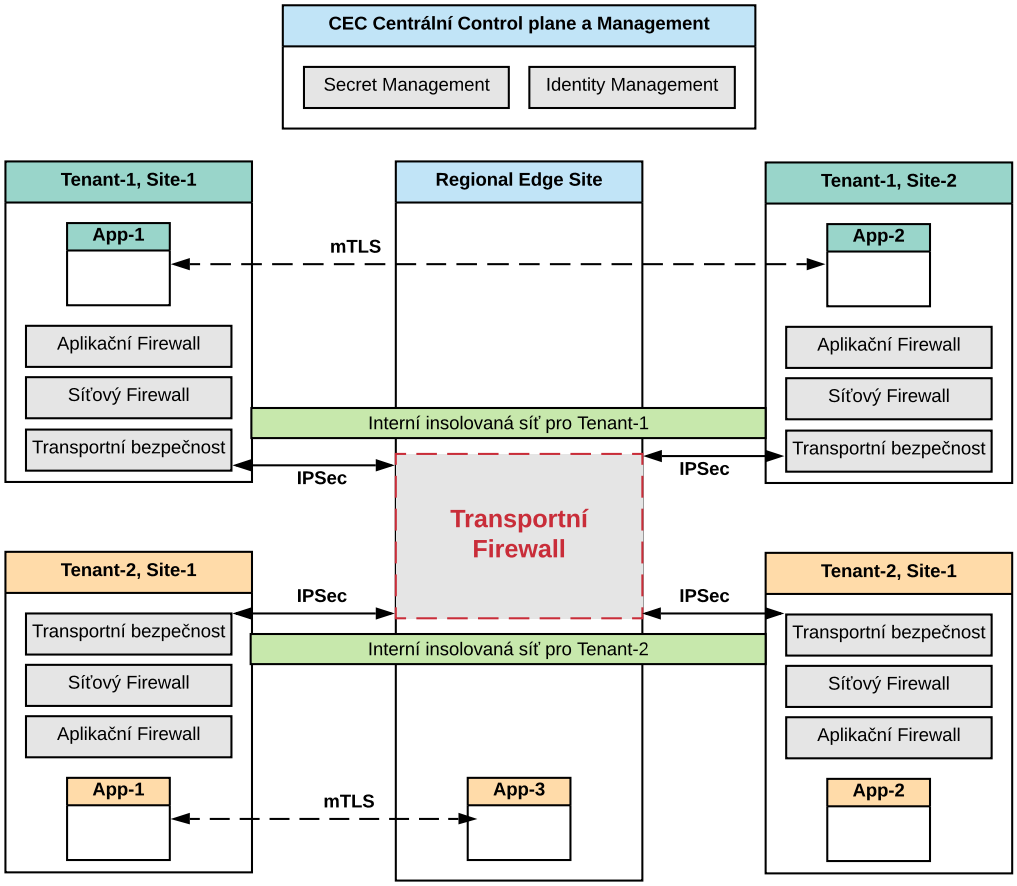
```
# kubectl get po -o wide
NAME                    READY  STATUS   NODE
productpage-cbb-r988m 1/1   Running  paris-node-0
productpage-cbb-pjtln 1/1   Running  paris-node-2
productpage-788-q4r87 1/1   Running  nyc-node-1
productpage-788-q4r87 1/1   Running  nyc-node-0
productpage-788-q4r87 1/1   Running  amsterdam-node-1
productpage-788-q4r87 1/1   Running  amsterdam-node-0
```

Tento příklad dokazuje, že lze použít standardní Kubernetes klient pro příkazovou řádku a deployment objekt bez jakékoli modifikace. Uživatelská zkušenost je tak naprosto identická při správě jednoho nebo tisíce clusterů.

4.6 Bezpečnost v CEC

Bezpečnost je rozsáhlá doména, která by mohla být řešena v rámci samostatné disertační práce a z toho důvodu se tato práce zabývá jen okrajovým popisem definovaných bezpečnostních vrstev edge platformy a neřeší všechny její aspekty. Dále nezachází ani do kryptografických a implementačních detailů. Aby bylo zajištěno bezpečné prostředí pro aplikace, data a konektivitu, je třeba zapojit mnoho aspektů zabezpečení dohromady, aby vznikl těžko prolomitelný systém podporovaný kryptograficky zabezpečenými stavebními bloky. Náš přístup k bezpečnosti spočívá v definování těchto stavebních bloků, které byly vybrány s ohledem na osvědčené postupy (Scott, 2018), (Keeriyattil, 2019), (Samaniego & Deters, 2018) v komerčním a akademickém prostředí tak, aby bylo možné aplikovat již zmíněnou koncepci Zero Trust. Náš návrh se zaměřuje především na zabezpečení aplikací, data a konektivity. Pohled zabezpečení a správa přístupu koncových uživatelů jako například IDM není součástí této práce.

Na základě zkoumání bezpečnostních vrstev v cloud computingu (Tene, 2014) a nejnovějších trendů zabezpečení aplikací v prostředích Service Mesh (Chandramouli & Butcher, 2020) jsme vytvořili obecný návrh bezpečnostních stavebních bloků CEC architektury, který je vidět na obrázku 4.24. Částečně jsme vycházeli i z již vlastního výzkumu zabývajícího se SIEM řešeními v prostředí cloud computingu (Pavlik, Komarek, & Sobeslav, 2014).



Obrázek 4.24: Obecný pohled na bezpečnostní architekturu CEC platformy, zdroj: vlastní tvorba

1. Identity management - musí poskytnout společnou identitu všem softwarovým komponentám, které běží v systému. Jde jak o aplikace uživatelů, tak řídicí komponenty platformy. Jednotná identita je zásadní pro práci napříč různými lokalitami a prostředími.

2. Autentizace a autorizace - identita tvoří základ pro autentizaci používanou pro bezpečnou komunikaci mezi jednotlivými edge lokalitami. Identitu lze také používat k nastavení mTLS spojení mezi aplikacemi, jež je podrobněji popsána v následujících kapitolách.

3. Secrets - existuje mnoho typů secret (například TLS certifikáty, hesla, tokeny atd.), které musí být bezpečně uloženy jak pro řídicí, tak pro uživatelské aplikace. CEC platforma musí obsahovat mechanismus, díky kterému je možné tyto secrety bezpečně uložit, aby nedošlo k jejich úniku.

4. Transportní bezpečnost - veškerá komunikace aplikací uvnitř platformy musí probíhat skrz zabezpečený kanál SSL nebo IPSec VPN, tak aby nebylo možné odposlechnout nebo zachytit komunikaci mezi lokalitami.

5. Síťová bezpečnost - CE stejně tak RE site nebo jakákoli jiná infrastruktura vyžaduje kombinaci síťových politik, aby bylo možné izolovat komunikaci napříč aplikacemi a podsítěmi tak, aby nedocházelo k různým útokům.

6. Aplikační bezpečnost - poslední aplikační vrstvou je ochrana a firewall na aplikační úrovni, jako jsou WAFS, DDoS, nebo detekce anomálií síťového provozu. Ta se obvykle provádí skrz servisní politiky, kde je možné izolovat konkrétní API nebo aplikaci.

Ne všechny tyto části jsou plně implementovány či kompletně navrženy. Bezpečnost je především kontinuální proces, který nikdy nekončí a je nutné ho neustále rozvíjet. Výše uvedené stavební bloky architektury jsou rozebrány v následujících podkapitolách včetně jejich stavu návrhu a implementace.

4.6.1 Identity management

Bezpečné nastartování identity je jednou z největších výzev a jedním z prvních kroků, které by měly být při návrhu bezpečné infrastruktury v souladu s konceptem Zero Trust podniknuty (Scott, 2018). Vydávání kryptografické ověřitelné identity doručené bezpečným způsobem, je problematické především z důvodu jejího získání (tzv. bootstraping identity) a důvěryhodnosti (root-of-trust).

Bootstraping identity se dá připodobnit ke skutečnému životu. Při narození jakékoli osoby je její totožnost prokázána rodným listem. To logicky bootstrapuje identitu osoby a pomocí tohoto certifikátu může osoba požadovat více identifikačních dokumentů, jako jsou cestovní pas, řidičský průkaz atd. Podobně v počítačovém světě je třeba pro každé spuštění aplikace (nebo mikroslužby) zavést identitu. Stanovení identity je jedním z prvních kroků, které musí jakýkoli spuštěný kód provést, aby se mohl integrovat s jinými službami. Kromě toho, že stejný kód aplikace může být spuštěn vícekrát (vývojář na notebooku, testovací prostředí nebo produkce) a v různých prostředích, je také nutné, aby každé z těchto spuštění vytvořilo samostatnou aplikační identitu.

Zatímco vydávání bootstrap identity se může zdát jako přímočarý proces, problém je v tom, kdo dostane tuto identitu. Například v reálném světě začíná zajišťování totožnosti v nemocnici, která potvrzuje, že se dítě narodilo k určitému datu a času. Předpokládá se, že nemocnice vytváří tento záznam o narození s řádnými procesem a kontrolami. Výsledkem je, že dokument o narození může být použit jako důvěryhodný zdroj

nebo kořen důvěry **root-of-trust**. Podobně když je aplikace spuštěna pomocí lidského nebo automatizovaného kódu, musí existovat kořen důvěry, který musí prokazatelně potvrdit identitu spuštěné aplikace. Toto potvrzení pak může být použito ke generování dalších dokladů totožnosti pro aplikaci.

Například, při spuštění VM ve veřejném cloudovém prostředí AWS mu je současně poskytnuta bootstrap identita a AWS metadatová služba zde vystupuje jako root-of-trust. Dokument s identitou podepsaný AWS kryptografickým klíčem pak vypadá jako ukázka kódu 4.4, I když instanceId může označovat jedinečnou identitu spuštěné aplikační instance, je třeba ji zřetězit s doménovým jménem jako např. mujserver.example.com, který ostatní aplikace použijí ke komunikaci s touto konkrétní instancí. Výsledkem je, že ani tato identita od AWS není dostatečná, ale může být použita k vydání další identity, kterou použijí aplikace na komunikaci mezi sebou (Scott, 2018).

Kód 4.4: Ukázka AWS identity dokumentu

```
{
    "devpayProductCodes" : null,
    "marketplaceProductCodes" : [
       "1abc2defghijklm3nopqrs4tu" ],
    "availabilityZone" : "us-west-2c",
    "privateIp" : "10.158.111.4",
    "version" : "2017-09-30",
    "instanceId" : "i-1234567876abcdef0",
    "billingProducts" : null,
    "instanceType" : "t3.small",
    "accountId" : "123454389012",
    "imageId" : "ami-5fbui835",
    "pendingTime" : "2018-11-19T16:32:12Z",
    "architecture" : "x86_64",
    "kernelId" : null,
    "ramdiskId" : null,
    "region" : "us-west-1"
}
```

CEC platforma a její komponenty spravují v tomto případě kryptografické materiály, které fungují jako root-of-trust pro mnoho dalších bezpečnostních vrstev. Všechny ostatní komponenty běžící v rámci CEC platformy získávají kryptografickou identitu od služby nazvané Idendity Authority, což je jedna z klíčových bezpečnostních složek.

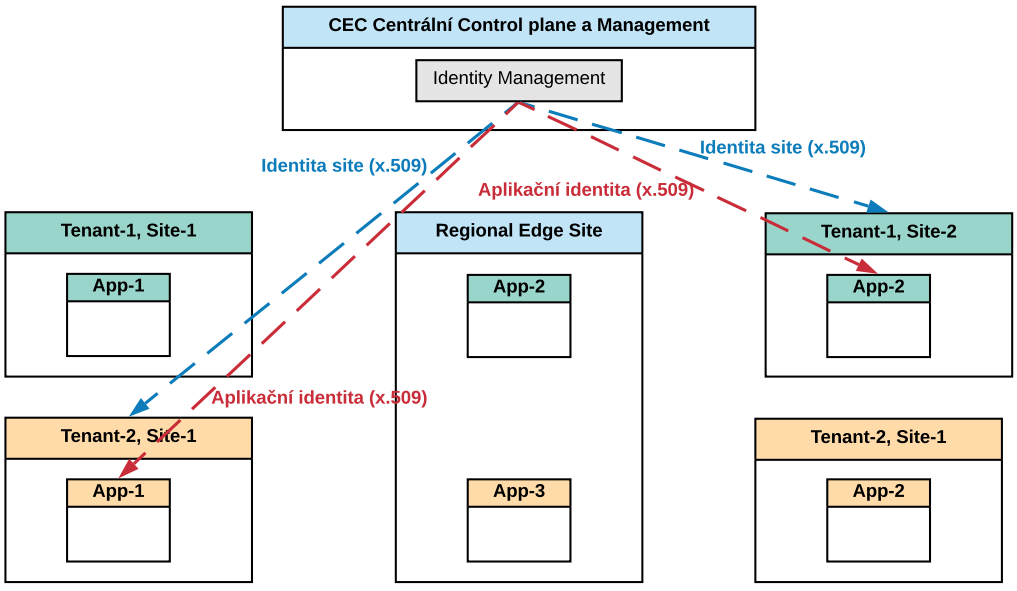
Tato kryptografická identita každé komponenty je nezbytná pro řízení jejího přístupu a současně také pro bezpečnou komunikaci.

Druhým blokem v bezpečnostním řetězci jsou komponenty, které jsou zodpovědné za registraci a správu CE site. Tyto komponenty provádějí bezpečnou registraci a generují přístupové údaje pro nové site tak, aby lokální řídicí služba každé z nich mohla zavést správnou identitu. Cílem v CEC platformě je provozovat na stejné úrovni zabezpečení řídicí i koncové aplikace. Každá komponenta je zaváděna s již zmíněnou kryptografickou identitou, kterou lze nadále použít pro bezpečnou komunikaci a autorizaci. To je jedna z největších výhod, jelikož jakákoli aplikace může tuto identitu (v tomto případě certifikát x.509) používat bez nutnosti překladu nebo transformace mezi různými prostředími. Obrázek 4.25 zachycuje komponenty v CEC platformě a jejich poskytnutí identity.

Existují dva typy identity v prostředí navržené platformy:

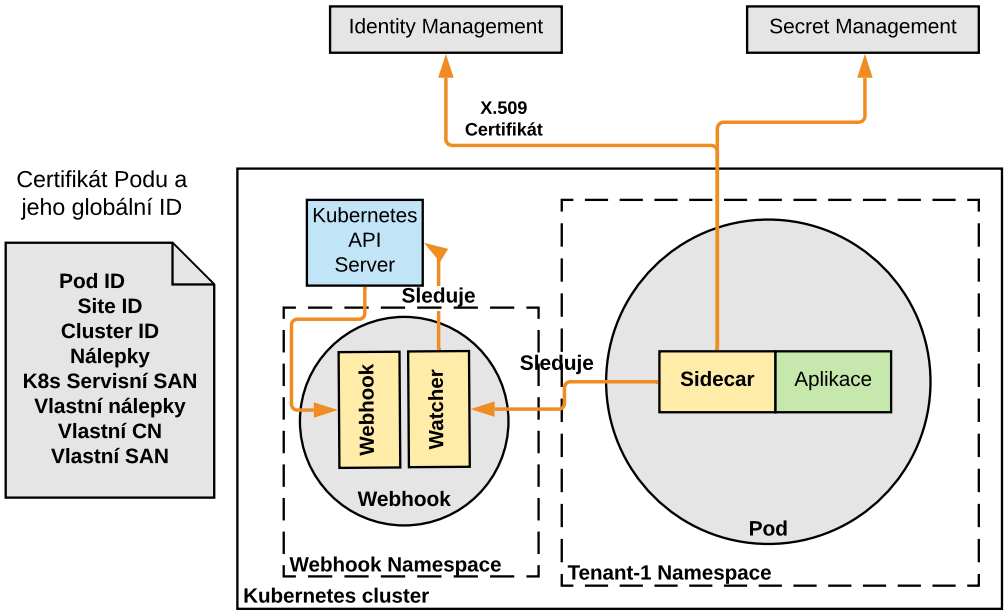
1. Site identita - ta je použita pro různé účely, například jedním z nich je navázání bezpečného komunikačního kanálu mezi ostatními site RE nebo CE pomocí IPSec nebo SSL VPN připojení.

2. Aplikační identita - aplikace koncových uživatelů běžící v rámci vk8s (kapitola 4.5). Dostávají bezpečnostní identitu skrze tzv. sidecar, který je automaticky vkládán do každého aplikačního podu. Více je vysvětleno v textu níže.



Obrázek 4.25: Identity management v CEC platformě, zdroj: vlastní tvorba

Jelikož CEC platforma je založena primárně na kontejnerové orchestraci Kubernetes, bootstrap identity a root-of-trust, řeší se pomocí technologie sidecar. Sidecar je koncept, který vznikl právě v prostředí Kubernetes. Jedná se o kontejner, který je automaticky vložen do každého podu a slouží jako jakýsi asistent běžících aplikací (Sayfan, 2018). Oba kontejnery sdílí síťovou i diskovou část, tudíž si mohou sdílet svá data. Nejznámějším typem je síťový sidecar pro správu komunikace každé aplikace, který vznikl společně s příchodem Service Mesh (Indrasiri & Siriwardena, 2018). Bezpečnostní sidecar je v tomto případě nastartován vedle každého aplikačního podu nebo kontejnerů a komunikuje s centrální identity serverem, který mu vrátí zmíněné X.509 certifikáty, jenž jsou následně dostupné aplikaci buď přes paměťový RAM disk nebo URL. Kromě této identity umožňuje sidecar také spravovat secrety a provádět jejich dešifrování nebo šifrování. Obrázek 4.26 zachycuje navrženou implementaci pomocí Kubernetes webhook, který při startu podu Kubernetes automaticky vloží sidecar s krátkodobým podepsaným tokenem, který je použit pro vyžádání X.509 certifikátu od centrální identity.



Obrázek 4.26: Root-of-trust v každém Kubernetes clusteru, zdroj: vlastní tvorba

Tento mechanismus poskytuje jedinečnou a univerzální identitu pro každou instanci aplikace běžící v CEC platformě. Jakmile je každá část v infrastruktuře vybavena jedinečnou kryptografickou identitou, lze na ní stavět další bezpečností požadavky, jako jsou autentizace, autorizace, zajišťování politik, správa klíčů atd.

V době našeho návrhu neexistovala implementace nebo koncepce bezpečnostního

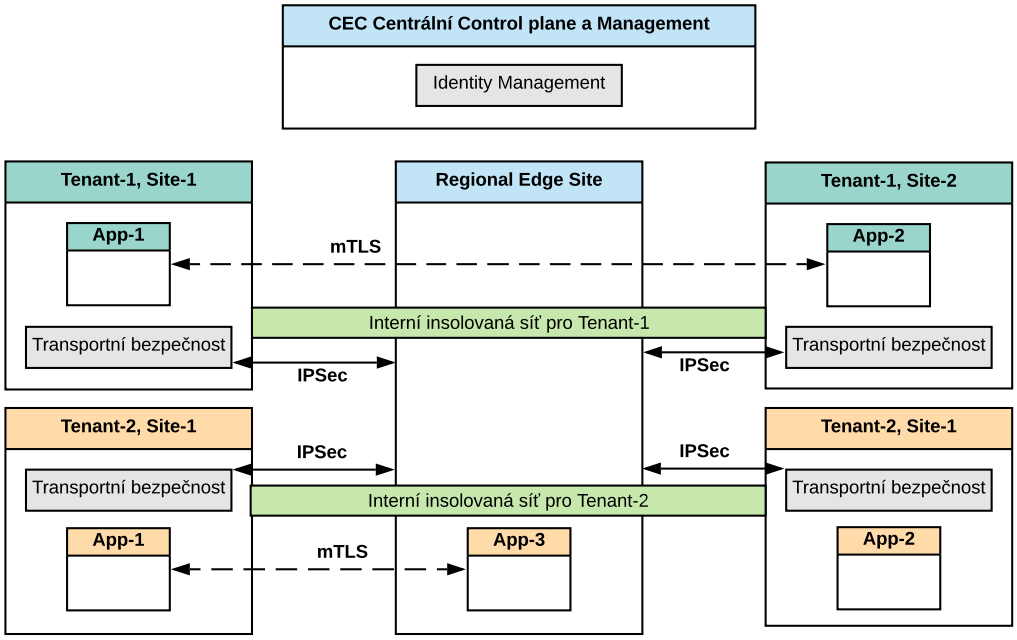
sidecar v prostředí Kubernetes, dokonce ani způsob zavádění kryptografické identity. Autorovým hlavním přínosem tak bylo navržení a částečné implementace koncepce bezpečnostního sidecar. Za potvrzení správnosti volby a zvolené cesty můžeme považovat oznámení z prosince roku 2019 společností Hashicorp, která představila vlastní bezpečnostní sidecar pro jedno z předních řešení pro secret management Hashicorp Vault (*Injecting Vault Secrets Into Kubernetes Pods via a Sidecar*, 2019).

4.6.2 Autentizace a autorizace

Mít jedinečnou identitu pro každý pod je dobrý začátek, protože velmi usnadňuje implementaci vzájemné autentizace mezi jednotlivými službami. CEC infrastruktura se skládá z mnoha různých služeb, které běží na různých protokolech jako jsou gRPC, REST, IPSec, BGP atd. Cílem při návrhu platformy bylo dosáhnout vzájemné autentizace a bezpečnosti komunikace skrze šifrovaný kanál pro všechny komunikující strany, a to bez ohledu na protokol. Z toho důvodu nebylo možné svázat se s některými existujícími technologiemi (např. Istio Service Mesh), které se omezují jen na konkrétní sadu protokolů, jako je například HTTP (*Istio Security Documentation*, 2019).

Jelikož všechny komponenty používají svoji X.509 identitu, je možné komunikovat autentizovaně a bezpečně pomocí mTLS nebo IPSec tunelů. Mutual TLS (mTLS) neboli obousměrná autentizace označuje dvě strany, které se současně navzájem autentizují, což bývá výchozí režim v některých protokolech (IKE, SSH) a volitelný například u TLS. TLS protokol při výchozím stavu prověřuje pouze identitu serveru přistupujícímu klientovi použitím X.509 certifikátu a jeho autentizace je ponechána na aplikační vrstvu. Nicméně TLS umožňuje právě použití také klientské autentizace pomocí X.509. Ta je velmi náročná, protože všichni klienti musí mít svůj certifikát a právě správa jejich identity bývá velmi náročná především v heterogenních prostředích (Chandramouli & Butcher, 2020). CEC platforma však právě díky navržené centrální identitě a konceptu zmíněném v kapitole 4.6.1, může tuto pokročilejší metodu aplikovat. Na základě toho je možné poskytnout ochranu od A do Z proti všem útokům, jako jsou například síťový spoofing, eavesdropping nebo man-in-the-middle.

Obrázek 4.27 zachycuje komunikaci mezi aplikacemi, která jsou jednak v šifrovaném IPSec/SSL tunelu z hlediska transportní bezpečnosti a současně také využívají šifrování a autentizaci na aplikační úrovní pomocí mTLS, ke kterému používá automaticky získanou identitu X.509 při svém startu.



Obrázek 4.27: Autentizace a autorizace v CEC platformě, zdroj: vlastní tvorba

Dalším logickým krokem pro dosažení vzájemně ověřeného zabezpečeného kanálu je autorizace. Jedná se o proces příjemce požadavku (serveru) k určení, zda povolit, či nepovolit požadavek přicházející od klienta. Důvodů, proč žádost nemůže být povolena, existuje mnoho - omezení kvót, oprávnění atd. Protože tyto důvody a jejich prahové hodnoty se dynamicky mění, není možné v distribuované platformě mít pevně zakódované sady pravidel pro autorizaci.

