Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Prototyp pro uložení binárních dat do MinIO

Bakalářská práce

Autor: Jakub Fogl
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Jan Krunčík

Odborný konzultant: Ing. Lubomír Andrle
Unicorn

Hradec Králové duben 2024

Prohlášení:

Prohlašuji, že jsem bakalářskou/diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne Jakub Fogl

_____

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Janu Krunčíkovi za metodické vedení práce, svému odbornému konzultantovi Ing. Lubomíru Andrlemu a firmě Unicorn jako zadavateli.

Abstrakt

Tato bakalářská práce má za cíl zmapovat možnosti objektového uložiště MinIO pro využití v produktu uuEnelane. V první části práce je představen produkt, jeho funkce a struktura. Následuje teoretický základ problematiky cloudu, porovnání s lokálním serverem a základní informace objektového uložiště. Dále je uveden srovnání možných alternativ uložiště a představení S3 API. Druhou částí je praktický projekt, jež je vytvořen podle požadavků uuEnelane, následně je testován jednotkovými testy a simulačním testem provozu. Praktický projekt realizuje způsob, jakým může být provedena implementace uložiště do aplikace v programovacím jazyce Java. Poslední částí práce je shrnutí výsledků a závěr, který nastiňuje, jakým směrem by se téma dalo rozvíjet. Výsledky práce ukazují, že pro produkt uuEnelane je vhodné využít uložiště MinIO, které splňuje dané požadavky.

Abstract

Prototype for storing binary data in MinIO

This bachelor thesis aims to describe options of object storage MinIO for using uuEnelane product. The product is introduced, the same as its function and structure in the first part of the work. The theoretical basis of the cloud problematics, comparison with local server, and object storage basic information follows. Then the comparison of possible alternative storage options and a description of S3 API are given. The second part of the work is the practical project, which is made by uuEnelane requirements, and then is tested by unit tests and traffic simulation tests. A practical project implements how the storage implementation could be done in the programming language Java. The last part contains the result summary and conclusion, which shows which way the topic could be further developed. The results say that the product is available for MinIO storage which satisfies the requirements.

Klíčová slova: MinIO, Cloud, S3 API, objektové uložiště, Java

Key words: MinIO, Cloud, S3 API, object storage, Java

Obsah

1. Úvod

Tématem této bakalářské práce je cloudové uložiště MinIO. Téma vzniklo nápadem firmy Unicorn integrovat uložiště do produktu uuEnelane. Produkt je middlewarem mezi dvěma systémy a jeho funkcí je připravení dat, která přijdou na vstup z jednoho systému, na takovou podobu, aby je druhá aplikace mohla bez obtíží využít. Primárním úkolem je audit příchozích a odchozích zpráv.

V současné době se v rámci produktu používají jiné technologie pro ukládání dat, ale cílem je rozšířit produkt o další technologii, kterou má být MinIO. Uložiště v rámci produktu bude provádět základní operace vytvoření, smazání a vrácení. Práce bude vymezovat základní zacházení s uložištěm a vytvoří teoretický základ pro rozhodnutí, zda je vhodné použít MinIO v produktu. Výstup bude podpořen teoretickými poznatky, které budou nejprve řešit základní teorii cloudu. Následně přijdou důležité kapitoly, které porovnají cloud s lokálním serverem a následně s alternativními technologiemi, které by mohly být použity. Poté bude představena technologie MinIO. Velkým tématem v rámci práce bude představení S3 API, které může do produktu přidat velkou výhodu. API bude integrováno vývojovým jazykem Java do projektu, který bude vytvořen podle podmínek firmy. Projekt bude řešen jako webová aplikace a bude mít vystavené API, přes které budou zpřístupněny operace. Aby se projekt více přiblížil skutečnému produktu, bude nutné nahrávat do projektu pouze vzorové soubory ze skutečného produktu. K tomu budou v projektu řešeny validace a rozdělení na payload a metadata. Tyto dvě části se následně budou ukládat do MinIO. V rámci instance MinIO bude řešen readonly režim proti neoprávněnému zápisu a základní konfigurace, která je potřebná k chodu uložiště. Celý projekt následně bude otestován jednotkovými testy, ale také simulací provozu.

2. uuEnelane

Pro správné vyhodnocení této bakalářské práce je potřeba rozebrat produkt, na kterým se má dané uložiště použít. Kapitola rozebírá základní funkce a význam produktu, následně vysvětluje požadavky, které cloud musí splňovat. [1] Aplikace, jež může být nazvána jako middleware, je dodávána na cloudu, nebo je instalována lokálně. Od společnosti Unicorn byly pro práci poskytnuty marketingové dokumenty, které jsou se svolením použity. Informace o produktu tedy vycházejí z těchto materiálů.

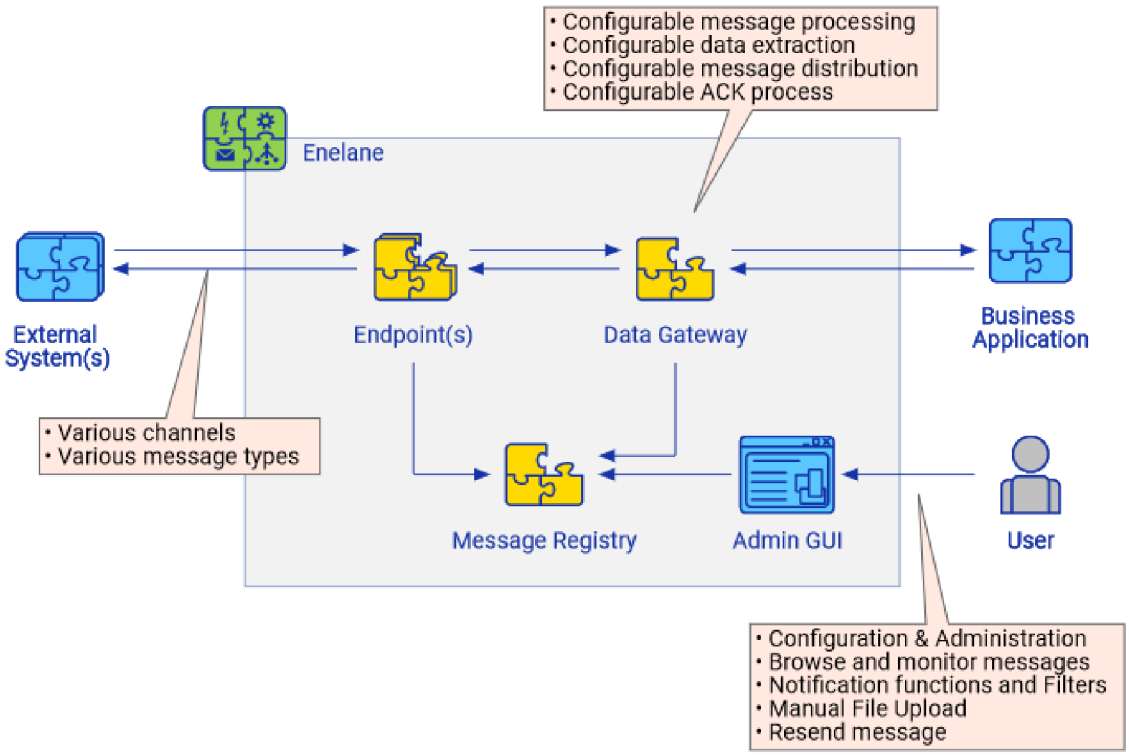
2.1 Funkce

[1] Produkt uuEnelane je aplikace ležící mezi dvěma koncovými body, konkrétně mezi externím systémem a byznys aplikací. Externí systém může být například systém senzorů, který poskytuje data byznys aplikaci, která je potřebuje pro svůj provoz. Dále to mohou být dva systémy, kde každý komunikuje jiným způsobem a uuEnelane se postará o jejich správné propojení.

[1] Hlavním úkolem produktu je přijímat proud dat v různých protokolech a různých formátů z externího systému a následně ho monitorovat, provádět validace, filtrovat, ukládat a také zpracovávat tak, aby výstupní data byla ve vhodné podobě pro byznys aplikaci. V konečném důsledku je byznys aplikace odstíněná od nutnosti znát všechny protokoly a formáty. To znamená, že po přijmutí už má data přesně v podobě, kterou vyžaduje, a může pokračovat v práci.

2.2 Základní architektura

[1] Obrázek 1 ukazuje základní architekturu aplikace. Ta je složena z mikroslužeb, které jsou odpovědné za jednotlivé prvky zpracování proudu dat. Mezi ně patří Endpoints, Data Gateway a Message Registry. Konkrétní vysvětlení mikroslužeb bude vysvětleno dále.



Obrázek 1: Architektura uuEnelane (zdroj: [1])

2.2.1 EndPoint

[1] Hlavní funkcí je přijmutí dat, nebo v opačném směru vyslání dat do externího systému. V případě přijímání datového proudu se v této části filtruje podle nastavených pravidel, a nakonec kompletuje interní enelane zprávu, která je uložena do MessageRegistry a následně poslána do Data Gateway. V opačném směru se datový proud odesílá. Nejdříve přijme enelane zprávu od Data Gateway. Následně vytvoří novou zprávu a vyplní její obsah podle již přijaté enelane zprávy. Posledními dvěma kroky je odeslání zprávy do externího systému a aktualizace zprávy v Message Registry.

Pro co nejvyšší univerzálnost příjmu a odeslání datového proudu je zde podporováno nespočet protokolů, jež můžou být například FTP, SFTP, IMAP, ECP a další.

2.2.2 Data Gateway

[1] Tato mikroslužba se stará o správu zprávy. Může probíhat dvěma způsoby. V prvním přepošle zprávu v nezměněné podobě do byznys aplikace, případně z byznys aplikace do externího systému. Ve druhé části přibude jednotka starající se o dodatečné zpracování zprávy. Zde probíhají validace, extrakce nebo další transformace zprávy podle potřeb koncové byznys aplikace.

2.2.3 Message Registry

[1] Tato část je zodpovědná za ukládání a archivování zpráv a příslušných doplňujících informací do odpovídajícího uložiště. Metadata, jež obsahují také extrahovaná data ze zprávy se ukládají do objektového uložiště. Další informace, které se ukládají do objektového uložiště, jsou list neúspěšných validací, které byly provedeny společně s jejich výsledkem a typem validace. Poslední věcí uloženou v tomto uložišti jsou citlivá data, kterými jsou například klíče, certifikáty a další. Ve druhém typu uložiště, jež je binární, se ukládá samotný obsah zprávy.

2.3 Zprávy

[1] Zprávy jsou v uuEnelane hlavním předmětem zpracování. V produktu nalezneme dva typy zpráv. Jedna vzniká přijímáním datového toku v Endpoint části a nazývá se Externí zprávy. Druhým typem jsou interní zprávy, které se jinak nazývají také Enelane zprávy. Po přijmu zprávy v Endpoint, ve formátu například XML, XLSX a JSON, se zpráva přetransformuje na Enelane zprávu s původním obsahem ve formátu JSON. Tyto zprávy se poté posílají mezi jednotlivými mikroslužbami, kde jsou zprávy zpracovávány.

2.4 Požadavky na MinIO

Na cloud produkt klade požadavky především v nutnosti podpory streamování, bez kterého by Message Registry v podobně cloudu nemohla efektivně fungovat. Dalším striktním požadavkem na cloud je dostatečně velká velikost objektu. V praxi se přes uuEnelane pohybují zprávy v řádech kilobajtů a cloud musí zprávu řádně uložit v celé její velikosti. Dalšími důležitými podmínkami pro bezpečný, rychlý a dobře fungující MessageStorage je podpora vysoké dostupnosti, rychlost čtení a zápisu, autorizovaný přístup, aby se nikdo nedostal k nepovolaným datům, a podpora archivace. Archivací je myšlena technika, kde se po uplynutí nakonfigurované doby odstraní všechny objekty, které jsou starší než daná doba. Toto mazání se automaticky provádí na straně cloudu bez nutnosti programování speciálních metod pro smazání více objektů. Dalším důležitou vlastností, kterou by cloud měl disponovat, je podpora komprimace. V uuEnelane se většinou zasílají textové zprávy, které je vhodné ukládat v komprimované podobě z důvodu efektivnějšího využití místa.

3. Cloud

[2] Protože je MinIO vztahováno k použití na cloudu, je důležité uvést teoretickou rovinu cloudu. Tato kapitola se nejprve zabývá základními informacemi týkajících se cloudu. Důležitým tématem obsaženým v této kapitole je analýza výhod a nevýhod spojených s nasazením cloudu oproti lokálnímu serveru, [2] na kterém může být MinIO také nasazeno. Dále se kapitola zabývá otázkou testování výkonu uložiště, což je v tomto případě složitější než provádění testů na jiných typech datových uložišť. Složitost spočívá v základním sestavení a ve škálování cloudu, kde se pro benchmarking musí uvažovat například virtualizace na sdílené infrastruktuře či různé metody škálování výkonu, které budou rozebrány později. Nakonec jsou porovnány cloudové služby od různých poskytovatelů, která by potenciálně mohly být použity v produktu uuEnelane.

3.1 Pojem Cloud vs Cloud computing

V následující kapitole bude vysvětlen pojem cloud a cloud computing, a to z důvodu lehké záměny jejich významů.

3.1.1 Cloud

[3] Pojem „cloud“ se používá pro označení serverů, softwaru a databází běžících na těchto serverech. [4] Jeden fyzický server je rozložen do několika virtuálních serverů pomocí virtualizace. [5] Virtualizace poskytovateli umožňuje rozdělit hardwarové prostředky mezi několik oddílů, ke kterým poté můžeme přistupovat pomocí API. Toto rozdělení umožňuje vykonávat cloud computing.

3.1.2 Cloud computing

Druhý pojem se týká jiného pohledu na cloud. Místo serverů, aplikací apod. bude v práci důležité, jak se s cloudem spojit a pracovat. [6] O cloud computingu se mluví v případě, kdy se myslí spojení mezi uživatelem a zdroji, které jsou součástí cloudu, tedy se realizuje přístup uživatele. Například IBM označuje také aplikace běžící na serverech pro správný chod

cloudu jako cloud computing. Mezi zmíněnými aplikacemi je například dříve zmíněná virtualizace.

3.2 Druhy cloud computingu podle správy infrastruktury

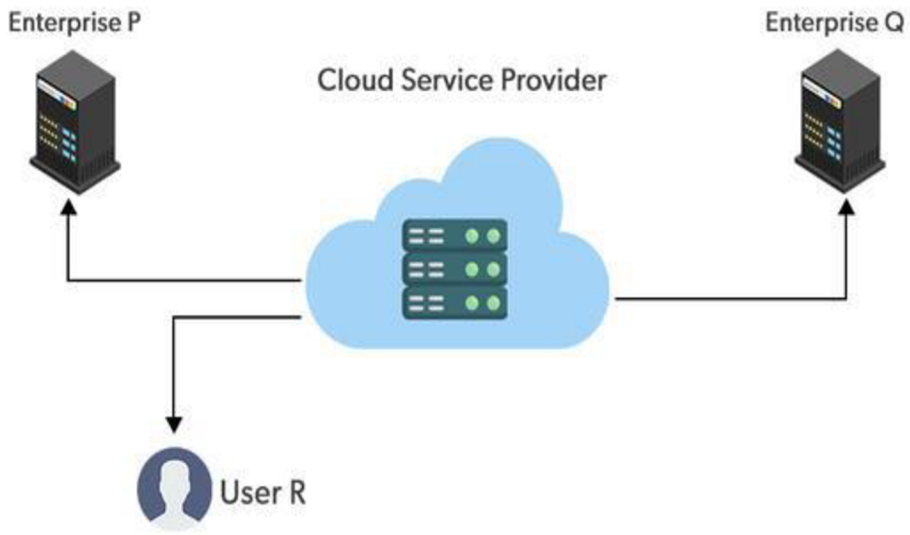
Cloud computing se může rozdělit do skupin podle umístění a spravování infrastruktury. Následující pod kapitoly vymezí pojmy „veřejný cloud“, „privátní cloud“ a „hybridní cloud“.

3.2.1 Veřejný cloud

Prvním zmíněným typem je cloud, který je v běžné společnosti nejvíce zastoupen. Je využíván milióny lidí ve světě na denní bázi. Možnost ukládání dat na cloud usnadňuje spoustu běžných činností člověka při kontaktu s výpočetní technikou. Jako příklad může být uveden přístup k jednotným datům z telefonu i počítače. Tato vymoženost šetří uživateli čas, který by byl potřeba pro přenos dat mezi jednotlivými zařízeními, které používá.

Data nejsou na fyzickém nosiči v úplném bezpečí. Uložiště ovlivňují různé okolní jevy, kterými může být například fyzické zničení nebo vystavení různým typům záření. Tato možnost ztráty dat může být kompenzována uložením duplicitních dat do vzdáleného uložiště. Pokud by nastala ztráta, uživatel si potřebná data pouze znova stáhne do svého zařízení.

Nejdříve byly pro názorné přiblížení veřejného cloudu vysvětleny ukázkové situace, ve kterých se lidé mohou s tímto typem cloudu setkat. Následně bude představen popis samotného veřejného typu. Podle Googlu [7] je veřejný cloud vlastněn společností, která má zázemí pro poskytování cloudových služeb odběratelům. Společnost se zároveň také stará o údržbu a chod serverů. Mezi nejznámější společnosti poskytující veřejný cloud patří například Google, Amazon nebo Microsoft.