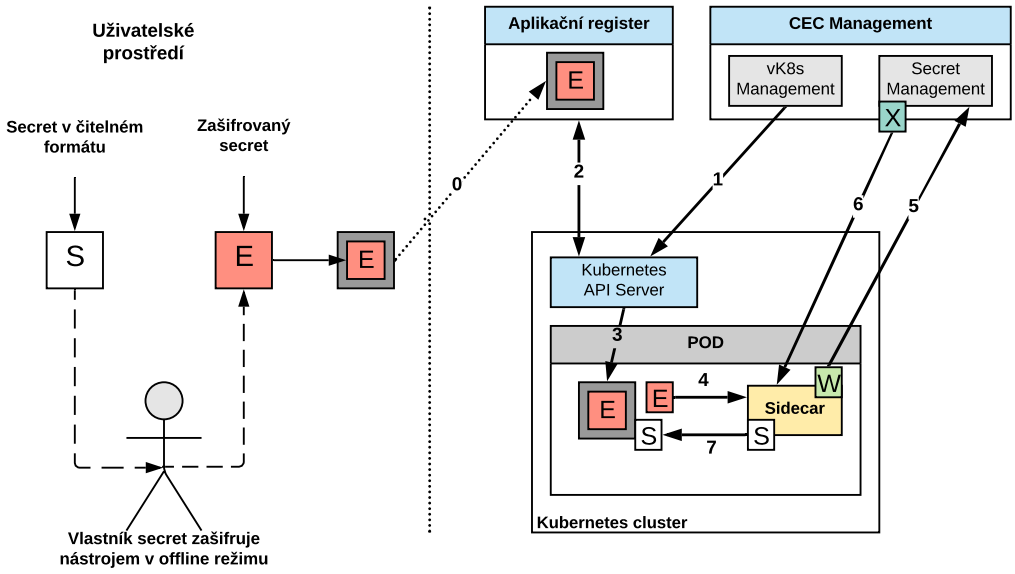
Systém autorizace v CEC platformě jsme tedy postavili na open source projektu Open Policy Agent (OPA), který slouží pro kontrolu politik v cloud native prostředích (*Open Policy Agent*, 2020). Tohoto agenta jsme museli obalit vlastním kódem tak, aby mohl běžet v již zmíněném sidecar společně s každým kontejnerem. To umožňuje dynamicky stahovat příslušné politiky a zásady, jež jsou rychle vyhodnocovány. Podobně jako u autentizace při oddělení autorizační části od identity (a autentizace) dává možnost prosazovat autorizační politiky při různých požadavcích a nejen bezprostředně při úvodní autentizaci aplikace. Pomocí jedinečné identity (vydané přes sidecar) pro autentizaci a programovatelného modulu politik (v rámci sidecar) pro autorizaci je CEC platforma schopna zabezpečit komunikaci pomocí mTLS a řídit každý přístup skrz robustní a programovatelné politiky. Tímto způsobem lze naprosto přesně naplnit již zmíněný princip Zero Trust *„nikdy nedůvěřovat, vždy ověřovat"* a docílit bezpečné komunikace mezi aplikacemi definované v (Chandramouli & Butcher, 2020).

4.6.3 Secret management

Téměř jakýkoli softwarový program musí pracovat s určitými senzitivními informacemi, jako jsou databázová hesla, TLS privátní klíče, API klíče, IAM přístupy atd. Bylo prokázáno, že každý den dělají vývojáři neúmyslné chyby ukládáním těchto klíčů a hesel do zdrojového kódu a ty se pak nějakým způsobem dostávají do veřejných úložišť. Správa secret je obtížná a bez snadno použitelné sady nástrojů a dobře definovaného procesu se od vývojářů očekává, že půjdou nejkratší možnou cestou. Podobným problémem se například zabývají (Bunyakiati & Sammapun, 2019), kde řeší správu secret při vývoji mobilních aplikací. Z těchto důvodů je nezbytné vytvořit snadnou a zabezpečenou cestu pro správu secret v rámci naší edge platformy.

Jelikož je celá platforma postavené na Kubernetes, bylo nutné prozkoumat, jaké jsou dostupné možnosti. Našli jsem dva nejpoužívanější způsoby pro správu secret v Kubernetes. Kubernetes secret jsou výchozí možností při jeho použití, avšak má to několik rizik. Především secret nejsou nijak šifrována, není možné definovat pokročilé politiky přístupu k nim a vůbec neřeší jejich centrální správu napříč více clustery (*Kubernetes Secrets*, 2019). Druhou možností je centralizované řešení Vault od společnosti Hashicorp (*Injecting Vault Secrets Into Kubernetes Pods via a Sidecar*, 2019). Funguje jako centrální úložiště pro secret a ta jsou přidělována autorizovaným žadatelům. Secret je chráněný jediným šifrovacím klíčem, který se používá pro šifrované uložení v centrálním místě. Problém s tímto přístupem je však v tom, že systém správy secret k nim má přímý přístup i přesto, že jsou uvnitř uložena v šifrované formě. V případě napadení centrálního úložiště existuje teoretická možnost získat všechny tyto secret. Dalším problémem je, že administrátor je schopen si dešifrovat libovolný secret. Jelikož CEC platforma může být provozována jako SaaS, zneužití nebo administrátorský přístup k zákaznickým senzitivním datům představuje velkou hrozbu.

Z těchto důvodů bylo navrženo vlastní řešení postavené na kryptografické technice slepého šifrování popsané v (Abe & Fujisaki, 2006). Tento přístup umožňuje vlastníkovi secretu jeho uzamknutí (zašifrování) takovým způsobem, že nemůže být jasně odhalen nežádoucí straně, a to i včetně dešifrovacího serveru. Secret tedy není uložen v centrálním místě, a to také jeho návrh výrazně zjednodušuje, jelikož přesouvá tuto odpovědnost na koncového uživatele. Obecný popis tohoto přístupu je zobrazen na obrázku 4.28. V této práci není nutné zabíhat do konkrétních implementačních detailů, avšak je dobré vysvětlit si základní princip.



Obrázek 4.28: Správa secret pomocí navrženého řešení, zdroj: vlastní tvorba

Uživatel použije nástroj, kterým kompletně v offline režimu provede zašifrování secretu (S), ten následně distribuuje se svojí aplikací a nahraje do kontejnerového registru. Poté během nasazení proběhne několik kroků:

1. vk8s spustí nasazení aplikace do fyzického Kubernetes,
2. lokální Kubernetes stáhne image z kontejnerového registru se zašifrovaným secret (S),
3. během spuštění je automaticky vložen zmíněný sidecar, který zavede příslušnou identitu pro pod,
4. aplikace zavolá lokální API sidecar pro dešifrování secret,
5. sidecar provede zaslepovací (blinding) operaci převedením S na W,
6. key-server v centrálním cloudu provede dešifrování zašifrovaného secret + zaslepovací operace, tedy převede W na X,
7. sidecar obdrží X a provede finální odslepovací transformaci z X na S, což je výsledný secret pro aplikaci.

Tímto je zajištěno, že centrální management nikdy nezíská přístup k dešifrované podobě (S), a ta je přístupná pouze během aktivní aplikace v její RAM paměti, nikoli na disku. Kromě toho je možné definovat i politiky přístupu k secret. Zásady lze definovat

na základě atributů identity jako její název, umístění atd. Tímto způsobem lze spravovat jakoukoli složitou aplikační skupinu a dosáhnou tak přesné kontroly přístupu. Tato politika je kryptograficky zavedena do procesu šifrování. Dalším krokem do budoucna by mohlo být confidential computing, které umožňuje HW šifrování u Intel SGX procesorů. Microsoft Azure na konci roku 2019 představil svoji integraci do Kubernetes[10].

4.6.4 Síťové politiky a firewall

Síťové politiky a firewall jsou klasickým nástrojem zabezpečení síťové komunikace. Jejich cílem je chránit určitý druh síťového provozu pomocí vytvoření bariéry mezi důvěryhodnou a nedůvěryhodnou sítí (Imran, Alghamdi, & Ahmad, 2015). Tyto funkce byly částečně představeny již v rámci návrhu EGW v kapitole 4.3. Z důvodu rozsahu práce se tato podkapitola věnuje pouze části, kterou se návrh významě liší od standardních síťových firewall zařízení.

Tradičně jsou síťové politiky navázány na síťové prvky infrastruktury jako například rozhraní, sítě a nebo konektory (Bavithra, Mahalakshmi, & Suganya, 2018). Nicméně v našem návrhu je cílem oddělit síťovou konfiguraci od síťových politik, což má umožnit lépe definovat záměr, než se zaměřovat na konkrétní síťovou topologii a přidělování adres. Tento přístup by také měl výrazně usnadnit opakované znovupoužití těchto politik. Podle (Keeriyattil, 2019) je důležité pro aplikaci konceptu Zero Trust, aby veškerý vstupní a výstupní síťový provoz přicházející na virtuální rozhraní kontejnerů byl porovnán s nastavenými pravidly. V případě, že paket není nalezen v žádném z definovaných pravidel, je automaticky zahozen. Tomuto přístupu se anglicky říká whitelisting nebo také pozitivní bezpečnostní model.

Síťové politiky se skládají z mnoha individuálních pravidel, avšak jejich základním konceptem v CEC návrhu jsou tři typy entit:

- Lokální a vzdálený koncový bod - lokální zde znamená z pohledu politiky. „Lokální koncový bod" je „síť", pro kterou je politika napsána. Vzdálený koncový bod je druhý konec připojení. Tyto body mohou být virtuální sítě, síťová rozhraní, sady IP adres, konkrétní IP adresa nebo podsíť.
- Výstupní pravidla - Výstupní je z pohledu lokálního koncového bodu. Tato pravidla jsou pro připojení navázaná z lokálního koncového bodu na vzdálený. Například pokud je lokální koncový bod rozhraní, potom všechna spojení z koncových bodů dosažitelná prostřednictvím tohoto rozhraní jsou klasifikována jako „výstupní". Pravidla definují koncový bod, protokol a port. Akce jdou povolit, nebo zakázat.

[10] https://azure.microsoft.com/en-us/blog/bringing-confidential-computing-to-kubernetes/

- Vstupní pravidla - vstupní je také z pohledu lokálního koncového bodu. Tato pravidla jsou pro připojení k němu ze vzdáleného koncového bodu. Pokud by byla lokálním bodem síť, všechna připojení směrem k němu jsou klasifikována jako vstupní. Podobně jako u výstupních pravidel definují koncový bod, protokol a port. Akce lze povolit, nebo zakázat.

Tradičně byla síťová a firewall pravidla psána z hlediska paketu pomocí pěti n-tice (zdrojová IP, cílová IP, protokol, zdrojový port, cílový port). Tato koncepce je velmi síťově orientována, a vyžaduje proto, aby operátor platformy rozuměl konkrétnímu síťovému provozu. Z toho důvodu bývá často velmi obtížné znovu použít definované politiky a zásady pro stejný záměr. Díky tomu jsou zmíněné koncové body, výstupní a vstupní pravidla konstruovány tak, aby nebylo nutné detailně rozumět směrování či toku síťového provozu. Jsou konstruovány tak, aby se mohl definovat záměr operátora. Navržené pravidla síťových politik podporují koncept porovnání štítků, v angličtině nazvaných label matcher. Jedná se o list štítků, který obsahuje vždy klíč a hodnotu. Ty musejí být stejné pro lokální i vzdálený koncový bod, aby byla povolena jejich vzájemná síťová komunikace.

Dobrým příkladem použití těchto štítků je při psaní pravidel, kde se bere v potaz typ nasazení. Předpokládejme, že existují dvě prostředí se stejnými aplikacemi *"deployment=testing"* a *"deployment=produkce"*. Pravidla říkají, že lokální koncový bod aplikace A může komunikovat směrem do vzdáleného koncového bodu B přes port 80 na protokolu TCP. Zároveň však nechceme, aby se aplikace A z testovacího prostředí mohla připojit do produkčního prostředí na aplikaci B. Právě na toto je vhodné použít pravidlo se štítky. Toto pravidlo obsahuje to, že lokální koncový bod A a vzdálený koncový bod B mají mít stejnou hodnotu klíče „deployment". Tímto způsobem lze velmi jednoduše definovat znovupoužitelná pravidla a není nutné měnit konkrétní IP adresy nebo podsítě jako u standardních síťových pravidel. Ukázka těchto pravidel je zachycena na 4.5, kde je vidět jedno pravidlo a poté umístění konkrétního štítku (label) v metadatech u Kubernetes aplikace.

Kód 4.5: Ukázka pro definici pravidla síťové politiky pomocí label matcher

```
# Definice pravidla pro povolení komunikace s
   label_matcher
metadata:
  name: pravidlo-1
spec:
  action: ALLOW
  ports:
```

```
    - "80"
  protocol: TCP
  label_matcher:
    keys:
      - deployment

# Ukázka umístění label na definici aplikace pro
   Kubernetes
metadata:
  name: aplikace-a
  labels:
    deployment: testing
```

4.6.5 Aplikační firewall

Aplikační bezpečnost je komplexní téma, které se skládá z mnoha vrstev. K jejich zabezpečení je potřeba mnoho technologií a nástrojů, a proto tento návrh jen okrajově nastiňuje některé z možností její implementace. Výzkum v této části nebyl zcela dokončen, takže je tato vrstva v době tvorby této práce spíše stručným shrnutím cílových funkcionalit, kterých má CEC platforma dosáhnout. Nástroje pro aplikační firewall používají kombinace tradičních technik statického podpisu, statistických algoritmů a dynamičtějšího strojového učení (Manaseer & Al Hwaitat, 2018). Víceméně jsme identifikovali tři funkce.

WAF je aplikační firewall určený pro HTTP aplikace. Funguje na principu aplikování různých pravidel na HTTP konverzaci. Obecně tato pravidla dokáží zachytit běžné útoky, jako jsou cross-site scripting (XSS) nebo SQL Injection. Zatímco proxy obecně ochraňuje klienty, WAF chrání servery. Je určen k ochraně konkrétních webových aplikací. Obvykle je ve formě server pluginu, zařízení nebo filtru (Manaseer & Al Hwaitat, 2018) . Aplikováním těchto pravidel na úrovni EGW Envoy řešení umožní, poskytnout tuto ochranu aplikačním serverům bez nutnosti její modifikace. Pomocí definování implicitních a explicitních WAF pravidel bude možné tuto funkci konfigurovat. Implicitní pravidla jsou uživatelem definované technologie jako například programovací jazyk, typ serveru apod. Tato pravidla jsou poté aplikována na konkrétní virtual-host v distribuované proxy. Explicitní umožní uživateli definovat pravidla, která chce naopak vyjmout z WAF detekce. Současně by mělo být možné také definovat mód, ve kterém WAF pracuje. V zásadě se jedná o monitorovací mód, který pouze zasílá události a upozornění o útocích nebo blokovací mód, při kterém WAF firewall automaticky blokuje požadavky klientů detekované jako útok.

Behaviorální analýza - ta by fungovala na principu strojového učení v rámci CEC namespace. Strojové učení by mohlo probíhat centrálně a používá data z naměřených metrik a sebraných logů všech EGW proxy, aby mohla vytvořit model. Tento model by mohl být následně distribuován do CE a RE site související s konkrétní aplikací. Síťový dataplane je potom schopný provádět odvození na základě tohoto modelu a blokovat určité typy komunikace. Identifikovali jsme nejméně tři typy behaviorální analýzy, které mohou být použity v rámci CEC platformy.

- Detekce anomálie podle požadavku - modely umělé inteligence jsou vytvořeny, aby se naučily základní chování různých druhů požadavků (Protic & Stankovic, 2019). Tento model se používá pro odvození požadavků na proxy, jejich označením od naučených modelů. Chování těchto požadavků je charakterizováno metrikami, jako jsou velikost požadavku, velikost odpovědi a doba odezvy mezi požadavkem a jeho odpovědí.
- Business logic markup - systém používá logy z přijatých požadavků (od klienta na server) a metriky z virtuálních hostů, aby byl schopen získat pravděpodobnostní distribuční funkci pro každé API rozhraní, čímž lze detekovat anomálie požadavků.
- Detekce anomálie časové řady pomocí metrik pro počet požadavků za vteřinu, chyb, dobu odezvy a propustnost.

Poslední navrhovanou funkcí je **Aplikační ochrana proti DDoS**, která kombinuje výstupy předchozích dvou. DDoS útoky mohou být detekovány pomocí událostí definovaných WAF pravidly, stejně jako události detekované anomálií z behaviorální analýzy (Y. Zhang, Liu, & Zhao, 2010). DDoS útoky ovlivňují pouze dataplane a ten musí být schopen zvládnout různé techniky na vyčerpaní zdrojů, např. tabulka flow pomocí syn flood, fragmentační buffer, nat pool apod. Dataplane musí umět rychle reagovat a nastavit přístupové listy na síťovém firewall, aby zabránil útokům na úrovni aplikace ze strany těchto klientů (Zeebaree, Hussein, & Muhamad, 2018).

5 Zhodnocení přínosů navržené edge computing platformy

Tato kapitola je zaměřena na zhodnocení přínosů navržené CEC platformy. Konkrétně se jedná o metody technického ověření a také ověření přínosu pro podnikový management z hlediska výkonu doručování softwaru. Nejprve jsou vymezeny jednotlivé metody a směry v rámci kapitoly 5.1 a následně je provedeno technologické ověření v kapitole 5.2 pomocí výkonnostního, funkčního a bezpečnostního testování. V kapitole 5.3 je poté vysvětlen vliv výkonnosti doručování softwaru na podnikový management a provedeno ověření skrz případovou studii restauračního řetězce s několika tisíci pobočkami ve Spojených státech amerických. Cílem je podívat se na přínosy výkonnosti společnosti skrz efektivnější doručování softwaru do restaurací. Přínosy a výsledky jsou diskutovány v závěrečné kapitole 5.4.

5.1 Vymezení metod pro ověření výsledků

V této části jsou vysvětleny metody, které byly použity k měření přínosů v rámci ověření CEC platformy. Existují různé pohledy, kterými se lze dívat na zhodnocení a ověření nové platformy. Jedním z nich je například zkoumání její efektivity, kterou lze dekomponovat na pohled ekonomický, manažerský nebo technologický. Lze se také zaměřit na problematiku vlivu platformy na efektivní řízení, budování a měření přínosů rozsáhlé enterprise architektury, jakou edge computig bezpochyby představuje. Další možností by bylo podívat se na vyhodnocení nebo zhodnocení investic provozu edge computingu a s tím souvisejících ekonomických metod. Z technologického pohledu by bylo možné se podívat na výkonnostní management informačních a komunikačních technologií nebo stanovení metrik a parametrů pro úroveň služeb SLA apod.

Jak lze vidět, existuje široká škála možností a pro nás bylo důležité v rámci této disertační práce se zaměřit na několik konkrétních míst, která přímo souvisí s cíli a definovanými problémy edge computing platformy z předcházejících kapitol. Zároveň jsme chtěli tato ověření částečně provádět na skutečné společnosti, abychom mohli tento výzkum i nadále rozvíjet a aplikovat do praxe. Z toho důvodu byly naše metody a sledované veličiny rozděleny do dvou základních pohledů:

- **Technologické ověření platformy** - ověřit přínosy a naplnění definovaných problémů, jako jsou doba odezvy, bezpečnost nebo multifunkčnost řešení.
- **Stanovení přínosu pro podnikový management z pohledu softwarové výkonnosti** - ověřit vliv platformy na softwarovou výkonnost společnosti a jakým způsobem se zlepší nebo zhorší jejím využitím, případně jaký vliv může mít na provoz ve společnosti.

5.2 Technologické ověření

Technologické ověření je rozsáhlé téma a lze na něj nahlížet z různých pohledů. Například v (Bouchenak et al., 2013) si pokládají otázku, co kromě tvrzených parametrů poskytované cloud computing služby (schopnost uložit data, provést výpočetní operace atd.) může zajímat koncové uživatele. Identifikují 4 základní oblasti zájmu. Důvěryhodný software a jeho identita - běží daná služba se správnou verzí softwaru a na správném místě? Funkční správnost - jakmile služba běží, chová se podle toho, jak by se měla chovat? Výkonnost služby a její závislost - jak efektivní je daná služba a je spolehlivá a dostupná? Bezpečnost - vyhovuje daná služba bezpečnostním politikám a standardům, pokud nějaké existují?

Pokusili jsme se vycházet z těchto kategorií a vzhledem k definovaným cílům disertační práce jsme technologické ověření rozdělili do 3 kategorií:

- Výkonnostní ověření CEC platformy - měření doby odezvy EGW v porovnání se standardním dostupným řešením,
- Funkční ověření CEC platformy - splnění certifikace standardního orchestračního API,
- Bezpečnost ověření - sada penetračních testů vůči edge zařízením se zaměřením na multi-tenanci a jejich výsledky.

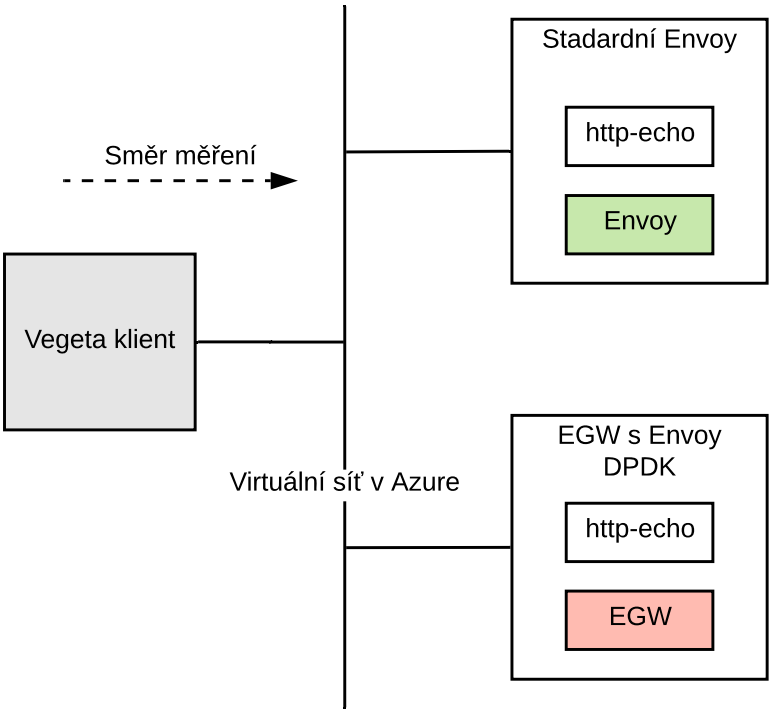
5.2.1 Výkonnostní ověření

Jak bylo několikrát v práci zmíněno, jedním z cílů je poskytnout řešení pro snížení doby odezvy. Jak je popsáno v kapitole 4.3, doba odezvy se dá snížit technologickou optimalizací nebo fyzickou vzdáleností klienta od serveru. Jelikož navržená CEC architektura počítá s dodatečnou vrstvou ještě blíže ke klientům, je zcela jasné, že umístěním EGW do místa ke klientovi dojde ke snížení doby odezvy, a tudíž takové měření nedává moc velký smysl. Zaměříme se tedy na navrženou edge gateway z pohledu technologické optimalizace, kde stěžejním přínosem je integrace L7 aplikační proxy s Intel DPDK.

Součástí ověření by mohlo být také porovnání výkonnosti data plane L3 založeném na vRouter vůči klasickému kernelovému data plane, avšak to proběhlo již před samotným návrhem a některé výsledky jsou součástí této práce 4.2.

Existuje několik scénářů, podle nichž lze ověřit výkonnost. Jednou z možností je zacílit na individuální propustnost a odezvu jedné site s EGW a porovnat to s již existujícím řešením, nebo se podívat na výkonnost z hlediska TLS offloading, tedy jak efektivní je navržená EGW při ukončování TLS spojení. Další možností je změřit optimalizaci trasy mezi různými site a zjistit, jak se změní latence při průchodu síťového provozu skrz VPN nebo přímým spojením. Kolik dokáže maximálně EGW propustit svým VPN tunelem a s jakou latencí. Nicméně v rámci této práce vybereme pouze jeden úhel pohledu ověření a přínosu EGW. Zaměříme se na vytvořenou integraci open source Envoy proxy s Intel DPDK v porovnání s tradiční Envoy proxy běžící v linuxovém kernelu.

K otestování doby odezvy a propustnosti EGW v rámci CEC site jsme použili simulaci prostředí v Azure veřejném cloudu, kde jsme spustili dvě VM F8-Series. Tento typ je optimalizován pro náročné výpočetní výkony a je založený na 2.4 GHz Intel Xeon® E5-2673 v3 (Haswell) procesoru. Konkrétně F8 obsahuje 8 vCPU, 16GB RAM a 128GB systémový disk. Tento typ také používá technologii SR-IOV, která umožňuje přímý přístup na fyzickou síťovou kartu. Díky tomu je možné lépe nasimulovat skutečné fyzické zařízení běžící v edge site. Pro testovací účely jsme použili jednoduchou HTTP aplikaci http-echo (Vargo, 2019), která poskytuje malý webový server napsaný v jazyce Golang. Obrázek 5.1 zachycuje kompletní testovací architekturu.



Obrázek 5.1: Topologie testovacího labu pro měření latence v Microsoft Azure, zdroj: vlastní tvorba

V rámci testování jsme měřili dobu odezvy pomocí nástroje Vegeta[1], kde jsme současně získali počet požadavků za vteřinu RPS. Výstup 5.1 ukazuje použitou konfiguraci, kde jsme využívali 8 procesorů současně s frekvencí maximálně 330 000 požadavků za vteřinu po dobu 5 minut. Nástroj Vegeta nám umožnil uložit data do jeho binárního souboru a následně ho exportovat jako CSV do statistického nástroje R. Současně také poskytl jednoduchý report výstup s kvantily, mediánem, počty požadavků za vteřinu a přenesenými bajty. Nicméně naším cílem bylo analyzovat tato data pomocí statistiky dále v textu a potvrdit nebo vyvrátit hypotézu o snížení doby odezvy.

Kód 5.1: Použití nástroje Vegeta na měření doby odezvy

```
echo 'GET https://<IP adresa server>' | vegeta -cpus 8
   attack -rate 330000 -duration 5m > result.bin
```

Při analýze dat o latenci jsme částečně vycházeli již z našich provedených statistic-

[1] https://github.com/tsenart/vegeta

kých analýz a automatizace pro měření dostupnosti cloudových služeb publikovaných v (Pavlik, Sobeslav, & Horalek, 2014) a (Pavlik, Sobeslav, & Komarek, 2014). Obecně statistici testují, zda data odpovídají Gaussovu normálnímu rozdělení, protože většina testů (t test, z test, F test, ANOVA) předpokládá normálnost dat. Proto jsme se rozhodli nejprve ověřit, zda latence pochází z normálního rozdělení. Pro toto ověření existuje několik testů, jako jsou Shapiro-Wilkův test, Jarque-Bera test, Pearsonův test chí-kvadrát atd. Všechny jsou nějakým způsobem interpretovány uvnitř statistického nástroje R, který byl použit. Rozhodli jsme se použít Shapiro-Wilkův test, protože je to jeden z nejsilnějších testů. Vzorec 5.1 ukazuje tento statický výpočet. Hodnoty a_i jsou dostupné ve statistických tabulkách.

$$W = \frac{(\sum_{i=1}^{n} a_i x_{(i)})^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{5.1}$$

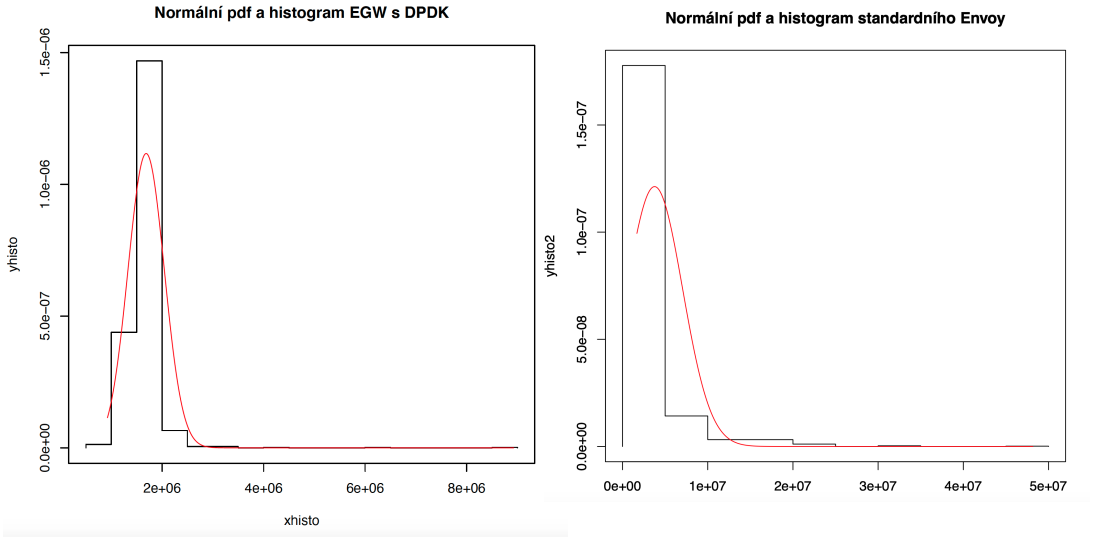
K provedení Shapiro-Wilkova testu byla použita funkce R shapiro.test(x), která vypočítala následující 5.2 hodnoty W a p pro standardní Envoy a EGW s DPDK.

Kód 5.2: Výsledky testu normálního rozdělení u standardní Envoy a EGW s DPDK

```
# Test stadardního Envoy
> shapiro.test(standard)

   Shapiro-Wilk normality test

data: without
W = 0.43029, p-value < 2.2e-16

# Test EGW s DPDK
> shapiro.test(dpdk)

   Shapiro-Wilk normality test

data: with
W = 0.60652, p-value < 2.2e-16
```

Nulová hypotéza říká, že výběr dat pochází z normálního rozdělení. V obou případech je však p hodnota (2.2e-16) nižší než 0,05 a nulová hypotéza se zamítá. Data nepochází z normálního rozdělení. Jelikož je p hodnota velmi nízká,vyplatí se podívat také na grafické porovnání pravděpodobnostního rozdělení. Jednou z těchto technik je graf křivky normálního pravděpodobnostního rozdělení a histogramu četností výběru dat. Diagram 5.2 zobrazuje oba naměřené vzorky četností dat a křivku normálního roz-

dělení, které by se měly obě dotýkat. Jak lze vidět na první pohled u vzorku dat EGW s DPDK, je výsledek opticky lepší, avšak oba výběry dat se nepřibližují normálnímu rozdělení, čímž se potvrzuje výsledek Shapiro-Wilkova testu.



Obrázek 5.2: Grafy normálního pdf a histogramu, zdroj: vlastní tvorba

Při nepotvrzení normálnosti dat existuje spousta dalších rozdělení pravděpodobnosti (Lognormální, Gamma, Weibull, Exponenciální atd.), my jsme se rozhodli pro použití neparametrického dvouvýběrového Wilcoxonova rank-sum (někdy označovaného také jako Mann-Whitney U-test) testu statické hypotézy, používaného při porovnávání dvou nezávislých vzorků, zda se jejich průměrné populační řady liší. Jedná se o alternativu pro parametrický t test v případě, že se jedná o nenormální rozdělení a vzorky dvou výběrů nemají výrazně odlišné rozdělení. Aby bylo možné oba naměřené výběry porovnat, vybrali jsme z každého testu 1200 nezávislých hodnot, a to z období mezi 1.-2. minutou testování z důvodu vyhnutí se zkresleným hodnotám při startu měření. Stanovili jsme následující hypotézu:

H_0: *standard* = *dpdk*, není rozdíl v latenci mezi standardním Envoy řešení a implementací v rámci EGW s DPDK
H_A: *standard* > *dpdk*, EGW s DPDK má nižší latenci než standardní Envoy (Pravostranná alternativa)
Tato hypotéza je řešena na hladině významnosti $\alpha = 0.05$

Princip výpočtu Wilcoxonova rank-sum spočívá v seřazení obou souborů dat podle pořadí. Tyto hodnoty jsou následně použity pro výpočet testové statistiky U_1 a U_2, de-

finovaných níže:

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1$$
$$U_2 = n_1 n_2 + \frac{n_2(n_2+1)}{2} - R_2 \tag{5.2}$$

kde R_1 = součet pořadí pro skupinu 1 a R_2 součet pořadí pro skupinu 2. Následně se vybere zvolí menší hodnota U a porovná se s kritickou hodnotou definovanou ve statistických tabulkách.

Výstup 5.3 zobrazuje celý tento výpočet ve statickém prostředí R. Je zřejmé, že zamítáme nulovou hypotézu díky $p < 0.05$. Výsledná hodnota 2.2e-16 ukazuje na výrazné rozdíly ve středních hodnotách, což bylo dobře vidět i na grafu četností obou vybraných souborů 5.2.

Kód 5.3: Použití nástroje Vegeta na měření doby odezvy

```
> wilcox.test(standard, dpdk, alternative="greater",
   paired=FALSE)

   Wilcoxon rank sum test with continuity correction

data: without and with
W = 1550432, p-value < 2.2e-16
alternative hypothesis: true location shift is greater
   than 0
```

Na základě výše uvedeného statistického ověření potvrzujeme, že navržené řešení EGW s Envoy na DPDK má nižší dobu odezvy než standardní dostupná Envoy proxy. Tabulka 5.1 ještě zachycuje shrnutí výsledků měření, kde je vidět rozdíl v mediánu naměřených hodnot téměř 1,2 ms. Kromě doby odezvy také vidíme, že EGW dokázalo v daném pětiminutovém intervalu obsloužit 3krát více požadavků než standardní kernel proxy.

Typ	**Počet požadavků za vteřinu**	**Latence (medián)**
Standardní Envoy	52k	2,91 ms
EGW s Envoy DPKD	172k	1,71 ms

Tabulka 5.1: Porovnání výkonnosti standardního Envoy a modifikovaného v rámci EGW, zdroj: vlastní tvorba

5.2.2 Funkční ověření

Pro funkční ověření nebo testování edge computing platformy neexistuje konkrétní standard. Je to především z důvodu jejich krátké existence a chybějících standardů popsaných v kapitole 3.3.2. Snažili jsme se proto inspirovat ve světě cloud computingu, který existuje o poznání déle. Podle (Bouchenak et al., 2013) se funkční testování skládá ze tří fází. *Vygenerování testovací sady*, při které se provede analýza testovaného softwaru a jeho specifikace za účelem získání efektivních testovacích případů. Ty jsou následně sestaveny do testovací sady. *Provedení testovací sady* kdy je software podroben vygenerované sadě testů. *Výsledek ověření*, kdy se porovnává provedená testovací sada s předepsanou specifikací. Pro každý testovací scénář se posuzuje úspěch nebo neúspěch. Vzhledem k cílům této práce jsme se snažili ověřit technický přínos standardizovaného API pro orchestraci edge aplikací, konkrétně navrženou komponentou vK8s 4.5, a najít vhodnou testovací metodiku k provedení uvedených tří kroků funkčního testování.

Testovací metodika a architektura testovacího systému

CEC platforma a její vK8s komponenta je založena na Kubernetes API, které se v posledních letech rozšířilo a existuje přes 90 jeho distribucí. Aby dokázalo CNCF foundation zajistit konzistenci a přenosnost mezi distribucemi, vytvořilo Kubernetes Software Conformance Certification program (CNCF, 2020). Všichni dodavatelé a výrobci jsou vyzváni, aby předložili výsledky testování a získali certifikaci CNCF, která formálně certifikuje vyhovující implementace. Tento program dává koncovým uživatelům jistotu, že při používání dané distribuce Kubernetes se mohou spolehnout na vyhovující funkční parametry. Současně také poskytuje nezávislým dodavatelům softwaru jistotu, že pokud jejich zákazník používá certifikovanou distribuci, bude se jejich software chovat dle očekávání. Z těchto důvodů je Kubernetes Conformance testovací sada přesně to, co chceme v rámci CEC platformy ověřit. Chceme rovněž prokázat, zda platforma umožňuje provozovat standardní cloudové aplikace přímo v edge koncových zařízení bez jejich výrazných změn.

Metodika testování je rozdělena do několika kroků, které jsou uvedeny níže:

- vytvoření vK8s objektu v CEC platformě,
- instalace testovacího nástroje,
- spuštění testů,
- stáhnutí výsledků,
- vyhodnocení výsledků.

Kubernetes Conformance testovací sada je založena na nástroji Sonobuoy[2], který pracuje přímo s Kubernetes API. Celé testování probíhá zhruba 60 minut a výstupem je zabalený adresář s logy a výsledky. Testy jsou rozděleny do 9 logických skupin podle typu testování a celkem obsahuje testovací sada 4413 testů. Ty se ovšem liší s ohledem na aktivované či deaktivované funkce, takže například u výchozího Google Kubernetes se provádí okolo 200 testovacích scénářů, který končí se 100% úspěšností. V případě testování navržené CEC platformy, konkrétně orchestrační části vK8s, bylo provedeno 215 testů. Tabulka 5.2 zobrazuje přehled výsledků Conformance testů dle jednotlivých kategorií s poznámkou o důvodech nesplnění testu. Z výstupů lze vidět, že jsme neuspěli ve 3 kategoriích s celkem 5 neúspěšnými testy.

Při detailnějším zkoumání těchto testů se jednalo o funkce nepodporovaného objektu pro QoS, který nebyl v současné době implementován ve vK8s, avšak jeho podpora může být velmi jednoduše přidána. Dále se jednalo o nemožnost mazání nebo vytváření Kubernetes namespace, což nemůže být z principu podporováno, jelikož je tento namespace vázán na namespace v globálním API a uživatel spravuje své namespace o úroveň výše. Namespace tedy může být vytvořen, nebo smazán, avšak tato operace není dovolena přes Kubernetes API. CRD neboli Custom Resource Definition byl specifikován již v kapitole 4.5 jako nepodporovaná třída. Jedná se o rozšíření Kubernetes API o vlastní typy tříd a objektů většinou z důvodu integrací na další systémy, které v kontextu vysoké škály edge computingu nedávají velký smysl. I přesto by bylo možné rozšířit vK8s v budoucích verzích o podporu základních CRD. Posledním neúspěšným testem bylo ověření limitů pro RAM a CPU u kontejnerů. Důvodem je vlastní logika přiřazování těchto limitů na základě vK8s profilů. Profil definuje CPU a RAM limity v anotaci objektů a ignoruje tak nastavené limity přímo pro PODy. Důvodem je orchestrace do potenciálních tisíců site, kde je nutné správně plánovat zdroje a neumožnit konfigurovat rozdílné limity různým uživatelům libovolně. Díky tomu test nebyl schopný ověřit stejnou velikost limitů mezi Deployment a Pod objektem. Každopádně vlastnost limitů funguje, avšak s mírně rozdílnou konfigurací oproti standardnímu Kubernetes API.

Z funkčního pohledu lze říci, že pouze jeden test na CRD nelze žádným způsobem obejít nebo docílit stejné funkce mírně rozdílnou konfigurací prostředí vK8s. Dle výsledků by ovšem CEC platforma jako oficiálně certifikovaná Kubernetes distribuce neuspěla z důvodu 5 neúspěšných testů i přesto, že dle výše uvedeného nemají na hlavní funkce Kubernetes příliš velký vliv.

[2]https://github.com/vmware-tanzu/sonobuoy

Testovací skupina	Popis testů	Počet provedených testů	Počet úspěšných testů	Počet neúspěšných testů	Poznámka
k8s.io	Testy týkající se funkcí samotných kontejnerů jako exec, získání IP adresy, nastavení příkazů, argumentů, atd.	44	43	1	Nepodporované QoS třídy pro Pod
sig-api-machine	Testy orientované na rozšiřující funkce API, odstranění nepoužívaných objektů, apod.	19	16	3	Nepodpora mazání namespaces a vytváření CRD
sig-apps	Testy ověřující práci s Deployment, Daemonset, Stafulset, Job a dalšími Kubernetes objekty.	22	22	0	
sig-auth	Testy ověřující práci se s Kubernetes servisními účty.	2	2	0	
sig-cli	Testy týkající se funkcí pro klientský nástroj kubectl.	22	22	0	
sig-network	Testy pro ověření síťových funkcí, jako je DNS, komunikace mezi kontejnery a jejich vystavení.	15	15	0	
sig-node	Testy orientované na vlastnosti serverů, na kterých běží kontejnery.	8	8	0	
sig-scheduling	Testy týkající se funkcí plánování a distribuce kontejnerů v rámci Kubernetes clusteru.	3	2	1	Chyby v testech s limity pro CPU a RAM u kontejnerů
sig-storage	Testy orientované na úložiště v Kubernetes	80	80	0	
Celkem		**215**	**210**	**5**	

Tabulka 5.2: Výsledky Kubernetes Conformance testů provedených na platformě CEC, zdroj: vlastní tvorba

5.2.3 Bezpečnostní ověření

Na bezpečnostní ověření lze nahlížet z pohledu technologického nebo procesního. Existuje spousta bezpečnostních certifikací a standardů, jako jsou například PCI DSS, GDPR nebo ISO27001. Například cílem zmíněné PCI DSS je usnadnit přijímání standardů v oblasti platebních karet a zamezit jejich zneužití (Bonner, O'Raw, & Curran, 2011). Naproti tomu ISO27001 je efektivní řízení a správa ISMS postavené na modelu Plan Do Check Action (PDCA), jehož cílem je neustálé zlepšování informační bezpečnosti skrz procesy napříč celou organizací (Candiwan, 2014). V rámci této části práce nás zajímá především technologické hledisko zabezpečení, nikoli procesní. Hlavním cílem je po-

tvrdit multi-tenancy navržené platformy v rámci edge site, tedy ověřit bezpečnou izolaci jednotlivých síťových a virtualizačních komponent.

Podobně jako u funkčního ověření jsme se v rámci bezpečnostního ověření CEC platfromy inspirovali u existujících přístupů cloud computingu. Konkrétně se jednalo o prameny (Felderer et al., 2016), (Krishnaveni, Prabakaran, & Sivamohan, 2015) a (Tian-yang, Yin-sheng, & You-yuan, 2010). Zvolená testovací metodika je především s ohledem na cíl práce multi-tenance v Edge Computing řešení a splněný běžných bezpečnostních standardů.

Zvolená testovací metodika

Obecně podle (Felderer et al., 2016) je metodika testování bezpečnosti cloudu sada technik, nástrojů a procesů, které je třeba dodržovat při testech cloudových služeb. Některé z těchto metod a technik jsou adaptací konvenčních technik a jiné byly speciálně vyvinuty tak, aby vyhovovaly testovacím potřebám edge computing služeb. Při testování cloudových aplikací je nutné vzít v úvahu jejich kontext. Zkoumali jsme několik výše uvedených článků o technikách testování bezpečnosti a v podstatě se jedná o níže uvedené.

- **Kontrola kódu** známá také jako statistická analýza je proces ruční kontroly bezpečnostních slabin zdrojového kódu. Mnoho vážných bezpečnostních chyb nelze detekovat žádnou jinou formou analýzy nebo testování. Většina odborníků na bezpečnost souhlasí s tím, že neexistuje náhrada za skutečné procházení zdrojového kódu. Se zdrojovým kódem může tester přesně určit, co se děje nebo co se má stát.

- **Fuzzy testování** vkládá do testu softwaru náhodná neplatná data (obvykle lehkým upravením vstupních dat) prostřednictvím testované nebo jiné softwarové komponenty. Termín fuzzing je odvozen od fuzz utitily, která je generátorem náhodných znaků pro testování aplikací vkládáním náhodných dat na jejich rozhraní. Je to jakési vytváření umělého šumu na testovanou aplikaci a pozorování jejího chování, čímž lze odhalit různé bezpečnostní hrozby.

- **Injekce poruchy zdrojového kódu** je testovací technika původně vytvořená komunitou pro bezpečnost software. Používá se k vyvolání stresu a simulaci poruch v software tak, aby došlo k odhalení míst ohrožujících bezpečnost. Účelem těchto simulovaných poruch je vyzdvihnout chyby, které mohou neúmyslně způsobit uživatelé při používaní softwaru.

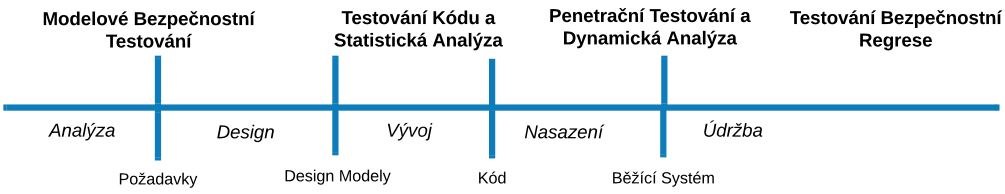
- **Risk analýza** se provádí během fáze vývoje softwaru, aby se přezkoumaly bezpečnostní požadavky a identifikovala se možná rizika. Modelování hrozeb je me-

todický proces, který se používá k identifikaci zranitelnosti v softwaru. Pomáhá architektům systémů analyzovat a uvědomit si bezpečnostní hrozby, kterým jejich systém může čelit. Proto je modelování hrozeb prováděno jako hodnocení rizik pro vývoj softwaru. Ve skutečnosti umožňuje vývojářům vyvinout strategie na zmírnění potenciální zranitelnosti a pomáhá se soustředit na ty nejohroženější části systému.

- **Skenování zranitelnosti** (vulnerability scanning) probíhá automatizovaně nástroji, které podporují aplikace různých úrovní od webových serverů přes databáze až po operační systémy. Jedná se o velmi užitečnou metodu pro testování zabezpečení softwaru. Tyto nástroje prohledávají aplikace a jejich vstupní a výstupní parametry porovnávají se známými hrozbami. Tyto různé vzory zranitelnosti a jejich stopy jsou porovnávány se stopami virových skenerů nebo s nebezpečnými kódovými konstrukty, které automaticky vyhledává skener zdrojových kódů. Jedná se tedy o automatizovaný způsob testování.

- **Penetrační testování** také známé jako etické hackování, je běžnou technikou testování bezpečnosti sítě. V penetračním testu je aplikace nebo systém testován zvnějšku z pohledu, který je srovnatelný se skutečným škodlivým útokem třetí strany. To znamená, že ve většině nastavení má entita, která provádí test, pouze omezené informace o testovaném systému a je schopna se integrovat pouze s veřejnými rozhraními testovaných služeb.

Když se podíváme na mapování technik bezpečnostního testování vzhledem k fázím vývoje softwaru na obrázku 5.3, vidíme, že penetrační testování a skenování zranitelnosti jsou techniky orientované na provoz systému. Z toho důvodu jsme se rozhodli pro účely této práce provést bezpečnostní ověření CEC platformy právě skrz penetrační testování a skenování hrozeb, jelikož vývojový cyklus nezapadá do námi definovaných cílů.



Obrázek 5.3: Techniky bezpečnostního testování v průběhu vývoje softwaru, převzato z (Felderer et al., 2016)

Testovací metodika pro penetrační testy a skenování hrozeb byla rozdělena do čtyř kroků na základě (Scarfone, Souppaya, Cody, & Orebaugh, 2008):

1. Plánování - v této fázi neprobíhá žádné konkrétní testování. Místo toho jsou definovány a zdokumentovány podmínky a hranice testování.

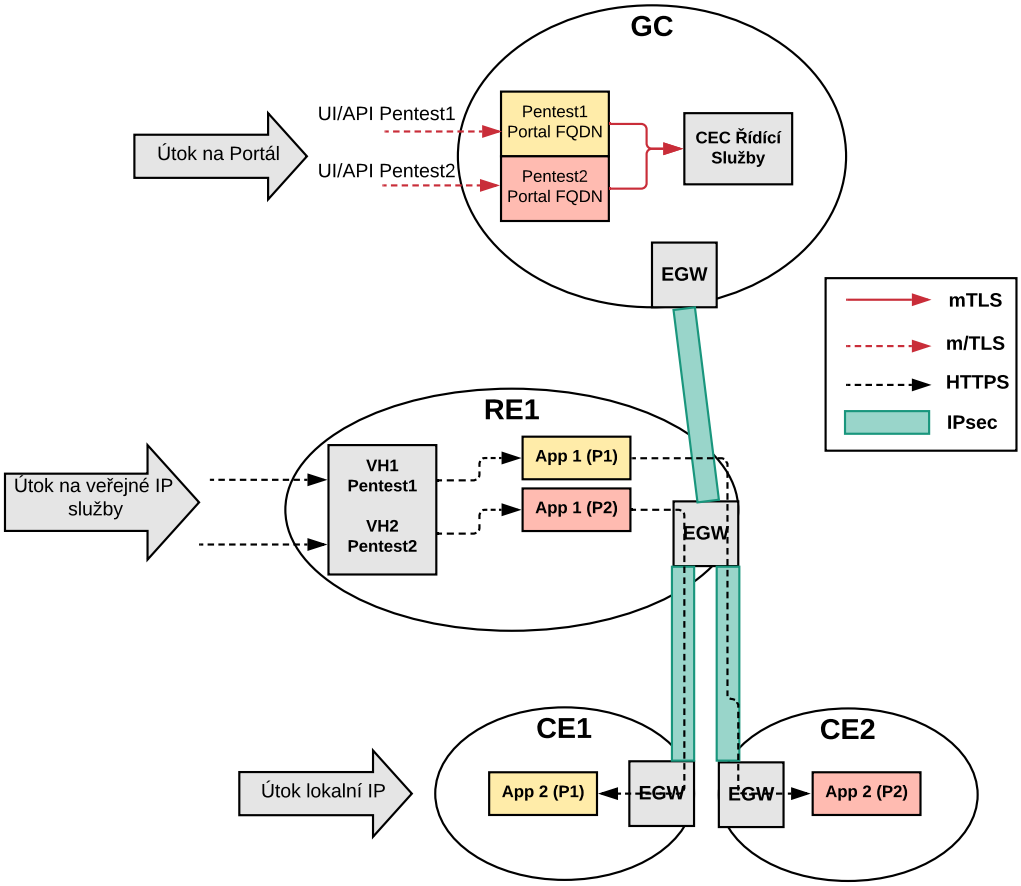
2. Objevování - tato fáze se skládá z několika kroků. Nejprve jsou systematicky nalezena a spočtena všechna dostupná externí rozhraní testovaného systému. Tato rozhraní tvoří základ pro počáteční útok. Druhou částí fáze zjišťování je analýza zranitelnosti, ve které jsou identifikovány použitelné třídy zranitelnosti, které odpovídají daným rozhraním, jako je například Cross-Site Scripting pro HTTP služby nebo SQL Injection pro aplikace s databázemi.

3. Útočení - nakonec jsou identifikovaná rozhraní testována pomocí řady útoků. Při těchto útocích se testeři aktivně pokoušejí ohrozit systém zasíláním škodlivého obsahu. V případě úspěchu jsou nalezené bezpečnostní díry zneužity za účelem získání dalších informací o systému tak, aby byl útok co nejúčinnější. Případné další objevené hrozby jsou znovu použity pro dodatečné útoky. Součástí této fáze je i použití nástrojů pro skenování hrozeb.

4. Hlášení - fáze hlášení probíhá současně s ostatními třemi fázemi penetračního testu a dokumentuje všechna zjištění spolu s jejich odhadovanou závažností.