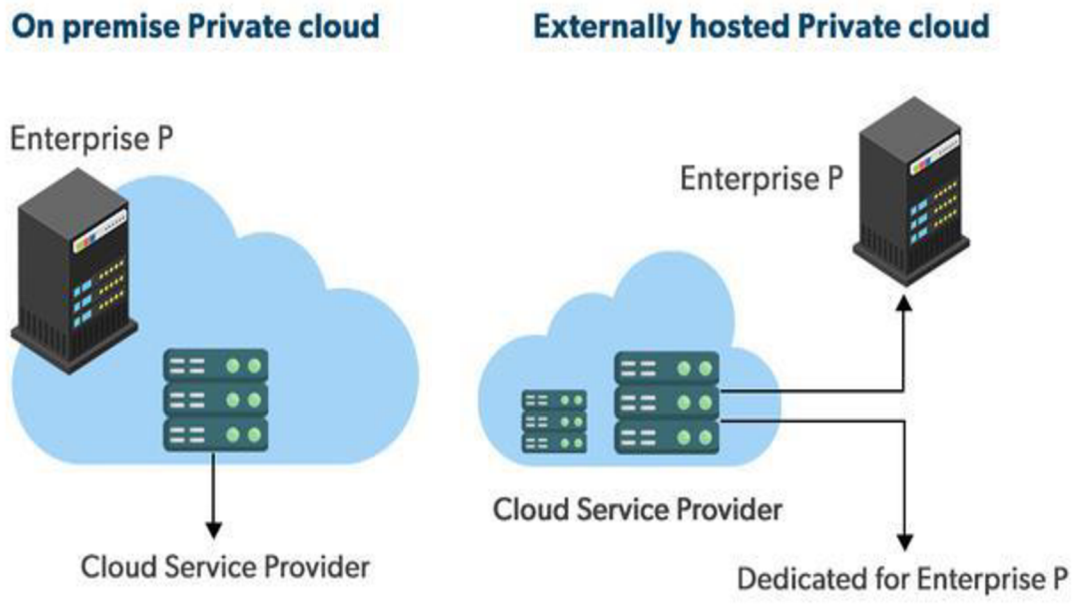


Obrázek 2: Veřejný cloud (zdroj: [8])

3.2.2 Privátní cloud

Veřejný cloud nemusí vyhovovat všem požadavkům, které jsou na něho kladeny od potencionálního uživatele nebo firmy. Tyto neduhy může vyřešit druhý typ cloudu, který se nazývá privátní cloud. V porovnání s veřejným cloudem jsou vidět rozdíly. [9] Největším rozdílem je umístění serverů, které v případě privátního serveru většinou náleží soukromé organizaci. Oproti předešlému typu má několik výhod. Cloud může benefitovat už z jeho definice. [10] Data zůstávají uvnitř organizace, což dává potenciál lepšímu zabezpečení citlivých dat. Pokud se bude mluvit o celkovém zabezpečení cloudu, tak odpověď není tak jednoduchá. [10] Veřejné cloudy spravují často přední bezpečnostní experti, investují do zabezpečení cloudu spoustu času, což způsobuje lepší zabezpečení. Při použití privátního cloudu má firma k dispozici lepší zabezpečení citlivých dat. [11] Privátní cloud se dá popsat jako uložiště, ve kterém má firma lepší kontrolu nad možným zabezpečením serverů, aplikací a sítě. Posuzování otázky bezpečnosti je tedy závislé na úhlu pohledu. [10] Může se jednat o citlivá data, které společnost chce mít na vlastních serverech díky fyzickému umístění v organizaci za používaným firewallem, nebo jde o jiná data, která není potřeba tímto způsobem zabezpečovat. Celková bezpečnost je tím pádem na pomezí, kdy privátní cloud bude bezpečnější vzhledem k určitým typům dat, ale pro celkovou bezpečnost by firma musela vynaložit dostatečné finanční prostředky, aby se dostala minimálně na úroveň světových hráčů. Právě tato skutečnost dělá privátní cloud pouze potencionálně bezpečnější,

ale v mnoha případech bude bezpečnější veřejný cloud od předních poskytovatelů. [11] Privátní cloud má další výhodu ve finanční stránce, kdy v dlouhodobém horizontu dokáže být úspornější.



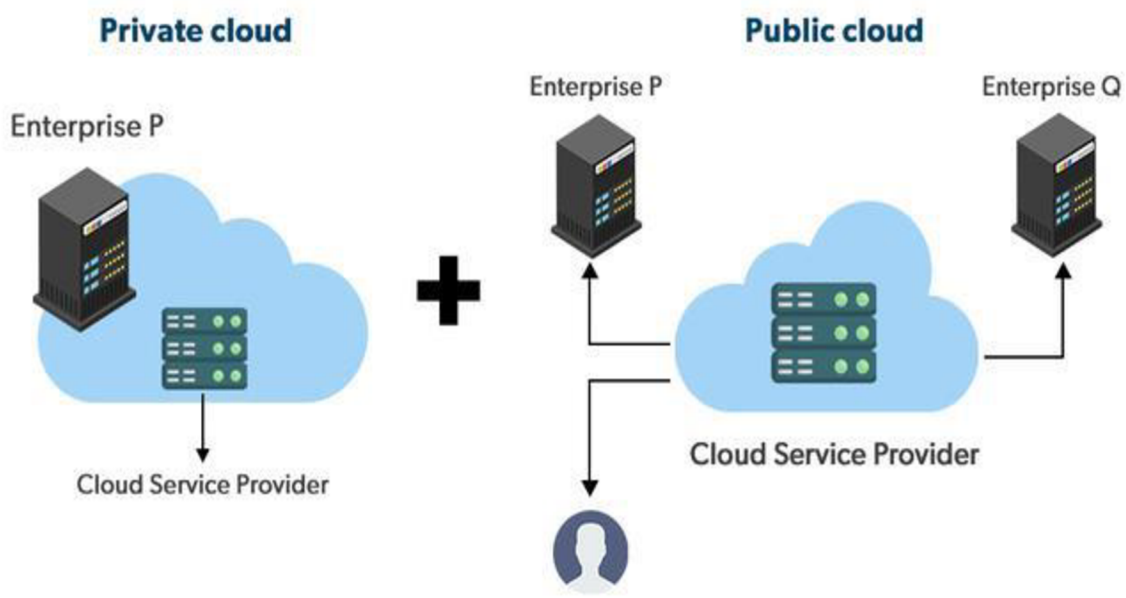
Obrázek 3: Privátní cloud (zdroj: [8])

3.2.3 Hybridní cloud

Posledním představeným typem je hybrid cloud. [7] Ten se dá považovat za nejpoužívanější typ cloudu ve firemním prostředí. V zásadě propojuje privátní s jedním nebo více veřejnými cloudy, aby získal nejlepší použitelnost pro danou společnost využívající cloudové služby.

[12] Hybridní cloud má výhody. Svou podstatou spojení privátního a veřejného cloudu umožňuje kontrolu a flexibilitu. V praxi například mohou být citlivá data ukládána do privátního cloudu, zatímco jiný obsah může být směrován na veřejný cloud z důvodu zvýšené možnosti přístupnosti z různých mobilních zařízení. Existence privátního cloudu také zaručuje přesné geografické místo, kam budou data uložena. To je výhoda v odvětvích, kde je kladen velký důraz na ochranu soukromí. Podmínky ochrany soukromí mohou nařizovat, kde se mají data ukládat nebo zpracovávat, a hybridní cloud v tomto ohledu dokáže pomoct. Poslední uvedenou výhodou zde [12] bude optimalizace nákladů. Při použití hybridní architektury se otevírá možnost optimalizovat spojení hybridního cloudu s veřejným a na základě toho optimalizovat náklady.

[12] Při realizaci hybridního cloudu se firmy mohou setkat s náročnou implementací. Samotná komunikace s možnými dodavateli a vyhodnocení firemních požadavků je stěžejní charakteristikou při realizaci hybridního cloudu. Po úspěšné realizaci přichází další nevýhoda, která se týká správy celého spojení. Využíváním více cloudů najednou vzniká složitost v komplexním vnímáním zdrojů a jejich stavu.



Obrázek 4: Hybridní cloud (zdroj: [8])

3.3 Cloud storage (základní parametry)

V této kapitole jsou rozebrány základní parametry cloudových uložišť. Podklady pro zpracování této kapitoly byly cenové konfigurátory uložišť Microsoftu [13] a Googlu [14].

Každá firma používá cloud v různých rolích. Tomu odpovídá základní dělení cloudu podle použití. Použití se může dělit na nespočet druhů od cloudu se zaměřením na velká data, cloudy podporující běh databáze, až po cloudy s vysokým výpočetním výkonem. Další možné použití souvisí přímo s touto prací. [15] Veřejné cloudy podporující Kubernetes umožňují spuštění MinIO.

Cloudu se dále může upřesňovat, jak má vypadat. Upřesnění se většinou týkají samotného cloudu. Datacentra jsou umístěna po celé planetě Zemi, a proto je potřeba lokalizovat přesné

datacentrum, pokud to poskytovatel umožňuje. [16] Například u Azure některé regiony nepodporují část produktů. Z tohoto důvodu je potřeba tento parametr nezanedbat. V opačném případě by se mohl při využívání jejich služeb vyskytnout problém.

Další možný parametr vychází z výkonnosti serveru, která se může týkat uložiště. Uložiště může používat pevné disky nebo SSD. S výjimkou typu uložiště se v cloudu vyskytuje výpočetní výkon. V případě Googlu lze výkon škálovat pomocí virtuálních CPU, pamětí a grafických karet.

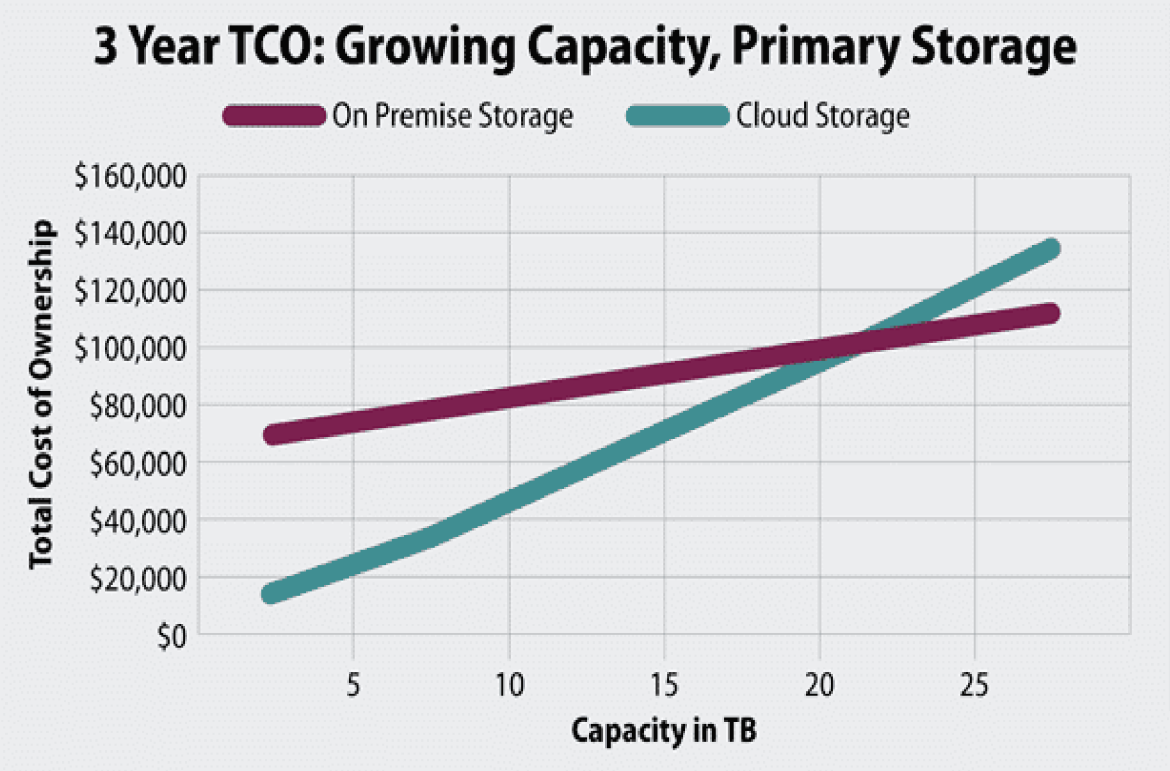
Redundance je pojem související se záchranou dat po výpadku. [17] Redundance se v zásadě stará o replikování uložených dat. Microsoft nabízí několik variant podle schématu míst uložení. Redundance se může nacházet na jediném fyzickém umístění. Toto schéma má zkratku LRS. Dalším typem je redundance, která probíhá v rámci třech zón, které mají rozdílnou fyzickou polohu v primární oblasti. V každé zóně se nachází jedna replika dat. Toto schéma je ukryto pod zkratkou ZRS a je lepší variantou než LRS. Další schémata jsou ve větších vzdálenostech a opět platí, že zabezpečení proti ztrátě dat je na lepší úrovni. Mezi ně patří schémata GRS a GZRS. GRS je druhou nejlepší volbou, replikace se provádí nejprve podle metody LRS, poté se data uloží do datacentra v sekundární oblasti, která je velmi vzdálena od primární oblasti. V sekundární oblasti se opět provede LRS. GRZS se oproti GRS liší replikací v primární oblasti, kde místo LRS používá ZRS. S rostoucím zabezpečením ztráty dat roste také cena. Odolnost se u všech zlepšuje pouze místem uložení. Lokální uložení je logicky náchylné na lokální jevy, které poté v globálním měřítku nehrají roli. Konkrétní úroveň závisí na konkrétním zadání a důležitosti dat.

3.4 Porovnání s lokálním serverem

Při zvažování, zda pro nasazení MinIO využít cloudu, je dobré porovnat cloud s lokálním serverem. Z tohoto porovnání může vyplynout, zda a jak moc výhodné či nevýhodné bude využití cloudových služeb. V následujících kapitolách budou vysvětleny hlavní rozdíly mezi těmito dvěma přístupy a budou zhodnoceny jejich výhody. V úvahu budou brány spíše firmy využívající cloud než jednotlivci. Existují firmy, které stále využívají pouze lokální servery a uvažují o přechodu na cloud. První, co bude potřeba řešit, bude to, zda se to ekonomicky vyplatí, jaké to má výhody, nebo naopak co by mohl být problém. V každém případě může být pro každou firmu použití vhodné v jiné míře. Z tohoto důvodu nelze s jistotou říci, že změnit infrastrukturu firmy na cloud je správné rozhodnutí.

3.4.1 Cena

Jedna výhoda cloudu se týká finanční stránky cloudové infrastruktury. Firma musí prosperovat, a k tomu pomáhá snižování výdajů. [18] Použitím cloudu se šetří nárazové výdaje za pořízení a poté údržbu vlastního vybavení potřebného pro chod firemní infrastruktury. Toto ušetření může hodně ulevit například startup firmám, které tyto peníze mohou investovat do svého dalšího rozvoje. [19] Postupem času nastane bod, ve kterém suma vynaložených peněz na pronájem cloudu za celé období bude srovnatelné s počáteční cenou serveru. Proto z dlouhého časového hlediska může být výhodnější vlastnit lokální server, což znázorňuje Obrázek 5.



Obrázek 5: Graf vývoje ceny lokálního serveru a cloudového uložiště (zdroj [20])

3.4.2 Zabezpečení

Důležitým faktorem pro každou firmu je zabezpečení. Nikdo nechce, aby se někdo nepovolaný dostal k citlivým údajům. Otázka zabezpečení je zde trochu kontroverzní z podobného důvodu, jako byl při problematice řešení veřejného a privátního cloudu.

Obecně je spíše náklon k názoru, že lokální server bude mít lepší zabezpečení než cloud. Tím se nijak neshazuje zabezpečení cloudu, [21] poněvadž jeho zabezpečení je také na velmi vysoké úrovni. Z toho lze vyvodit závěr, že mluvit obecně o zabezpečení není nejlepší volba. Pro srovnání bezpečnosti lze mluvit pouze u konkrétního serveru, kde jsou známa jeho fyzické či softwarové zabezpečení. Avšak také je pro správné usouzení potřeba znát citlivost ukládaných dat. Pokud by však mělo být vyzdvihnuto potenciálně nejlepší zabezpečení, kterého lze dosáhnout, má lokální server oproti cloudu výhodu.

[22] Bezpečnost lze vést na mnoha pilířích, které ve výsledku budují kvalitu zabezpečení. Servery, ať už lokální či cloudové, se musí nacházet na bezpečném místě, které je chráněno proti fyzickému vstupu neoprávněných lidí. Z tohoto důvodu je používáno ověření totožnosti při vstupu, stavební nároky na odolnost stěn, nebo ošetření přístupu pomocí pomocných infrastruktur, jako jsou například kanalizace, střecha či vzduchotechnika. S ochranou místa souvisí také rozmístění kamerového systému, který bude eliminovat slepá místa, a v případě útoku může pomoci jeho detekci nebo analýze průběhu. Dalším pilířem, na kterém bezpečnost stojí, je monitorování aktuálního stavu. Bezpečnostní technici by měli monitorovat aktuální běh, který jim pomůže rozhodnout o podezřelé aktivitě, případně urychlit odezvu na případné vniknutí. Dále by se technici měli zabývat aktuálností softwaru kvůli možným bezpečnostním chybám ve starších verzích. Bezpečnost také zvyšuje pravidelné obměňování použitého hardwaru. Dále je velmi důležité používat role pro možné uživatele, které je nutné udržovat v platnosti. Další pilíř se týká síťové bezpečnosti, která musí být vytvořena tak, aby používala různé nástroje jako firewally, ochrany proti DDOS útoku nebo kontrole IP adres. V dnešní době existuje řada způsobů, kterými hackeři mohou atakovat zaměstnance. Toho se týká poslední pilíř. [22] Je nutné dbát důraz na rozšíření bezpečnostních zvyků mezi zaměstnanci, aby nechtěně nepomohli vniku do systému. [23] Ostatně pro bezpečnost cloudu jsou také důležití samotní uživatelé. Ti musí tedy udržovat pevná a silná hesla. Dále je zvýšena bezpečnost cloudu z pozice uživatele používáním VPN, monitorováním stavu pronajatých prostředku, aby bylo možné objevit podezřenou aktivitu, nebo dvoufázovým ověřením. Bezpečnost lze také zlepšit nakonfigurováním cloudu týkající se možnosti sdílení či řízení přístupu.

3.4.3 Využití hw zdrojů

Jedním z rozdílů mezi serverem a cloudem je odlišné využití dostupných hardwarových prostředků. [19] Tento rozdíl je uskutečněn pomocí virtualizace, která se využívá na cloudu. Výsledkem virtualizace na cloudu by měla být výpočetní a paměťová optimalizace. [24] Virtualizace zařídí přiřazení jen těch prostředků, které jsou potřeba, zbytek HW prostředků je dostupný ostatním uživatelům.

3.4.4 Škálovatelnost

[18] Cloud přináší do společnosti agilitu vzhledem k jejím potřebám. Pomocí cloudu lze snadno škálovat požadované zdroje v závislosti s aktuální kondicí firmy. V případě velkých firem může být škálovatelnost užitečná například při uvedení nového produktu na trh. V tomto případě může nastat vysoký nárůst poptávky o tyto služby. S rostoucí poptávkou bude potřeba reagovat škálováním cloudu.

Na následujících řádcích budou rozebrány různé metody škálování cloudových služeb a také to, co znamená automatické škálování. V poslední řadě bude představeno, jaké charakteristiky by měla splňovat aplikace nasazená na cloudu. [25] První metodou, která se používá, je tzv. vertikální škálování. Týká se změny konfigurace daného serveru. Lze jím být například přidání dalších modulů nebo záměna jednotlivých komponent. Nejedná se o moc vhodné řešení v dlouhodobém horizontu. Druhou metodou je tzv. horizontální škálování. Zde se škáluje pomocí přidání dalších serverů. Tyto dvě metody je možné vzájemně míchat a každou využívat pro jinou část aplikace.

Pro adekvátní reagování cloudových prostředků na požadované potřeby lze používat automatické škálování. [25] Na rozdíl od ručního zde není potřeba interakce pověřené osoby, protože vše probíhá automaticky. Hned na počátku vývoje aplikace lze cloud nastavit tak, aby dokázal automaticky reagovat na aktuální potřeby a tento proces udržovat celé roky.

Další důležitou problematikou je návrh samotné aplikace, která bude nasazena v cloudovém prostředí. [26] Aby byla aplikace dobře škálovatelná, je potřeba dodržet pár zásad, které se s tímto vážou. První problematika je použití správné architektury, která má důležitý dopad na celkovou škálovatelnost celé aplikace. Aplikace by měla být založena na použití tzv. micro services. Pro snadné pochopení si můžeme tento typ architektury představit jako jednotlivé kostky, jež jsou sdružené do jedné velké aplikace. Důležitou vlastností je

nezávislost jednotlivých kostek. Každá kostka má svůj vlastní význam a funkčnost. Tím se otevírá možnost škálovat jednotlivé funkční části aplikace, tedy jednotlivé micro service, na místo škálování celkové aplikace.

[26] Pro správné vyhodnocování, v jakém stavu se aplikace nachází, jsou používány monitorovací nástroje, které sbírají aktuální informace o běhu. Tento monitoring může například usnadnit včasné odhalení problému.

Škálování cloudu dodá produktu uuEnelane nepopiratelnou výhodu v oblasti snadné správy velikosti uložiště. Umožní využívat jen takové zdroje, které jsou opravdu potřebné bez velkých prostojů. Ukládání logů tedy nebude mít problémy týkající se nedostatečné infrastruktury, namísto toho bude mít plnohodnotný typ ukládání pro jakkoli velkou klientelu.

3.4.5 Přístupnost

[18] Další funkcí cloudu je možnost přístupu ke cloudu odkudkoli pomocí různých zařízení. Tato vlastnost pomáhá kvalitnímu průběhu práce ze vzdáleného pracoviště bez závislosti na umístění zaměstnance. Lokální server také umožňuje vzdálený přístup skrze specializované programy, které jsou k tomu určeny. Tyto metody přístupu jsou závislé na použitém operačním systému.

3.4.6 Snadná údržba

V případě, že se jedná o veřejný cloud, se lze pozastavit nad výhodou snadné údržby, kterou cloud poskytuje. Už bylo vysvětleno, co je to veřejný cloud. [7] Pro připomenutí, jeho hlavním rysem je zmíněná infrastruktura patřící poskytovateli. Poskytovatel se tedy také sám stará o údržbu serverů. [18] Z toho vyplývá, že firma nemusí mít početné týmy spojené s údržbou, stačí pouze pár zaměstnanců se znalostmi týkajícími se cloudu. S tím souvisí lepší pozice na trhu práce, kde firma nepotřebuje shánět další odborníky na infrastrukturu, a místo toho se může soustředit na shánění pracovní síly v produktové oblasti.

3.4.7 Zotavení z chyb

Jedním důležitým termínem nejen v cloudu, ale celých informačních technologií je zvládnutí vzniklých chyb. Pokud nastane nečekaný problém, tak je potřeba na něj adekvátně reagovat. Na následujících řádcích budou představeny způsoby konfigurace RAID pro využití na serveru. Dalším tématem bude užitečnost zálohy dat a také principy toho, jak předejít výpadku služeb na cloudu.

Nejdříve bude vysvětlena technologie RAID. Podle definice [27] je RAID složen ze skupiny úložných zařízení, které jsou vzájemně propojeny za účelem lepšího výkonu a ochraně proti výpadku úložných zařízení. Dalším důvodem je usnadnění přístupu k datům. Úložná zařízení mohou být klasické HDD, SSD nebo SAS.

Prvně budou blíže představeny benefity. [27] RAID chrání proti výpadku disků. Pokud by nebyl nepoužit RAID mohl by se dokonce zhroutit celý systém a ochromit poskytované služby, nebo při nejlepším jen ztratit některá data. Při použití určitých typů RAID vyřazený disk nezpůsobí žádný problém. Může se vyměnit a systém funguje jako před vyřazením disku z provozu. Některé typy mohou podporovat zlepšení výkonosti z hlediska zápisu a čtení. V tomto případě řadič RAID pracuje paralelně s více disky.

[28] RAID je rozdělen do několika druhů, kde každý nabízí jinou kombinaci vylepšení výkonnosti a ochraně proti výpadku. Hlavní druhy používají alespoň jeden z principů, které RAID používá pro svoji práci. [27] Prvním principem je zrcadlení, které data připravená k zápisu na disk zapíše duplicitně na více disků. Další princip nazývající se distribuce se opět týká ukládání informací na disk. Na rozdíl od prvního uvedeného rozdělí data na části a každou část zapíše na jiný disk. Poslední princip se týká obnovy dat po zničení jednoho z disků, jež používá vypočítanou paritu. Parita poté slouží pro obnovení dat z nefunkčního disku.

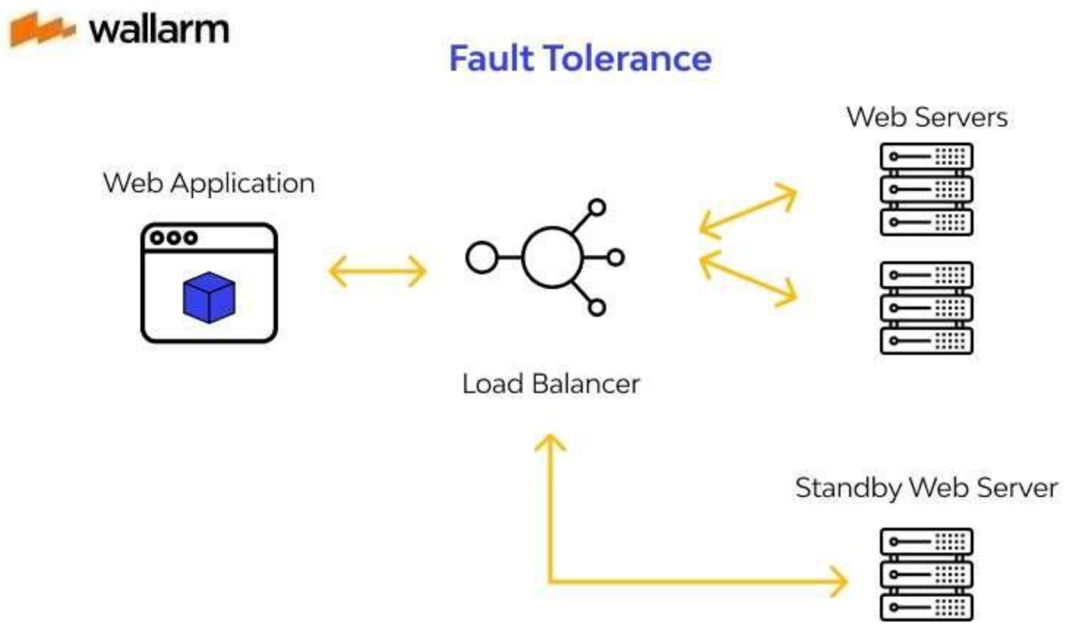
Nyní budou vysvětleny základní levely, které představují jednotlivé varianty RAID. [29] První typ se nazývá RAID 0 a jeho použití lze označit za čistě výkonnostní. Neposkytuje žádnou redundanci, tedy ani ochranu dat. Je složen ze dvou či z více disků, na které jsou paralelně zapisována data. Data jsou zapisována po blocích na odpovídající disky.

[29] Druhý RAID nazývající se RAID 1 na rozdíl od minulého je postavený na redundanci. Čtení probíhá souběžně z obou disků, díky čemuž je rychlost čtení zlepšená. V opačném případě rychlost zápisu není nijak ovlivněna, protože stejná data se zapisují na oba disky

zároveň. Jako jeho nevýhoda by se dalo označit využití pouze poloviční celkové kapacity. Složení je striktně pouze ze dvou disků.

Dalším typem je RAID 5. [29] Tento typ používá vypočítanou paritu. Paritu používá především pro zvýšení odolnosti proti výpadku disku bez nutnosti zapsání duplicitních dat. Parity se ukládají na jednotlivé disky tak, že v případě výpadku jednoho z disků je možné pomocí uložených parit znovu vytvořit data, která byla ztracena. [30] Po výpadku disku je sice systém funkční, ale je výkonnostně omezený. Dále se po výměně disku dopočítávají ztracená data, čímž je způsobeno další snížení výkonu. [31] RAID je sestaven minimálně ze tří disků. [29] Z hlediska využití celkové kapacity je tento typ výhodnější než RAID 1. [29] K tomuto typu RAID existuje obdoba RAID 6, která využívá dva disky pro paritu. Tím dokonce dokáže vyřešit výpadek dvou disků zároveň.

[29] Doteď se kapitola věnovala primárně co se skrývá pod zkratkou RAID a souvisejícími vlastnostmi a principy. RAID se běžně používá na vysoce výkonných serverech. Na následujících řádcích budou obsaženy informace týkající se použití RAID na cloudu. Jeho použití se potýká s protichůdnými názory. [32] Existují situace, kde se princip RAID uplatňuje, ačkoliv se netýká přímo diskových zařízení ale jakousi virtuální nad vrstvou. Princip pracuje s virtuálními objekty, které jsou přidružené nějaké aplikaci. Tyto objekty jsou umístěny v různých segmentech, ale pracující aplikace je vidí jako jednotný segment. Při zapisování do zmíněného jednotného segmentu virtuální vrstva zařídí zapsání do odpovídajících segmentů. Ostatní principy počítání parity a dalších jsou stejné jako u klasického RAID, pouze jsou vykonávány na úrovni virtuální vrstvy. Tento přístup aplikuje firma ShardSecure, jejíž web se stal zdrojem pro popis tohoto přístupu. [33] Podobný přístup také používá uložiště od Amazonu EBS, ve kterém je RAID zřízen na úrovni softwaru. Hlavním důvodem pro jeho použití je značné zlepšení výkonu IO operací. Z toho si můžeme převzít fakt, že RAID se minimálně na cloudu od určitých poskytovatelů může použít ve formě softwaru. Nastává nám tím otázka, jakým způsobem je v prostředí cloudu realizována výdrž proti výpadků službám, když se zde nepoužívá RAID na klasická úrovni, jak jsme si ho představili.



Obrázek 6: Odolnost proti chybám Faul tolerance (zdroj [34])

[35] Existují dva termíny, které se týkají problematiky vyřešení možných chyb, které mohou vzniknout. Jedním z nich je odolnost proti chybám, druhým je vysoká dostupnost. Nejdříve bude vysvětlena odolnost proti chybám kvůli podobnosti s principem RAID. V tomto si vezmeme na pomoc AWS S3 cloud. [35] V této problematice se mluví o replikaci dat přes více zón dostupnosti. Uložená data jsou mezi zónami synchronizována pro jejich aktuálnost. V případě, kdy vypadne používaná zóna, se využije kopie v jiné zóně a služby jsou dále funkční. Také existuje speciální typ, kdy se replikuje v rámci jedné zóny, ovšem to má oproti předešle zmíněnému nevýhody. Stejná zóna způsobí závislost na jednom místě, kde problém s tímto místem může také způsobit výpadek služeb. Oproti tomu vysoká dostupnost rozděluje pracovní zátěž mezi více serverů. [36] O toto se stará vyvažování zátěže. Pokud se opět vezme v potaz vznik chyby, vyvažování zátěže zařídí přepnutí na jiné dostupné servery. [35] Nevýhoda vysoké dostupnosti je, že zátěži trvá nějaký čas, než se přesune na jiný funkční server, také nemusí zátěž být zcela aktuální, ale hrozí tedy ztráta dat. Na druhou stranu při využití odolnosti proti chybám ztráta dat nehrozí díky synchronizovaným kopiím, ale existuje tu druhý problém, a to cena za infrastrukturu uložiště. Dalo by se říct, že si to lze představit stejně jako u RAID 1, kde kapacita pro naše data je jen část z celkové kapacity. [35] Musí se tedy pro stejně objemnou zátěž použít mnohem více uložiště než při vysoké dostupnosti, a tím pádem se odolnost proti chybám značně prodražuje.

Závěr z toho plyne takový, že jak lokální server, tak cloud může poskytovat vyřešení vzniklých chyb při jejich používání ve vysoké míře. Každý to však provádí jinými technologiemi a je na nějakém hlubším posouzení či testování, která varianta vychází lépe pro požadovaný účel použití cloudu. [27] Je vhodné si pamatovat, že RAID chrání pouze před výpadkem disku, ale ne jiných komponent. [35] Na rozdíl metody použité na cloudu dokážou vyřešit také výpadky důležitých komponent.

3.4.8 Shrnutí

Následuje Tabulka 1, která obsahuje souhrn důležitých informací obsažených v textu. Její porovnání cloudu s lokálním serverem společně s celou kapitolou lze vyhodnotit jako velkou převahu výhod využití cloudu pro nasazení MinIO.

	Cloud	Lokální server
Cena	Dle cenových konfigurátorů cloudu, lze říct, že cena bude menší než při použití lokálního serveru.	[37] Kupování celého hardwaru je finančně náročné.
Zabezpečení	[21] Na velmi vysoké úrovni.	[21] Může být lepší než na cloudu, ale záleží na konkrétním řešení posuzovaného serveru.
Využití HW zdrojů	[24] Využívá jen takovou část hardwarového výkonu, která je pro uživatele potřebná.	[19] Využívá všechny hardwarové prostředky, které má server k dispozici.
Škálovatelnost	[24] Velmi vysoká díky použití virtualizace. [37] Škálování je rychlé a relativně snadné oproti škálování serveru.	[37] Nutno nakoupit celý další server, zdlouhavé a drahé.
Přístupnost	[18] Dosažitelný odkudkoliv s podmínkou internetového připojení.	Lze přistupovat vzdáleně pomocí specializovaných programů.
Snadná údržba	[18] Cloud je servisován poskytovatelem, to znamená, že je ušetřena režie uživatele v tomto ohledu.	[21] Všechen servis je obstaráván vlastníkem serveru. Nutnost mít specializované pracovníky.
Zotavení z chyb	[35] Využívá metody vysoké dostupnosti a	[30] Používá technologii RAID. Ta chrání před

	Cloud	Lokální server
	tolerance chyby. Zaručují velmi dobrou ochranu proti výpadku služeb. Chrání proti výpadku komponent jako jsou procesory, paměti atd.	výpadkem jednoho či více disků. Důležité je, že nechrání proti výpadku jiných komponent než disků.

Tabulka 1: Porovnání cloudu s lokálním serverem (zdroj: vlastní)

3.5 Testování výkonu cloudu

V této kapitole budou uvedeny informace týkající se testování výkonu cloudu. Obecně lze říct, že výkon je důležitý aspekt mnoha výrobků, a ne jinak tomu je u cloudu. Vzhledem k široké problematice tohoto zaměření si uvedeme jen základní informace. Konec kapitoly bude věnovaný uvedení některých benchmarků, které se mohou použít. Musí se brát v potaz, že benchmarků existuje celá řada a jsou specializované na určité vlastnosti cloudu. Z toho vychází, že uvedené benchmarky jsou jen minimálním zlomkem, který má zajistit praktickou ukázku toho, jak mohou vypadat.