Provedené testování

Pro penetrační testování jsme vytvořili topologii zachycenou na obrázku 5.4, která se fyzicky skládala z globálního řídicího controlleru GC, jednoho regionálního edge RE a dvou consumer edge CE. Abychom dokázali ověřit mutli-tenancy řešení, vytvořili jsme dva tenanty pentest1 a pentest2, z nich každý vlastnil po jednom CE. Tato topologie je zjednodušená z pohledu pro penetrační test a ukazuje typy síťové komunikace (veřejnou, lokální, mTLS, IPSec), aby bylo možné zvolit příslušné metody útoku. Jak lze vidět, v jednom RE běží aplikace obou tenantů (app1 P1 a app1 P2), aby bylo možné ověřit kompletní izolaci a multi-tenancy dvou tenantů v rámci jedné site. Toto ověření je klíčové pro potvrzení výsledků naší práce.



Obrázek 5.4: CEC topologie pro penetrační testování, zdroj: vlastní tvorba

Penetrační testování bylo rozděleno do dvou základních kategorií. **Aplikační penetrační testování** prováděné na konkrétních instancích, aby bylo možné identifikovat zranitelnost rozhraní webových a klientských aplikací, jako je například přístupový portál. Vybrali jsme několik testovacích technik, které byly provedeny na CEC prostředí:

- Obcházení vstupní validace - pokus o obejití klientských validačních kontrol a poslat neimplementované prvky aplikačnímu serveru.
- SQL injection - technika odesílání speciálně vytvořených příkazů SQL k ověření vstupních ovládacích prvků pro ochranu dat databáze.
- Cross-site scripting - odesílání aktivního obsahu do aplikace ve snaze způsobit, že webový prohlížeč uživatele spustí neautorizovaný a nefiltrovaný kód. Účelem tohoto testu je ověřit ovládací prvky vstupu od uživatele.

- Manipulace s parametry - odesílání upravených dotazů, parametrů a skrytých polí ve snaze získat neoprávněný přístup k uživatelským datům nebo funkcím aplikace.
- Otrava cookies - technika odesílání upravených cookie souborů za účelem testování reakce aplikace na příjem neočekávaných hodnot.
- Eskalace uživatelských oprávnění - snaha pokusit se získat neoprávněný přístup k oprávnění správce nebo jiných uživatelů v systému.
- Manipulace s přístupovými údaji - úprava identifikačních a autorizačních údajů ve snaze získat neoprávněný přístup k datům a aplikačním funkcím ostatních uživatelů.
- Násilné procházení - procházení souborů umístěných na webovém serveru ve snaze o přístup k souborům a uživatelským datům, které se uživateli výslovně nezobrazují v aplikačním rozhraní.
- Možnosti v ladění a zpětných přístupů - snaha identifikovat kód, který vývojáři zanechali pro účely ladění, což by potenciálně mohlo útočníkovi umožnit získat další úrovně přístupu.
- Konfigurace subversion - posouzení webových a aplikačních serverů z hlediska nesprávných konfigurací, které by mohly vytvořit potenciální vektory útoku.
- Pokus o útěk z Docker kontejnerů skrz známé způsoby, jako je například použití privilegovaného přepínače, zneužití schopností nebo známé zranitelnosti s veřejně dostupným škodlivým kódem.
- Pokus o přístup na interní Kubernetes API nebo jeho ETCD databázi.
- Pokus o získání přístupu nebo ohrožení aplikace běžící v jiným tenantovi v rámci edge site.
- Pokus o detekci jakékoli nesprávné konfigurace sítě, která by mohla ohrozit základní infrastrukturu.

Druhou kategorií testování bylo **Síťové penetrační testování**, jehož cílem bylo provést externí skenování na identifikaci nebo prolomení síťových a serverových hrozeb. Jednalo se především o části, které jsou veřejně dostupné, nebo je možné na ně zaútočit z lokální sítě jako například u CE zařízení umístěného v restauraci. Tato kategorie měla následující fáze:

1. Aktivní identifikace zařízení neboli objevování, při kterém testeři obdrží síťové rozsahy a pokusí se objevit všechna dostupná zařízení z venku.
2. Skenování hrozeb na nalezených zařízeních a jejich analýza.
3. Validace nalezených hrozeb z důvodu identifikace a odebrání pozitivních nálezů, které nesouvisí se systémem.
4. Zneužití systémů testery na základně pochopení rolí externích zařízení, potenciálních vztahů jejich důvěryhodnosti a zranitelnosti. Cílem je získat přístup k systému.
5. Pokračování ve zneužití při úspěšném dosažení přístupu u zranitelných hostů. Jedná se o pokus eskalace oprávnění zneužitých hostitelů. Cílem je získat přístup k senzitivním datům, jako jsou hashovaná hesla nebo autentizační tokeny, které následně umožňují přístup do uživatelské sítě CEC platformy a jejich dat.

Výsledky bezpečnostního testování

Výstupem penetračního testování byl více než 40stránkový dokument, popisující detailně všechny provedené akce a jejich případné hrozby. Celkem se jednalo o zhruba 200 testovacích scénářů. V rámci této práce nedává smysl procházet každé objevené CVE nebo díru v konkrétní verzi softwaru stejně jako přidávat tyto výsledky do příloh disertační práce. Důvodem je, že bezpečnostní skenování a odhalování hrozeb je kontinuální proces a nelze v určité chvíli říci, že systém je bezpečný a není potřeba ho sledovat (Felderer et al., 2016). Během několika týdnů tak mohou být výsledky a verze úplně jiné díky aktualizacím jednotlivých softwarových komponent. Šlo nám především o koncepční ověření vybraných komponent pro CEC platformu a případné porovnání s dalšími alternativami, které řeší bezpečnostní prvky jiným způsobem. Jedním z cílů této práce je multi-tenance a multifunkčnost, která již byla několikrát zmíněna v souvislosti s jejími nedostatky u existujících řešení 3.4.5. Šlo nám tedy především o izolaci síťovou, procesorovou a konfigurační uvnitř edge site.

Z výsledků testování nás nejvíce z pohledu této práce zajímá část oddělení virtuálních zdrojů, a to konkrétně Docker, jelikož další části jako síťová izolace pro EGW izolace neukázaly žádné větší nedostatky kromě konfiguračních chyb v podobě otevřených síťových portů apod. Právě u Docker jsme objevili největší problém v podobě CVE-2019-5736[3], které by samo o sobě nevadilo, jelikož CVE jsou běžnou součástí života jakékoli softwarové komponenty včetně kernelu. Ten například v roce 2019 zaznamenal 177 hrozeb (MITRE, 2019). Nicméně tato konkrétní bezpečnostní díra umožnila

[3] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-5736`

získat nejvyšší právo root v hostovském systému skrz libovolný kontejner. Jednoduše řečeno - jakýkoli uživatel, který by použil tento exploit[4], mohl komprimovat celý systém, zmocnit se certifikátů a pokračovat dále v útoku na další části CEC platformy. Samozřejmě každá takováto známá díra má obvykle dostupnou záplatu v řádech hodin. Nicméně tento fakt nás přinutil k zamyšlení, zda je tato klíčová vrstva kontejnerové virtualizace dostatečně izolovaná a nejsou dostupné nějaké jiné alternativy.

Při průzkumu relevantních pramenů jsme zjistili, že například jedním z vysoce rizikových útoků, které ovlivňují kontejnery, byl například Meltdown. V rámci (Lipp et al., 2018) byli autoři schopni se úspěšně dostat do Docker, LXC i OpenVZ. Tento útok umožnil protivníkovi získat informace o jádře hostitelského OS a všech ostatních kontejnerů běžících v systému. Šlo tak prakticky získat plný přístup k informacím ze všech kontejnerů, což by v případě naší platformy mělo katastrofální dopady. Dalším příkladem vážné hrozby pro kontejnery byl Spectre, který dokáže podvrhnout jinou aplikaci a získat přístup k jejich paměti, odkud lze dostat citlivé informace (Kocher et al., 2018). Podle (Sultan, Ahmad, & Dimitriou, 2019) je nutné se podívat na bezpečnost kontejnerů ze 4 hlavních pohledů:

- Ochrana kontejneru před aplikacemi uvnitř IT - v tomto případě se jedná o škodlivý software běžící uvnitř kontejneru, který může provádět vzdálenou exekuci, neautorizovaný přístup, vir, trojský kůň apod. Jedná se tedy o útok zevnitř kontejneru a nejčastěji nastává spuštěním kontejnerového image, který nebyl prověřen. Z toho důvodu je nutné dbát na skenování hrozeb kontejnerových image před jejich spuštěním. Tento pohled se tedy týká více uživatelů CEC platformy, jelikož není v jejím rámci sledovat nebo skenovat každou spuštěnou aplikaci a její image.

- Ochrana mezi kontejnery - v tomto případě se jedná o útok zevnitř nebo zvenčí na ostatní kontejnery v rámci hostitele. Například škodlivý kontejner může spotřebovat většinu zdrojů hostitele a tím učinit zbytek kontejnerů nepoužitelnými. Dále sem spadají i zmíněné útoky Meltdown nebo Spectre, a proto je nutné chránit kontejnery mezi sebou, aby o sobě vzájemně nevěděly, jako je tomu u standardních VM.

- Ochrana hostitele (a aplikací uvnitř) před kontejnery - v tomto případě se škodlivý kontejner snaží poškodit svého hostitele ať už konzumací všech zdrojů, nebo přístupem k senzitivním informacím. Řešením tohoto problému by bylo opět poskytnout stejnou izolaci, jako mají VM.

- Ochrana kontejnerů před hostitelem - tento způsob se týká důvěryhodnosti hostitele, kde je kontejner spouštěn. Tento problém v případě navržené platformy CEC

[4] volně přeloženo škodlivý kód

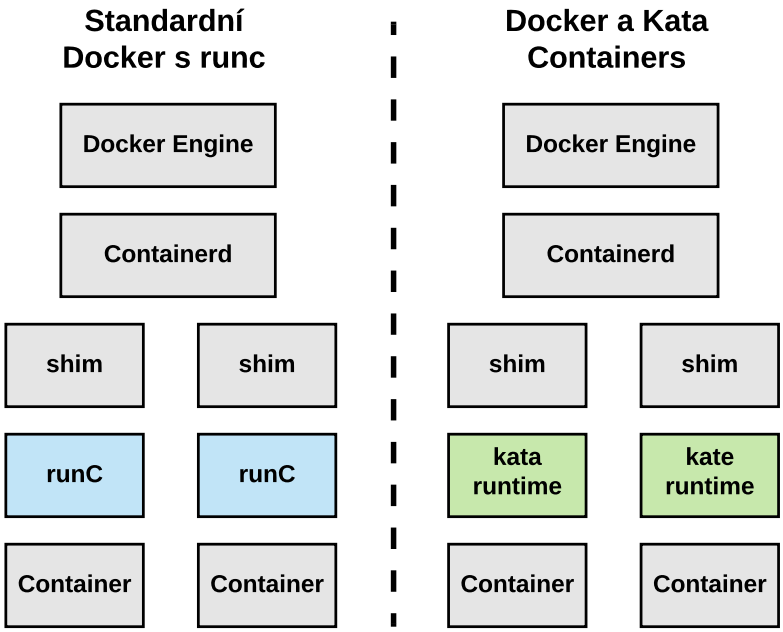
má velmi nízkou pravděpodobnost, jelikož by vyžadoval napadení orchestračního systému anebo OS v edge site. Nicméně tento pohled umožňuje získání senzitivních informací z běžících aplikací, nebo dokonce změnit jejich chování.

Z výše uvedeného je patrné, že ochrana mezi kontejnery a ochrana hostitele před kontejnery jsou nejvíce relevantní pro naši CEC platformu. Zmíněné problémy lze řešit pomocí softwaru nebo hardwaru nástrojů. Ochranný mechanismus hardwaru je cílený především na ochranu kontejnerů před hostitelem, která je pro CEC platformu až druhořadý problém. I přesto můžeme zmínit, že existují dva dostupné mechanizmy, kterými jsou vTPM a Intel SGX. Na softwarový mechanizmus ochrany spoléhá většina kontejnerových řešení. Jedná se o především o LFS, kam patří funkce Linux jádra jako Namespaces, CGroups nebo Seccomp.

Do této kategorie spadá i naše řešení s Docker, které jsme částečně představili již v kapitole 4.4.1 návrhu platformy, avšak nyní se zaměřme více na bezpečnostní aspekt. Podle (Ali Babar & Ramsey, 2017) nebyl Docker navržen primárně s myšlenkou bezpečnosti na prvním místě, avšak cílil na jednoduchost a použití různých uživatelů. Největší jeho problém vidí v tom, že veškeré operace s kontejnery jdou skrze Docker Engine, který musí běžet pod právy uživatele root. Problémem zůstává, že ostatní alternativy, jako je rkt nebo LXC, mají podobné bezpečnostní problémy anebo zaostávají z hlediska funkcionalit nutných pro provoz (Nanobox, 2017). V rámci (Bui, 2015) provedli detailní analýzu Docker a zjistili, že jeho multi-tenance je na akceptovatelné úrovni, avšak zdaleka nedosahuje stejně bezpečné izolace jako standardní virtuální servery.

Právě standardní virtuální servery VM se ukazují jako vzor dostatečné izolace a bezpečnostního mechanismu. VM jsou však oproti kontejnerům značně neefektivní, jelikož každá instance musí mít svoji vlastní kopii operačního systému. Tím pádem konzumují více zdrojů a jejich startovací doba je velmi pomalá oproti kontejnerům, které dokáží nastartovat i za 50 milisekund (Kaur, Dhand, Kumar, & Zeadally, 2017). Řešením do budoucna pro naši CEC platformu by tak mohlo být například použití Kata Containers, jehož cílem je spojit to nejlepší z obou světů neboli *"rychlost kontejnerů, bezpečnost VM"*[5]. Obrázek 5.5 ukazuje, jakým způsobem lze upravit standardní Docker za integraci s Kata Containers. To je možné díky jejich kompatibilitě se OCI standardem, kdy dochází k nahrazení výchozího runC, který používá tradiční LFS (CGroups, Namespaces, atd.) za kata-runtime, který spouští každý kontejner ve vlastním nenáročném VM. Tato technologie však existuje teprve od roku 2017, a proto je nutné provést detailní testování a porovnání parity funkcionalit v naší CEC platformě nebo jejich výkonnost.

[5] `https://katacontainers.io/`



Obrázek 5.5: Docker s Kata Containers runtime namísto výchozího runc, převzato z (*Kata Containers Architecture*, 2019)

Z hlediska bezpečnosti by bylo dobré projít i ostatní výsledky jednotlivých komponent, avšak rozsah této práce to neumožňuje, a proto byl rozebrán ten nejviditelnější problém. Závěrem bezpečnostního testování tedy lze říci, že platforma CEC splňuje základní prvky multi-tenance na úrovni běhu mikroslužeb díky použití Kubernetes a vlastnímu EGW CNI pluginu pro síťovou izolaci. Současně nemůžeme považovat její bezpečnost za optimální a budeme pokračovat v našem výzkumu nahrazení Docker runc například za zmíněné Kata Containers.

5.2.4 Výsledky technologického ověření

V rámci technologického ověření CEC platfromy jsme se zaměřili na výkonnostní, funkční a bezpečnostní ověření jejich přínosu. Vybraná ověření se vztahovala na některé cíle disertační práce a její technologickou část. Naším cílem bylo snížit dobu odezvy v rámci edge computingu jako takového. Zvolili jsme fyzickou a softwarovou optimalizaci, kdy přidáním další vrstvy do edge architektury oproti cloudlet nebo fog computing došlo k umístění consumer edge zařízení přímo ke klientovi nebo do jeho blízkosti, čímž logicky dochází ke snížení doby odezvy. Návrhem vlastní edge gateway postaveném na Intel DPDK jsme chtěli docílit větší síťové propustnosti a nižší doby odezvy i z pohledu

softwarové optimalizace. EGW je postavená na různých open source komponentách, jakými jsou Tungsen Fabric vRouter nebo Envoy proxy. Z tohoto důvodu jsme provedli statistické ověření doby odezvy standardního open source Envoy řešení s navrženou EGW obsahující integraci Envoy s Intel DPDK. Výsledkem bylo zpracování téměř 3krát většího počtu požadavků za vteřinu a rozdíl 1,2 ms v mediánu obou vzorků. Na pravděpodobnostní hladině 5% a pomocí Wilxcon rank sum testu jsme zamítli nulovou hypotézu a prokázali, že EGW má nižší dobu odezvy než dostupná open source řešení. Lze tedy říci, že CEC platforma jako taková snižuje dobu odezvy.

Dalším cílem bylo navrhnout standardizované API pro orchestrace edge aplikací, jež existují ve veřejných cloudech a datových centrech. API jsme v CEC platformě postavili na Kubernetes API, jenž patří prokazatelně ke standardu kontejnerové orchestrace poslední let. K dosažení tohoto cíle jsme navrhli a vytvořili vK8s komponentu pro distribuci Kubernetes objektů napříč edge site s Kubernetes API rozhraním dostupným pro uživatele. Abychom dokázali ověřit, že vK8s řešení splňuje požadavky standardu, využili jsme Kubernetes Conformance testovací sady pro certifikace distribucí různých výrobců. Výsledkem bylo 215 provedených testů, z nichž 5 skončilo neúspěšně. Po detailním rozebrání jednotlivých neúspěšných testů jsme zjistili, že pouze jeden z nich, a to týkající se Kubernetes CRD, nelze žádným způsobem provést v rámci dnešní implementace CEC platformy. Nicméně již v části návrhu bylo zmíněno, že tato funkce rozšířeného API nebude podporována. CEC platforma by tak nezískala oficiální certifikaci Kubernetes distribuce, avšak naplnila většinu funkčních požadavků. CEC platforma tak dokáže přistupovat k tisícům edge site stejným způsobem, jako je přistupováno k jednomu Kubernetes clusteru. Tento fakt umožňuje plynulý přechod aplikací z veřejného cloudu do zařízení běžících u koncových klientů bez velkých změn v jejich způsobu nasazení a její celkový dopad bude popsán dále v kapitole 5.3.

Posledním fází ověření jsme chtěli prokázat splnění multi-tenance v edge řešení a její multifunkčnost, tedy přechod z modelu klient-server na peer-to-peer[6] mezi consumer edge a centrálním cloudem. Multi-tenance je zajištěna izolací na různých vrstvách, jako je síťová, procesová, API atd. K jejímu prokázání je nutné ověřit bezpečnost a skutečnou separaci jednotlivých služeb. Z tohoto důvodu jsme se rozhodli pro penetrační testování a skenování bezpečnostních hrozeb CEC platformy. Provedli jsme celkem 200 testovacích scénářů a objevili největší mezery v bezpečnosti Docker runtime, konkrétně runC. Při průzkumu relevantních pramenů jsme dospěli k názoru, že ochranné mechanismy pro kontejnery jsou rozsáhlé téma nejen na akademické půdě. Detailním rozebráním jednotlivých pohledů na typy útoků zevnitř kontejneru, mezi kontejnery nebo z hostitele jsme dospěli k názoru, že bychom se měli zabývat nahrazením kontejnerového runtime. Kata Containers se ukazuje jako potenciální možnost

[6] volně přeloženo jako rovný s rovným

přivést do kontejnerového světa vysokou míru izolace ze použití plnohodnotné serverové virtualizace. Kromě tohoto problému nevykazovala platforma nějaké výraznější problémy a lze usoudit, že splňuje prvky slibované multi-tenance, které nejsou dnes dostupné u většiny IoT platforem představených v kapitole 3.4.5.

5.3 Stanovení přínosu pro podnikový management z pohledu softwarové výkonnosti

Softwarová akcelerace se stala jedním z klíčových kritérií pro ověření navržené CEC platformy. Podle (Humble et al., 2018) dochází v posledních letech k transformaci standardního vývoje produktů a služeb. Organizace ve všech odvětvích od finančního sektoru po maloobchod, telekomunikace a dokonce i státní správu se odvracejí od způsobu vývoje produktů a poskytování služeb skrz velké projekty s dlouhou dodací lhůtou. Namísto toho vytvářejí vytvářejí malé týmy, které pracují v krátkých cyklech s okamžitou zpětnou vazbou od zákazníka, čímž se jim daří zvyšovat jejich spokojenost a hodnotu. Tito tzv. „high performers" neboli vysoce výkonné organizace pracují na tom, aby se zlepšili v tom, co dělají a rychle odstranili všechny překážky. To je potvrzeno například i podle (Bessen, 2018), kde zjistili, že strategickým použitím technologií dochází k většímu růstu příjmů a produktivitě než v případě dlouhodobých spojení společností nebo jejich akvizicí. K tomu aby organizace zůstala konkurenceschopná a vynikala na trhu, je podle (Humble et al., 2018) nutné urychlit:

- dodávku zboží a služeb k uspokojení potřeb zákazníků,
- interakci s trhem za účelem flexibilní reakce na poptávku,
- implementaci nových předpisů a regulačních změn v odvětí, které dopadá na jejich systémy,
- reakci na možná rizika, jako jsou bezpečnostní hrozby nebo změny vývoje ekonomiky.

5.3.1 Metodika pro měření softwarové akcelerace

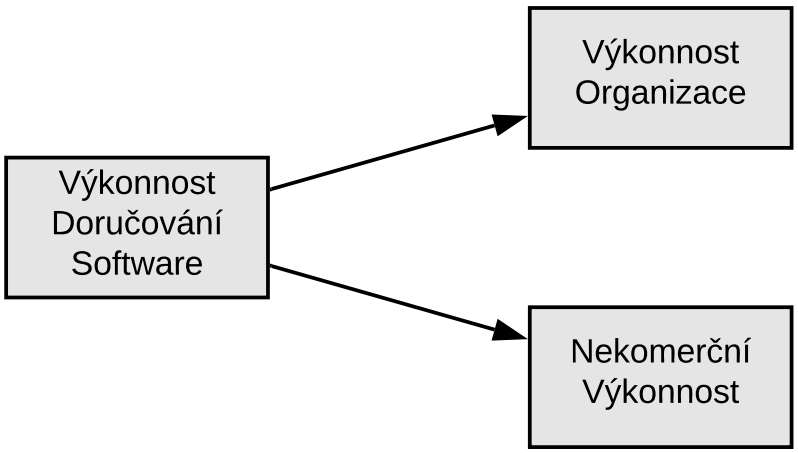
Srdcem této akcelerace se tak stává software a to platí pro organizace v jakémkoli odvětví. Banky již nevytváří hodnotu tím, že uchovávají zlaté pruty v trezorech, ale tím, že rychleji a bezpečněji obchodují a objevují nové prodejní kanály a produkty, kterými zaujmou zákazníky. Maloobchod získává a udržuje své zákazníky tím, že jim nabízí vynikající sortiment a služby, které umožňují sledování stavu objednávek v reálném čase

nebo bezproblémové nakupování online i offline. Toto vše je možné právě díky technologii. Vládní organizace uvádějí využívání technologií jako klíč k efektivnějšímu poskytování služeb veřejnosti a daňovým poplatníkům. Software a technologie jsou tedy klíčové k uspokojení potřeb svých zákazníků. Jejich vazba na produktivitu a příjmy společnosti byla prokázána ve výzkumech (Bessen, 2018), (McAfee & Brynjolfsson, 2008).

Hlavní inspirací pro měření inovací a akceleraci vývoje aplikací v rámci této práce se stal především čtyřletý průzkum (Humble et al., 2018), který se zaměřil na zkoumání toho, jaké schopnosti a postupy jsou důležité pro urychlení vývoje a dodávku softwaru. Následně pak je jejich výsledek promítnutý v ziskovosti, produktivitě a růstu podílu na trhu. Přímo tak byla prokázána vazba těchto hodnot na DevOps přístup, jelikož společnosti s tímto přístupem jsou dlouhodobě úspěšnější bez ohledu na oblast působení. DevOps se zrodil z několika společností, které čelily problémům: jak budovat bezpečné, odolné a rychle se vyvíjející distribuované systémy v obrovském měřítku (Erich, Amrit, & Daneva, 2017).

Celkem analyzovali 23000 vstupů od více než 2000 společností. Za účelem měření výkonnosti organizace byli respondenti průzkumu požádáni, aby ohodnotili relativní výkonnost organizace v několika dimenzích: ziskovost, podíl na trhu a produktivita. Tento způsob byl několikrát ověřen například v (Widener, 2007). Bylo také prokázáno, že míra organizační výkonnosti je vysoce korelována s mírou návratnosti investic (ROI). Analýza za několik let ukazuje, že výkonné organizace trvale dosahovaly svých cílů s dvakrát větší pravděpodobností oproti ostatním organizacím. Díky tomu prokázali, že výkonnost v oblasti softwaru poskytuje významnou konkurenční výhodu. Bylo také prokázáno, že ať už se organizace snaží dosáhnout zisku, nebo ne, vždy závisí na IT technologii, aby byla schopna dosáhnout svého poslání a poskytovala hodnotu svým zákazníkům rychle, spolehlivě a bezpečně. Ať už je tedy mise společnosti jakákoli, její technologická stránka předpovídá celkový výkon a její inovaci. Právě z toho důvodu jsme zapracovali jejich kritéria a metriky do hodnocení CEC platformy a mohli tak pozorovat zvýšení, nebo snížení softwarové výkonnosti.