[38] Jedním z parametrů, který je důležitý při rozhodování o použití cloudu, je právě jeho výkonnost. Výkonnost je závislá na hardwaru, který je na cloudu přidělený. [39] Může se tedy měřit přímo výkon serveru. Ovšem cloud se používá vzdáleně a k přístupu se používají i vlastní zařízení všeho druhu. Tímto se zabývá práce [39], která měří výkon cloudu z uživatelského zařízení. To znamená, že se netestuje pouze hardware serveru, ale v testu je zahrnuta také cesta end-to-end. Výsledky studie potvrzují závislost I/O výkonu na uživatelském zařízení a poloze. Na uživatelském zařízení je to způsobeno také samotným výkonem, kde největší dopad má procesor, paměť a uložiště. Také má na výkon vliv samotná síť, přes kterou ke cloudu přistupuje. Při porovnávání výkonů různých poskytovatelů by se mělo spíše použít pro ohodnocení výkonu samotného serveru z důvodu podobných podmínek při testování.

3.5.1 SPEC

[40] Jedním z důležitých hráčů na poli testování je SPEC. Je to korporace, která se zaměřuje také na vyvíjení benchmarků použitelných na test cloudu. Jejich nabízené produkty jsou například SPECvirt, SPECweb a další. Skupina pod SPEC se zaobírá lepším porozuměním cloudového benchmarkingu a možných problému s ním spojených. Její pohled na věc se týká tří různých scénářů. Prvním scénářem je pohled z pozice firmy používající cloud pro své vlastní aplikace. V dalším scénáři se dívá na problematiku jako dodavatel cloudových služeb. V posledním scénáři se aplikuje to, co již bylo zmíněno výše, tedy porovnání výkonnosti cloudů různých poskytovatelů, který pomáhá v rozhodování o konkrétním výběru.

V následujících kapitolách bude nejdříve představena korporace SPEC, dále bude uveden jejich benchmarkový produkt, následně budou uvedeny další příklady benchmarků, které mohou být použity. Z hlediska počtu je vysoký počet různých benchmarků, a proto budou uvedeny jen některé pro představu. Benchmarky byly převážně převzaty z tohoto zdroje [40].

3.5.2 SPEC cloud IaaS 2018

[41] Jak název napovídá, patří k produktům korporace SPEC. V rámci tohoto benchmarku je testován cloud z více hledisek. Mezi ně patří výpočetní výkon, uložiště, posledním je síť. Cloud musí být typu IaaS, aby bylo možné benchmark použít. Benchmark ohodnotí cloud na základě spuštěných typů zátěže. Podle zjištěných informací z webu SPEC lze říct, že možnost použití benchmarku má vlastně kdokoli, kdo nějakým způsobem interaguje s cloudem. To znamená, že ho může použít spotřebitel, ale také poskytovatel, či například akademický výzkumník.

3.5.3 CloudHarmony

[40] Benchmark s potenciálem pro vytvoření podkladů na porovnání různých cloudů. Vzhledem k tomu, že cloud je velmi škálovatelný systém používající virtualizaci, je vhodné ho testovat také různými metodami. CloudHarmony používá rozsáhlou sbírku různých benchmarků, které se při testu specializují na určité části cloudu. Lze mezi nimi nalézt i klasický přístup, který se týká hardwarových součástek, jako je například procesor nebo

paměti. Další charakteristika, která se testuje, je síť. Zde opět využívá benchmarky, které měří základní síťové parametry, latenci a propustnost. Pro test výkonnosti mezi cloudem a uživatelem jde zde používána aplikace v prohlížeči. Poslední charakteristiku, kterou CloudHarmony testuje, jsou nepříjemné výpadky služeb.

3.5.4 C-MART

[40] Benchmark určený pro testování cloudu pomocí webových aplikací. S webovými aplikacemi souvisí technologie, které benchmark využívá. [40] Mezi ně patří typické technologie typu HTML 5.0, jQuery, AJAX nebo SQLlite. Architektura benchmarku se skládá ze samotné webové aplikace. Pro simulaci přístupu a požadavků jednotlivých uživatelů je zde obsažen uživatelský emulátor. K vytvoření prostředí pro testování je využíván server, který zařídí správnou konfiguraci. V poslední řadě je zde použito tzv. škálovatelné API. API je zde využito pro nadefinování požadovaného datacentra.

3.6 Výzvy implementace cloudu ve firemní infrastruktuře

Při budování cloudového řešení běžně vznikají problémy. V této kapitole budou vysvětleny některé konkrétní výzvy, které vychází z průzkumu, provedeného firmou Flexera. [42] Průzkum probíhal mezi respondenty z různých firem se zkušenostmi s cloudem, kteří měli buď rozhodovací pravomoci, nebo byli jen obyčejnými uživateli.

Příloha 1 ukazuje výsledky průzkumu na obrázku NejvětšíVýzvyImplementace.png. Mezi největší výzvy spadá správa výdajů, zabezpečení a neznalost odbornosti cloudu. Existuje však mnoho dalších výzev, které se s touto problematikou pojí. V následujících kapitolách se podíváme na některé z nich.

3.6.1 Správa výdajů

[42] Největší problematikou se stala práce starající se o finanční stránku cloudu. Firmy se snaží snížit náklady kladené na cloudové služby, ale i přes to se jedná o jeden z největších problémů. Podle Flexery respondenti uvádí, že jejich stav výdajů je o 18 % vyšší, než tomu

mělo být podle výhledu. Existují metody, které pomáhají snížit výdajové zatížení firem. Mezi ně může patřit například využívání slevových akcí. Pokud firmy správně a efektivně monitorují svoje využití cloudu, dokážou poté nastavit automatizované vypínání zátěže v určitých hodinách. Způsobů, jak snížit náklady na provoz cloudu, existuje mnoho.

3.6.2 Uživatelská bezpečnost používání cloudu

Další palčivým tématem při implementaci cloudu a jeho dalším používání je řešení zabezpečení. Existuje mnoho rizik z pohledu uživatele. [42] Může nastat například prolomení zabezpečení účtu, poškozený přihlašování, únik citlivých dat, hacknutí API, nebo zneužití účtu. Metody, kterými lze lépe zabezpečit cloud z pohledu uživatele, již byly zmíněny výše.

3.6.3 Nedostatek odbornosti

[42] Při přecházení na cloud se může stát problémem odborná neznalost zaměstnanců s cloudovými službami. Na základě této neznalosti je poté omezena správné řešení požadavků, které firma má. S čím dal větším rozvíjením cloudu se tento problém stále prohlubuje a v důsledku může firmám způsobovat finanční ztráty. Řešení tohoto problému je například přijmutí nových IT specialistů, kteří mají zkušenosti. Tito zkušení odborníci do firmy přináší i další pozitiva. Méně zkušení či mladí pracovníci mohou vedle zkušeného kolegy růst. [42] Toto řešení je pro menší či střední podniky finančně až moc nákladné. Proto by firma měla zvážit zavedení automatizačních nástrojů, například Puppet nebo Chef. Tyto nástroje automatizují běžné úkony, které IT specialista provádí při práci s cloudem. Mezi ně by se dalo zařadit například sledování vzorců, využití zdrojů, automatické zálohování apod. Další důležitá forma zkvalitnění znalostí ve firmě je například využití nabídek různých školení.

3.6.4 Kvalita sítě

[42] Další faktor, který se stává problematickým, je získaní požadované kvality sítě. Aby nedocházelo ke zpomalování, či dokonce k výpadkům kvůli kvalitě sítě, investují velké podniky do síťové infrastruktury s hlavním cílem zvýšit šířku pásma. V případě menších

podniků se více dívá na výdaje v oblasti cloudu a toto investování nemusí být v plně potřebné šíři.

3.6.5 Výkon

[42] Důležitým faktorem pro správný chod služeb v cloudovém prostředí je výkon řešení a možné výpadky, které jsou svázány s použitím cloudových technologií. Následky výpadků můžeme zmírnit pomocí výběru správného dodavatele. Důležitým aspektem je možnost monitoringu v reálném čase. Monitoring umožňuje na základě dat, které poskytuje, zjistit aktuální kondici cloudu, případně vyvodit nějaká změny, které pomůžou celkovému chodu.

3.6.6 Migrace

[42] Dalším velkým problémem se stala migrace. Tato problematika se týká nasazení aplikace do cloudového prostředí. Lze uvažovat aplikaci, která se používala na jiném druhu infrastruktury a v současné době trend a postup technologií umožňuje použití cloudu. Nasazení aplikace do cloudu může znamenat zdlouhavý proces. Může zde vzniknout velké množství vzniklých problémů. Dalšími složitými vyskytujícími se problémy je zabezpečení, mapování složitostí aplikace, prostoje aplikace, pomalá migrace dat a další.

3.7 Porovnání konkurence MinIO

V této kapitole bude porovnáno objektové uložiště MinIO s cloudovými objektovými uložišti od Amazonu, Googlu a Microsoftu. Sledované parametry jsou především podpora S3 API, podpora streamingu a maximální velikost objektu, který může být uložen. Jako doplňující parametry jsou zde uvedeny podpora, jež je velmi důležitá v provozu používajícím cloudovou infrastrukturu. Také je zde uvedena případná minimální velikost uložiště, která může rozhodovat o rozhodnutí využití konkrétního produktu oproti konkurenci z důvodu malého požadavku na místo. Z následující Tabulka 2 bude vycházet zbytek kapitoly.

	MinIO	Amazon Simple Storage	Google Cloud Storage	Azure blob storage
S3 API	[43]Plně kompatibilní s Amazon S3	[44]Plně kompatibilní	[45] Dokáže podporovat nástroje určené pro práci s S3 API	[46] Částečně kompatibilní, nutnost S3Proxy. [47] použití MinIO
Typ uložiště	[43] Objektové	[48] Objektové	[49] Objektové	[50] Objektové
Minimální objem uložiště	[51] 200 TiB – placená, neomezené – verze zdarma	neznámý	[52] Nabízí 15 GB každému účtu.	Neznámý
Streaming	[53] Podporuje	[54] Podporuje	[55] Podporuje	[56] Podporuje
Podpora	[51] portál, telefonická, webové konference	[57] Automatizační nástroje, optimalizace nákladů, nepřetržitá odborná podpora, poradenské služby	[58] 24/7 podpora, různé jazyky, API podpory	[59] 24/7 podpora, telefon, email, Azure portály, speciální API, kurzy od Azure
Cena	[51] placené 20 $ za TiB, další	[60] závisí na počtu operací,	[61] závisí na pronajaté	[50] Závisí na metodách

	MinIO	Amazon Simple Storage	Google Cloud Storage	Azure blob storage
	příplatky za konfiguraci, podporu	odvíjí se od dalších parametrů, typů uložiště.	kapacitě, jednotlivých datacentrech, druhy uložišť	redundance, regionech, datacentrech, velikosti uložiště, počet provedených operací
Maximální velikost objektu	[62] 5OTiB po rozložení na více objektů, maximální velikost objektu 5TiB, Velikost části 5MiB – 5GiB	[63] Maximální velikost 5TiB, velikost části 5MiB – 5GiB	[64] Maximální velikost 5TiB, velikost části 5MiB – 5GiB	[65] Maximální velikost složeného z bloků 190,7TiB, max bloku 4000 MiB

Tabulka 2: Porovnání konkurence MinIO (zdroj: vlastní)

Konkrétní produkt se musí vybírat podle vztahu k použití v praxi. Aplikace či uživatel má určité požadavky, které uložiště musí obsahovat, či se k nim v některých případech alespoň přibližovat. Již byly představeny požadavky produktu uuEnelane na požadovanou technologii, kde byla zmíněna nutnost využívání streamů. Tuto vlastnost splňují všechna uvedená uložiště. Dalším velmi důležitým parametrem byla maximální velikost jednotlivých objektů. Zde opět všechny porovnávané uložiště velmi převyšují podmínky, jelikož MinIO má největší podporovaný objekt o velikosti 5OTiB a Microsoft dokonce 190,7TiB. Dalším velmi užitečným parametrem je podpora S3 API. S tou má největší problém blob storage od Microsoftu, kde pro správné fungování API musí být použita ještě S3Proxy. Výhody využití S3 API budou zmíněny v jedné z dalších kapitol. Také jsou v Tabulka 2 uvedeny některé druhy podpor, které jsou k dispozici u jednotlivých firem. Firma si může zaplatit služby podpory, jež poté budou důležitou oporou při řešení problémů nebo dalších témat souvisejících s využívaným uložištěm. Například Amazon poskytuje také podporu pro optimalizaci nákladů, která může být pro firmy z finančního hlediska užitečná. Konkrétní detailní podoby podpor se nachází na stránkách poskytovatelů. V tabulce jsou

uvedeny jen ukázkové druhy či možnosti, jak lze s poskytovatelem komunikovat. V tabulce není uveden žádný řádek týkající se výkonu. To má odůvodnění v již probrané kapitole benchmarku cloudu. Existují benchmarky MinIO, jako je například [66], kde naměřili se zapnutým šifrováním 161,99 GB/S GET a 114,7GB/S PUT. Tyto hodnoty ale byly naměřeny v roce 2019 na nejlepším hardwaru, který byl k dispozici na Amazon AWS. Na jiném hardwaru budou výsledky jiné. Podobná metodika se týká i ostatních poskytovatelů. Pro adekvátní porovnání by bylo potřeba otestovat jednotlivé cloudy a vztáhnout jejich výkon k ceně pro lepší analyzování celkové situace. Také se však musí vzít v potaz možnosti využití MinIO právě na každém poskytovateli, který je vhodný k potřebám. Z tohoto vyplývá mnoho možností, jakými se dá konkrétní konfigurace sestavit.

Ve sledovaných parametrech si jsou uložiště velmi podobná a závisí spíše na konkrétní preferenci. Ovšem také lze používat více druhů zároveň například pro záložní uložiště, a v tomto případě právě S3 API poskytuje skvělou službu pro spotřebitele cloudu. MinIO je tedy vhodné pro použití v produktu uuEnelane.

4. Virtualizace

V práci již zazněl termín virtualizace a zběžné vysvětlení jejího použití v oblasti cloudu. Ovšem použití virtualizace se neváže pouze ke cloudu, ale k mnohem více oborům. V této kapitole bude vysvětlena problematika důležitých vlastností virtualizace v použití na cloudové infrastruktuře. Také budou uvedeny základní typy virtualizace z důvodu lepšího pochopení celého principu.

4.1 Vlastnosti v cloudu

V této kapitole uvedeme důležité rysy virtualizace pro použití v Cloudu.

4.1.1 Izolovanost

[67] Tato vlastnost je extrémně důležitá v použití na cloudu. Izolovanost cloudu si lze představit jako skříň se šuplíky. Každý šuplík má jeden operační systém a je zde puštěná nějaká aplikace. Všechny šuplíky jsou ve skříni odděleny stěnou, kdy jeden šuplík nezasahuje do jiného. To samé platí na cloudu, kde jsou všechny virtuální počítače izolovány od okolí a které mají dostupnou svoji část hardwarových prostředků. Tato vlastnost způsobuje efektivnější práci na cloudu, kde napomáhá při ochraně dat proti narušení. Dále vytváří prostředí pro efektivní práci se zdroji.

4.1.2 Přenositelnost

[67] Poskytuje bezproblémové spuštění stejných aplikací na jednotlivých virtuálních počítačích.

4.1.3 Snadná konfigurace

[67] Velkým znakem virtualizace je usnadnění vytvoření potřebné infrastruktury. Samozřejmě uživatel vytvářející tuto strukturu musí mít znalosti pro takovou konfiguraci. Bez těchto znalostí by mohl mít problémy s pochopením konfiguračních vlastností. [67]

V každém případě virtualizace umožňuje IT specialistovi sestavit infrastrukturu, či ji změnit, během krátkého časového úseku, který se může pohybovat v řádech pár hodin. Samozřejmě čím budou vyšší nároky na cloud, tím se může také zvýšit doba konfigurace. Virtualizace však hraje velkou roli a celý proces velmi usnadňuje.

4.1.4 Lepší rozložení zátěže

[67] Virtualizace umožňuje rozložit příchozí požadavky na odpovídající servery. Tím se dokáže zrychlit celá služba.

4.2 Typy virtualizace

Pro lepší pochopení virtualizace budou ukázány její typy, což napomůže pochopení, jak virtualizace funguje, či proč ji používat. Posledním důvodem vysvětlení rozdělení virtualizace podle typu je použití jednotlivých druhů v cloudu. [67] Může se totiž říci, že v cloudu je virtualizace použita na více vrstvách. Využívá se u komponent serveru, sítě, uložiště a spuštěných aplikací.

4.2.1 Virtualizace serveru

Tento typ, jak název napovídá, se týká serveru. [67] Virtualizace rozděluje jeho hardwarové prostředky, kterými mohou být například kapacity disků, výkon procesorů nebo paměť RAM, mezi nasazené virtuální stroje. V případě cloud computingu se mluví také o hardwarové virtualizaci. Obě tyto virtualizace jsou shodné, jediným rozdílem je jejich název.

4.2.2 Virtualizace sítě

[67] Tento typ umožňuje vytvořit rozhraní, přes které může být konfigurováno více komponent. [24] Použití této virtualizace snižuje počet potřebných zařízeních, jako například přepínače, kabely či routery.

4.2.3 Virtualizace uložiště

[67] Při virtualizaci uložiště se rozděluje dostupná hardwarová kapacita disků mezi různé virtuální stroje. Při použití na cloudu je toto uložiště dostupné skrze API, které je k tomuto vytvořeno nebo další možností je použití specializovaného softwaru od poskytovatele cloudu. Pro mnohé uživatele bude rozhodně přínosnější a jednodušší použití právě zmíněného softwaru od poskytovatele.

4.2.4 Virtualizace počítače

[67] Virtualizaci počítače se používá pro vzdálenou práci například s firemními zařízeními. Nabízí bezpečnou cestu k práci z jakéhokoliv kouta světa, a tím také flexibilitu týkající se ztráty závislosti být na určitém místě v daný časový okamžik.

5. MinIO

V této kapitole bude představeno MinIO objektové uložiště. Nejdříve bude MinIO představeno. Po představení se představí velká síla MinIO, která tkví v podpoře S3 API od Amazonu. Následně se kapitole bude týkat kódování, které se zde používá, a jako poslední bude představen princip cloud native.

5.1 Představení

[68] MinIO je open source produkt, který nabízí vysoce výkonné objektové uložiště, které lze aplikovat také v distribuované formě. Samotné MinIO dokáže fungovat na veřejných cloudech, ale také může být implementováno jako privátní cloud na soukromém hardwaru. [69] Tato univerzálnost je daná podporou kubernetes. To znamená, že MinIO lze spustit na jakékoli spuštěné instanci kubernetes [70] ve verzi k8s. Dále MinIO podporuje S3 API, což bude lépe vysvětleno v následující kapitole.

[71] Uložiště disponuje několika výhodami. Jednou z velkých výhod je výkonnost, která aplikaci, jež používá toto uložiště, umožňuje rychlejší zpracování dat. Ve spojení s další výhodou škálování a distribuovanou architekturou je MinIO vhodné pro různé obory zacházející s velkými daty. Vhodnými obory jsou analýzy velkých dat, strojové učení či další obory, jež potřebují uložiště, které bude rychlé. Dále je MinIO způsobilé pro využití v roli archivačního nebo zálohovacího nástroje, jež jsou umožněny odolností proti chybám. [43] Uložiště je také platné jako testovací nástroj, jež svou univerzálností v možnosti nasazení, podporovaným S3, poskytuje dobré podmínky pro testování či vývoj.

5.2 API

[72] MinIO v rámci světového trhu poskytuje nejlepší podporu k S3 API od Amazonu. [73] API umožňuje aplikaci práci s uložištěm. Obsahuje funkce pro správu objektů a bucketu. To můžou být například operace pro vznik a smazání. S ohledem na tuto podporu bude představena práce s metodami, jež tvoří důležitou část pro produkt uuEnelane při práci s S3 API, ačkoliv součástí API je mnohem větší škála funkcionalit pro interakci s MinIO. [74] Pro komunikaci s API bude využit MinioClient, který obsahuje dané metody.

5.2.1 getObject(GetObjectArgs args)

[74] Pomocí tohoto dotazu na API lze získat data konkrétního objektu, který je identifikovaný pomocí třídy pro atributy souboru. Mezi ně patří jméno konkrétního objektu a jméno bucketu, ve kterém se nachází. Data jsou přijata ve formě streamu.

5.2.2 putObject(PutObjectArgs args)

[74] Důvod použití je pro upload objektu do MinIO. Opět tu lze nalézt v parametrech metody třídu, která se zapouzdřuje vlastnosti každého objektu. Mezi tyto vlastnosti zde opět patří jméno objektu a bucketu, dále informace o velikosti, konkrétní stream dat, který se má nahrát do MinIO.

5.2.3 removeObject(RemoveObjectArgs args)

Pokud již v MinIO jsou nahrané nějaké soubory ve formě objektu, v některých případech je také potřeba je odstranit. [74] Tato metoda je zodpovědná za smazání objektu v MinIO a stejně jako předchozí obsahuje parametrech metody třídu. Její funkce je stejná jako u předchozích. Tato třída má další zajímavý atribut označující verzi daného objektu.

5.2.4 statObject(StatObjectArgs args)

V souvislosti s použitím na produktu uuEnelane bude velmi důležitou součástí API vracení metadat o konkrétním objektu. [74] MinIO pro tuto potřebu obsahuje tuto metodu, která vrací právě tyto informace. Ve třídě obalující argumenty se opět nachází jméno bucketu a objektu, a k tomu dále například verze.

5.3 Struktura objektového uložiště

Pro správné uchopení základní struktury, jakou se data ukládají, je potřeba vymezit dva pojmy. Těmi jsou objekty a buckety.

5.3.1 Bucket

[75] Uložiště MinIO je složeno z bucketů a objektů. Bucket lze chápat jako kontejner, ve kterém se nacházejí uložené objekty. Je tedy obdobou adresáře v operačních systémech.

5.3.2 Objekt

[75] Objekty se nachází v bucketech. Pod objektem si lze představit hned několik věcí. Můžou to být například videa, obrázky a zvukové soubory. To znamená, že objekt má vždy reprezentaci v původním souboru, který byl nahrán. Jsou to tedy vždy soubory, které byly na cloud nahrány, na rozdíl od bucketů, které dané soubory pouze třídí do určitých skupin podle preferencí uživatele.

5.4 Erasure Coding

Již v práci zazněly metody zotavení z chyb při běhu cloudu. MinIO v této problematice používá Erasure Coding. [76] Erasure coding se stará o redundanci dat a jejich dostupnost. [77] Lze říct, že Erasure coding přináší vysokou dostupnost. [76] Tato technika by se dala připodobnit k technikám RAID z důvodu, že MinIO používá také paritní informace k obnově ztracených dat. Celá problematika se však rozděluje do konkrétní konfigurace, která určuje, jak moc bude uložiště odolné proti výpadku. V první části MinIO server sloučí celkový objem disků do erasure set. Konkrétní velikost a počet erasure set vytváří MinIO automaticky.

[76] Disky v erasure set se rozdělují do datových a paritních částí, můžou být také celé paritní a celé datové disky. V případě, kdy se ukládá objekt, je rozdělen na dvě části, a ty jsou následně zapsány do odpovídajících částí: parita do paritní části a data do datové části. Velikost paritní části může nabývat od nulové parity až do parity, která nabývá polovině erasure set.

[76] Samotné zvládání výpadku je závislé na minimálním počtu funkčních disků. Tomuto počtu se říká K. V praxi to znamená, že pokud nastane situace, kde je počet disků menších než K, není jedna z operací umožněna. V případě čtení hodnota K substituuje read quorum, které definuje podmínky, kdy je dostupné čtení. V opačném případě K substituuje write

quorum tedy minimální počet disků potřebných pro úspěšný zápis. [78] Velikost K pro případy čtení a zápisu se může lišit.

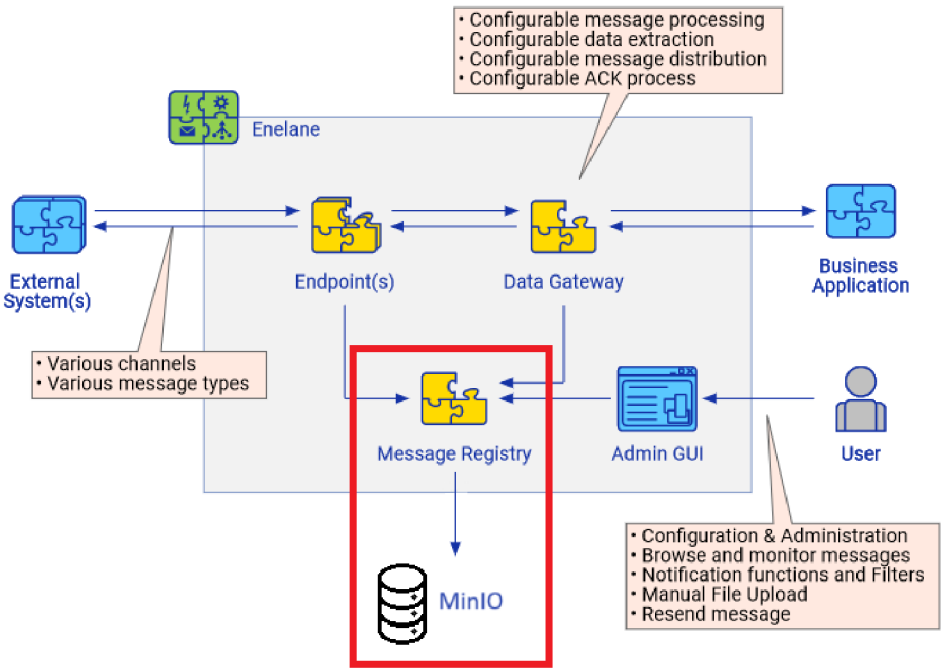
5.5 Cloud native

[68] MinIO je postaveno na cloud native architektuře. [79] Tato architektura se vyznačuje využíváním cloudové vysoké škálovatelnosti, odolnosti proti chybám a flexibilitě. Uložiště je tedy napsané s ohledem na budoucí využití cloudu pro její běh. Dále je obecně aplikace naprogramovaná tímto způsobem složena z mikro služeb, které jsou následně vhodné ke kontejnerizaci a k nasazení na cloudu za použití nějakého orchestračního nástroje. [68] V případě MinIO to zaručuje hostování velkého množství tenantů na společném hardwaru.

6. Návrh a vývoj aplikace

V následujících kapitolách bude představen praktický projekt, který ukazuje možnost implementace MinIO do stávající aplikace. Řešení projektu se dá rozdělit na několik fází, kterými je úspěšné spuštění a osvojení práce s objektovým uložištěm MinIO, dále naprogramování aplikace, která využívá klienta pro komunikaci s objektovým uložištěm. Dalším krokem je vytvoření testů, které ověřují správnou funkčnost každého modulu aplikace.

6.1 Představení



Obrázek 7: Použití MinIO v rámci produktu uuEnelane (zdroj: vlastní)

Aplikace bude vytvořena s ohledem na aplikaci uuEnelane, jež je adeptem na využití MinIO ve svých službách. Aplikace uuEnelane je již velmi rozsáhlou aplikací, jejíž vnitřní struktura je velmi komplikovaná a provázaná. Tato problematika vede k velmi složité implementaci pro studenta, jež není součástí vývojářské skupiny zabývající se vývojem této aplikace.

Po konzultaci se zástupcem firmy to v praxi znamená vytvoření úplně samostatné aplikace, jež bude vyvíjena podle nastíněných podmínek či vlastností, které jsou spojeny s produktem uuEnelane. Hlavní nastíněné podmínky jsou využití objektového jazyka Java, jež je použit právě v produktu uuEnelane. Obrázek 7 ukazuje architekturu produktu. Ohraničená část ukazuje místo, která je předmětem vypracování aplikace. Samotná aplikace bude muset podporovat část rozhraní, které je využito v produktu. Konkrétní části budou vysvětleny dále v práci. [1] V současné době je rozhraní využité pro komunikaci s jinými typy technologií určenými na uchovávání potřebných souborů. Úspěšná aplikace by tedy měla zachovat strukturu požadovaného rozhraní a na jeho základě postavit vrstvu aplikace, která bude úspěšně komunikovat s instancí objektového uložiště MinIO. Tato nová vrstva bude na základě jednotného rozhraní sloužit jako vzor pro zpracování do reálného produktu, kam přinese variabilitu v možnostech výběru podporovaného uložiště. Soubory určené k zachování budou ve formě objektů uloženy do jednoho bucketu. Důvod je takový, že v době psaní práce není potřeba třídění mezi více bucketů. Součástí vytvoření bucketu je také nutnost vytvoření pravidel týkajících se životnosti objektů. Používání pravidel odstíní tuto problematiku od aplikace a zaručí snadné odstraňování objektů, namísto programování speciální logiky, která by se o to starala. Tato pravidla budou odstraňovat objekty, které mají svůj život delší než expirační dobu, případně jsou starší než na pevno zadané datum.

Součástí aplikace bude vytvořené jednoduché API, které bude odpovídat funkcionalitě již zmíněného rozhraní. Na tomto API se budou přijímat soubory, které následně projdou skrz aplikaci do MinIO. Tato varianta získávání souborů byla zvolena z důvodu rozšířenosti a univerzálnosti využívání API. Nabízí se také varianta, kde by se na API posílala pouze data ve formě JSON formátu. To by ale postrádalo přiblížení ke skutečnému využití produktu. Vybraná varianta posílající celé soubory bude lépe přibližovat zasazení do skutečného produktu. Aplikace tedy bude fungovat následujícím způsobem. V případě ukládání se nejprve na API přijme soubor, který dále bude validován, rozparsován na jednotlivé části, následně bude po tomto zpracování poslán k uložení. V ostatních případech bude přijat požadavek ve formátu JSON, následně bude obsloužen odpovídajícími vrstvami aplikace, opět bude zvalidován, a až poté bude vykonána konkrétní akce s instancí MinIO a vrácen výsledek. Implementace také bude ošetřovat všechny výjimky, které se za běhu aplikace budou moci vyskytnout, a na základně nich buď informovat o chybě do konzole, nebo budou přímo vystaveny jako http odpověď uživateli.

Součástí aplikace budou také testy. Testy budou primárně jednotkové a budou testovat všechny důležité metody, které jsou v cestě od přijetí požadavku přes jeho vyřízení na straně

MinIO klienta až po úspěšné vrácení odpovědi, či chybové http odpovědi. V rámci metod se bude testovat správnost výstupů či vyhazování správných výjimek. Součástí budou také testy, které otestují aplikaci jako celek, tedy celkovou spolupráci všech částí aplikace posouzených na dosažení očekávaných výsledků.

6.2 Použité technologie

Technologie	Verze	Popis
MinIO	RELEASE.2023-12-09T18-17-51Z	Verze použitého image MinIO
Java	21	Použitá Java na aplikaci
Spring Boot	3.2.1	Použitý framework
InteliJ IDEA	2022.3.2	Použité IDE
Postman	-	Testování API endpointů.
Docker desktop	-	Spuštění kontejnerů MinIO a aplikace.
Docker engine	20.10.12	-
Maven	3.8.4	Build aplikace a závislosti

Tabulka 3: Tabulka použitých technologií (zdroj: vlastní)

Následující odstavce budou vycházet převážně z Tabulka 3. V rámci vytvoření celého praktického projektu bude použito hned několik technologií, které umožní či usnadní práci. Použité technologie v projektu jsou vybrány s ohledem na produkt. To v praxi znamená jazyk Java ve verzi 21. Verze 21 bude použita především kvůli její aktuálnosti. Java bude pracovat s frameworkem Spring Boot, který bude velmi důležitý v snadném a rychlém vývoji celé aplikace. Jeho hlavní předností bude vytvoření REST API, na kterém se budou přijímat požadavky vyslané z aplikace Postman. Tato aplikace zajišťuje kvalitní dotazování na jakékoli API. Aplikace je velmi snadno použitelná a intuitivní. Na potřebu dotazování je z těchto důvodů použití velmi vhodné. Především MinIO bude muset běžet v době vývoje ve formě kontejneru na lokálním počítači, později bude moci být takto kontejnerizované nasazeno na dedikovaný server. Ke kontejnerizaci se budou používat dvě velmi svázané

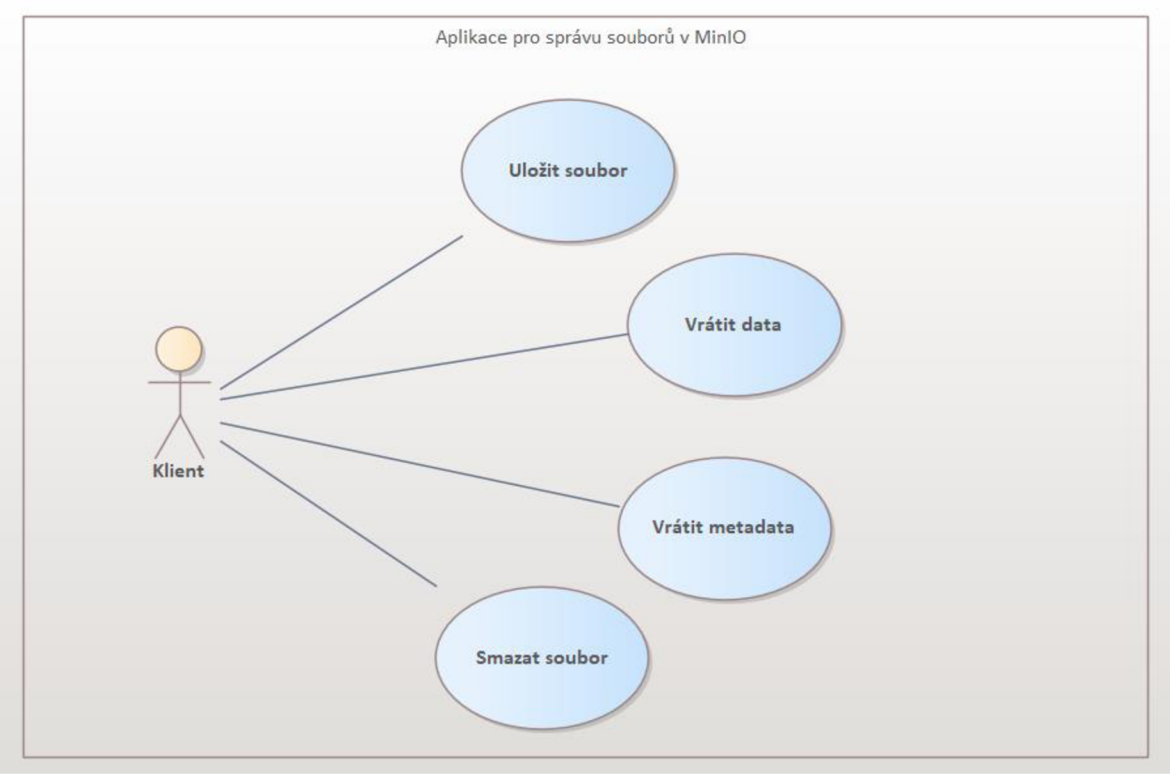
technologie. Konkrétně bude použit Docker Desktop a Docker engine. Tabulka 3 ukazuje image, podle kterého bude kontejner MinIO vytvořen.