Měření výkonu v oblasti softwaru je obtížné - částečně proto, že na rozdíl od výroby je inventář neviditelný. Nejdůležitějším faktorem podle (Humble et al., 2018) je výkonnost doručování softwaru (Software Delivery Performance). Vazba na výkonnost je znázorněna na obrázku 5.6.



Obrázek 5.6: Dopad výkonnosti doručování softwaru na výkonnost společnosti, převzato (Humble et al., 2018)

Existuje mnoho rámců a metodik, jejichž cílem je zlepšit způsob, jakým jsou poskytovány a vyvíjeny softwarové produkty a služby (Young, 2013). Většinou se jedná o metodiky projektového managementu pro vývoj softwaru, kam patří například dva nejznámější - Waterfall nebo Agile (Van Casteren, 2017). Naším cílem nebylo analyzovat celý proces projektového managementu pro vývoj softwaru, ale zaměřit se pouze na část týkající se způsobu doručování softwaru, a to především z praktického hlediska. Inspirovali jsme se proto několika průzkumy a metodami (Mann, Stahnke, Brown, & Kersten, 2018) (CircleCI, 2018). Pro naši případovou studii restauračního řetězce představenou v kapitole 5.3.2 bylo nutné upravit vybrané metody měření, jelikož v rámci zkoumání vlivu CEC platformy nemůžeme posuzovat celkovou výkonnost společnosti, protože zde působí další faktory jako například firemní kultura nebo vnitropodnikové procesy pro produktový design či development. Z těchto důvodů je cílem porovnávat především výsledky bez využití CEC platformy a s využitím.

Nakonec jsme se rozhodli vybrat následující metriky pro měření výkonnosti doručování softwaru, u kterých budeme zachycovat jejich hodnotu před využitím a po využití CEC platformy: time to production, frekvence nasazování aplikace, počet manuálních kroků během nasazení aplikace, průměrný čas obnovení služby. Jejich výběr byl proveden na základě několika nezávislých zdrojů (Mann et al., 2018) (CircleCI, 2018) a (Humble et al., 2018). Z těchto metrik jsme se potom rozhodli vytvořit vlastní jednotné skóre nazvané CDVelocity, které je popsáno v další části této kapitoly. V následujících odstavcích probereme jednotlivé metriky.

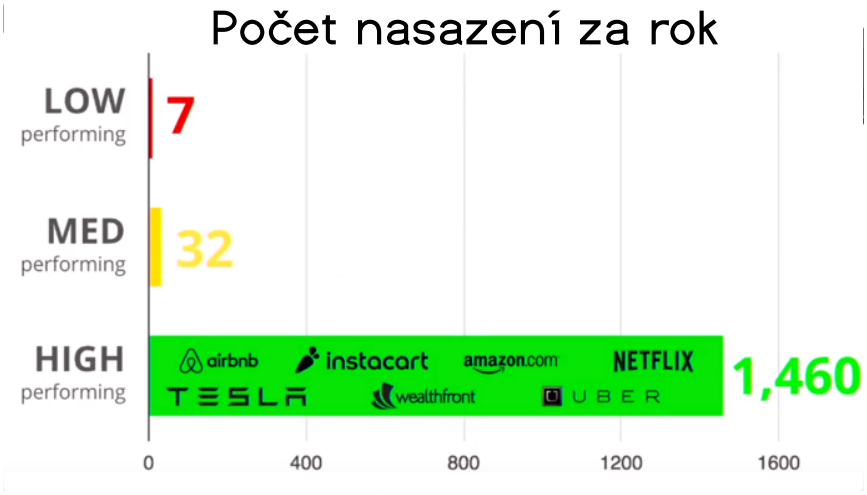
Obrázek 5.7: Myšlenka - Kód - Produkční kód, převzato z (Bello et al., 2018)

Jednou z klíčových metrik při měření výkonnosti bývá dodací lhůta[7] udávající dobu od zadání zákazníkova požadavku do doručení. Obecně lze říci, že kratší dodací lhůta je lepší z důvodu rychlejší zpětné vazby a případné změny směru vývoje produktu. Optimalizaci dodací lhůty také naznačuje obrázek 5.7, jenž cílí na zlepšení tří klíčových milníků v rámci vývoje - myšlenka, kód a produkční kód. Kratší dodací lhůta také umožňuje rychlejší opravení závad a chyb. V souvislosti s vývojem produktu je nutné zakomponovat požadavky více zákazníků, a proto se dodací lhůta rozděluje na dvě části: čas potřebný k návrhu produktu a čas samotné implementace a doručení. V první části je často velmi nejasné, kdy začít měřit, a současně je tato část nerelevantní pro naše ověření. V optimálním případě by měla být tato první část stejná, nehledě na použití nebo nepoužití CEC platformy. Z tohoto důvodu jsme zvolili metriku **Time to production** neboli jak dlouho trvá, než se kód od vývojáře objeví v produkci. Měří se od chvíle, kdy je kód nahrán do verzovacího systému, až do chvíle, kdy je kód nasazen v Consumer Edge.

Druhou metrikou, která nás zajímá, je velikost dávky[8]. Snížení velikosti dávek bylo jedním z ústředních prvků Lean paradigmatu, který byl jedním z klíčových úspěchů výrobního systému společnosti Toyota. Díky tomu bylo možné zkrátit dobu cyklu, variabilitu procesu, získat rychle zpětnou vazbu a celkově zvýšit výrobní efektivitu (Reinertsen, 2009). Ve světě softwaru je složité měřit velikost dávek, jelikož to není hmatatelné jako u výrobního procesu. Z tohoto důvodu jsme zvolili **Frekvenci nasazování aplikace**, protože se jednoduše měří a je jakýmsi zprostředkovatelem velikosti dávek.

[7] v angličtině lead time z metodiky štíhlé výroby
[8] v angličtině batch size

Nasazením je myšleno nasazení softwaru do consumer edge. Frekvence nasazování mohou být na vyžádání (několik nasazení denně), jednou za hodinu nebo den, jednou denně, jednou týdně apod. Obrázek 5.8 vychází ze studie společnosti Armory (Bello et al., 2018), která se snažila kategorizovat společnosti podle výkonnosti právě z hlediska počtu nasazení aplikací do produkce za jeden rok. Jak je vidět, nejprogresivnější jsou novodobé rychle rostoucí společnosti jako Tesla, Uber nebo Netflix, které využívají naplno výhod cloud computingu a neprovozují historickou infrastrukturu.



Obrázek 5.8: Počet nasazení do produkce ročně, převzato (Bello et al., 2018)

Čas doručení do produkce nebo frekvence nasazování jsou dva výkonnostní ukazatele. Nicméně důležité je porovnat také, zda zlepšení výkonnosti není na úkor stability celého systému. Obvykle je spolehlivost měřena dobou mezi výpadky, avšak v moderním vývoji a poskytování softwarových služeb dochází k rychlým změnám a komplexitě, při níž jsou výpadky nevyhnutelnou součástí. Díky tomu se tak otevírá spíše otázka: Jak rychle může být služba obnovena? Zajímá nás tedy **Průměrný čas obnovení služby** označovaný také jako **MTTR**. MTTR je ukazatel kvality, který vyjadřuje průměrný čas, za který je možné systém opravit nebo vyměnit. Vzorec 5.3 ukazuje způsob jeho výpočtu. V našem případě nás zajímá také počet kroků nutných k obnově po výpadku hlavní služby v edge lokalitě.

$$MTTR = \frac{\text{Celkový čas výpadků}}{\text{Počet výpadků}} \tag{5.3}$$

Poslední důležitou metrikou je **Počet manuálních kroků během nasazení aplikace**, který velmi často bývá podceňován. Například nasazení aplikace do serveru je prezen-

továno jako jeden krok, avšak ve skutečnosti musí systémový administrátor provést kroky tři - stáhnutí nové verze, aplikace změny a restart služby. Manuální kroky také zpomalují vývoj a zvyšují chybovost nasazení, a to z důvodu lidského faktoru. Nepřímo tak počet manuálních kroků souvisí s vysokou hodnotou MTTR, resp. nízkou spolehlivostí aplikace.

Pro účely porovnání a měřitelnosti jsme vytvořili vlastní **Confident Deployment Velocity** (CDVelocity) skóre (vzorec 5.4), které pomáhá kvantifikovat počet nasazení konkrétní aplikace za měsíc vyčíslený skrz její SLA. Cílem tohoto skóre je poskytnout jednu hodnotu pro podnikový management, podle kterého bude možné porovnávat změny v akceleraci doručení softwaru. SLA udává v tomto případě dostupnost dané aplikace nejčastěji specifikované počtem devítek, např. 99,9%. Nepřímo tak udává i důležitost dané aplikace pro společnost, čímž vyvažuje sílu skóre. Vyšší skóre CDVelocity přináší bezpečnější a stabilnější doručení softwaru a inovace. Velocity neboli frekvence je obecně bezpečnější přístup, protože doručuje aktualizace softwaru spíše po malých rozdílech namísto velkých a pomalých vývojových cyklů. Podle (Bello et al., 2018) manuální nasazení softwaru a nízká frekvence způsobují časté výpadky a současně také nízkou inovativnost softwaru, která má podíl nejen na rozvoji obchodu společnosti.

$$CDVelocity = \frac{\text{Počet nasazení aplikace měsíčně}}{(1 - \text{SLA Aplikace})} \tag{5.4}$$

Dosaďme si nyní reálný příklad 5.5, kdy se provede 10 nasazení nové verze aplikace měsíčně a požadovaná dostupnost SLA je 99,5%: Koeficient nebo skóre je v tomto případě 2000. Je vždy zajímavé srovnávat, o kolikrát se změní daný koeficient při zavedení nové technologie nebo úpravě procesu. Současně se podle něj dají také kategorizovat společnosti podle softwarové výkonnosti.

$$\frac{10}{(1 - 0{,}995)} = 2000 \tag{5.5}$$

5.3.2 Případová studie ABC

Implementace edge computing platformy je rozsáhlý projekt, který má několik fází od úvodních diskuzí o potřebách ověření přes pilotní nasazení až po přechod do produkce. Délka těchto fází k přechodu do produkce může být okolo jednoho roku. Z tohoto důvodu bylo velmi složité získat velký vzorek společností pro statistické ověření jejich softwarové výkonnosti při využití CEC platformy. Zároveň bylo naším cílem získat skutečná čísla a výsledky konkrétní společnosti, abychom mohli pokračovat v jejím rozvoji a měli zpětnou vazbu z komerčního sektoru. Z těchto důvodů jsme se rozhodli pro zpracování případové studie společnosti ABC. Tato společnost disponuje více než 2000 pobočkami restaurací po Spojených státech amerických a byla ideálním kandidátem na

ověření naší teorie. Autor práce přímo působil na pozici jednoho z hlavních architektů celé implementace platformy do společnosti ABC, jejíž jméno nemůže být z důvody ochrany obchodního tajemství zveřejněno, avšak všechna uvedená čísla jsou skutečná.

Společnost ABC provozuje síť restaurací s rychlým občerstvením ve Spojených státech amerických. V největších špičkách v průměru prodají jeden sendvič každých 16 vteřin, krabičku kuřecích nuget každých 25 vteřin nebo například jim projede jedno auto skrz car drive každých 22 vteřin. Kompletní řetězec má zhruba 2300 restaurací, 100000 IoT zařízení a miliardu MQTT zpráv za den. Jejich dlouhodobým cílem je: *„Zjednodušit procesy restaurace pro majitele / provozovatele a jejich týmy a dosáhnout tak skvělého, teplého, chutného, osobitého, servírovaného jídla rychle, a to vše zároveň při zvýšení kapacity."*

Trochu kontextu, proč byl vybrán právě řetězec rychlého občerstvení. Aniž bychom si to uvědomovali, vzniká nová generace aplikací běžící přímo okolo nás. Jako jednoduchý příklad si představme předpovědní model, který se snaží o prognózu, kolik hranolek by mělo být uvařeno za každou minutu v rámci dne. Prognóza je vytvořena analytickým procesem běžícím v cloudu, který využívá údaje o prodeji na úrovni transakcí z mnoha restaurací. Tuto prognózu lze s jistotou vyrobit bez větších problémů. Bohužel ale nebude dost přesná na to, aby se podle ní mohla skutečně řídit produkce potravin. Prodej v restauracích ABC je náchylný k mnoha různým špičkám návštěvnosti a je výrazně ovlivněn místními událostmi (provoz, sport, počasí atd.). Pokud by však dokázali shromažďovat tato data v reálném čase z lokálních prodejních systémů, bylo by možné pochopit současnou poptávku, zároveň korelovat data z údajů od fritéz o stavu nedokončené výroby bylo by možné upravit mikro předpověď přímo v restauraci. Ta by byla mnohem přesnější a bylo by možné podle ní v daném okamžiku vařit různá množství. Současně tato data mohou být použita k poskytování mnohem inteligentnějšího zobrazení pro členy týmu restaurace, který je například zodpovědný za hranolky, nebo dokonce k řízení automatického vaření v budoucnosti. Tyto důvody vedou společnost ABC k potřebě platformy pro IoT zařízení přímo v restauracích, která podpoří jejich byznys sběrem dat a současně povede k řízené automatizaci restaurace. Přesně takovéto potřeby byly ideální pro částečné ověření přínosu CEC platformy.

Hypotéza vedení společnosti ABC: *„Díky chytřejšímu kuchyňskému vybavení mohou sbírat více dat. Z těchto dat je možné vytvořit inteligentnější systémy. Budováním inteligentnějších systémů bude možné lépe škálovat jejich byznys."*

Cíle projektu definované společností ABC:

- Provoz kritických aplikací s nízkou latencí,
- Vysoká dostupnost těchto aplikací,
- Použití edge computingu umožňuje rapidní inovaci a doručování nových byznys funkcí do produkce co nejrychleji,

- Horizontální škálovatelnost infrastruktury a aplikačních vývojových týmů.

V této případové studii nás nejvíce zajímají výsledky měření právě zmíněné inovace a akcelerace doručování softwaru ve společnosti, které byly popsány na začátku této kapitoly. Technologické přínosy byly analyzovány zvlášť v kapitole 5.2. Z toho důvodu jsou technologické přínosy vynechány z výsledků měření případové studie.

5.3.3 Popis testování a nasazení platformy

Implementace do společnosti ABC byla rozdělena do několika fází, kdy nejprve probíhaly demo ukázky a sběr požadavků, následně probíhal PoC[9] na 2 vybraných restauracích a pilot na celkem 100 restauracích. Současně jsme pozorovali stejná kritéria i u 100 restaurací se stávajícím systémem, aby bylo možné vytvořit srovnání a získat přínos pro podnikový management. Během 6 měsíců jsme vyhodnocovali stanovená kritéria u třech vybraných aplikací. Před začátkem bylo nutné zanalyzovat typy aplikací běžící v restauraci a kategorizovat je do čtyř skupin:

- platformní služby - jedná se o autentizační službu na vydávání tokenů, pub/sub messsaging (MQTT), sběr logů a jejich filtrace (FluentBit), monitoring (Prometheus) atd.,
- integrační služby - aplikace interagující se s „věcmi" uvnitř restaurace,
- webové služby - jednoduché mikroslužby, které zpracovávají HTTP požadavky,
- služby pro modely strojového učení - syntetizují předpovědi zpracované v cloudu s událostmi v reálném čase z MQTT, aby se mohli rozhodovat přímo v restauraci a umožňovali automatizaci.

Ze třech skupin jsme následně vybrali reprezentativní typ aplikace pro účely našeho ověření. Ty jsou pro účely této práce pojmenovány obecně podle typu:

- HTTP aplikace - webová služba, která se používá na spouštění stopek pro dobu, kdy jsou jídla udržována v teplém stavu,
- IoT aplikace - aplikace pro sběr dat z fritéz údaje o spotřebě atd.,
- ML aplikace - aplikace strojového učení pro předpověď smaženého množství kuřete atd.

[9] z anglického Proof Of Concept - fáze ověření technologie

Je také nutné pochopit strukturu a odpovědnost za provoz různých komponent v rámci společnosti, abychom mohli zkoumat případné procesní změny. Společnost ABC má strukturované IT oddělení na 3 základní týmy:

- Infrastruktura - do infrastrukturního týmu zapadají víceméně všechny činnosti týkající se IT hardwarové infrastruktury jako fyzická správa zařízení v restauracích a jejich provoz,
- Platforma - tento tým provozuje veškerou softwaru IT část ve společnosti od cloudových aplikací přes interní systémy až po platformní služby běžící v restauracích,
- Vývoj - vývoj je rozdělen do dalších týmů, avšak pro účely našeho výzkumu vystupují jako jedna skupina. Jejich úkolem je vyvíjet stávající i nové aplikace nejen do restaurací, ale i podpůrné systémy pro dodávky surovin, CRM, ITSM apod.

5.3.4 Výsledky měření přínosu softwarové akcelerace

Tato podkapitola se zabývá analýzou výsledků přínosů akcelerace doručování softwaru pro společnost ABC. Nejprve porovnává výsledky metrik pro měření softwarové akcelerace na třech sledovaných aplikacích a následně analyzuje vliv změn na strukturu a některé procesy související s provozem těchto aplikací v restauracích.

Výsledky akcelerace aplikací

Jak již bylo řečeno měření probíhalo 6 měsíců na vzorku dvou stovek restaurací, kde jedna polovina běžela na CEC platformě a druhá ne. Pro měření softwarové akcelerace jsme použili metody stanovené v kapitole 5.3.1. První vzorek bez využití CEC dosáhl výsledků zachycených v tabulce 5.3.

Metrika	HTTP Aplikace	IoT Aplikace	ML Aplikace
Time to production (dny)	35 dní	118 dní	x
Frekvence nasazování za jeden měsíc	0,9 měsíčně	0,3 měsíčně	x
Počet výpadků	53	8	x
Průměrný čas obnovení služby v minutách	19,56603774	95,75	x
Počet manuálních kroků během nasazení aplikace	10	12	x
SLA	99,9%	99,8%	x
CDVelocity skóre	900	150	x

Tabulka 5.3: Výsledky jednotlivých metrik za 6 měsíců ze 100 restaurací bez použití CEC platformy, zdroj: vlastní tvorba

Druhý vzorek běžící na platformě CEC dosáhl výsledků shrnutých v tabulce 5.4. Podívejme se nyní na porovnání a důvody změny u jednotlivých aplikací.

Metrika	HTTP Aplikace	IoT Aplikace	ML Aplikace
Time to production (dny)	5 dní	20 dní	4 dny
Frekvence nasazování za jeden měsíc	3,8 měsíčně	1,9 měsíčně	9,8 měsíčně
Počet výpadků	20	20	60
Průměrný čas obnovení služby v minutách	8,5	60,8	58,6
Počet manuálních kroků během nasazení aplikace	4	5	2
SLA	99,9%	99,8%	99%
CDVelocity skóre	3800	950	980

Tabulka 5.4: Výsledky jednotlivých metrik za 6 měsíců ze 100 restaurací s použitím CEC platformy, zdroj: vlastní tvorba

V původní restauraci byla HTTP aplikace statickou stránkou, která se moc neměnila. Tabulka 5.5 zachycuje hodnoty bez využití a s využitím CEC platformy. Průměrná doba doručení změny byla okolo 35 dnů a měnila se téměř jednou měsíčně. Důvodem byl především proces i typ aplikace. Celý proces měl zhruba 10 manuálních kroků napříč třemi odděleními vývoje, platformy a infrastruktury. Vývojář nejprve nasadil verzi na vývojové prostředí, následně platformní tým zreplikoval aplikaci do staging prostředí.

V této části byla verze schválena produktovým týmem a po 2 týdnech schválena k nasazení do produkce. Verze byla následně přebrána a nasazena infrastrukturním týmem na sdílený mini server v restauraci pomocí konfiguračního managementu. Tento proces byl velmi rigidní a pomalý, protože měl závislost na dostupnosti ostatních týmů a nebylo možné dát vývoji přímý přístup k aplikaci běžící v restauraci. Právě proto při zavedení CEC platformy došlo k velkým změnám. Především se snížil počet kroků, a to výrazně díky automatizaci manuálních kroků při předávkách mezi třemi týmy a také se využily nástroje pro CI/CD, které již vývoj používal pro aplikace běžící v cloudu. Platformní tým vytvořil namespace v rámci edge platformy a dal přímý přístup vývoji ke správě a běhu aplikace v restauraci. Současně se tak mohla výrazně zkrátit doba nasazení do produkce, jelikož bylo možné rychleji a bez vazeb na další subjekty provádět aktualizace webu. Frekvence nasazování se tím zvýšila 4krát. Po dobu 6 měsíců jsme také naměřili celkem 53 výpadků napříč stovkou těchto restaurací, přičemž průměrná doba na obnovení služby byla 19,6 minut. Výpadky se pohybovaly od rozmezí 10-60 minut, kde důvodem byla nejčastěji chyba na serveru, která vyžadovala ruční restart nebo výměnu hardwaru. Podle záznamů z restaurace byl nejdelší interval detekce chyb a notifikace patřičné osoby, která provedla vzdálený nebo lokální restart serveru. SLA definované na 99,9% umožňuje maximální dobu výpadku na 43.2 minut za jeden měsíc. Z toho vyplývá, že je možné mít téměř 2 výpadky s MTTR v rámci jedné restaurace a stále dostát definovanému SLA. Porovnáme-li to s restauracemi běžícími na CEC platformě, lze vidět, že doba MTTR se zkrátila na třetinu na přibližně 8,5 minuty. Při hlubší analýze jsme zjistili, že díky automatické detekci chyb a přepnutí CE tříserverový cluster automaticky přemigroval aplikaci na jiný funkční server. Zároveň lepší monitoring upozornil okamžitě klíčové vlastníky aplikace a ti byli schopni hned vzdáleně zasáhnout. V tuto chvíli je tedy možné pro dodržení stávajícího SLA zaznamenat téměř 6 výpadků měsíčně. Náš CDVelocity koeficient definovaný vzorcem 5.4 se změnil z 900 na 3800, což je více než 4krát více, a tedy značný nárůst software produktivity v případě HTTP aplikace.

Metrika	HTTP bez CEC	HTTP s CEC
Time to production (dny)	35 dní	5 dní
Frekvence nasazování za jeden měsíc	0,9 měsíčně	3,8 měsíčně
Počet výpadků	53	20
Průměrný čas obnovení služby v minutách	19,56603774	8,5
Počet manuálních kroků během nasazení aplikace	10	4
SLA	99,9%	99,9%
CDVelocity skóre	900	3800

Tabulka 5.5: Porovnání výsledků akcelerace HTTP aplikace, zdroj: vlastní tvorba

IoT aplikace měla nejpomalejší cyklus doručení změn do restaurace, kdy se doba doručení do produkce pohybovala okolo 4 měsíců. Důvodem byl jednoduchý mechanismus sběru dat z fritéz, které není nutné příliš často měnit a v kombinaci rigidního procesu pro aktualizaci byla snaha systém měnit jen ve velmi důležitých situacích (např. bezpečnostní záplata). Současně také testovací fáze zahrnovala simulaci skutečného HW, v tomto případě fritézy. Jelikož byl proces výměny příliš rigidní a složitý, postupovalo se v dlouhých iteracích vývoje změn a celý proces měl zhruba 12 manuálních kroků. Aplikace procházela velmi podobným procesem skrz oddělení vývoje, platformy a infrastruktury, avšak rozšířený o 2 kroky ověření na skutečném zařízení ve staging prostředí. Při nasazení pak musel infrastrukturní tým provést o manuální krok navíc. Oproti HTTP aplikaci byla IoT aplikace tedy velmi statická a neumožňovala téměř žádný prostor pro inovaci či rozšíření, jelikož doba doručení aktualizací byla velmi dlouhá. Zavedením CEC platformy ve zkušebních restauracích došlo k dramatickému snížení času pro doručení změn a frekvenci jejich nasazení téměř 2krát měsíčně. Hlavním důvodem bylo přesunutí tohoto softwaru z jednoúčelového dedikovaného zařízení pro IoT aplikaci na CE cluster, který provozoval již HTTP aplikaci. Došlo tak vlastně k virtualizaci jednoúčelového fyzického serveru s procesem sběru dat do lehkého docker kontejneru. Tato změna umožnila naprosto stejný mechanismus doručování změn jako u HTTP aplikace a otevřela možnosti pro rozšíření statické aplikace například o dynamické funkce vzdáleného ovládání fritéz z dalších aplikací. Obě aplikace v tomto případě tedy sdílí HW, avšak díky multi-tenancy se tváří jako kompletně oddělené. Není tedy nutné spravovat specifický hardware pro každou takovou aplikaci, jelikož má svůj virtuální namespace v rámci CE clusteru s detailním nastavením práv a přístupů k nim. Po dobu 6 měsíců jsme také naměřili 8 výpadků napříč stov-

kou původních restaurací s dobou obnovení mezi 75-115 minutami. Průměrná doba výpadku byla okolo 95 minut. Nejčastějším důvodem výpadku byla HW chyba, a jelikož charakter aplikace byl spíše analytický a neovlivňoval tak přímo provoz samotné restaurace, i jeho SLA bylo o desetinu volnější tedy 99,8%. To umožňuje výpadek až 87 minut měsíčně, avšak dle získaných údajů o výpadcích je zřejmé, že SLA bylo v 5 restauracích nedodrženo. V restauracích s CEC platformou došlo k nárůstu výpadků o více než dvojnásobek na 20, a to z důvodu častějších aktualizací aplikace, čímž se zvýšila její chybovost. Zatímco v původním světě se aplikace téměř nikdo nedotýkal a její chyby byly způsobeny pouze kvůli chybě hardwaru nebo výpadku elektřiny, v novém světě docházelo k chybám softwaru z důvodu častějších změn. Důležitým ukazatelem je ovšem MTTR, které se snížilo zhruba o 35 minut v průměru na jednu hodinu. Reakce na softwarovou chybu mohla být tedy provedena v řádech minut namísto hodin či dní. Snížením došlo i k dodržením SLA u všech 20 případů výpadku. Díky snížení MTTR se také změnilo naše CDVelocity skóre z 150 na 950, což je více než 6krát, a tedy ještě větší poměrový nárůst softwarové produktivity v porovnáním s HTTP aplikací.