V rámci aplikace budou důležité taky konkrétní závislosti na použité další knihovny. K tomuto účelu bude celá aplikace zastřešena použitím Maven. Maven umožní snadné zásobování projektu požadovanými knihovnami, později také vytvoření buildu celé aplikace.

Tabulka 3 ukazuje dohledané verze. Verze jsou velmi významné především kvůli nastavení podmínek, při kterých aplikace bude vytvořena a pod kterými bude správně fungovat. Jak se software, který bude využívat aplikace, vyvíjí, tak současně s tím přichází změny, které mohou být i většího rozsahu. Například mohou změnit způsob dotazování, nebo odebrat podporu některé z použitých funkcí. Právě z těchto důvodů je zaznamenání použitých verzí velmi důležité. Mohlo by se totiž stát, že při změně verze nějaké použité technologie by se mohlo způsobit omezení funkčnosti, či by mohlo dojít k úplnému odstavení funkce. Tato charakteristika technologií a verzí bude v budoucnu umožňovat opakované vytvoření podobné aplikaci či její spuštění a otestování.

6.3 Use case

Aplikace bude sloužit pouze jako prototyp, který bude předlohou pro skutečnou implementaci v produktu uuEnelane. To znamená, že funkce aplikace budou pouze v rozsahu požadavků na schopnosti manipulace s daty pomocí vybraného objektového uložiště. Konkrétní uživatelský přístup k aplikaci bude realizován pomocí REST API, které bude kopírovat podporovanou manipulaci s objekty v uložišti. Každá obsluha požadavku na API má v URL pro názornost obsaženo jméno operace odpovídající názvu v DAO vrstvě. Na konkrétní řešení nastavení URL nebyly z Unicornu žádné požadavky. V následujících kapitolách budou vysvětleny konkrétní případy, ve kterých uživatel bude interagovat s API.

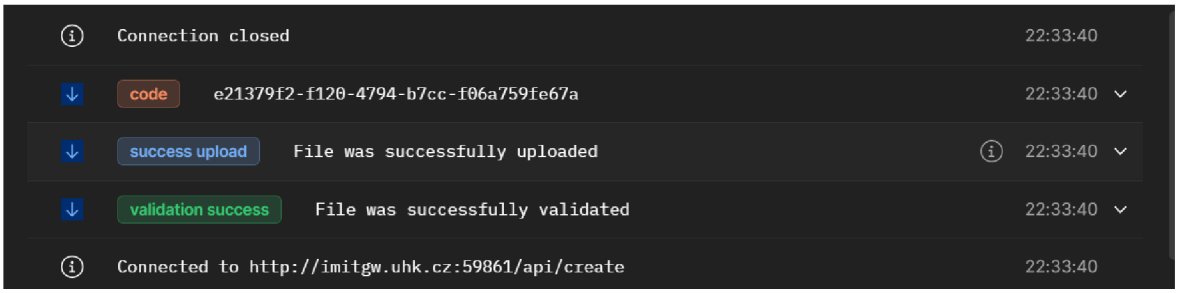


Obrázek 8: Use case diagram mapující funkcionality projektu (zdroj: vlastní)

6.3.1 api/create

UuEnelane potřebuje uložit message v průběhu jejího životního cyklu. V aplikaci bude ukládání realizováno na adrese api/create, která poté spustí předzpracování souboru přijatého na API, poté má na starosti asynchronní spuštění ukládání do MinIO. Soubor přijatý na API musí mít identifikaci souboru pod parametrem file. V opačném případě se vyskytne problém s přijetím.

Vstupní soubory musí být podporovaného formátu a jejich obsah musí splňovat všechny povinné atributy. Virtuální Příloha 1 s testovacími soubory ukazuje strukturu několika typů souborů. Soubory musí být formátů JSON nebo XML, aby je aplikace dokázala zpracovat. Formáty byly zvoleny z formátů, které podporuje produkt uuEnelane. Soubory musí být menší než 500 kilobajtů, naopak nesmí být prázdné. V případě, že jsou splněny formáty a velikost, je nutné soubor rozparsovat na metadata. Zde jsou určené atributy, které jsou vyžadované pro zápis do metadat. Mezi ně patří messageId, sendingAccountID, receivingAccountID a numberOfCertificates.



Obrázek 9: Výstup při správném uložení (zdroj: vlastní)

V procesu celého zpracování a ukládání bude uživatel informován o průběhu procesu. Obrázek 9 ukazuje výstup aplikace při úspěšném uložení. Úspěšné vykonání požadavku vrací http status 202. Je třeba si povšimnout vráceného kódu. Pod tímto kódem je soubor uložen v uložišti. V souvislosti s dalšími koncovými body API je velmi důležité tento kód zapamatovat pro případné další dotazy nad konkrétním souborem. Uložení souboru může selhat na několika místech. Může se například vyskytnout problém s přijetím souboru či jiný interní problém, jako například vyhození výjimky od MinIO. Z tohoto důvodu bude průběžné informování klienta zahrnovat také zpětnou vazbu o problému. To je důležité zejména k identifikaci problému a jeho následnému vyřešení. Jelikož ukládání probíhá asynchronně, bylo nutné vyřešit problém s obeznámením uživatele o stavu uložení. To je ošetřeno pomocí SseEmitteru, který posílá uživateli podrobné informace o stavu ukládání.

Hláška	Řešení
File is empty (400)	Soubor je prázdný, pro přijetí je nutné, aby soubor obsahoval požadovanou strukturu.
The file is larger than is allowed (400)	Soubor je větší než 500 kB. Soubor je nutné zmenšit.
In the file is syntax error (400)	V rámci souboru je chyba v syntaxi, jež může být například chybějící středník.
This file format server cannot processed (400)	Soubor je nepodporovaného formátu. Pro zpracovaní je nutné transformovat soubor do správného formátu.
Required attribute missing in the file (400)	V souboru chybí povinný atribut. V rámci výjimky se vypisuje také jeho konkrétní název.

Hláška	Řešení
Problem with file transfer to api (500)	Naskytl se problém s přijetím souboru.
There is error response from repository during uploading, file wasn´t upload. (202)	Společně s hláškou se vypíší také podrobnosti souboru. Důležité zkontrolovat velikost metadat nebo název bucketu.

Tabulka 4: Chybové hlášky koncového bodu create (zdroj: vlastní)

Problém může nastat hned na několika místech. Tabulka 4 ukazuje přehled nejdůležitějších unikátních hlášek a jejich http statusů, které může uživatel dostat jako odpověď od aplikace. Součástí jsou chybové hlášky týkající se samotného souboru, čímž je myšlen například chybějící atribut, nebo špatný formát. Dále poskytne uživateli chybovou hlášku týkající se vzniklého problému na úrovni přijetí souboru. Velkou pozornost si zaslouží problém, kdy MinIO neuloží objekt, ale vrátí ErrorResponseException. V tomto případě jsou možné dva případy, jež za ní mohou stát. První nepravděpodobná možnost z důvodu použití pouze jednoho na pevno zvoleného bucketu v aplikaci je ukládání do neexistujícího bucketu. Druhou možností je překonání maximální velikosti metadat, které mohou být svázány k jednomu objektu. Při vrácení hlášky je spolu s ní vrácen http status 202. To je způsobeno typem řešení, kterým je odesílání odpovědí pomocí instance SseEmitter. Toto řešení nemá možnost nastavit status v případě vrácení odpovědi z asynchronní metody. Z tohoto důvodu je vrácen chybový stav, ale status zůstává stejný jako v případě úspěšného požadavku.

6.3.2 api/getByCode a api/getDataByCode

V průběhu životnosti zprávy v uuEnelane je nutné se umět dotázat na její metadata a obsah. Tyto dvě akce jsou si velmi podobné a z tohoto důvodu jsou uvedeny společně. Jejich konkrétní služby jsou dostupné na adresách api/getDataByCode a api/getByCode. Pro správné vrácení metadat v případě metody getByCode nebo obsahu v případě getDataByCode je nutné nejdříve poskytnout správný kód ve správném formátu. Samotný kód musí být uveden v těle požadavku pod názvem code.

Hláška	Řešení
File with code: <code> doesn´t exist or bucket doesn´t exist (404)	Podle zadaného kód nebyla nalezena žádná shoda. Soubor tedy neexistuje, nebo je neexistující bucket.
The selected code have bad pattern, try again (400)	Zadaný kód má nesprávný formát, jež musí být před dalším spuštěním opraven.

Tabulka 5: Hlášky koncového bodu getByCode/getDataByCode (zdroj: vlastní)

Také chyby jsou u obou metod stejné. V případě neúspěchu je důležité nejvíce věnovat pozornost dvěma chybám. Tabulka 5 ukazuje chyby, které mohou být způsobeny uživatelem. Je důležité si pamatovat přesný kód, protože aplikace kontroluje přesnou syntaxi pomocí regex výrazu. V případě jakékoli neshody je uživatel s nastalou situací obeznámen. Druhou situací je, že zadaný kód neodpovídá žádnému objektu v uložišti.

6.3.3 api/delete

Na adrese api/delete je dostupný koncový bod pro mazání objektu v uložišti. Stejně jako u předchozích GET metod je v rámci těla požadavku uložen pod názvem code kód objektu, jež je určen ke smazání. S ním se váží stejná úskalí chyb jako v předchozím případě.

Hláška	Řešení
There is error response from repository. It may be non existent bucket. (500)	Značí neexistující bucket.

Tabulka 6: Hláška koncového bodu delete (zdroj: vlastní)

Zajímavostí ohledně mazání objektu je také chování MinIO při zadání kódu neexistujícího objektu. Zde se liší, oproti předchozím požadavkům, protože MinIO neuvažuje v potaz neexistující objekt. To znamená, že i v případě, kdy je poslán požadavek na smazání neexistujícího objektu, instance uložiště vrátí úspěšné vykonání.

6.4 Implementace

Po úvodních setkáních se zástupcem firmy byly stanoveny podmínky praktické části bakalářské práce, které již byly představeny. Další kroky vývoje byly stanoveny do tří hlavních částí.

První část se týká zadání firmy a jeho zpracování. Tato část je primární pro zhodnocení výsledků bakalářské práce a je celá vypracovaná pod odborným dohledem zadavatelské firmy. Část je postavena na požadavcích firmy a samotný průběh vývoje byl průběžně diskutován. V průběhu vývoje byly také diskutovány rozšiřující funkce, které byly potřeba popsat, aby se v aplikaci mohly zajistit potřebné funkce či vlastnosti. Konkrétně se zde řešila primárně vstupní data, tedy soubory. Zde byl důležitý jejich konkrétní formát a informace, které mají být interpretovány jako metadata a payload. Také byly konzultovány různé typy validací, které by se v aplikaci mohly nacházet. Po kompletním dokončení DAO vrstvy byl výsledek předveden zástupci firmy.

Další částí jsou dodatečné funkce týkající se zpříjemnění pochopení účelu práce a také pro variabilitu vstupních dat, která již nebyla pod dozorem firmy. Zde se objevovalo více možností, jakými to šlo vyřešit. Prvním případem bylo základní načítání souboru z lokálního uložiště. Toto řešení bylo zamítnuto z důvodu jeho jednoduchosti a také jeho méně užitečného využití do budoucna. V případě, kdy by se aplikace měla testovat, by se řešení ukázalo jako velmi nevhodné. Z tohoto důvodu bylo přijato jiné řešení, které dalo jako vstupní bránu do aplikace REST API. Použití API mělo několik pozitivních vlastností, a to nejen kvůli jeho rozsáhlému využití v dnešním světě. V tomto konkrétním případě API přináší další dva benefity. Prvním je přiblížení aplikace více lidem bez nutnosti řešit lokální uložiště v aplikaci. Tímto může být aplikace lépe přenositelná. To znamená, že v případě sdílení aktuálního stavu se zkouška aplikace mohla provádět skrze poslání požadavku na API. Aplikace v tomto řešení může být spuštěna lokálně na počítači skrze vývojové prostředí, v kontejneru či kontejneru na serveru, což znamená, že řešení má větší univerzálnost nasazení. Dalším benefitem je samotná manipulace s aplikací, která je při použití API mnohem příjemnější na obsluhu. Také například podporuje snáze vyrobitelný benchmark, jež by umožňoval podložit výsledky vývoje aplikace také zátěžovým testem, který by ukazoval chování aplikace a uložiště při částečné simulaci provozu.

Další částí, která je součástí výstupu bakalářské práce, je testování celé aplikace. Testování se provádělo jednotkovými testy, poté proběhlo otestování aplikace od přijetí požadavku na API až po adekvátní odpověď. V celém testování jsou zachyceny chybné stavy,

ve kterých se aplikace může nacházet. Může to být způsobeno například špatnými vstupními daty.

Při vývoji, ale také při následném testování vždy bylo potřeba mít spuštěnou instanci MinIO v dockeru. Důležité při implementaci bylo především pochopit, jakým způsobem uložiště funguje a jakým způsobem lze aplikaci na uložiště napojit. Zprovoznění úvodní verze aplikace byla poměrně rychlá. To z velké části závisí na intuitivním zacházením s uložištěm, ale také na logicky vytvořeném klientovi, přes kterého se volá S3 API uložiště. Klient je získán externí závislostí, která byla získána pomocí mavenu. Důvod k použití tohoto klienta je jeho prezentování v dokumentaci od [74] MinIO, a tedy úzké propojení klienta s uložištěm. Zároveň tento klient podporuje všechny potřebné operace, a tak splňuje podmínky vypracování práce.

V následujících kapitolách budou podrobněji rozebrány zmíněné části a přehled o funkčnosti a potřebných znalostech API pro uživatele.

6.4.1 DAO

Nejstěžejnější částí celé bakalářské práce je DAO vrstva. Ta by se dala představit jako brána do požadovaného uložiště. Vrstva pro svoji univerzálnost využívá nadefinovaných rozhraní mapujících všechny požadované operace uuEnelane. Samotné vystavené API MinIO klienta zde tuto univerzálnost ještě vylepšuje. Samotný kód metod v DAO vrstvě je pevně napsán. Zbylé parametry jako například URL či přihlašovací údaje jsou dodány externě. To je v rámci aplikace řešeno pomocí proměnných prostředí. Z toho plyne, že pokud by firma byla nucena, či chtěla změnit použití MinIO na jinou technologii používající S3 API, například AWS, stačilo by jen pár změn, ale samotné DAO by nemuselo být nutné výrazně měnit.

Aby ovšem DAO mohlo fungovat, je také nutné mít instanci MinIO, které bude využito pro aplikaci. V prvotní fázi vývoje se zde počítalo s využitím lokálně umístěného uložiště, jež je spuštěno jako kontejner v dockeru na lokálním uložišti. Tento způsob se hodil pro prvotní seznámení s uložištěm. MinIO lze konfigurovat dvěma způsoby. První je skrze příkazovou řádku, druhým způsobem je konfigurace přes konzoli, kterou MinIO nabízí. Konzole je grafické rozhraní, které se dá použít právě pro práci s uložištěm. Použitím konzole se proces učení výrazně zlepšil a urychlil. Základní konfigurace, jako jsou porty a místo ukládání, se definovala přímo v dockercompose. V rámci konfigurace je důležité dbát pozor právě na vystavené porty a také nastavení uživatelských parametrů, jež poté bude

potřebovat DAO vrstva pro komunikaci. V pozdější fázi bylo důležité objasnit si problematiku konfigurace přes příkazový řádek pomocí mc příkazů. Tento způsob se hodí pro rozšířenou konfiguraci MinIO přímo při vytváření kontejneru. Součástí toho je vytvoření defaultního bucketu. Důvod, proč je to řešeno následovně, je nutnost pouze jednoho bucketu. V případě, kdy by bylo nutné v budoucnu využívat další buckety, lze poté jednoduše buď nakonfigurovat přímo skrze konzoli, nebo pomocí příkazového řádku a také upravit logiku zápisu v aplikaci. Další důležitou konfigurací, která vyplývá i ze zadání, je samostatné čištění uložiště od nepotřebných objektů. Tato konfigurace se nalézá přímo v dockercompose souboru pro okamžité nastavení při vytvoření image. Konkrétní nastavení přímo koresponduje s požadavky, protože [80] MinIO umožňuje mazání objektů v závislosti na jejich stáří.

Samotné DAO je vytvořeno podle zmíněného rozhraní. Na následujících řádcích nebude vysvětleno konkrétní řešení daných funkcí. Implementace těchto funkcí je velmi podobná a větší přidanou hodnotu má obecné vysvětlení principu, na jakém funguje volání komunikace s uložištěm. K navázání komunikace s MinIO existuje speciální klient, který obsahuje všechny podporované operace. Všechny metody vrstvy jsou tedy implementovány pomocí tohoto klienta. K jeho vytvoření je nutné znát právě URL, na jehož konci bude dostupné MinIO a přístupové údaje některého z uživatelů. Následně lze velmi jednoduše analogicky volat metody, jež jsou potřeba. Je důležité dbát na uvolňování zdrojů po přijmutí streamu ze strany uložiště. Také je zde důležité věnovat pozornost chybám, které se vracejí z uložiště v případě problému. Nejen že na základě nich může být uživatel informován o tom, co se děje, ale také podle toho vytvořit opatření k nápravě, ale v rámci aplikace, ale i dalších projektů, implementaci další potřebné metody. Existence objektu se v bakalářské práci řeší v předzpracování souboru. Metoda na existenci objektu není realizována v S3 API, což v konečném důsledku znamená vytvoření vlastní implementace. Princip je postaven na vrácení specifické chyby z MinIO, jež se jmenuje ErrorResponseException, při volání metody na vrácení dat z MinIO. V případě, kdy se nevrátí data, ale vrátí se tato chyba, lze z toho vyvodit problém neexistujícího objektu. V průběhu testování bylo zjištěno, že stejnou chybu můžou způsobovat velikost metadat spojených se souborem.

V produktu je výhodné ukládat data v komprimované podobě z důvodu úspory místa. [81] MinIO disponuje možností komprimace dat. Komprimace probíhá před uložením do uložiště a následně při vracení dat uživateli se soubor dekomprimuje. Komprimace má v základu povolené jen některé formáty souborů a je nastavována pro celé uložiště, to znamená pro všechny buckety. Základními formáty jsou například JSON, XML nebo CSV. Příloha 1

obsahuje soubory vypnutiKomprese.txt a zapnutiKomprese.txt, které ukazují, jakým způsobem lze pomocí příkazové řádky zapnout, nebo vypnout kompresi.

6.4.2 Logika předzpracování

Z důvodu principu interakce uživatele s aplikací bylo nutné vymyslet základní předzpracování vstupního souboru. MinIO je schopné pracovat se souborem i bez předzpracování, ale v uložišti by nebylo možné mít explicitně vyjádřená metadata k danému objektu. To vedlo k nutnosti, aby aplikace byla schopna rozparsovat vstupní soubor a uložit metadata v explicitní formě. V rámci aplikace bylo rozhodnuto o podpoře vstupních formátů JSON a XML. Jako zkušební soubory jsou k dispozici vzorové soubory poskytnuté zástupcem firmy.

Dříve, než je celý soubor uložen, je ještě nutné vytvořit unikátní kód, pod kterým se bude nacházet v uložišti. Toto se děje v rámci sekce základních validací celého souboru. Jelikož originální aplikace přijímá soubory velmi široké škály, od malých po velké a přes mnoho formátů a s ohledem na důležitost validací v aplikaci, jsou aplikovány pouze základní opatření, která musí být splněna, než bude soubor poslán dále ke zpracování. To je kontrola správného formátu, které se zde nachází z důvodu výskytu jiného typu formátu. Nesprávný formát znemožňuje aplikaci daný soubor rozparsovat a korektně uložit. Dalším opatřením, které je v aplikaci zahrnuto, je maximální velikost souboru, jež je pevně definována na maximálně 500 kilobajtů. U ostatních případů užití není potřeba dělat předchozí validace, na místo toho je pouze zkontrolován správný syntax kódu, který je důležitý k identifikaci objektů.

V rámci celé aplikace jsou ošetřovány nepříznivé stavy, což znamená, že i v průběhu zpracovávání souboru mohou vzniknout chyby. Za nimi se mohou skrývat například zmíněné nesplněné validace, nebo dále nesprávné rozparsování souboru. Pro každou takovou situaci je vytvořena adekvátní reakce, jež informuje uživatele o problému. Problém v parsování souboru může být zapříčiněn chybějícími atributy, nebo špatným syntaxem souboru. V rámci projektu jsou vytvořeny i chybné soubory vycházející ze souborů vzorových, které pak vyvolávají právě zmíněné chyby.

6.5 Readonly režim

Součástí podmínek, které klade produkt na MinIO, byl readonly režim, který lze nastavit na straně uložiště. Konfigurace může probíhat oběma způsoby, jak bylo představeno dříve. Obecně lze říct, že pro zapnutí přístupu omezeného na čtení jsou nutné dvě věci. První je mít uživatele, který k tomu bude určen. To znamená mít přístupové údaje, které budou sloužit pro přístup do uložiště. Druhá věc je nastavit uživateli správné oprávnění. Jelikož má první uživatel všechna práva, tak je vhodné vytvořit nového uživatele. Pomocí konzole lze vytvořit nového uživatele a zároveň přidělit oprávnění readonly. Po tomto přidělení má uživatel dovoleno číst konkrétní soubory. To v praxi znamená, že tento uživatel má přístup ke čtení souborů přes S3 API. Pokud by byl uživatel přihlášen do konzole, není možné vylistovat celý bucket. Takto je definován základní režim. Pokud by bylo potřeba vypisovat všechny objekty v bucketu, jsou dvě varianty, jak toho dosáhnout. První variantou je upravit stávající oprávnění, druhou variantou je vytvořit úplně nové oprávnění. [82] MinIO definuje oprávnění na základě JSON souboru. Pro změnu akcí, které uživatel může dělat pod oprávněním, je nutné tento soubor upravit a přidat s3:ListBucket akci. Tato akce je dostupná v konzoli a lze ji tak upravit. Příloha 1 obsahuje soubor readOnlyWithList.json, který ukazuje, v jaké podobě musí být definice oprávnění, aby bylo možné listovat objekty v bucketu spojené s režimem readonly. K jednotlivým souborů, v konzoli ovšem nejsou vidět žádná metadata a nemá možnost je ani číst. Za použití dotazu na S3 API je uživatel schopen číst. Tato definice oprávnění je použita pro vytváření úplně nového oprávnění v aplikaci. V příloze je také soubor konfiguraceOpravneni.txt, který ukazuje, jakými příkazy lze vytvořit oprávnění a následně ho přidělit uživateli.

6.6 Testování

Aby bylo aplikaci možno správně zhodnotit, jsou v celé šíři aplikace aplikované jednotkové testy. Již v průběhu vývoje přineslo testování přínosy v ušetření času při testování celkové funkčnosti aplikace při dílčích úpravách. Ovšem nejdůležitější roli hrají testy jako podklad pro výstup celé bakalářské práce. Aby byl pohled na aplikaci kompletní, jsou k jednotkovým testům doplněny také další testy, které testují součinnost všech částí aplikace a akce provedené v uložišti. Pro spuštění testů zabývajících se také integrací uložiště do aplikace je potřeba mít k dispozici spuštěnou instanci MinIO. Situace, na které jsou testy zaměřené, jsou otestovány komplexně a na správné očekávané chování aplikace. Mezi ně se řadí situace,

ve kterých vše proběhne podle úspěšného scénáře, ale také situace, kde se vyskytne chyba a kontrolují se například správné zpětné hlášky uživateli či vyhazované výjimky.

Posledním krokem testování je simulace reálného provozu. Pro test byl zvolen lokální počítač z důvodu odstranění nutnosti posílat data na server. Lokálního zařízení ovšem nemá distribuční hardware a zároveň je testována pouze aplikace. Kvůli tomu nelze provést simulaci v plné míře. Počítače, na kterém bylo testováno, má procesor Intel core i7 1165G a 16 gigabajtů operační paměti. Z předchozího důvodu bylo potřeba nadefinovat podmínky, které by měl test v této formě splňovat. Konkrétní podmínky jsou uložení alespoň 50 souborů, jež budou velikosti odpovídat maximálně desítkám kilobajtů, během 5 sekund. Z důvodu přiblížení reálnějšímu provozu jsou zde použity dvě velikosti souborů, menší se pohybuje kolem 16 kilobajtů. Jeho zastoupení je použito ve velmi velké míře. Druhou velikostí je 32 kilobajtů, která hraje roli přijetí občasně většího souboru, než je menší. Roli v problematice testování hrají také různé druhy formátu. V aplikaci uuEnelane je velmi používaným formátem JSON. Z tohoto důvodu je jeho zastoupení znatelně větší než zastoupení formátu XML, které se i v reálné aplikaci nepoužívá v tak velké míře. Celkový počet uložených souborů, potažmo uživatelů, kteří posílají požadavky na server, často i paralelně, je 300. Protože v reálné aplikaci se musí alespoň dvakrát přistupovat k datům, jsou v celém testu zahrnuty také GET dotazy. Zde se zanedbávají druhy formátů. Je tu 300 uživatelů, kteří posílají 2 požadavky v celkovém časovém horizontu. Konkrétní počet požadavků v určitém čase běhu aplikace se mění pomocí ramp-up period.

Koncový bod	Ramp-up period	Počet požadavků	Souhrnná chyba	Souhrnná propustnost
getByCode	7 s	600	0,00 %	61,3/s
getDataByCode	7 s	600	0,00 %	61,2/s
create	7 s	300	26,12 %	29,503/s
getByCode	8 s	600	0,00 %	60,4/s
getDataByCode	8 s	600	0,00 %	60,4/s
create	8 s	300	0,00 %	29,412/s

Tabulka 7: Souhrn výsledků z 5 opakování testu (zdroj: vlastní)

Aplikace byla testována opakovaně. Souhrnné výsledky ukazuje Tabulka 7. Sloupec Ramp-up period a počet požadavků jsou uvedeny v rámci jednoho opakování. Další dva sloupce

ukazují souhrnnou chybu a souhrnnou propustnost. Lze vidět jedinou chybu při ukládání souboru. Ta je způsobena přehlcením aplikace. To vedlo k prodloužení ramp-up period z důvodu lepšího rozložení požadavků v čase. Ukazuje se, že při delší periodě již žádné chyby nevznikají. Důležitý sloupec je souhrnná propustnost. Jsou zde dva údaje o propustnosti požadavků na uložení souboru. Ty vznikly sečtením hodnot jednotlivých skupin. Tím se rozumí skupiny rozdělené podle formátu a velikosti souboru. Výsledky tohoto koncového bodu jsou uvedeny celkově, protože malý počet zaslaných požadavků některých skupin by velmi zkresloval možnou propustnost těchto skupin. Oba souhrnné údaje propustnosti se pohybují kolem hodnoty 29,5 požadavků za sekundu. To v konečném zhodnocení znamená, že aplikace splňuje podmínky 50 uložení během 5 sekund. Požadavky GET byly v obou periodách úspěšnými a jejich hodnoty se pohybovaly na vysoké úrovni kolem 61,3 požadavků za sekundu. Výsledky testování se musí brát s ohledem na lokální zařízení a aplikace. Je vidět, že MinIO s aplikací zvládly zátěžový test bez vzniku nějakého vážného problému.

6.7 Server

Již dříve v práci zaznělo nasazení MinIO na server. Použitý server má operační systém Ubuntu Server 22.04.3 LTS. V oblasti hardwaru je dostupných 6 gigabajtů operační paměti a 60 gigabajtů místa na pevném disku. Svými parametry server plně dostačuje účelům. Dříve než se mohla migrovat aplikace s MinIO na server, bylo nutné doinstalovat docker. Následně mohla proběhnout migrace na server. Na serveru není nasazeno jen uložiště, ale také celá aplikace, která tímto krokem umožňuje vzdálený přístup a otestování aplikace bez nutnosti řešit lokální spuštění. Jediná změna, která se mění při dotazování na API, je základní URL, která bude směrovat dotaz na školní server, který ho poté poskytne aplikaci. Ta ho následně zpracuje a podá výsledky.

7. Shrnutí výsledků

V tématu MinIO jsou uvedené podmínky, které byly nezbytné pro zpracování tématu vztahu k produktu uuEnelane. Práce se zaměřila na cloudové prostředí, ve kterém MinIO může být nasazeno, následně bylo porovnáno s lokálním serverem. Byl uveden a vypracován teoretický podklad pro rozhodování, zda je výhodné použít MinIO pro produkt, také bylo zpracováno srovnání s dalšími technologiemi. Výstupy z tohoto teoretického přehledu byly ve prospěch posuzovaného uložiště MinIO. Hlavní přednost, která stojí za zmínku, je S3 API, které je uložištěm poskytováno. Pro produkt to znamená, že na úrovni DAO vrstvy může vzniknout přístup k datům pomocí tohoto API. V případě, kdy by v rámci produktu bylo nutné změnit používané uložiště na jiné s podporou S3 API, je uzpůsobení lehčí a rychlejší než předělávat či vytvářet úplně novou část pro přístup k jiným uložištím. Dále ze shrnutí porovnání lokálního serveru s cloudem vznikl důležitý závěr, který vyzdvihl cloud, ve kterém MinIO může být nasazené. Hlavní přednosti jsou škálování, využití hardwarových zdrojů a snadná údržba. MinIO má také důležitou vlastnost, [70] kdy může být nasazeno stejně na jakýkoli kubernetes k8s, čímž se zvyšuje škála možností, ve kterých může být nasazeno. MinIO splnilo všechny podmínky, jež byly definovány v průběhu celé práce.

Dále byla úspěšně zpracována aplikace napojená na uložiště. Aplikace úspěšně provádí operace, které jsou převzaté z originální DAO vrstvy produktu. V průběhu práce bylo nutné se prakticky seznámit s fungováním uložiště a osvojit si samotnou práci s ním. Také bylo vytvořeno předzpracování souboru. Aplikace byla vyvíjena s ohledem na nepříznivé stavy, které mohou během procesů vzniknout. Tyto stavy byly ošetřeny a vráceny uživateli pro přehled, co může za vzniklou chybou stát. Také byly objeveny některé příčiny vyhazování konkrétních výjimek od uložiště, například neexistující bucket, nebo moc velká velikost metadat. Posledním dílem při vývoji aplikace bylo úspěšné otestování jednotkovými testy, také pak následně vytvoření simulačního testu, který měl za úkol otestovat aplikaci ve spojení s uložištěm pod rozsáhlejší zatížením na lokálním hardwaru. V tomto testu aplikace splnila definovanou hranici. Poté, co byla aplikace vyvinuta, byla zkontejnerizována a spolu s instancí MinIO nasazena na školním serveru. Důležité je zmínit fakt, že i takto nasazená verze má stejný základ znalostí. To znamená, že prvky zachycené v projektu, jako je například práce s API a konfigurace MinIO, bude stejná i při nasazení do cloudu. Rozdíly lze nalézt pouze prostředí, ve kterém se uložiště nachází.

V práci se nepovedlo přesně zmapovat příčiny vzniku konkrétních výjimek od uložiště. Pro lepší porozumění by bylo potřeba se setkat v delším časovém úseku se všemi chybami v praxi. Dále bylo MinIO nasazené pouze na vzdálený server, jelikož nasazení na cloud by bylo finančně náročné. V tomto případě stačí využití serveru, který pokryje všechny potřeby aplikace a MinIO.

8. Závěr

Začátek práce se věnoval uvedení a vysvětlení základní architektury produktu uuEnelane. Následně byl po teoretické stránce představen cloud, jeho výhody, nevýhody, zabezpečení a srovnání s lokálním serverem. Také byly zmíněny nejtěžší kroky k implementaci cloudového řešení ve firmě a jeden ze základních prvků cloudu, kterým je problematika virtualizace. Další nastíněným tématem byl benchmark cloudu, u kterého byly také zmíněny některé možné nástroje, které by bylo možné použít. Z teoretického rozboru vyplynulo, že je MinIO velmi vhodné pro využití v produktu uuEnelane.

Funkcionalita praktické části odpovídá zadání definované zástupcem firmy a funguje korektně. MinIO je ovšem používáno pouze v jedné instanci kontejnerizované v dockeru. Práce by se tedy dala více směrovat ke cloudovému řešení, což znamená přesunout celé uložiště MinIO na cloudovou infrastrukturu. Zde by bylo stěžejní pochopit celý ekosystém cloudu a jeho konfiguraci. Také by bylo důležité přidat monitorovací nástroje, které by umožnovaly včasnou reakci na chybu, nebo jen zjištění, v jakém stavu se celé uložiště nachází. Zde by se daly použít nástroje jako Grafana či Prometheus. Na opačné straně by se také dalo upustit od cloudu a nasadit MinIO na server. V obou případech by bylo potřebné doplnit monitorovací nástroje a vyřešit problematiku škálování. Další vhodné rozšíření by se mohlo týkat více MinIO, přesněji proniknout více do základních principů a zkoumat ve větší míře, co vše je tam možné a jakým způsobem, ať už přes konzoli, nebo přes příkazový řádek. Zajisté má co nabídnout mnohem více, než bylo řešeno v práci, kde byla pozornost upnuta na základní vlastnosti, které bylo potřebné objasnit ve spojení s budoucím možným integrováním do produktu.