Metrika	**IoT bez CEC**	**IoT s CEC**
Time to production (dny)	118 dní	20 dní
Frekvence nasazování za jeden měsíc	0,3 měsíčně	1,9 měsíčně
Počet výpadků	8	20
Průměrný čas obnovení služby v minutách	95,75	60,8
Počet manuálních kroků během nasazení aplikace	12	5
SLA	99,8%	99,8%
CDVelocity skóre	150	950

Tabulka 5.6: Porovnání výsledků akcelerace IoT aplikace, zdroj: vlastní tvorba, zdroj: vlastní tvorba

U ML aplikace nelze porovnávat rozdíl bez využití a s využitím CEC platformy, jelikož ML aplikaci nebylo v současné době možné vůbec provozovat na existující infrastruktuře. To byl také jeden z důvodů, proč byl zájem o provedení pilotu na použití edge computingu u této společnosti. Každopádně když se podíváme na čísla, tak má nejrychlejší time to production. Hlavním důvodem je, že se jedná o dynamický vývoj, je nutné nasazovat nový kód velmi rychle, aby bylo možné získat zpětnou vazbu a reagovat na požadavky restaurací. Současně je to nový projekt, takže za sebou nemá dlouhou historii a bylo možné začít od nuly podle nejnovějších trendů vývoje cloud native, což

značně celý proces tohoto cyklu zjednodušuje. Současně SLA je velmi nízké, jelikož se jedná o novou aplikaci pro analýzu dat, a nemůže tedy její výpadek přímo ovlivnit chod celé restaurace. Je povolen výpadek maximálně 7,2 hodiny měsíčně. CDVelocity skóre je téměř shodné jako u IoT aplikace, tedy mezi 900-1000. I přes vysokou frekvenci nasazení její skóre zhoršuje nízké SLA, které je u HTTP aplikace nižší o 0,9%. Pokud by se ML aplikace zapracovala a stala se klíčovou aplikací jako HTTP, měla by skóre okolo 9 800. Taková frekvence nasazení je však při tak vysokém SLA velmi nepravděpodobná.

Metrika	**ML bez CEC**	**ML s CEC**
Time to production (dny)	x	4 dny
Frekvence nasazování za jeden měsíc	x	9,8 měsíčně
Počet výpadků	x	60
Průměrný čas obnovení služby v minutách	x	58,6
Počet manuálních kroků během nasazení aplikace	x	2
SLA	x	99%
CDVelocity skóre	x	980

Tabulka 5.7: Porovnání výsledků akcelerace ML aplikace, zdroj: vlastní tvorba

Podívejme se nyní na procentní nárůst, nebo pokles u jednotlivých kritérií při běhu s CEC platformou a bez ní zobrazené v tabulce 5.8. Toto porovnání lze provést pouze pro HTTP a IoT aplikaci, jelikož běžely v obou zkoumaných vzorcích. Je zajímavé, že Time to production se téměř shodně zkrátil pro dvě zcela nezávislé aplikace o přibližně 85%. Nárůst u frekvence nasazení je shodný jako pro CDVelocity skóre, jelikož je frekvence jeho základem. Obě tyto hodnoty výrazně stouply a zvýšily tak značně akceleraci doručení softwaru ve společnosti. Zajímavá metrika je procentní nárůst výpadků u IoT aplikace, který jsme již vysvětlili v předchozí části. Zde lze polemizovat, co je pro společnost lepší. Vyšší frekvence nasazení s nižší dobou obnovení MTTR, a to za cenu vyššího počtu kratších výpadků anebo pomalý neměnný proces s potenciálně velkými výpadky a nízkou mírou flexibility doručování softwaru. Z těchto výsledků je tedy vidět, jakým způsobem CEC platforma zlepšuje akceleraci vývoje a nepřímo tak i růst inovace a výkonnosti ve společnosti, jak bylo vysvětleno v kapitole 5.3.1.

Metrika	HTTP Aplikace	IoT Aplikace
Time to production	klesl o 85%	klesl o 83%
Frekvence nasazování	navýšení o 322%	navýšení 533%
Počet výpadků	snížen o 43%	zvýšen o 150%
Průměrný čas obnovení služby v minutách	klesl o 56%	klesl o 37%
Počet manuálních kroků během nasazení aplikace	klesl o 60%	klesl o 58%
CDVelocity skóre	navýšení o 322%	navýšení 533%

Tabulka 5.8: Procentuální rozdíly v metrikách po zavedení CEC platformy, zdroj: vlastní tvorba

Závěrem očekáváme, že v následujících 18–24 měsících dojde ke zvýšení počtu inteligentních a připojených zařízení v restauracích na více než 100 000 a dokončení implementace edge computing platformy do všech zbývajících restaurací. Počet případů pro použití umělé inteligence, které se budou ovládat v přímo restauraci, stále roste a je jasné, že růst této platformy bude nezbytný.

Vliv na strukturu a odpovědnost za provoz

Nyní se podívejme, jak vypadalo stávající řešení z pohledu odpovědnosti a správy aplikací. Každá restaurace byla tvořena několika hybridními zařízeními, která se spravovala nezávisle a většinou systémem od konkrétního výrobce. Každá restaurace tak měla malou skříň s různými řídicími jednotkami a počítači s jednolitými aplikacemi od různých výrobců. Problém byl v jejich správě, jelikož bylo jen velmi těžko možné je zpřístupnit různým vlastníkům ve společnosti. Tento model vyžadoval kompletní správu celé této části infrastrukturním týmem, jak bylo vysvětleno v kapitole 5.3.3. Bylo prakticky nemožné dát přístup do systému běžícího přímo v restauraci vývojářů, a to kromě možností sdílení obrazovky při výpadcích. Existovala tak oddělená organizační „sila" jednotlivých oddělení vývoje infrastruktury a platformy, které si předávají práci skrz service management systém a nejsou navzájem propojeny. Vývojář tak prakticky neví, co se děje v restauraci, a současně infrastrukturní provoz neví téměř žádné detaily o napsané aplikaci. Podobným mechanismem tzv. legacy infrastrukturního světa jsme se zabývali v našem výzkumu (Komarek, Sobeslav, & Pavlik, 2015) transformace ICT enterprise společnosti na agilní prostředí, kde jsme zaváděli metody konfiguračního managementu do standardů a rámců, jako jsou ITIL, TOGAF nebo COBIT.

Zavedením CEC platformy se změnila odpovědnost a proces řízení a doručování aplikací do restaurací. Tabulka 5.9 zachycuje tyto rozdíly bez a při využití CEC platformy na vzorku 100 restaurací pro jednotlivé kategorie aplikací ve společnosti před-

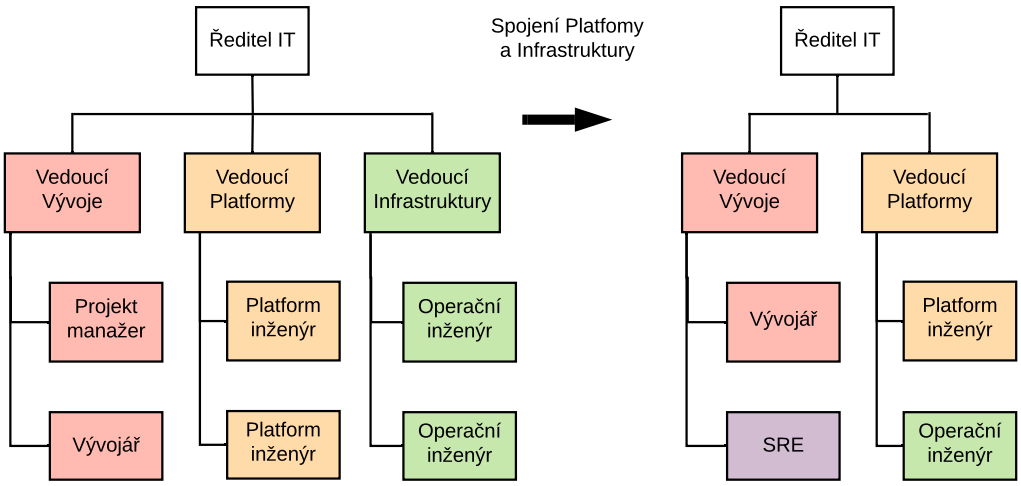
stavené v kapitole 5.3.3. Zajímavé je se podívat, že bez CEC platformy byla odpovědnost za provoz u všech aplikací běžících v restauracích na infrastrukturním týmu. Ten měl v některých případech výhradní přístup a byl zcela oddělený od vývoje. U aplikací běžících mimo restauraci, tedy ve veřejném cloudu nebo datovém centru, byly plně v odpovědnosti platformního týmu. S přechodem na CEC platformu lze pozorovat přesun odpovědnosti na vývojové týmy. V případě webových služeb a služeb strojového učení přešla odpovědnost plně na vývoj. Infrastrukturní nebo platformní tým se stal spíše podporou v případě výpadků hardware. Odpovědnost infrastrukturního týmu tak téměř zmizela, nebo přešla do vývoje, kde vznikla nová pracovní pozice SRE. Site Reliability Engineering je koncept, kdy vývojáři provozují svoje vlastní aplikace a přistupují obecně k provozu stejně jako k vývoji. SRE se stalo velmi moderním trendem v oblasti provozu softwaru po vzoru společnosti Google (Beyer, Jones, & Petoff, 2016). Implementovat tento koncept provozu do edge computingu, konkrétně CEC platformy, je možný právě díky multi-tenancy a multifunkčnosti zařízení, jež umožňují přistupovat a spravovat aplikace nezávislým a plně izolovaným týmům.

Odpovědnost za provoz	**Bez využití CEC**	**Při využití CEC**
Platformní služby	infrastruktura a platforma	platforma
Integrační služby	infrastruktura	platforma a vývoj
Webové služby	infrastruktura a platforma	vývoj
Služby strojového učení	platforma (pouze v cloudu)	vývoj

Tabulka 5.9: Porovnání odpovědnosti za provoz jednotlivých aplikací bez a s využitím CEC platformy, zdroj: vlastní tvorba

Z výše uvedeného vyplývá, že role infrastrukturního i platformního týmu se při využití CEC platformy značně překrývají a bylo by možné je částečně spojit a zjednodušit celou strukturu organizace. Potenciální změna této struktury je zachycena na obrázku 5.9, kde je vidět současná situace a situace v případě zavedení CEC platformy. Hlavním rozdílem je tedy přechod části operačních inženýrů infrastruktury pod platformu a zrušení vedoucí pozice infrastruktury společně s částí operačních inženýrů. Ve vývoji by vznikla úplně nová pozice Site Reliability Engineer, který je součástí vývojového týmu s orientací na provoz aplikace. Tento způsob provozu aplikací se ukázal jako velmi účinný v několika velkých společnostech (Beyer, Murphy, Rensin, Kawahara, & Thorne, 2018), jelikož zeštíhlil nebo zrušil nákladné operační týmy a přenesl zodpovědnost přímo do vývoje. To zvýšilo kvalitu a současně také došlo i ke snížení mzdových nákladů na provoz, jejichž prostředky se dají použít například na rozvoj IT. Samozřejmě ke konkrétním závěrům vlivu CEC platformy na náklady společnosti by bylo nutné zmapovat změny za delší období než jeden rok a také provést tento výzkum

na větším vzorku společností.



Obrázek 5.9: Potencionální změna organizační struktury IT oddělení při využití CEC platformy, zdroj: vlastní tvorba

5.4 Shrnutí přínosů a diskuze výsledků

Závěrem této kapitoly je důležité provést shrnutí dosažených přínosů a jejich diskuzi. Hlavním cílem disertační práce bylo navrhnout kompletní edge computing platformu, která dokáže efektivně provozovat globálně distribuované aplikace vyžadující nízkou dobu odezvy pomocí standardizovaného aplikačního rozhraní pro jejich orchestraci. Dílčí cíle v kapitole 2 byly stanoveny následovně:

1. Analyzovat a prozkoumat problematiku Edge Computingu.
2. Návrh řešení dílčích problémů Edge Computingu, které zahrnují následující body:
   - Optimalizace doby odezvy a nestabilita internetového připojení.
   - Multi-tenance v edge řešení a přechod od modelu klient-server k modelu multifunkčního edge.
   - Standardizované API pro orchestrace edge aplikací, jež existují ve veřejných cloudech a datových centrech.
   - Stanovení přínosu nové Edge Computing platformy pro podnikový management z hlediska výkonnosti doručování softwaru.

3. Návrh samotné Edge Computing platformy včetně jednotlivých modulů a její ověření z pohledu informačního managementu, tedy technického a podnikového přínosu v rámci případové studie nebo laboratorního měření.

První z dílčích cílů byl naplněn na základě analýzy současného stavu v kapitole 3, kde byla představena problematika a motivace k edge computingu včetně jeho využití pro internet věcí, rozšířenou realitu nebo optimalizace v rámci optimalizací médií a příchodem 5G sítí. Následně byly představeny dostupné standardy pro edge computing jakými jsou fog computing, MEC a cloudlet. Podkapitola 3.3 provedla detailní rešerši v oblasti vědeckých prací týkajících se problémů a příležitostí, tak aby bylo možné vydefinovat prostor pro výzkum této práce. Výstupem této části bylo vydefinovaných 5 nevyřešených problémů a 5 příležitostí, které tato oblast otevírá pro zkoumání. Ty byly v rámci podkapitoly 3.4 potvrzeny přehledem a porovnáním existujících řešení od výrobců jako Cisco, Microsoft atd. Na tomto základě byla v závěrečném shrnutí vysvětlena potřeba nového návrhu edge computing platformy, která vidí jako jeden ze základních problémů model se třemi vrstvami. Nevyřešené problémy a příležitosti byly přiřazeny k dílčím cílům vlastní navržené CEC platformy a shrnuty především v rámci tabulky 4.1. Kapitola 4 obsahovala samotný návrh CEC platformy strukturované právě s důrazem na tyto cíle namísto obecného popisu architektury a jejích komponent.

Rozeberme nyní jednotlivé dílčí definované problémy v cílech práce a jejich naplnění. **Optimalizace doby odezvy** byla řešena z pohledu fyzického a softwarového. Fyzická optimalizace je provedena právě zavedením dodatečné vrstvy CEC architektury, a to konkrétně CE site, běžící přímo v dané lokalitě s obsluhujícími klienty. Tento fakt prakticky sám o sobě snižuje odezvu bez ohledu na konkrétní technologii. Softwarový návrh optimalizace této části je vysvětlen v kapitole 4.3, hlavním klíčem je integrace existujících open source projektů s Intel DPDK. V rámci návrhu je vytvořena softwarová síťová edge gateway (EGW), jež pokrývá funkce vrstev L3-L7 modelu ISO-OSI a je základním kamenem každé edge site a distribuované Service Mesh, která řeší kompletní komunikaci v rámci celé platformy. Díky tomu dokáže poskytnout plnohodnotnou komunikaci IP-IP namísto pouze vybraných síťových protokolů ve vztahu klient-server. Jak již bylo zmíněno, je tvořena nejvýkonnějšími dostupnými open source nástroji, jako je Tungsten Fabric vRouter, Envoy proxy apod. Její přínos byl prokázán výkonnostním statistickým ověřením a porovnáním se standardní nemodifikovanou verzí Envoy proxy. Výsledkem bylo zpracování téměř 3krát většího počtu požadavků za vteřinu a rozdíl 1,2 ms mediánu obou vzorků. Provedenou statistickou metodou Wilxon rank sum testu (používanou také v autorských publikacích (Pavlik, Sobeslav, & Komarek, 2014) a (Pavlik, Sobeslav, & Horalek, 2014)) a na 5% hladině významnosti lze říci, že EGW je výkonnější než dostupné open source řešení používané pro distribuovanou proxy. Samozřejmě je tento způsob ověření pouze jedním ukazatelem a existuje celá

škála dalších možností pro testování latence jako například výkonnost z hlediska TLS offloading, optimalizace trasy mezi lokalitami nebo měření latence v rámci VPN spojení. Tato práce však nahlíží na Edge Computing jako celek, a proto bylo možné zvolit pouze jeden testovací scénář.

Multi-tenance v edge řešení a přechod od modelu klient-server k modelu multifunčkního edge bylo dalším z dílčích cílů této práce. Z analýzy současného stavu vyplynuly dvě příležitosti spadající do této kategorie, a to vytvoření obecných výpočetních nodů 3.3.1 a použití mikro operačních systémů a virtualizace 3.3.2. Navrhovaná CEC platforma tedy v kapitole 4.4 zvolila vhodnou kontejnerovou virtualizaci a způsob její orchestrace, kde bylo čerpáno především z autorských publikací (Mercl & Pavlik, 2019a), (Mercl & Pavlik, 2019b) na téma kontejnerových orchestrátorů a jejich úložišť. Pro vytvoření multifunkčního obecného návrhu bylo snahou přenést nejpoužívanější open source kontejnerovou orchestraci Kubernetes s Docker kontejnery do prostředí edge zařízení. Zároveň bylo nutné zajistit izolaci nejen na procesové úrovni, ale také síťové, a proto bylo nutné vytvořit důkladný návrh integrace s EGW. Celkový návrh tak není závislý na konkrétním hardwaru a celé řešení může být distribuováno pouze jako software. Dalším přínosem je především izolovanost jednotlivých kontejnerů v rámci tzv. namespace, čímž mohou běžet nezávisle různé aplikace uvnitř stejného zařízení. K ověření tohoto tvrzení bylo provedené penetrační testování v kapitole 5.2.3 s pokusy o útěk z Docker kontejnerů nebo získání přístupů k běžícím aplikacím v edge zařízení. To odhalilo některé bezpečnostní díry v konkrétních verzích softwaru a doporučilo použít pokročilejší izolaci v podobě výměny kontejnerového runtime z runC používané Dockerem například za Kata Containers, které se snaží poskytnout stejný typ emulace jako u standardní serverové virtualizace. Tato iniciativa bude předmětem budoucího rozvoje platformy a bude nutné provést důkladné ověření a porovnání jednotlivých kontejnerových runtime. Jelikož multi-funkční návrh edge je základní stavební kámen celé CEC platformy, jeho hlavní přínos byl ověřen společně s testováním standardizovaného API pro orchestrace a přínosu pro doručování výkonnosti softwaru popsaném níže.

Vytvořit a objasnit přínos **Standardizovaného API pro Orchestrace edge aplikací** bylo dalším dílčím cílem této práce, který navazoval na příležitosti s definicí standardů 3.3.2 a objevování běžících služeb v edge zařízení 3.3.1. Nejprve bylo definováno, co je myšleno standardním API v prostředí cloud computingu, jenž úzce souvisí s výběrem kontejnerového orchestrátoru pro edge. Cílem bylo poskytnout stejný přístup orchestrace do stovek či tisíců edge lokalit jako v případě jednoho datového centra nebo cloudového regionu. Díky tomu je možné zrychlit adaptaci aplikací do edge, jelikož není nutné provádět jejich výrazné změny. Po analýze dostupných řešení na orchestraci do Kubernetes bylo navrženo vlastní řešení vK8s emulující standardní Kubernetes API

v rámci kapitoly 4.5. K ověření tohoto tvrzení bylo využito testovací sady pro certifikaci různých Kubernetes distribucí. Bylo provedeno 215 funkčních testů, z nichž 5 skončilo neúspěšně. Po detailním rozebrání jednotlivých neúspěšných testů bylo možné konstatovat naplnění tohoto standardu řešením vK8s (Kapitola 5.2.2). Tento fakt potvrdil, že je možné přenášet aplikace mezi cloudem a edgem bez větších problémů. Tento přínos byl viditelný především v části zlepšení výkonnosti doručování softwaru uvnitř podniku.

Posledním dílčím cílem disertační práce bylo **Stanovení přínosu edge computing platformy pro podnikový management z hlediska výkonnosti doručování softwaru** neboli jaký přínos může mít CEC platforma pro podnikový management. Jak bylo vysvětleno v kapitole 5.3, výkonnost doručování softwaru má podle výzkumu (Humble et al., 2018) vliv na výkonnost celé organizace, což nepřímo ovlivňuje její růst a inovaci. Na tomto základě bylo tedy postaveno měření přínosu, kdy měla CEC platforma poskytnout větší akceleraci pro společnosti skrz využití moderního přístupu DevOps využívaného především ve světě cloud computingu. V rámci případové studie restauračního řetězce ABC provedené autorem práce jsme vydefinovali 5 metrik pro měření doručování softwarové akcelerace a ty jsme porovnávali na vzorku 100 poboček bez nasazení a s nasazením CEC platformy po dobu 6 měsíců (Kapitola 5.3.4). Na 2 vybraných konkrétních aplikacích běžících v restauracích byl vidět signifikantní nárůst ve zrychlení doručení softwaru a to především v Time to production shodně o 85%. Organizace se rázem podle rozdělení v (Bello et al., 2018) dostala z kategorie Low Performers do High Performers, tedy vysoce výkonné. Ovšem rychlejší doručení mělo i negativní dopady, a to konkrétně u vybrané IoT aplikace, kde se zvýšila chybovost a i počet výpadků. Nicméně zkrátila se doba těchto výpadků v průměru o 37%, což by v delším horizontu při zlepšení testování aplikace mohlo vést ke snížení počtu výpadků. I přestože se jednalo pouze o jednu konkrétní organizaci a o velmi krátký časový interval 6 měsíců, lze považovat CEC platformu za technologii zvyšující výkonnost v doručování softwaru ve světě Edge Computingu.

Kromě softwarové akcelerace bylo také sledováno v rámci kapitoly 5.3.4, jakým způsobem se mohou měnit role, pozice a struktura uvnitř společnosti s nasazením CEC platformy. V organizacích s podobným případem použití existují dnes pevné hranice mezi odděleními vývoje a provozu těchto aplikací. Stává se tak, že vývojář nedostává zpětnou vazbu a neví, jak se jeho aplikace chová v reálném provozu, což značně komplikuje a zpomaluje celý proces. Důvody jsou především v jednoúčelovosti koncových zařízení a nedostupných nástrojů pro řízení přístupu různých pracovních oddělení. Díky multifunkčnosti a multi-tenanci CEC platformy došlo u společnosti ABC k přenesení odpovědnosti za provoz směrem k vývoji a umožnilo tak provozovat aplikace běžící v restauracích přímo vývojářům. Tomuto trendu se říká Site Reliability Engineering a ten

ruší klasická oddělení provozu a vytváří nové pozice přímo v týmech vývojářů. Podle (Beyer et al., 2016) je tak možné snížit náklady na provoz IT systémů a přenést je do jejího rozvoje. CEC platforma se tak může stát hybatelem implementace DevOps a SRE přístupu do Edge Computingu.

Celkově tak v rámci této práce byl **navržen nový typ consumer edge computing platformy** s konkrétní implementací a ověřením jak technologického přínosu, tak přínosu pro akceleraci doručování softwaru v podnicích na případové studii restauračního řetězce. To vše bylo možné provést díky přidání čtvrté vrstvy ke standardní edge computing architektuře v podobě vytvoření multifunkční, multi-tenantní softwarové CE lokality. Autor této práce provedl její kompletní návrh, ověření a části implementace v rámci svého působení na pozici jednoho z hlavních architektů v nadnárodní americké společnosti. Předepsaný rozsah práce bohužel neumožňuje do detailů popsat veškeré aspekty platformy, a proto musely být některé části nastíněné pouze okrajově. V budoucnu se chceme zaměřit na identitu, autorizaci a autentizaci koncových aplikací, způsoby úložiště persistentních dat v edge computingu nebo rozšíření aplikačního firewallu. Velmi zajímavá je také otázka nahrazení Docker runtime za pokročilejší mechanismus izolace nebo statistická analýza anomálií v síťovém provozu mezi lokalitami a detekce útoků. Dále tato práce vůbec neobsáhla části sběru metrických dat, jejich monitorování a notifikace v případě výpadků jednotlivých komponent. Edge Computing je stále velmi nová doména a umožňuje tak široké možnosti v oblasti výzkumu.

6 Závěr

Tato disertační práce měla jako hlavní cíl navržení edge computing platformy, která umožní efektivně provozovat globálně distribuované aplikace vyžadující nízkou dobu odezvy pomocí standardizovaného aplikačního rozhraní pro jejich orchestraci. Edge computing je velmi aktuální téma především v souvislosti s příchodem internetu věcí a 5G sítí, jež připojují k síti stále více zařízení produkující data. Hlavní a dílčí cíle práce vzešly z detailně provedené rešerše vědeckých pramenů a jejich zkoumání na dostupných komerčních řešení od výrobců jako Cisco, Microsoft nebo Amazon.

Hlavním přínosem této práce bylo rozšíření standardní edge computing architektury o další vlastní vrstvu nazvanou consumer edge computing (CEC). Přidáním čtvrté vrstvy v podobě malého nenáročného datového centra umístěného přímo u koncového uživatele (domácnost, maloobchodní prodejna apod.) bylo možné optimalizovat dobu odezvy jednak díky bližšímu fyzickému umístění, ale také softwarovou optimalizací Intel DPDK prostřednictvím vlastní edge brány (EGW). Tyto výsledky byly ověřeny statistickou analýzou výkonnostního porovnání se standardním dostupným open source řešením. Spojením EGW s kontejnerovou virtualizací a její orchestrací bylo navrženo multifunkční a multi-tenantní edge zařízení, jež není závislé na konkrétním hardwaru a umožňuje provozovat stejný typ aplikací jako v prostředí běžného cloud computingu. Výrazně se tak CEC platforma odlišuje od všech dostupných jednoúčelových nebo komplexních řešení na trhu představených v této práci. Její bezpečnost a funkčnost byla ověřena komerčním a penetračním testováním zaměřujícím se právě na izolaci a orchestraci aplikací. Následným návrhem a implementací standardizovaného aplikačního rozhraní pro orchestraci edge aplikací bylo prokázáno zvýšení výkonnosti doručování softwaru v podniku, což je významným přínosem pro podnikový management z hlediska výkonnosti a plnění podnikových cílů. Výběr a návrh těchto komponent byl podpořen několika autorskými publikacemi.

Výsledky této práce také prokázaly, že využití této platformy může potenciálně pozměnit organizační strukturu IT oddělení pomocí změny odpovědnosti za provoz aplikací běžících v koncových lokalitách (například restaurace, nemocnice atd.). CEC platforma se tak může stát hybatelem implementace DevOps a SRE přístupu ve světě edge computingu. Autor této práce během 4 let výzkumu působil na pozici jednoho z hlavních architektů v nadnárodních společnostech Mirantis a Volterra, kde postupně pra-

coval na kompletním návrhu, ověření a implementaci částí navržené Consumer Edge Computing platformy. Díky tomu mohl provést ověření v reálném prostředí restauračního řetězce ve Spojených státech amerických a ukázat i její aplikovaný přínos v praxi. Podrobnější shrnutí přínosů a diskuze o naplnění cílů této práce je provedeno v kapitole 5.4.

Tato disertační práce nemohla z důvodu určeného rozsahu detailním způsobem projít a ověřit všechny části edge computing platformy a jejích souvisejících služeb. Detailnější informace je proto možné nalézt v citovaných zdrojích, publikacích a projektech, na kterých se autor dlouhodobě podílí.

Literatura

Aazam, M., & Huh, E. (2014, 08). Fog computing and smart gateway based communication for cloud of things. In (p. 464-470). doi: 10.1109/FiCloud.2014.83

Abe, M., & Fujisaki, E. (2006, 10). How to date blind signatures. In (p. 244-251). doi: 10.1007/BFb0034851

Aecc: General principle and vision. (2018). Retrieved 2019-02-02, from `https://aecc.org/wp-content/uploads/2019/04/AECC_White_Paper_v2.1_003.pdf`

Ahmed, E., Gani, A., Khurram Khan, M., Buyya, R., & Khan, S. U. (2015). Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges. *Journal of Network and Computer Applications*, *52*, 154-172. doi: 10.1016/j.jnca.2015.03.001

Ahmed, E., Gani, A., Sookhak, M., Hamid, S. H. A., & Xia, F. (2015). Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges. *Journal of Network and Computer Applications*, *52*, 52-68. doi: 10.1016/j.jnca.2015.02.003

Ahmed, E., & Rehmani, M. H. (2017). Mobile edge computing: Opportunities, solutions, and challenges. *Future Generation Computer Systems*, *70*, 59-63. doi: 10.1016/j.future.2016.09.015

Aijaz, A., & Sooriyabandara, M. (2019, Feb). The tactile internet for industries: A review. *Proceedings of the IEEE*, *107*(2), 414-435. doi: 10.1109/JPROC.2018.2878265

Ali Babar, M., & Ramsey, B. (2017, 01). Understanding container isolation mechanisms for building security-sensitive private cloud.. doi: 10.13140/RG.2.2.34040.85769

Amadeo, M., Campolo, C., Molinaro, A., & Ruggeri, G. (2018, May). Iot data processing at the edge with named data networking. In *European wireless 2018; 24th european wireless conference* (p. 1-6).

Amazon web services (aws) - cloud computing services. (2019). Retrieved 2019-09-09, from `https://aws.amazon.com/`

Andrus, J., Dall, C., Hof, A., Laadan, O., & Nieh, J. (2011, 10). Cells: A virtual mobile smartphone architecture. In (p. 173-187). doi: 10.1145/2043556.2043574

App engine – build scalable web & mobile backends in any language. (2019). Retrieved 2019-09-09, from `https://cloud.google.com/appengine/`

Ashton, K. (2019). *That 'internet of things' thing - 2009-06-22 - page 1 - rfid journal.* Retrieved 2019-01-09, from https://www.rfidjournal.com/articles/view?4986

Awada, U. (2018, 06). Application-container orchestration tools and platform-as-a-service clouds: A survey.. Retrieved from https://www.researchgate.net/publication/325737011_Application-Container_Orchestration_Tools_and_Platform-as-a-Service_Clouds_A_Survey

Aws iot documentation. (2018). Retrieved 2019-01-02, from https://aws.amazon.com/documentation/iot/

Aws vpc networking. (2018). Retrieved 2019-01-02, from https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Networking.html

Azure iot suite. (2018). Retrieved 2019-01-02, from https://opbuildstorageprod.blob.core.windows.net/output-pdf-files/en-us/Azure.azure-documents/live/iot-suite.pdf

Azure networking concepts. (2018). Retrieved 2019-01-02, from https://docs.microsoft.com/en-us/azure/networking/networking-overview

Bachhuber, C., Martinez, A., Pries, R., Eger, S., & Steinbach, E. (2019, 09). Edge cloud-based augmented reality. In (p. 1-6). doi: 10.1109/MMSP.2019.8901715

Bahl, V. (2015). *Micro datacenter middleware for mobile computing.* http://research.microsoft.com/en-us/um/people/bahl/Present/Bahl_mDC_Keynote_May.pdf. Microsoft.

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016, 05). Microservices architecture enables devops: an experience report on migration to a cloud-native architecture. *IEEE Software*, *33*, 1-1. doi: 10.1109/MS.2016.64

Barker, A., Varghese, B., Ward, J., & Sommerville, I. (2014, 06). Academic cloud computing research: Five pitfalls and five opportunities..

Baset, S. (2012, 07). Cloud slas: present and future. *Operating Systems Review - SIGOPS*, *46*. doi: 10.1145/2331576.2331586

Baset, S., Silva, M., & Wakou, N. (2017, 04). Spec cloud™ iaas 2016 benchmark. In (p. 423-423). doi: 10.1145/3030207.3053675

Bavithra, G., Mahalakshmi, V., & Suganya, R. (2018, 1). An review on firewall and its attacks. *International Journal of Advanced Research in Computer and Communication Engineering*, *7*. doi: 10.17148/IJARCCE.2018.7145

Beck, M., & Maier, M. (2014a). *Mobile edge computing: Challenges for future virtual network embedding algorithms.* Retrieved 2019-02-02, from http://research.microsoft.com/en-us/um/people/bahl/Present/Bahl_mDC_Keynote_May.pdf

Beck, M., & Maier, M. (2014b). Mobile edge computing: Challenges for future virtual

network embedding algorithms..

Beck, M., Werner, M., Feld, S., & Schimper, T. (2014, 01). Mobile edge computing: A taxonomy..

Bello, A., Mosquera, I., & Rogers, E. (2018). *How spinnaker enables fast, repeatable and safe multi-cloud deployments.* Armory. Retrieved 2019-02-02, from `https://armory.io`

Bessen, J. (2018, 01). Automation and jobs: When technology boosts employment. *SSRN Electronic Journal*. doi: 10.2139/ssrn.2935003

Beyer, B., Jones, C., & Petoff, J. (2016). *Site reliability engineering: How google runs production systems* (1st ed.). O'Reilly Media.

Beyer, B., Murphy, N. R., Rensin, D. K., Kawahara, K., & Thorne, S. (2018). *The site reliability workbook: Practical ways to implement sre*. O'Reilly Media, Inc.

Bezemer, C.-P., & Zaidman, A. (2010, 12). Multi-tenant saas applications: Maintenance dream or nightmare? position paper abstract. In (p. 88-92). doi: 10.1145/1862372.1862393

Bi, H., & Wang, Z.-H. (2016, 01). Dpdk-based improvement of packet forwarding. *ITM Web of Conferences*, *7*, 01009. doi: 10.1051/itmconf/20160701009

Bonner, E., O'Raw, J., & Curran, K. (2011, 08). Implementing the payment card industry (pci) data security standard (dss). *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, *9*. doi: 10.12928/telkomnika.v9i2.709

Bonomi, F., & Milito, R. (2012, 08). Fog computing and its role in the internet of things. *Proceedings of the MCC workshop on Mobile Cloud Computing*. doi: 10.1145/2342509.2342513

Bonomi, F., Milito, R., Natarajan, P., & Zhu, J. (2014, 03). Fog computing: A platform for internet of things and analytics.. doi: 10.1007/978-3-319-05029-4_7

Borcoci, E. (2018). *Fog computing, mobile edge computing, cloudlets – which one?* Retrieved 2019-01-02, from `https://www.iaria.org/conferences2016/filesICSNC16/Softnet2016_Tutorial_Fog-MEC-Cloudlets-E.Borcoci-v1.1.pdf`

Botta, A., de Donato, W., Persico, V., & Pescape, A. (2014). On the integration of cloud computing and internet of things. *2014 International Conference on Future Internet of Things and Cloud*. doi: 10.1109/ficloud.2014.14

Bouchard, M. (2019). *The state of container and kubernetes security.* StackRox. Retrieved 2019-12-02, from `https://security.stackrox.com/rs/219-UEH-533/images/StackRox-Report-State_of_Container_and_Kubernetes_Security_Spring_2019.pdf`

Bouchenak, S., Chockler, G., Chockler, H., Gheorghe, G., Santos, N., & Shraer, A. (2013, 07). Verifying cloud services: Present and future. *ACM SIGOPS Operating Systems*

Review, *47*, 6-19. doi: 10.1145/2506164.2506167

Bui, T. (2015, 01). Analysis of docker security..

Bunyakiati, P., & Sammapun, U. (2019, 11). On secret management and handling in mobile application development life cycle: A position paper. In (p. 77-80). doi: 10.1109/ASEW.2019.00033

Cach, J. (2019). *Implementace hybridního multi-cloud konceptu pro běh distribuovaných aplikací v kontejnerech a virtuálních serverech do enterprise prostředí* (Master's thesis). https://theses.cz/id/pkz0ug/.

Candiwan, C. (2014, 11). Analysis of iso27001 implementation for enterprises and smes in indonesia..

Casalicchio, E. (2019, 01). Container orchestration: A survey. In (p. 221-235). doi: 10.1007/978-3-319-92378-9_14

Chandramouli, R., & Butcher, Z. (2020, 01). *Building secure microservices-based applications using service-mesh architecture.* doi: 10.6028/NIST.SP.800-204A-draft

Chaves, S., Uriarte, R. B., & Westphall, C. (2011, 12). Toward an architecture for monitoring private clouds. *IEEE Communications Magazine*, *49*, 130-137. doi: 10.1109/MCOM.2011.6094017

Chen, S., Wang, Y., & Pedram, M. (2013). A semi-markovian decision process based control method for offloading tasks from mobile devices to the cloud. *2013 IEEE Global Communications Conference (GLOBECOM)*. doi: 10.1109/glocom.2013.6831512

Chen, S., Xu, H., Liu, D., Hu, B., & Wang, H. (2014). A vision of iot: Applications, challenges, and opportunities with china perspective. *IEEE Internet of Things Journal*, *1*(4), 349-359. doi: 10.1109/jiot.2014.2337336

Chen, W., & Deelman, E. (2012, 05). Integration of workflow partitioning and resource provisioning. *Proceedings - 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012*. doi: 10.1109/CCGrid.2012.57

Chen, X., Jiao, L., Li, W., & Fu, X. (2016). Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, *24*(5), 2795-2808. doi: 10.1109/tnet.2015.2487344

CircleCI. (2018). *Three critical development metrics for engineering velocity.* Retrieved 2019-12-02, from https://circleci.com/resources/velocity-report/

The cisco iot system. (2018). Retrieved 2019-01-02, from https://www.cisco.com/c/dam/en/us/products/collateral/wireless/industrial-wireless-3700-series/at-a-glance-c45-734164.pdf

Clinch, S., Harkes, J., Friday, A., Davies, N., & Satyanarayanan, M. (2012). How close is close enough? understanding the role of cloudlets in supporting display appropriation by mobile users. *2012 IEEE International Conference on Pervasive Computing*

and Communications. doi: 10.1109/percom.2012.6199858

Cloudflare. (2019). *What is a reverse proxy? | proxy servers explained.* Retrieved 2019-01-02, from https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/

Cloud native continuous deliver. (2019). Retrieved 2019-05-02, from https://www.spinnaker.io/

CNCF. (2020). *Cncf k8s conformance.* Retrieved 2019-12-02, from https://github.com/cncf/k8s-conformance

Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010, 09). Benchmarking cloud serving systems with ycsb. In (p. 143-154). doi: 10.1145/1807128.1807152

Corcoran, P., & Datta, S. K. (2016). Mobile-edge computing and the internet of things for consumers: Extending cloud computing and services to the edge of the network. *IEEE Consumer Electronics Magazine*, *5*(4), 73-74. doi: 10.1109/mce.2016.2590099

Dastjerdi, A. V., & Buyya, R. (2016). Fog computing: Helping the internet of things realize its potential. *Computer*, *49*(8), 112-116. doi: 10.1109/mc.2016.245

Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P., ... Wenger, K. (2014, 10). Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, *46*. doi: 10.1016/j.future.2014.10.008

Desertot, M., Escoffier, C., Lalanda, P., & Donsez, D. (2006). Autonomic management of edge servers. *Self-Organizing Systems*, 216-229. doi: 10.1007/11822035_18

Dey, S., Mukherjee, A., Paul, H. S., & Pal, A. (2013). Challenges of using edge devices in iot computation grids. *2013 International Conference on Parallel and Distributed Systems*. doi: 10.1109/icpads.2013.101

Edge computing market size, share & trends analysis report by component (services, hardware, software, edge-managed platforms), by end use (transportation & logistics, retail, datacenters, wearables), and segment forecasts, 2019 - 2025. (2019). Retrieved 2019-10-02, from https://www.grandviewresearch.com/industry-analysis/edge-computing-market

Elijah. (2017). *Cloudlet-based edge computing.* Retrieved 2019-02-02, from http://elijah.cs.cmu.edu/

Envoy proxy. (2018). Retrieved 2019-12-02, from https://www.envoyproxy.io/

Erich, F., Amrit, C., & Daneva, M. (2017, 06). A qualitative study of devops usage in practice. *Journal of Software: Evolution and Process*, *00*. doi: 10.1002/smr.1885

ETSI. (2016). *Second white paper on mobile edge computing from etsi.* Retrieved 2018-02-02, from http://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf

ETSI. (2020). *Etsi mobile edge computing portal.* Retrieved 2020-01-02, from http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing

Felderer, M., Büchler, M., Johns, M., Brucker, A., Breu, R., & Pretschner, A. (2016, 03). Security testing: A survey. In (p. 1-51). doi: 10.1016/bs.adcom.2015.11.003

Ferrer, A., Hernández, F., Tordsson, J., Elmroth, E., Ali-Eldin, A., Zsigri, C., ... Sheridan, C. (2012, 01). Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, *28*, 66-77. doi: 10.1016/j.future.2011.05.022

F-stack. (2019). Retrieved 2019-05-02, from http://www.f-stack.org/

Garg, S., Versteeg, S., & Buyya, R. (2013, 06). A framework for ranking of cloud computing services. *Future Generation Computer Systems*, *29*, 1012–1023. doi: 10.1016/j.future.2012.06.006

Garrett, O. (2016). *Introducing nginx 1.10 and 1.11.* NGINX. Retrieved 2019-01-02, from https://www.nginx.com/blog/nginx-1-10-1-11-released/

Google. (2019). *The service mesh era: Architecting, securing and managing micro-services with istio. whitepaper.* Retrieved 2019-12-02, from https://services.google.com/fh/files/misc/the_service_mesh_era_architecting_securing_and_managing_microservices_with_istio_white_paper.pdf

Greenberg, A., Hamilton, J., Maltz, D. A., & Patel, P. (2008). The cost of a cloud. *ACM SIGCOMM Computer Communication Review*, *39*(1), 68. doi: 10.1145/1496091.1496103

Grieco, R., Malandrino, D., & Scarano, V. (2006). A scalable cluster-based infrastructure for edge-computing services. *World Wide Web*, *9*(3), 317-341. doi: 10.1007/s11280-006-8559-x

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645-1660. doi: 10.1016/j.future.2013.01.010

Hassan, N., Yau, K.-L., & Wu, C. (2019, 08). Edge computing in 5g: A review. *IEEE Access*, *PP*, 1-1. doi: 10.1109/ACCESS.2019.2938534

Honbo, Z. (2013). *The internet of things in the cloud*. CRC Press.

Hromic, H., Phuoc, D., Serrano, M., Antonić, A., Zarko, I., Hayes, C., & Decker, S. (2015, 06). Real time analysis of sensor data for the internet of things by means of clustering and event processing.. doi: 10.1109/ICC.2015.7248401

Humble, J., Kim, G., & Forsgren, N. (2018). *Accelerate: The science of lean software and devops: Building and scaling high performing technology organizations*. IT Revolution Press.

IDC. (2019). *he growth in connected iot devices is expected to generate 79.4zb of*

data in 2025, according to a new idc forecast. Retrieved 2019-01-02, from https://www.marketwatch.com/press-release/the-growth-in-connected-iot-devices-is-expected-to-generate-794zb-of-data-in-2025-according-to-a-new-idc-forecast-2019-06-18?mod=mw_quote_news

Imran, M., Alghamdi, A., & Ahmad, B. (2015, 12). Role of firewall technology in network security. *International Journal of Innovation and advancement in Computer Science*, 3-6.

Indrasiri, K., & Siriwardena, P. (2018, 01). Service mesh: Designing, developing, and deploying. In (p. 263-292). doi: 10.1007/978-1-4842-3858-5_9

Injecting vault secrets into kubernetes pods via a sidecar. (2019). Retrieved 2020-01-02, from https://www.hashicorp.com/blog/injecting-vault-secrets-into-kubernetes-pods-via-a-sidecar/

Iot security and scalability on intel® iot platform. (2018). Retrieved 2019-01-02, from https://www.intel.com/content/www/us/en/internet-of-things/iot-platform.html

Ismail, B. I., Mostajeran Goortani, E., Ab Karim, M. B., Ming Tat, W., Setapa, S., Luke, J. Y., & Hong Hoe, O. (2015, Aug). Evaluation of docker as edge computing platform. In *2015 ieee conference on open systems (icos)* (p. 130-135). doi: 10.1109/ICOS.2015.7377291

Istio security documentation. (2019). Retrieved 2020-01-02, from https://istio.io/docs/concepts/security/

Jararweh, Y., Doulat, A., Alqudah, O., Ahmed, E., Al-Ayyoub, M., & Benkhelifa, E. (2016, 05). The future of mobile cloud computing: Integrating cloudlets and mobile edge computing.. doi: 10.1109/ICT.2016.7500486

Jararweh, Y., Tawalbeh, L., Ababneh, F., & Aldosari, F. (2013, 12). Resource efficient mobile computing using cloudlet infrastructure. In (p. 373-377). doi: 10.1109/MSN.2013.75

Kartakis, S., & Mccann, J. (2014, 06). Real-time edge analytics for cyber physical systems using compression rates real-time edge analytics for cyber physical systems using compression rates..

Kata containers architecture. (2019). Retrieved 2019-12-02, from https://github.com/kata-containers/documentation/blob/master/design/architecture.md

Kaur, K., Dhand, T., Kumar, N., & Zeadally, S. (2017, 06). Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers. *IEEE Wireless Communications*, *24*, 48-56. doi: 10.1109/MWC.2017.1600427

Keeriyattil, S. (2019, 12). Microsegmentation and zero trust: Introduction. In (p. 17-31). doi: 10.1007/978-1-4842-5431-8_2

Khan, K., Wang, Q., & Grecos, C. (2012, 11). Experimental framework of integrated cloudlets and wireless mesh networks.. doi: 10.1109/TELFOR.2012.6419180

Khan, K. A., Wang, Q., Luo, C., Wang, X., & Grecos, C. (2014). Comparative study of internet cloud and cloudlet over wireless mesh networks for real-time applications. *Real-Time Image and Video Processing 2014*. doi: 10.1117/12.2052474

Kindervag, J. (2010). *Build security into your network's dna: The zero trust network architecture.* Forester Research. Retrieved 2019-09-02, from `http://www.virtualstarmedia.com/downloads/Forrester_zero_trust_DNA.pdf`

Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., ... Yarom, Y. (2018, 01). Spectre attacks: Exploiting speculative execution..

Komarek, A., Sobeslav, V., & Pavlik, J. (2015, 01). Enterprise ict transformation to agile environment. In (p. 326-335). doi: 10.1007/978-3-319-24306-1_32

Krebs, R., Momm, C., & Kounev, S. (2012, 04). Architectural concerns in multi-tenant saas applications. In (pp. 426–431).

Krishnaveni, S., Prabakaran, & Sivamohan, S. (2015, 01). Analysis of software security testing techniques in cloud computing. *International Journal of Modern Trends in Egineering and Research*, *02*, 10-23. doi: 10.1145/2506164.898743

Kubernetes secrets. (2019). Retrieved 2019-12-02, from `https://kubernetes.io/docs/concepts/configuration/secret/`

Lewis, G., Echeverria, S., Simanta, S., Bradshaw, B., & Root, J. (2014, 10). Tactical cloudlets: Moving cloud computing to the edge. In (p. 1440-1446). doi: 10.1109/MILCOM.2014.238

Li, H., Shou, G., Hu, Y., & Guo, Z. (2016). Mobile edge computing: Progress and challenges. *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. doi: 10.1109/mobilecloud.2016.16

Li, K.-C., Li, Q., & Shih, T. K. (2019). *Cloud computing and digital media*. CRC PRESS.

Li, R. (2018). *Benchmarking envoy proxy, haproxy, and nginx performance on kubernetes.* Ambassador. Retrieved 2019-01-02, from `https://security.stackrox.com/rs/219-UEH-533/images/StackRox-Report-State_of_Container_and_Kubernetes_Security_Spring_2019.pdf`

Li, W., Lemieux, Y., Gao, J., Zhao, Z., & Han, Y. (2019, 04). Service mesh: Challenges, state of the art, and future research opportunities. In (p. 122-1225). doi: 10.1109/SOSE.2019.00026

Li, W., Lu, S., & Chen, D. (2015, 03). Mechanisms and challenges on mobility-augmented service provisioning for mobile cloud computing. *Communications*

Magazine, IEEE, *53*, 89-97. doi: 10.1109/MCOM.2015.7060487

Li, Y., & Wang, W. (2013, 12). The unheralded power of cloudlet computing in the vicinity of mobile devices. In (p. 4994-4999). doi: 10.1109/GLOCOMW.2013.6855742

Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., ... Hamburg, M. (2018, 01). Meltdown: Reading kernel memory from user space..

Liu, Y., Lee, M., & Zheng, Y. (2016, 10). Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system. *IEEE Transactions on Mobile Computing*, *15*, 1536-1233. doi: 10.1109/TMC.2015.2504091

Luan, T., Gao, L., Li, Z., Xiang, Y., & Sun, L. (2015, 02). Fog computing: Focusing on mobile users at the edge. *Comput. Sci.*.

Luan, T. H., Gao, L., Li, Z., Xiang, Y., We, G., & Sun, L. (2015, 10). Fog computing: Focusing on mobile users at the edge..

LXC. (2019). *lxcfs.* Author. Retrieved 2019-09-02, from `https://github.com/lxc/lxcfs`

Lynch, J., & Matthews, L. (2017). *Taking zero-downtime load balancing even further.* Yelp. Retrieved 2019-01-02, from `https://engineeringblog.yelp.com/2017/05/taking-zero-downtime-load-balancing-even-further.html`

Machen, A., Wang, S., Leung, K. K., Ko, B. J., & Salonidis, T. (2016). Migrating running applications across mobile edge clouds. *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking - MobiCom '16*. doi: 10.1145/2973750.2985265

Mahjabin, T., Xiao, Y., Sun, G., & Jiang, W. (2017, 12). A survey of distributed denial-of-service attack, prevention, and mitigation techniques. *International Journal of Distributed Sensor Networks*, *13*, 155014771774146. doi: 10.1177/1550147717741463

Manaseer, S., & Al Hwaitat, A. (2018, 09). Centralized web application firewall security system. *Modern Applied Science*, *12*, 164. doi: 10.5539/mas.v12n10p164

Mann, A., Stahnke, M., Brown, A., & Kersten, N. (2018). *State of devops report.* Puppet. Retrieved 2019-02-02, from `https://puppet.com/resources/report/state-of-devops-report/`

McAfee, A., & Brynjolfsson, E. (2008, 07). Investing in the it that makes a competitive difference. *Harvard Business Review*, *86*, 98-107.

McNamara, J. (2018). *Dpdk test suite.* Retrieved 2019-12-02, from `https://readthedocs.org/projects/dpdk-test-suite/downloads/pdf/stable/`

Mercl, L., & Pavlik, J. (2019a, 01). The comparison of container orchestrators: Icict 2018, london. In (p. 677-685). doi: 10.1007/978-981-13-1165-9_62

Mercl, L., & Pavlik, J. (2019b, 08). Public cloud kubernetes storage performance analysis. In (p. 649-660). doi: 10.1007/978-3-030-28374-2_56

Meurisch, C., Seeliger, A., Schmidt, B., Schweizer, I., Kaup, F., & Mühlhäuser, M. (2015, 11). Upgrading wireless home routers for enabling large-scale deployment of cloudlets. In (Vol. 162, p. 12-29). doi: 10.1007/978-3-319-29003-4_2

Microsoft azure cloud computing platform & services. (2019). Retrieved 2019-09-09, from https://azure.microsoft.com/en-gb/

MITRE. (2019). *Cve: Vulnerability statistics.* Retrieved 2019-12-02, from https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33

Montes, J., Sanchez, A., Memishi, B., Pérez, M., & Antoniu, G. (2013, 03). Gmone: A complete approach to cloud monitoring. *Future Generation Computer Systems*. doi: 10.1016/j.future.2013.02.011

Morabito, R. (2017, 05). Virtualization on internet of things edge devices with container technologies: a performance evaluation. *IEEE Access*, *PP*. doi: 10.1109/ACCESS.2017.2704444

Multi-access edge computing (mec). (2018). Retrieved 2019-01-02, from https://networks.nokia.com/solutions/multi-access-edge-computing#tab-highlights

Nanobox. (2017). *Docker container use adoption devops clusterhq survey.* Author. Retrieved 2019-09-02, from https://content.nanobox.io/content/images/2017/06/docker-container-use-adoption-devops-clusterhq-survey.png

Network edge smart cell solutions revolutionize mobile services. (2018). Intel. Retrieved 2019-01-02, from https://www.intel.com/content/www/us/en/communications/smart-cells-revolutionize-service-delivery.html

Niyato, D., Wang, P., Joo, P., Han, Z., & Kim, D. (2015, 01). Optimal energy management policy of a mobile cloudlet with wireless energy charging. *2014 IEEE International Conference on Smart Grid Communications, SmartGridComm 2014*, 728-733. doi: 10.1109/SmartGridComm.2014.7007734

Nunna, S., & Ganesan, K. (2017, 01). Mobile edge computing. In (p. 187-203). doi: 10.1007/978-3-319-47617-9_9

Odun-Ayo, I., Misra, S., Abayomi-Alli, O., & Ajayi, O. (2017, 12). Cloud multi-tenancy: Issues and developments. In (p. 209-214). doi: 10.1145/3147234.3148095

OECI. (2017). *Tactical-cloudlets.* Open Edge Computing Initiative. Retrieved 2019-02-02, from https://www.govtechworks.com/tactical-cloudlets-mobile-computing-readies-for-battle/

Open edge computing. (2018). Open Edge Computing Initiative. Retrieved 2019-02-02, from http://openedgecomputing.org

Open policy agent. (2020). Retrieved 2020-01-02, from https://www.openpolicyagent.org/

Open vswitch documentation. (2018). Retrieved 2019-01-02, from http://docs.openvswitch.org/en/latest/

Padit, K., Prasad, V. M., Bian, B., & Kwatra, A. (2018). *Modeling the impact of cpu properties to optimize and predict packet-processing performance.* Intel Corporation. Retrieved 2019-01-02, from https://www.intel.com/content/dam/www/public/us/en/documents/case-studies/att-cpu-impact-on-packet-processing-perfomance-paper.pdf

Patel, M., Naughton, B., Chan, C., Sprecher, N., & Abeta, S. (2014). *Mobile-edge computing introductory technical white paper.*

Pavlik, J., Komarek, A., & Sobeslav, V. (2014, 11). Security information and event management in the cloud computing infrastructure. In *2014 ieee 15th international symposium on computational intelligence and informatics* (*cinti*) (p. 209-214). doi: 10.1109/CINTI.2014.7028677

Pavlik, J., & Sobeslav, V. (2018). Open iot platform design for urban environments..

Pavlik, J., Sobeslav, V., & Horalek, J. (2014, 07). Statistics and analysis of service availability in cloud computing.. doi: 10.1145/2628194.2628222

Pavlik, J., Sobeslav, V., & Komarek, A. (2014, 11). Measurement of cloud computing services availability. In (Vol. 144, p. 191-201). doi: 10.1007/978-3-319-15392-6_19

Peng, M., Yan, S., Zhang, K., & Wang, C. (2015, 06). Fog computing based radio access networks: Issues and challenges. *IEEE Network*, *30*. doi: 10.1109/MNET.2016.7513863

Povedano-Molina, J., Lopez-Vega, J., Lopez-Soler, J., Corradi, A., & Foschini, L. (2013, 10). Dargos: A highly adaptable and scalable monitoring architecture for multi-tenant clouds. *Future Generation Computer Systems*, *29*, 2041-2056. doi: 10.1016/j.future.2013.04.022

Project atomic. (2019). Retrieved 2019-12-02, from https://www.projectatomic.io/

Protic, D., & Stankovic, M. (2019, 01). Anomaly-based intrusion detection: Feature selection and normalization influence to the machine learning models accuracy. *European Journal of Engineering and Formal Sciences*, *2*. doi: 10.26417/ejef.v2i3.p101-106

Reinertsen, D. G. (2009). *The principles of product development flow: Second generation lean product development*. Celeritas Publishing.

Routaib, H., Badidi, E., Elmachkour, M., Sabir, E., & ElKoutbi, M. (2014, 05). Modeling and evaluating a cloudlet-based architecture for mobile cloud computing. In (p. 1-7). doi: 10.1109/SITA.2014.6847290

Saeid, S., & Ali Yahiya, T. (2018, 10). Load balancing evaluation tools for a private cloud: A comparative study. *ARO-The Scientific Journal of Koya University*, *6*, 13-19. doi: 10.14500/aro.10438

Samaniego, M., & Deters, R. (2018, 07). Zero-trust hierarchical management in iot. In (p. 88-95). doi: 10.1109/ICIOT.2018.00019

Santos, I., Tilly, M., Chandramouli, B., & Goldstein, J. (2013, 08). Dial: Distributed streaming analytics anywhere, anytime. *Proceedings of the VLDB Endowment*, *6*, 1386-1389. doi: 10.14778/2536274.2536322

Satyanarayanan, M., Bahl, V., Caceres, R., & Davies, N. (2011). The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*. doi: 10.1109/mprv.2009.64

Sayfan, G. (2018). *Mastering kubernetes* (2nd ed.). PACKT Publishing Limited.

Scarfone, K., Souppaya, M., Cody, A., & Orebaugh, A. (2008). *Technical guide to information security testing and assessment.* NIST. Retrieved from `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf`

Schuster, R., & Ramchandran, P. (2016). *Open edge computing – from vision to reality.* OPNFV Design Summit, Berlin, Germany.

Schwarzkopf, M., Murray, D. G., & Hand, S. (2012). The seven deadly sins of cloud computing research. In *Presented as part of the unisex.* USENIX. Retrieved from `https://www.usenix.org/conference/hotcloud12/seven-deadly-sins-cloud-computing-research`

Scott, B. (2018, 03). How a zero trust approach can help to secure your aws environment. *Network Security*, *2018*, 5-8. doi: 10.1016/S1353-4858(18)30023-0

Shahzadi, S., Iqbal, M., Dagiuklas, T., & Qayyum, Z. U. (2017). Multi-access edge computing: open issues, challenges and future perspectives. *Journal of Cloud Computing*, *6*(1). doi: 10.1186/s13677-017-0097-9

Shi, W., & Dustdar, S. (2016). The promise of edge computing. *Computer*, *49*(5), 78-81. doi: 10.1109/mc.2016.145

Simoens, P., Herzeele, L., Vandeputte, F., & Vermoesen, L. (2015, 06). Challenges for orchestration and instance selection of composite services in distributed edge clouds. *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*, 1196-1201. doi: 10.1109/INM.2015.7140466

Stojmenovic, I., & Wen, S. (2014, 09). The fog computing paradigm: Scenarios and security issues. In (p. 1-8). doi: 10.15439/2014F503

Sultan, S., Ahmad, I., & Dimitriou, T. (2019). Container security: Issues, challenges, and the road ahead. *IEEE Access*, *7*, 52976-52996. doi: 10.1109/ACCESS.2019.2911732

Suryawanshi, R., & Mandlik, G. (2015). Focusing on mobile users at edge and internet

of things using fog computing. In (Vol. 17, p. 3225-3231).

Sysel, M., & Doležal, O. (2014, 12). An educational http proxy server. *Procedia Engineering*, *69*, 128–132. doi: 10.1016/j.proeng.2014.02.212

Systems, C. (2019). *Cisco iox network infrastructure products.* Retrieved 2019-01-02, from https://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html

Tang, W., Jenkins, J., Meyer, F., Ross, R., Kettimuthu, R., Winkler, L., ... Desai, N. (2015, 02). Data-aware resource scheduling for multicloud workflows: A fine-grained simulation approach. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, *2015*, 887-892. doi: 10.1109/CloudCom.2014.19

Tarreau, W. (2017). *Truly seamless reloads with haproxy – no more hacks!* HAProxy. Retrieved 2019-01-02, from https://www.haproxy.com/blog/truly-seamless-reloads-with-haproxy-no-more-hacks/

Tene, T. (2014, 01). The need for a dynamic, multi-layered cloud security. , 2347-2812.

Tian-yang, G., Yin-sheng, S., & You-yuan, F. (2010, 09). Research on software security testing. In (Vol. 70, p. 647-651).

Tran, T. X., Hajisami, A., Pandey, P., & Pompili, D. (2017). Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges. *IEEE Communications Magazine*, *55*(4), 54-61. doi: 10.1109/mcom.2017.1600863

Tungsten fabric architecture. (2018). Retrieved 2019-01-02, from https://tungstenfabric.github.io/website/Tungsten-Fabric-Architecture.html

Ullah, R., Ahmed, S. H., & Kim, B.-S. (2018, 12). Information-centric networking with edge computing for iot: Research challenges and future directions. *IEEE Access*, *PP*, 1-1. doi: 10.1109/ACCESS.2018.2884536

Van Casteren, W. (2017, 02). The waterfall model and the agile methodologies : A comparison by project characteristics.. doi: 10.13140/RG.2.2.36825.72805

Vaquero, L. M., & Rodero-Merino, L. (2014). Finding your way in the fog. *ACM SIGCOMM Computer Communication Review*, *44*(5), 27-32. doi: 10.1145/2677046.2677052

Varghese, B., Akgun, O., Miguel, I., Thai, L., & Barker, A. (2014, 11). Cloud benchmarking for performance. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, *2015*. doi: 10.1109/CloudCom.2014.28

Varghese, B., Subba, L., Thai, L., & Barker, A. (2016, 03). Doclite: A docker-based lightweight cloud benchmarking tool..

Varghese, B., Wang, N., Barbhuiya, S., Kilpatrick, P., & Nikolopoulos, D. (2016, 11). Challenges and opportunities in edge computing.. doi: 10.1109/SmartCloud.2016.18

Vargo, S. (2019). *http-echo.* Ambassador. Retrieved 2019-10-02, from `https://github.com/hashicorp/http-echo`

Verbelen, T., Simoens, P., Turck, F., & Dhoedt, B. (2012, 06). Cloudlets: Bringing the cloud to the mobile user. *MCS'12 - Proceedings of the 3rd ACM Workshop on Mobile Cloud Computing and Services*. doi: 10.1145/2307849.2307858

Wang, X., Han, G., Du, X., & Rodrigues, J. J. (2015). Mobile cloud computing in 5g: Emerging trends, issues, and challenges [guest editorial]. *IEEE Network*, *29*(2), 4-5. doi: 10.1109/mnet.2015.7064896

Whaiduzzaman, M., Gani, A., & Naveed, A. (2014, 12). Pefc: Performance enhancement framework for cloudlet in mobile cloud computing.. doi: 10.1109/ROMA.2014.7295892

Widener, S. K. (2007). An empirical analysis of the levers of control framework. *Accounting, Organizations And Society, 32(7-8), 757-788*.

Xiao, Y., Jia, Y., Liu, C., Cheng, X., Yu, J., & Lv, W. (2019, 06). Edge computing security: State of the art and challenges. *Proceedings of the IEEE*, *PP*, 1-24. doi: 10.1109/JPROC.2019.2918437

Xu, L., Wang, Z., & Chen, W. (2015, 07). The study and evaluation of arm-based mobile virtualization. *International Journal of Distributed Sensor Networks*, *2015*. doi: 10.1155/2015/310308

Yadav, A., Garg, M., & Mehra, R. (2019, 01). Docker containers versus virtual machine-based virtualization: Proceedings of iemis 2018, volume 3. In (p. 141-150). doi: 10.1007/978-981-13-1501-5_12

Yang, B., Wu, D., & Wang, R. (2019, 05). Cue: An intelligent edge computing framework. *IEEE Network*, *33*, 18-25. doi: 10.1109/MNET.2019.1800316

Yin, D., & Kosar, T. (2011, 07). A data-aware workflow scheduling algorithm for heterogeneous distributed systems.. doi: 10.1109/HPCSim.2011.5999814

Young, D. (2013, 08). Software development methodologies. *White paper*.

Zavodovski, A., Mohan, N., & Kangasharju, J. (2018). edisco: Discovering edge nodes along the path. *CoRR*, *abs/1805.01725*. Retrieved from `http://arxiv.org/abs/1805.01725`

Zeebaree, S., Hussein, K., & Muhamad, R. (2018, 12). Application layer distributed denial of service attacks defense techniques : A review. *Academic Journal of Nawroz University*, *7*, 113. doi: 10.25007/ajnu.v7n4a279

Zhang, K., Mao, Y., Leng, S., He, Y., & Zhang, Y. (2017). Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading. *IEEE Vehicular Technology Magazine*, *12*(2), 36-44. doi: 10.1109/mvt.2017.2668838

Zhang, Y., Liu, Q., & Zhao, G. (2010, 08). A real-time ddos attack detection and prevention system based on per-ip traffic behavioral analysis. In (Vol. 2, p. 163 - 167).

doi: 10.1109/ICCSIT.2010.5563549

Zhou, X., Peng, X., Xie, T., Sun, J., Li, W., Ji, C., & Ding, D. (2018, 09). Delta debugging microservice systems. In (p. 802-807). doi: 10.1145/3238147.3240730

Zhu, W., Li, P., Luo, B., Xu, H., & Zhang, Y. (2018, 12). Research and implementation of high performance traffic processing based on intel dpdk. In (p. 62-68). doi: 10.1109/PAAP.2018.00018

Zhu, W., Luo, C., Wang, J., & Li, S. (2011). Multimedia cloud computing. *IEEE Signal Processing Magazine*, *28*(3), 59-69. doi: 10.1109/msp.2011.940269

Zhuang, W. (2017). Emerging technologies, applications, and standardizations for connecting vehicles (part ii) [from the guest editor]. *IEEE Vehicular Technology Magazine*, *12*(2), 23-25. doi: 10.1109/mvt.2017.2682683

Seznam Zkratek

AP Access Point. 14

API Application Programming Interface. 40

AR Augmented Reality. 8

CC Cloud Computing. 5, 6

CE Consumer Edge. 42

CEC Consumer Edge Computing. 2, 37

CI/CD Continuous Integration / Continuous Delivery. 123

CLI Command Line Interface. 64

CNI Container Network Interface. 111

CRD Kubernetes Custom Resource Definition. 76

CRM Customer Relationship Management. 27, 121

CVE Common Vulnerabilities and Exposures. 108

DDoS Distributed Denial-of-Service. 44, 81

DPDK Data Plane Development Kit. 45, 46

DSP Digital Signal Processor. 19

EdgeC Edge Computing. 6

EGW Edge Gateway. 45

ERP Enterprise Resource Planning. 27

ETSI The European Telecommunications Standards Institute. 12, 13

GC Global Controller. 41

IaaS Infrastructure-as-a-Service. 15

ICT Information and Communications Technology. 127

IDM Identity Management. 79

IoT Internet of Things. 1

ISMS Information Security Management System. 102

ITSM IT Service Management. 121

LACP Link Aggregation Control Protocol. 47

MCC Mobile Cloud Computing. 6, 12, 13

MEC Mobile Edge Computing. 2, 11, 12, 37

mTLS Mutual TLS. 80

MTTR Mean time to repair. 123

OCI Open Container Interface. 110

OPNFV Open Platform for NFV. 13

OS Operační Systém. 110

PoP Point of Presence. 42

PPS Packet Per Second. 47

PVC Persistent Volume Claim. 76

QoE Quality of Experience. 11, 20

QoS Quality of Service. 20, 39

QPS Query Per Second. 51

RAN Radio Access Network. 7

RE Regionální Edge. 42

ROI Return Of Investment. 114

RPS Request Per Second. 96

RSU Roadside Units. 8

SaaS Software-as-a-Service. 41

SDK Software Development Kit. 30, 33

SDN Software Defined Networking. 71

SEcS Scalable Edge Computing Services. 12

SIEM Security Information and Event Management. 79

SLA Service Level Agreement. 118

SR-IOV single root input/output virtualization. 95

SRE Site Reliability Engineering. 128

V2X Vehicle-to-everything. 8

VIP Virtual IP address. 50

vK8s Virtual Kubernetes. 101

VM Virtual Machine. 62

VRF Virtual Routing and Forwarding. 47, 51

WAFS Web Application Firewalls. 44, 81

XSS cross-site scripting. 91

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu

Seznam všech publikovaných prací disertanta

Publikované práce

Pavlik, Jakub & Sobeslav, Vladimir & Horalek, Josef. (2014). Statistic and analysis of service availability in cloud computing. 10.1145/2628194.2628222.

Pavlik, Jakub & Komarek, Ales & Sobeslav, Vladimir & Horalek, Josef. (2014). Gateway redundancy protocols. 459-464. 10.1109/CINTI.2014.7028719.

Pavlik, Jakub & Komarek, Ales & Sobeslav, Vladimir (2014), Security information and event management in the cloud computing infrastructure.

Sobeslav, Vladimir & Horalek, Josef & Pavlik, Jakub. (2015). Utilisation of Cloud Computing in Education with Focus on Open-Source Technologies. 10.1007/978-3-319-11104-9_94.

Pavlik, Jakub & Sobeslav, Vladimir & Komarek, Ales. (2014). Measurement of Cloud Computing Services Availability. 144. 191-201. 10.1007/978-3-319-15392-6_19.

Komarek, Ales & Sobeslav, Vladimir & Pavlik, Jakub. (2015). Enterprise ICT Transformation to Agile Environment. 10.1007/978-3-319-24306-1_32.

Komarek, Ales & Pavlik, Jakub & Sobeslav, Vladimir. (2015). Network Visualization Survey. 10.1007/978-3-319-24306-1_27.

Komarek, Ales & Pavlik, Jakub & Sobeslav, Vladimir. (2015). High Level Models for IaaS Cloud Architectures. 598. 209-218. 10.1007/978-3-319-16211-9_22.

Cimler, Richard & Doležal, Ondrej & Kühnová, Jitka & Pavlik, Jakub. (2016). Herding Algorithm in a Large Scale Multi-agent Simulation. 10.1007/978-3-319-39883-9_7.

Komarek, Ales & Pavlik, Jakub & Mercl, Lubos & Sobeslav, Vladimir. (2017). Hardware Layer of Ambient Intelligence Environment Implementation. 325-334. 10.1007/978-3-319-67077-5_31.

Komarek, Ales & Pavlik, Jakub & Sobeslav, Vladimir. (2017). Performance Analysis of Cloud Computing Infrastructure. 303-313. 10.1007/978-3-319-65515-4_25.

Komarek, Ales & Pavlik, Jakub & Mercl, Lubos & Sobeslav, Vladimir. (2017). VNF Orchestration and Modeling with ETSI MANO Compliant Frameworks. 121-131. 10.1007/978-3-319-67380-6_11.

Komarek, Ales & Pavlik, Jakub & Sobeslav, Vladimir. (2017), Hybrid System Orchestration with TOSCA and Salt. 2396-2401. ISSN 1816-949X.

Komarek, Ales & Pavlik, Jakub & Mercl, Lubos & Sobeslav, Vladimir. (2018). Metric Based Cloud Infrastructure Monitoring. 391-400. 10.1007/978-3-319-69835-9_37.

Komarek, Ales & Pavlik, Jakub & Sobeslav, Vladimir. (2018). Orchestration and Automation of NVF. 664-672. 10.1007/978-3-319-69835-9_62.

Mercl, Lubos & Pavlik, Jakub. (2019). The Comparison of Container Orchestrators: ICICT 2018, London. 10.1007/978-981-13-1165-9_62.

Mercl, Lubos & Pavlik, Jakub. (2019). Public Cloud Kubernetes Storage Performance Analysis. 10.1007/978-3-030-28374-2_56.

Komarek, Ales & Pavlik, Jakub & Sobeslav, Vladimir. (2020). Time-Based Change Visualization Techniques Survey. Journal of Engineering and Applied Sciences. 14. 10538-10543. 10.36478/jeasci.2019.10538.10543.

Nepropsané práce v databázi

Pavlik, Jakub & Sobeslav, Vladimir, (2018). Cloud Computing Platform for Internet of Things. *ADIBUM*

Pavlik, Jakub & Sobeslav, Vladimir, (2018). Open IoT Platform Design for Urban Environments. *ADIBUM*

Seznam odborných vědecko-výzkumných aktivit

Shrnutí publikačních výstupů

	publikací	h-index	citací celkem
WoS	13	4	22
Scopus	15	4	39
Google Scholar	17	5	80

Participace na projektech

[1] Smart networking & cloud computing solutions. SPEV 2014. Řešitel. prof. Ing. Ondřej Krejcar, Ph.D. Role: návrh a implementace OpenStack řešení.

[2] Smart řešění v bioinformatice. Grant Excelence. 2014. Řešitel prof. Ing. Kamil Kuča, Ph.D. Role: návrh platformy pro stanovení bioaktivních látek.

[3] Věda na dosah ruky. Projekt EU. Role: vypracování pěti obsahových kapitol o virtualizaci.

[4] Smart Solutions for Ubiquitous Computing Environments. SPEV. 2015. Řešitel prof. Ing. Ondřej Krejcar, Ph.D. Role: nasazení systému pro sběr senzorických dat.

[5] SCM, Control of Markets and Production in Agent-based Computational Economics. SPEV. 2015. Řešitel RNDr. Petr Tučník, Ph.D.. Role: analýza a návrh využití cloud computingu v multi agentních systémech.

[6] GAČR 15-117245. 2016. Řešitel prof. RNDr. Peter Mikulecký, Ph.D. Role: návrh a implementace cloudové platformy pro výpočty.

[7] Smart Solutions for Ubiquitous Computing Environments SPEV. 2016. Řešitel prof. Ing. Ondřej Krejcar, Ph.D. Role: návrh a implementace otevřené IoT platformy pro chytrá města.

[8] SPEV 2017 - Počítačové sítě pro cloud a distribuované výpočty – řešitel Ing. Karel Mls, Ph.D. Role: návrh a implementace OpenStack platformy pro distribuované výpočty.

Odborné přednášky

[1] Přednáška na SmartCity IoT on Kubernetes v Londýně - https://youtu.be/_L_pR6ZYwEs

[2] Přednáška na OpenStack Summitu v Austinu 2016 - https://youtu.be/Ym_CZ8-crD8

[3] Přednáška na OpenStack Summitu v Bostonu 2017 - https://youtu.be/o1RgIK1N4mo

[4] Přednáška na KubeCon v San Diego 2019 - https://youtu.be/Sz_RCstTlE0

Další činnost

[1] Příprava studijních dokumentů pro podporu vzdělávání v předmětu OS1 a OS2 v roce 2014, vedoucí: Mgr. Josef Horálek, Ph.D.

[2] Instalace OpenStack cloudu pro výuku 2013, vedoucí: Ing. Vladimír Soběslav, Ph.D.