9. Seznam obrázků

10. Seznam tabulek

11. Reference

[1] Unicorn a.s. (b.r.). *Enelane -Manage your business data*. Získáno 29. říjen 2023, z https://uuapp.plus4u.net/uu-webkit-maing02/11a144ea98fb44a4912513ba1a3c3feb/

[2] MinIO, Inc. (b.r.). *MinIO | S3 & Kubernetes Native Object Storage for AI*. MinIO. Získáno 13. duben 2024, z https://min.io

[3] Cloudflare, Inc. (b.r.). *What is the cloud? | Cloud definition*. Získáno 7. červenec 2023, z https://www.cloudflare.com/learning/cloud/what-is-the-cloud/

[4] Hazard, K. (b.r.). *What is a Cloud Server? | IBM*. IBM. Získáno 8. červenec 2023, z https://www.ibm.com/topics/cloud-server

[5] Amazon Web Services, Inc. (b.r.). *What is Virtualization? - Cloud Computing Virtualization Explained - AWS*. Získáno 8. červenec 2023, z https://aws.amazon.com/what-is/virtualization/

[6] IBM. (b.r.). *What is cloud computing? | IBM*. Získáno 8. červenec 2023, z https://www.ibm.com/topics/cloud-computing

[7] Google Cloud. (b.r.). *What are the different types of cloud computing? | Google Cloud*. Získáno 6. červenec 2023, z https://cloud.google.com/discover/types-of-cloud-computing

[8] sameekshakhandelwal1712. (2021, červenec 11). *Cloud Deployment Models*. GeeksforGeeks. Získáno 22. srpen 2023, z https://www.geeksforgeeks.org/cloud-deployment-models/

[9] Google Cloud. (b.r.). *What Is a Public Cloud? | Google Cloud*. Získáno 7. červenec 2023, z https://cloud.google.com/learn/what-is-public-cloud

[10] Novotný, A. (b.r.). *A Comparison of Private Cloud Security & Public Cloud Security*. StormIT. Získáno 7. červenec 2023, z https://www.stormit.cloud/blog/private-cloud-security-vs-public-cloud-security/

[11] VMware. (b.r.). *What is a Private Cloud? - Definition*. Získáno 7. červenec 2023, z https://www.vmware.com/topics/glossary/content/private-cloud.html

[12] Susnjara, S. (2023, prosinec 11). *The advantages and disadvantages of hybrid cloud*. IBM Blog. Získáno 11. prosinec 2023, z https://www.ibm.com/blog/hybrid-cloud-advantages-disadvantages/www.ibm.com/blog/hybrid-cloud-advantages-disadvantages

[13] Microsoft. (b.r.). *Pricing Calculator | Microsoft Azure*. Získáno 8. červenec 2023, z https://azure.microsoft.com/en-us/pricing/calculator/

[14] Google Cloud. (b.r.). *Google Cloud Pricing Calculator*. Získáno 9. červenec 2023, z https://cloud.google.com/products/calculator

[15] MinIO, Inc. (b.r.). *High Performance, Kubernetes Native Object Storage*. Získáno 9. červenec 2023, z https://min.io/

[16] Microsoft. (b.r.). *Azure Products by Region | Microsoft Azure*. Získáno 8. červenec 2023, z https://azure.microsoft.com/en-us/explore/global-infrastructure/products-by-region/

[17] jimmart-dev. (2023, srpen 4). *Data redundancy - Azure Storage | Microsoft Learn*. Microsoft. Získáno 21. srpen 2023, z https://learn.microsoft.com/en-us/azure/storage/common/storage-redundancy

[18] Globaldots, A. (2018, červenec 5). *13 Benefits of Cloud Computing for Your Business | GlobalDots*. GlobalDots. Získáno 10. červenec 2023, z https://www.globaldots.com/resources/blog/cloud-computing-benefits-7-key-advantages-for-your-business/

[19] Datacenters.com Cloud. (2020, září 23). *What's the Difference Between Dedicated Server and Cloud Servers?*. Získáno 8. srpen 2023, z https://www.datacenters.com/news/what-s-the-difference-between-dedicated-server-and-cloud-servers

[20] Morgan, L. (2019, únor 14). *On Premise vs Cloud | Key Differences, Cost, Pros & Cons | ESF*. Enterprise Storage Forum. Získáno 22. srpen 2023, z https://www.enterprisestorageforum.com/cloud/on-premise-vs-cloud-storage/

[21] Singh, V. (2020, březen 18*). Difference Between Cloud Servers & Dedicated Servers*. Cloud Academy. Získáno 8. srpen 2023, z https://cloudacademy.com/blog/difference-between-cloud-servers-dedicated-servers/

[22] Weinberg, A. D. (2021, srpen 8). *7 Steps to Secure Your Data Center | CybeReady Blog*. CybeReady. Získáno 27. srpen 2023, z https://cybeready.com/7-steps-to-secure-your-data-center

[23] Stouffer, C. (2023, červenec 14). *What is cloud security? An overview + best practices - Norton*. Norton. Získáno 27. srpen 2023, z https://us.norton.com/blog/privacy/what-is-cloud-security

[24] Red Hat, Inc. (2018, březen 2). *What is virtualization?*. Získáno 26. červenec 2023, z https://www.redhat.com/en/topics/virtualization/what-is-virtualization

[25] Machowski, M. (2022, březen 21). *How to Successfully Scale your Web Application in 2022*. 10Clouds. Získáno 3. září 2023, z https://10clouds.com/blog/devops/how-to-successfully-scale-your-web-application/

[26] Silva, D. da. (2023, červenec 24). *Scalable Applications: Why They Matter And How To Build Them*. Cheesecake Labs. Získáno 5. září 2023, z https://cheesecakelabs.com/blog/scalable-applications/

[27] BasuMallick, C. (2022, listopad 16). *The 8 Types of RAID Storage and How They Work - Spiceworks*. Spiceworks, Inc. Získáno 14. července 2023, z https://www.spiceworks.com/tech/data-management/articles/what-is-raid-storage/

[28] Smith, J. (2022, duben 12). *What is the Best RAID Configuration for Your Server?* HostDime Global corp. Získáno 16. červenec 2023, z https://www.hostdime.com/blog/best-raid-configuration/

[29] Deft. (2021, listopad 1). *The Levels of RAID. Deft | Managed Data Center, Cloud, Network, and Disaster Recovery Services*. Získáno 17. červenec 2023, z https://deft.com/blog/the-levels-of-raid/

[30] Steadfast. (2020, květen 26). *Almost Everything You Should Know About RAID | Steadfast*. Získáno 19. červenec 2023, z https://www.steadfast.net/blog/almost-everything-you-need-know-about-raid

[31] Gillis, A. S., Sullivan, E., & Posey, B. (b.r.). *What is RAID?*. TechTarget. Získáno 19. červenec 2023, z https://www.techtarget.com/searchstorage/definition/RAID

[32] Weinberger, J. (2022, listopad 21). *Applying RAID to the Cloud for Data Resilience*. ShardSecure. Získáno 24. červenec 2023, z https://shardsecure.com/blog/applying-raid-to-the-cloud

[33] Kovacs, G. (2020, září 20). *AWS EBS Ultimate Guide & 5 Bonus Features to Try*. NetApp, Inc. Získáno 7. srpen 2023, z https://bluexp.netapp.com/blog/ebs-volumes-5-lesser-known-functions

[34] Lee, I. (2023, únor 20). *What is Fault Tolerance? 3 Techniques & Definition*. Wallarm. Získáno 22. srpen 2023, z https://www.wallarm.com/what/what-is-fault-tolerance

[35] Tozzi, C. (2022, březen 18). *Compare high availability vs. Fault tolerance in AWS | TechTarget*. Tech Target. Získáno 7. srpen 2023, z https://www.techtarget.com/searchcloudcomputing/tip/Compare-high-availability-vs-fault-tolerance-in-AWS

[36] Avi Networks. (b.r.). *What is Fault Tolerance? Definition & FAQs | Avi Networks*. Získáno 7. srpen 2023, z https://avinetworks.wpengine.com/glossary/fault-tolerance/

[37] Makeitfuture. (2021, únor). *Cloud vs Server: Why you don't need a system house*. Získáno 9. srpen 2023, z https://www.makeitfuture.com/automation/<link rel="canonical" href="www.makeitfuture.com" />//automation/cloud-vs-server

[38] Zheng, Q., Chen, H., Wang, Y., Zhang, J., & Duan, J. (2013). COSBench: Cloud object storage benchmark. *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, 199–210. Získáno 14. srpna 2023, z https://doi.org/10.1145/2479871.2479900

[39] Hou, B., Chen, F., Ou, Z., Wang, R., & Mesnier, M. (2017). Understanding I/O Performance Behaviors of Cloud Storage from a Client's Perspective. *ACM Transactions on Storage*, *13*(2), 16:1-16:36. Získáno 14. srpen 2023, z https://doi.org/10.1145/3078838

[40] Ficco, M., Rak, M., Venticinque, S., Tasquier, L., & Aversano, G. (2015). *Cloud Evaluation: Benchmarking and Monitoring* (s. 175–200). Získáno 16. srpen 2023, z https://doi.org/10.1002/9781119131151.ch7

[41] Standard performance evaluation corporation. (2023, červenec 24). *SPEC Benchmarks and Tools*. Získáno 16. srpen 2023, z https://www.spec.org/benchmarks.html#cloud

[42] Calzon, B. (2023, červen 6). *Learn The Top Cloud Computing Challenges, Risks & Issues*. RIB Software GmbH. Získáno 8. září 2023, z https://www.datapine.com/blog/cloud-computing-risks-and-challenges/

[43] Pratt, M. (2023, květen 29). *Introduction to MinIO | Baeldung*. Baeldung. Získáno 27. říjen 2023, z https://www.baeldung.com/minio

[44] Amazon Web Services, Inc. (b.r.). *Amazon S3 API Reference—Amazon Simple Storage Service*. Získáno 27. říjen 2023, z https://docs.aws.amazon.com/AmazonS3/latest/API/Type_API_Reference.html

[45] Google Cloud. (b.r.). *Interoperability with other storage providers | Cloud Storage | Google Cloud*. Získáno 27. říjen 2023, z https://cloud.google.com/storage/docs/interoperability

[46] Zhang, R. (2016, květen 22). *Access Azure Blob Storage from Your Apps using S3 Java API - ISE Developer Blog*. Microsoft. Získáno 28. říjen 2023, z https://devblogs.microsoft.com/ise/2016/05/22/access-azure-blob-storage-from-your-apps-using-s3-api/

[47] Azure storage. (2023, srpen 14). *Storage Partners | Azure Storage*. Získáno 27. říjen 2023, z https://azure.github.io/Storage/docs/storage-partners/

[48] Amazon Web Services, Inc. (b.r.). *What is Amazon S3? - Amazon Simple Storage Service*. Získáno 27. říjen 2023, z https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

[49] Google Cloud. (b.r.). *Cloud Storage | Google Cloud*. Získáno 27. říjen 2023, z https://cloud.google.com/storage

[50] Microsoft. (b.r.). *Azure Storage Blobs Pricing | Microsoft Azure*. Získáno 28. říjen 2023, z https://azure.microsoft.com/en-us/pricing/details/storage/blobs/

[51] MinIO, Inc. (b.r.). *MinIO | SUBNET Subscription & MinIO Pricing*. Získáno 15. duben 2024, z https://min.io/pricing

[52] Google. (b.r.). *How your Google storage works—Google One Help*. Získáno 28. říjen 2023, z https://support.google.com/googleone/answer/9312312?hl=en

[53] Ramakrishnan, S. (2023, březen 23). *Time to First Byte and Streaming Media*. MinIO, Inc. Získáno 27. říjen 2023, z https://blog.min.io/time-to-first-byte-streaming-media/

[54] Amazon Web Services, Inc. (b.r.). *Using the AWS SDK for Java—Amazon Simple Storage Service*. Získáno 27. říjen 2023, z https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingTheMPJavaAPI.html

[55] Google Cloud. (b.r.). *Streaming uploads | Cloud Storage | Google Cloud*. Získáno 27. říjen 2023, z https://cloud.google.com/storage/docs/streaming-uploads

[56] pauljewellmsft. (2023, září 22). *Upload a blob with Java - Azure Storage | Microsoft Learn*. Microsoft. Získáno 28. říjen 2023, z https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blob-upload-java

[57] Amazon Web Services, Inc. (b.r.). *Efficient Cloud Support Services - AWS Support for Optimal Performance*. Získáno 27. říjen 2023, z https://aws.amazon.com/premiumsupport/

[58] Google Cloud. (b.r.). *Customer Care | Google Cloud*. Získáno 27. říjen 2023, z https://cloud.google.com/support

[59] Microsoft. (b.r.). *Azure Support Plans Comparison | Microsoft Azure*. Získáno 28. říjen 2023, z https://azure.microsoft.com/en-us/support/plans

[60] Amazon Web Services, Inc. (b.r.). *Amazon S3 Simple Storage Service Pricing - Amazon Web Services*. Získáno 27. říjen 2023, z https://aws.amazon.com/s3/pricing/

[61] Google Cloud. (b.r.). *Pricing | Cloud Storage | Google Cloud*. Získáno 27. říjen 2023, z https://cloud.google.com/storage/pricing

[62] MinIO, Inc. (b.r.). *Thresholds and Limits — MinIO Object Storage for Azure Kubernetes Service*. Získáno 27. říjen 2023, z https://min.io/docs/minio/kubernetes/aks/operations/concepts/thresholds.html

[63] Amazon Web Services, Inc. (b.r.). *Amazon S3 multipart upload limits—Amazon Simple Storage Service*. Získáno 27. říjen 2023, z https://docs.aws.amazon.com/AmazonS3/latest/userguide/qfacts.html

[64] Google Cloud. (b.r.). *Quotas & limits | Cloud Storage | Google Cloud.* Získáno 27. říjen 2023, z https://cloud.google.com/storage/quotas

[65] akashdubey-ms. (2023, duben 3). *Scalability and performance targets for Blob storage - Azure Storage | Microsoft Learn*. Získáno 28. říjen 2023, z https://learn.microsoft.com/en-us/azure/storage/blobs/scalability-targets

[66] MinIO. (2019). *MinIO S3 Throughput Benchmark on NVMe SSD - 32 Node.* Získáno 28. říjen 2023, z https://min.io/resources/docs/MinIO-Throughput-Benchmarks-on-NVMe-SSD-32-Node.pdf

[67] Kumar, P. (2023, červenec 14). *What is Virtualization in Cloud Computing?* Získáno 26. červenec 2023, z https://www.knowledgehut.com/blog/cloud-computing/virtualization-in-cloud-computing#how-does-virtualization-work%C2%A0in-cloud-computing?%3Cimg-alt=%22working-of-virtualization-in-cloud-computing-%22-sizes=%22(min-width:767px)-50vw,-100vw%22-srcset=%22/_n

[68] MinIO, Inc. (b.r.). *MinIO | Enterprise Grade, High Performance Object Storage*. Získáno 30. říjen 2023, z https://min.io/product/overview

[69] MinIO. (b.r.). *High-Performance Object Storage for AI Data Infrastructure*. Získáno 30. říjen 2024, z https://min.io/resources/docs/MinIO-High-Performance-Multi-Cloud-Object-Storage.pdf

[70] MinIO, Inc. (b.r.). *Deploy the MinIO Operator—MinIO Object Storage for Kubernetes.* Získáno 8. duben 2024, z https://min.io/docs/minio/kubernetes/upstream/operations/installation.html

[71] Demir, M. E. (2023, červenec 25). *Breaking Barriers in Data Storage: Exploring the Versatility of MinIO | by Muhammed Eren Demir | Medium*. Medium. Získáno 8. duben 2024, z https://muhammederendemir.medium.com/breaking-barriers-in-data-storage-exploring-the-versatility-of-minio-35e87603ff07

[72] MinIO, Inc. (b.r.). *MinIO | AWS S3 Compatible Object Storage*. Získáno 31. říjen 2023, z https://min.io/product/s3-compatibility

[73] Evans, C. (2016, únor 19). *Object Storage: Standardising on the S3 API - Architecting IT*. Architecting IT. Získáno 3. listopad 2023, z https://www.architecting.it/blog/object-storage-standardising-on-the-s3-api/

[74] MinIO, Inc. (b.r.). *Java Client API Reference—MinIO Object Storage for Linux*. Získáno 3. listopad 2023, z https://min.io/docs/minio/linux/developers/java/API.html#getObject

[75] MinIO, Inc. (b.r.). *Core Administration Concepts—MinIO Object Storage for Container*. Získáno 4. listopad 2023, z https://min.io/docs/minio/container/administration/concepts.html

[76] MinIO, Inc. (b.r.). *Erasure Coding—MinIO Object Storage for Linux*. Získáno 5. listopad 2023, z https://min.io/docs/minio/linux/operations/concepts/erasure-coding.html

[77] MinIO, Inc. (b.r.). *Core Operational Concepts—MinIO Object Storage for Linux*. Získáno 18. duben 2024, z https://min.io/docs/minio/linux/operations/concepts.html

[78] MinIO, Inc. (b.r.). *Availability and Resiliency—MinIO Object Storage for Container*. Získáno 5. listopad 2023, z https://min.io/docs/minio/container/operations/concepts/availability-and-resiliency.html

[79] Carey, S. (2021, srpen 17). *What is cloud native? The modern way to develop software | InfoWorld*. InfoWorld. Získáno 4. listopad 2023, z https://www.infoworld.com/article/3281046/what-is-cloud-native-the-modern-way-to-develop-software.html

[80] MinIO, Inc. (b.r.). *Object Lifecycle Management—MinIO Object Storage for Linux*. Získáno 21. březen 2024, z https://min.io/docs/minio/linux/administration/object-management/object-lifecycle-management.html

[81] MinIO, Inc. (b.r.). *Data Compression—MinIO Object Storage for Linux*. Získáno 18. duben 2024, z https://min.io/docs/minio/linux/administration/object-management/data-compression.html

[82] MinIO, Inc. (b.r.). *Access Management—MinIO Object Storage for Linux*. Získáno 11. duben 2024, z https://min.io/docs/minio/linux/administration/identity-access-management/policy-based-access-control.html

Příloha č.1

Příloha 1: Na přiloženém flash disku jsou obsaženy přílohy a aplikace.

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2022/2023

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení:	**Jakub Fogl**
Osobní číslo:	**I2100198**
Adresa:	Za Humny 702, Sezemice, 53304 Sezemice, Česká republika
Téma práce:	Prototyp pro uložení binárních dat do MinIO
Téma práce anglicky:	Prototype for storing binary data in MinIO
Jazyk práce:	Čeština
Vedoucí práce:	Ing. Jan Krunčík Katedra informatiky a kvantitativních metod

Zásady pro vypracování:

Externí téma, odborné vedení společností Unicorn.

Cíl
Tvorba prototypu uložení binárních dat do cloudového úložiště MinIO s cílem ověřit použitelnost na produktech Unicornu – primárně uuEnelane

Zaměřeno na

- Zmapování možností produktu
  - Možnost readOnly režimu
  - Konfigurace MinIO pro velké objemy dat (v řádech TB)
- Integraci úložiště pomocí S3 API v Javě

Seznam doporučené literatury:

Podpis studenta: Datum:

Podpis vedoucího práce: Datum: