VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Václav Gryc

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

INTEGRACE ITSM APLIKACÍ PROSTŘEDNICTVÍM WEBOVÝCH SLUŽEB A IBM CLOUDU
INTEGRATION OF ITSM APPLICATIONS USING WEB SERVICES AND IBM CLOUD

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Václav Gryc

VEDOUCÍ PRÁCE
SUPERVISOR
Mgr. Ing. Pavel Šeda

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Václav Gryc ***ID:*** 174203

Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Integrace ITSM aplikací prostřednictvím webových služeb a IBM cloudu

POKYNY PRO VYPRACOVÁNÍ:

Diplomová práce se bude zabývat možnostmi integrace ITSM (Information Technology Service Management) řešení prostřednictvím webových služeb. Práce se zde bude soustředit zejména na softwarové architektury v cloudovém prostředí včetně vhodné rešerše k nim. Cílem práce bude navrhnout vhodnou softwarovou architekturu včetně její implementace s ohledem na možnosti integrace ITSM řešení na základě široké škály zákaznických požadavků. Vytvořené řešení bude implementováno v systému Node.js, za použití frameworku Loopback včetně diskuze nad alternativními programovacími jazyky či frameworky. Výsledná aplikace bude nasazená na IBM Cloud včetně vhodných schémat a diskuze nad možnostmi kontinuální integrace a dodávky (continuous integration and delivery) s ohledem na její bezpečnost.

DOPORUČENÁ LITERATURA:

[1] BREWSTER, Ernest. IT service management: a guide for ITIL® foundation exam candidates. 2nd ed. Swindon: BCS, The chartered institute for IT, c2012. ISBN isbn:9781906124939.

[2] Mardan A. (2014) Sails.js, DerbyJS, LoopBack, and Other Frameworks. In: Pro Express.js. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-0037-7_18

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: Mgr. Ing. Pavel Šeda

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Tato diplomová práce se zabývá vývojem software s možností integrace ITSM řešení. Cílem práce je návrh vhodné softwarové architektury integrační aplikace v cloudovém prostředí. Z důvodu využití řešení pro mnoho zákazníku je kladen důraz na udržitelnost a znovupoužitelnost. Po rozsáhlé analýze technologií byla vytvořena webová aplikace, která je plně konfigurovatelná pro uspokojení široké škály zákaznických potřeb. Pro vývoj aplikace je rozhodnuto využití prostředí Node.js a framework Loopback 4. To zajišťuje nižší náklady na provoz cloudových zdrojů, oproti konkurenčním technologiím, jako je například Java. Celý systém je nasazen do IBM Cloud. Výsledná aplikace je uvedena do produkce ve firmě IBM.

KLÍČOVÁ SLOVA

IT Service Management, Integrace, IBM Cloud, Mikroslužby, Node.js

ABSTRACT

This thesis is focused on developing software with ITSM integration features. The aim of this work is to design a suitable software architecture for integration applications in a cloud environment. Due to the use of the solution for many customers, emphasis is placed on sustainability and reusability. After wide technology analysis, a web application has been created that is fully configurable to meet a wide range of customer needs. It is decided to use the Node.js environment and the Loopback 4 framework for application development. To ensure lower costs for the operation of cloud resources, compared to competing technologies such as Java. The entire system is deployed in IBM Cloud. The resulting application is put into production by IBM.

KEYWORDS

IT Service Manafement, Integrations, IBM Cloud, Microservices, Node.js

GRYC, Václav. *Integrace ITSM aplikací prostřednictvím webových služeb a IBM cloudu*. Brno, 2021, 69 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Mgr. Ing. Pavel Šeda

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Integrace ITSM aplikací prostřednictvím webových služeb a IBM cloudu“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval Bc. Jan Schafer za seznámení s problematikou integrace aplikací, cenné rady při návrhu softwarové architektury, věcné připomínky a odborný přístup při konzultacích nápomocných k vypracování diplomové práce. Dále bych rád poděkoval vedoucímu diplomové práce Mgr. Ing. Pavel Šeda za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Nakonec bych chtěl poděkovat celé své rodině za trpělivost a psychickou podporu, kterou mi poskytovali po celou dobu studia.

Obsah

Seznam obrázků

Úvod

Závislost organizací na informačních a komunikačních technologiích (ICT) každým rokem roste [1]. Málokterá organizace by si v dnešní době dokázala představit fungování procesů bez podpory IT služeb. S tím je logicky spojován požadavek na neustále zlepšování kvality IT služeb a infrastruktury. Na základě těchto požadavků byl v 80. letech britskou vládní agenturou CCTA (Central Computer and Telecommunications Agency) vydán svazek pravidel pro řízení IT služeb pod názvem ITIL [2]. První část práce je zaměřena na aktuální verzi ITIL verze 4. Jsou popsány teoretické základy důležité k pochopení, jak řízení IT služeb funguje v dnešní době.

Jelikož IT odvětví se stává čím dál rozsáhlejším, tak organizace mají na výběr z mnoha různých řešení pro řízení IT služeb. Pro poskytování kvalitních služeb je vyžadováno interakce mezi jednotlivými organizacemi. K tomu slouží integrace. Možnosti integrace mezi více organizacemi jsou popsány v druhé části práce.

Firma IBM, mimo jiné, zajišťuje integrace mezi ITSM nástroji mnoha zákazníků. Cílem diplomové práce je návrh konfigurovatelného software pro uspokojení všech jejich potřeb. Požadavkem na software je jeho znovupoužitelnost, pro dodávání integračních aplikací v co nejkratším čase za co nejmenší prostředky.

Z praxe je známo, že provoz mezi ITSM nástroji je nahodilý. V noci je provoz většinou velmi nízký a ve špičce může docházet až k tisícům požadavků za minutu. To je jeden z hlavních důvodů, proč byl požadavek na provozování softwaru v cloudu. Cloudové systémy jsou nastíněny ve třetí části práce.

Poslední část práce je zaměřena zejména na architektonická rozhodnutí a návrh plně konfigurovatelné aplikace. Výsledná aplikace je nasazena na IBM Cloud. V závěrečné práci je diskutovány možnosti kontinuální integrace a nasazení na IBM Cloud.

1 IT Service Management

Základním úkolem řízení IT služeb (anglicky IT Service management, dále jen ITSM) je řídit implementaci a spravovat kvalitu poskytovaných IT služeb s využitím vhodné kombinace lidí, procesů a informačních technologií [3]. Z praktického hlediska ITSM představuje soubor procesů, které umožňují vytvářet, navrhovat nebo spravovat IT služby. Hlavní přínosy ITSM jsou následující:

- Snížení nákladu na IT,
- zvýšení spokojenosti zákazníků,
- zvýšení kvality služeb,
- zlepšení řízení a snížení rizika,
- získání konkurenční výhody,
- zvýšení flexibility [4].

1.1 IT služba

Knihy ITIL, popsané v části 1.3, definují IT službu jako *prostředek pro poskytování hodnoty zákazníkovi prostřednictvím výstupů, kterých zákazník chce dosáhnout bez vlastnictví specifických nákladů a rizik* [5]. IT službu tvoří kombinace informačních technologií, lidí a procesů a liší se od produktu v tom, že nemusí vždy poskytovat hmatatelnou věc. Její význam spočívá v poskytování hodnoty příjemci služby, který za její pomoci může dosáhnout požadovaného cíle bez vynaložení specifických nákladů a bez podstupování rizika. Poskytovatelé IT služeb však musejí ovládat soustavu organizačních způsobilostí potřebných pro poskytování daných hodnot [6]. IT služby jsou poskytovány za účelem přímé podpory firemních procesů zákazníka tak, aby dosahovaly smluvně stanovených cílů, které jsou zahrnuty v dohodě o úrovni služby. Všechny služby nemusí být určeny přímo zákazníkům. Existují také podpůrné IT služby, které umožňují poskytování těch IT služeb, které zákazníkovy určeny jsou [6].

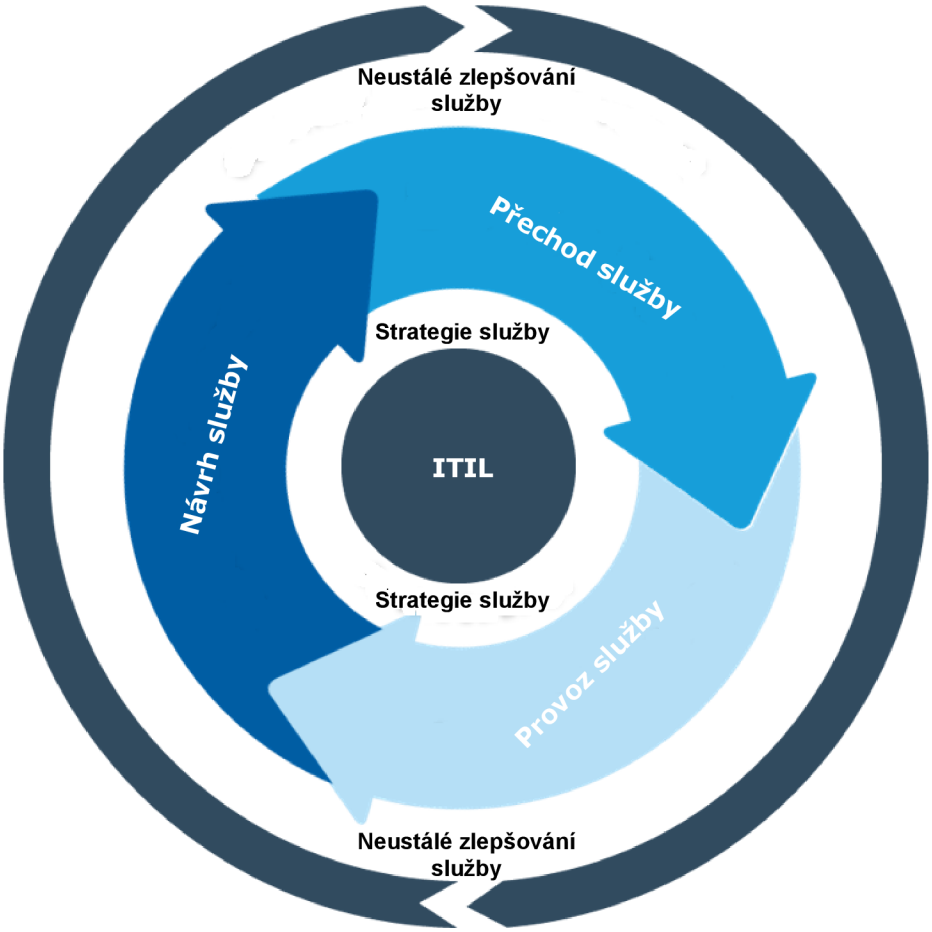
1.2 Poskytovatel IT služeb

Organizace poskytovatele se skládá z lidí pracujících v procesech s kontrolními mechanismy, přičemž je vše umístěno v organizační struktuře [6]. V závislosti na strategii zajištění zdrojů, kterou uplatňuje zákazník, k němu může organizace poskytovatele IT služeb zaujímat pozici:

- externí – jde o dva různé subjekty, kde se zákazník rozhodl zajistit požadované IT služby formou outsourcingu.
- interní – poskytovatel i zákazník jsou součástí stejné organizace.

1.3 ITIL

Knihovna infrastruktury IT (anglicky Information Technology Infrastructure Library, dále ITIL) je knihovna obsahující praxí ověřené postupy plánování a využití informačních technologií. Vychází ze zkušeností mnoha společností po celém světě. Jedná se téměř o mezinárodní standard pro řízení IT služeb [2]. Tato knihovna byla poprvé vydána v podobě 31 knih, které byly publikovány v letech 1989 – 1995 v UK agenturou *Central Communications and Telecommunications Agency (CCTA)* [2]. V dnešní době je původní verze zredukována do 5 knih, které poskytují organizacím rady, jak dosáhnout poskytování kvalitních IT služeb a je označována jako ITIL 4. Základní předpoklady k plnému využití potenciálu ITIL spočívají v osvojení jeho přístupů a v jejich následnému přizpůsobení specifickým potřebám a požadavkům dané organizace. Tímto postupem by měla být zajištěna tvorba hodnoty, jak pro poskytovatele IT služeb, tak i pro jeho zákazníky [5]. Na obr. 1.1 je graficky zobrazen životní cyklus IT služby v pěti fázích. Každá fáze definuje své cíle a principy, jak jich dosáhnout.



Obr. 1.1: Životní cyklus služby dle ITIL

1.3.1 Strategie služby

Cílem strategie služeb je nabízet lepší služby než konkurence. Aby byla organizace dlouhodobě úspěšná, musí myslet dlouhodobě, protože průmyslové odvětví se přizpůsobuje nevyhnutelným ekonomickým, sociálním, technologickým a politickým změnám [7]. Strategie služeb není jen o strategii pro jednotlivé služby, ale hlavně o dlouhodobém poskytování všech IT služeb. Do fáze strategie služeb je zahrnut také návrh, vývoj a implementace jako součást základny strategických aktiv organizace [5]. Klíčové procesy řešeny ve správě služeb jsou [8]:

- Správa financí,
- správa portfolia služeb,
- správa požadavků.

1.3.2 Návrh služby

Další fází životního cyklu správy IT služeb je návrh služeb. Tato fáze slouží k vytvoření nových služeb, které pak přechod služby zavede do produkčního prostředí. Návrh služby klade za cíl podniknout nezbytné kroky k zajištění toho, aby nová služba fungovala podle plánu a poskytovala funkcionalitu a výhody určené pro organizaci. Tento princip je srdcem přístupu ITIL a proto je většina procesů návrhu služeb zaměřena na řízení operací [5]. Procesy zahrnuté ve fázi návrh služeb jsou následující:

- Správa katalogu služeb,
- správa úrovně služeb,
- správa kapacit,
- správa dostupnosti,
- správa kontinuity služeb IT,
- správa bezpečnosti informací,
- správa dodavatelů.

Příspěvek, který fáze životního cyklu návrh služeb přináší, lze tedy shrnout jako zajištění vytvoření nákladově efektivních služeb poskytujících úroveň kvality, požadovanou k uspokojení zákazníků a zúčastněných stran po celou dobu životnosti služby. Skutečnost, že se obchodní požadavky v průběhu času mění a vytvářejí potřebu nebo příležitost k dalšímu zlepšování, však znamená, že i organizace s vyspělými procesy navrhování služeb budou muset provádět změny ve službách po celou dobu jejich životnosti. Návrh služeb proto hraje důležitou roli při podpoře neustálého zlepšování služeb a je stejně důležitý pro řízení změn stávajících služeb i při navrhování nových služeb [5].

1.3.3 Přechod služby

Ve společnostech častokrát dochází k rozpojení vývoje a provozního oddělení. Toto rozpojení vede k neúspěšnému vývoji nových služeb či špatné implementaci změn ve stávajících službách. Fáze přechod služby zajišťuje, aby byly provozní požadavky plně zohledněny a uspokojeny před tím, než bude cokoli přesunuto do produkčního prostředí a to včetně dokumentace, školení pro uživatele a podpůrný personál. Fáze přechod služby je také zodpovědná za vyřazení z provozu a odstranění služeb, které již nejsou nutné [5]. Na následující procesy se vztahuje fáze přechod služby:

- Správa změn,
- správa aktiv a konfigurace,
- správa znalostí,
- plánování a podpora přechodu,
- správa verzí a nasazení,
- ověření a testování služby,
- vyhodnocení.

1.3.4 Provoz služby

Provoz služby je klíčová fáze životního cyklu správy IT služeb. Pokud služby nejsou využívány nebo nejsou efektivně poskytovány, nedosáhnou svého plného potenciálu bez ohledu na to, jak dobrý je jejich návrh. Provoz služby je fáze, ve které se dodává skutečná hodnota, která byla modelována ve strategii služeb a potvrzena prostřednictvím návrhu služby a přechodu služby. Určení smysluplných metrik provozování služby je základem a také výchozím bodem pro fázi zlepšování služby. Provoz služby je nejblíže uživatelům služeb a současně zákazníkům společnosti. Tato fáze je tady „tváří“ IT organizece [5]. Procesy prováděné ve fázi provoz služby jsou:

- Správa událostí,
- správa incidentů,
- správa problémů,
- správa přístupů,
- provádění požadavků,
- správa provozu IT,
- správa aplikací,
- technická správa.

1.3.5 Neustálé zlepšování služby

Jakmile je řešení navržené služby implementováno, je zásadní nepřestat službu vyvíjet. Všechny aspekty prostředí se neustále mění a poskytovatel služeb musí vždy

usilovat o zlepšení. Kontinuální zlepšování služeb je odpovědné za zajištění toho, aby byla tato zlepšení identifikována a implementována [9]. Výkon poskytovatele IT služeb se neustále měří a zlepšují se procesy, IT služby a IT infrastruktura za účelem zvýšení efektivity a nákladů [5]. V této fázi najdeme procesy zvané:

- Měření služeb,
- vykazování služeb.

1.4 Nejpoužívanější ITSM nastroje

K zjednodušení správy IT služeb je využívána spousta nástrojů, které pomáhají regulovat způsob, jakým jsou IT služby poskytovány v rámci společnosti.

1.4.1 ServiceNow

Společnost ServiceNow nabízí mimo jiné i nástroj pro správu IT služeb. Vize ServiceNow je agregace IT služeb do jediné cloudové platformy. Populárnost řešení společnosti ServiceNow dokazují reference velkých společností jako jsou GE Digital, Broadcom, Overstock.com a mnoho dalších [10].

V nabídce nechybí implementace funkcí jako je správa incidentů s automatickým směrováním, správa aktiv a nákladů, které mohou sledovat náklady a smlouvy na aktiva, a moderní samoobslužný portál pro přístup spotřebitelů [10].

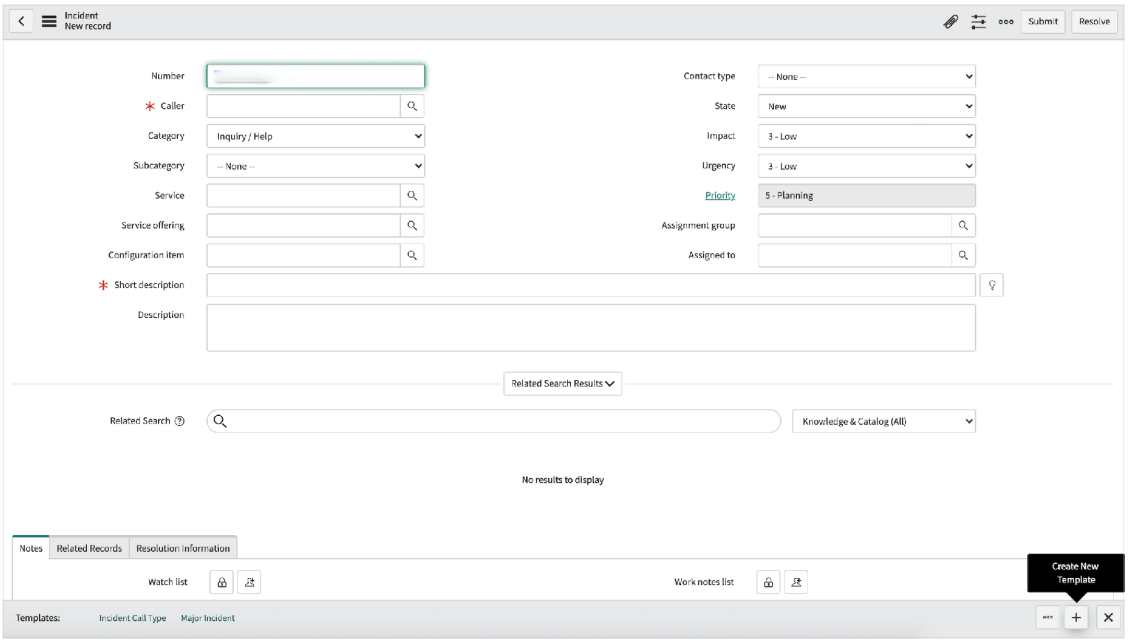
ServiceNow IT Service Management lze také integrovat do analytických řídicích panelů výkonu v reálném čase se schopností srovnávat, jak si vedete mezi průměry průmyslového odvětví [11].

Mezi nevýhody můžeme řadit skutečnost že se jedná pouze o placený nástroj a ne zrovna nejlevnější. Není možnost ani zkušebního bezplatného období [11].

1.4.2 Další ITSM nástroje

Existuje velká řada ITSM nástrojů. Každý nástroj má své klady a zápory. Nelze jednoznačně určit, který z nástrojů je nejvhodnější. Každý nástroj má své benefity a také určité nevýhody. Některé další nástroje jsou:

- Freshservice – ITSM nástroj zaměřený na řešení problémů a prevenci. Nabízí 21 denní bezplatnou zkušební dobu, aplikace pro mobilní zařízení a analýzu příčin incidentů.
- Cherwell – vyznačuje se hlavně jednoduchostí ovládání. Má „drag and drop“ uživatelské rozhraní a přívětivou cenu.



Obr. 1.2: Detail incidentu v ServiceNow.

2 Možnosti integrace více společností

Hlavním úkolem integrace je spolupráce více aplikací. Některé aplikace mohou být vyvíjeny interně v podniku, zatímco jiné jsou nakupovány od dodavatelů třetích stran. Všechny tyto aplikace pravděpodobně běží na různých počítačích, které mohou být vedle sebe, nebo také na druhé straně planety. Některé aplikace mohou být provozovány obchodními partnery podniku nebo zákazníky. Existují i takové aplikace, které nebyly navrženy pro integraci. Tyto problémy a mnoho dalších jim podobných dělají z integrace obtížný úkol. V následující kapitole jsou popsány možnosti pro integraci aplikací.

2.1 Kritéria pro integraci aplikací

Pokud by potřeby integrace byly vždy stejné, existoval by pouze jeden integrační styl. Integrační aplikace zahrnuje řadu úvah a důsledků, které je třeba vzít v potaz. První kritérium, které je třeba zvážit, je integrace aplikace. Pokud aplikace nepotřebuje spolupracovat s žádnou další aplikací, je možné se integraci úplně vyhnout [12]. V dnešní době využívají i ty nejmenší společnosti řadu aplikací, které spolu potřebují komunikovat, aby poskytly jednoduché prostředí buď pro zaměstnance, nebo zákazníky. Další hlavní kritéria integrace jsou:

- **Spojení aplikací** – Integrované aplikace by měly minimalizovat závislosti mezi sebou, aby se každá z aplikací mohla vyvíjet nezávisle.
- **Jednoduchost integrace** – Při integraci aplikací je většinou dobré co nejméně zasahovat do původní aplikace.
- **Integrační technologie** – Různé integrační techniky vyžadují různé množství specializovaného softwaru a hardwaru.
- **Datový formát** – Integrované aplikace se musí shodnout na formátu dat, která si vyměňuji. Další možností je implementace překladače pro sjednocení datového formátu.
- **Doba integrace** – Doba integrace je pro různá řešení odlišná. Existují integrace real-time systémů, kde by měla být minimalizována doba mezi okamžikem sdílení dat a okamžikem, kdy druhá aplikace data příjme. Na druhou stranu existují systémy, kde doba integrace nehraje významnou roli. V takových systémech může být integrační proces prováděn jednou za delší časový úsek, například jednou za rok.
- **Integrovaní funkce** – Integrované aplikace nemusejí vždy požadovat sdílení dat. Mohou chtít sdílet funkce tak, aby každá aplikace mohla vyvolat funkci jiné aplikace.

- **Asynchronicita** – Některé integrované aplikace nemohou čekat na zpracování procesu integrování. To platí zejména pro integrované aplikace, kde nemusí být vzdálená aplikace spuštěna nebo může být nedostupná síť – zdrojová aplikace může chtít jednoduše zpřístupnit sdílená data nebo zaznamenat požadavek na volání podprocesu, ale poté přejít k další práci s jistotou, že vzdálená aplikace bude fungovat později [12].

2.2 Styl návrhu integrační aplikace

K integraci aplikací lze přistupovat mnoha způsoby. Každý přístup řeší některá integrační kritéria lépe než jiná. Různé přístupy lze shrnout do čtyř hlavních integračních stylů [12]:

- Integrace přenášením souborů,
- integrace sdílením databáze,
- integrace vzdáleným vyvoláním procedury,
- integrace přenášením zpráv.

Každý z přístupů má stejný cíl, nutnost integrovat aplikace a velmi podobný kontext. To, co jednotlivé přístupy odlišuje, je různá míra elegantnosti. Úkolem není vybrat jeden styl, který se má vždy použít, ale vybrat nejlepší styl pro konkrétní integrační příležitost [12]. Každý styl má své výhody a nevýhody. Dvě aplikace se mohou integrovat pomocí více stylů, takže každý bod integrace využívá styl, který mu nejlépe vyhovuje. Podobně může aplikace používat různé styly k integraci s různými aplikacemi tak, aby si vybrala styl, který nejlépe vyhovuje jiné aplikaci. Některé integrační přístupy lze považovat za hybrid více stylů.

Výše zmíněné ITSM nástroje nabízejí komunikační rozhraní založené na webových službách. Z tohoto důvodu se zbývající část práce zaměřuje na integrace přenášením zpráv.

2.3 Integrace přenášením zpráv

Díky přenášení zpráv jsou aplikace volně spojeny asynchronní komunikací, což také zvyšuje spolehlivost komunikace, protože tyto dvě aplikace nemusí být spuštěny současně [13]. Díky zasílání zpráv je systém odpovědný za přenos dat z jedné aplikace do druhé, takže se aplikace mohou soustředit na to jaké údaje potřebují ke sdílení, ale nemusí se tolik starat o to jak je sdílet. Systém zasílání zpráv se skládá z několika hlavních konceptů [12]:

- Kanál zpráv,
- zprávy,

- vícestupňové doručení,
- směrování,
- transformace,
- koncové body.

2.3.1 Kanál zpráv

Aplikace pro zasílání zpráv přenášejí data prostřednictvím virtuálního kanálu, který spojuje odesílatele s příjemcem. Nově nainstalovaný systém zasílání zpráv neobsahuje žádné kanály. Je potřeba nakonfigurovat jak má aplikace komunikovat a poté vytvořit kanály, které komunikaci usnadní [12].

2.3.2 Zprávy

Zpráva je balíček dat který lze přenášet na kanálu. Aby mohla aplikace přenášet data, musí je rozdělit na jeden nebo více paketů, zabalit každý paket jako zprávu a poté odeslat na kanál. Podobně aplikace přijímače obdrží zprávu, ze které je potřeba zpracovat data. Zejména u synchronních systémů bývá očekáváno potvrzení o přijetí zprávy od aplikace přijímače. Při nedoručení kladného potvrzení se systém většinou snaží doručit zprávu opakovaně a to do té doby, dokud nebude úspěšně doručena [12].

2.3.3 Vícestupňové doručení

V nejjednodušším případě doručuje systém zpráv zprávu přímo z počítače odesílatele do počítače příjemce. Akce však často musí být provedeny se zprávou poté, co ji odešle její původní odesílatel, ale než ji přijme její konečný příjemce. Například může být nutné zprávu ověřit nebo transformovat, protože příjemce očekává jiný formát zprávy než odesílatel [12].

2.3.4 Směrování

Ve velkém podniku s mnoha aplikacemi a kanály pro jejich připojení je žádoucí, aby zpráva procházela několika kanály pro dosažení svého konečného cíle. Trasa, kterou musí zpráva následovat, může být tak složitá, že původní odesílatel neví, jaký kanál doručí zprávu konečnému příjemci. Místo toho původní odesílatel odešle zprávu do směrovače zpráv, který určí, jak směrovat zprávu ke konečnému příjemci, nebo alespoň k dalšímu směrovači. [12].

2.3.5 Transformace

Různé aplikace se nemusí shodnout na formátu stejných koncepčních dat. Odesílatel naformátuje zprávu jedním způsobem, přesto příjemce očekává, že bude naformátována jiným způsobem. Aby to bylo možné sladit, musí zpráva projít zprostředkujícím filtrem a překladačem zpráv, který převádí zprávu z jednoho formátu do jiného [12].

2.3.6 Koncové body

Aplikace může a nemusí mít integrovanou schopnost rozhraní se systémem zasílání zpráv. Pokud aplikace integrovanou schopnost nemá, je potřeba implementovat vrstvu, která zprostředkovává interakci mezi aplikací a systémem zasílání zpráv. Tato vrstva je sada koordinovaných koncových bodů, které aplikaci umožňují odesílat a přijímat zprávy [12].

2.4 Typy integrací z pohledu architektury

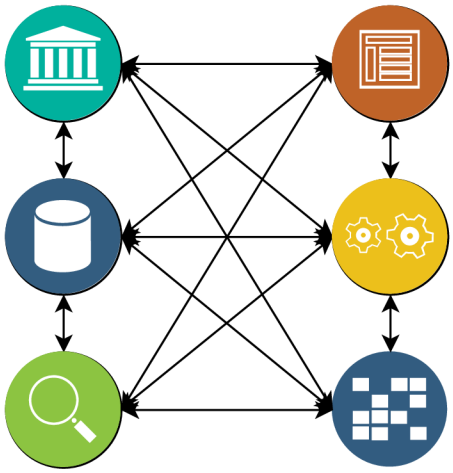
V problematice integrace jsou historicky známé 4 typy návrhů [14]. V této části textu jsou všechny typy znázorněny a popsány.

2.4.1 Integrace Point-to-Point

Integrace point-to-point znamená, že každá aplikace musí být přizpůsobená ke komunikaci s dalšími aplikacemi a současně s dalšími částmi IT systémů dané společnosti. Takto přizpůsobeny musí být všechny aplikace, které spolu chtějí komunikovat. Vytvářet přizpůsobení pro každou aplikaci zvlášť je velmi zdlouhavý proces a také velmi náchylný na vznik chyb. Integrace point-to-point je nejstarším typem integrací, ale v určitých případech je užitečný. Tento typ integrace se využívá pro úzce svázaná řešení [14].

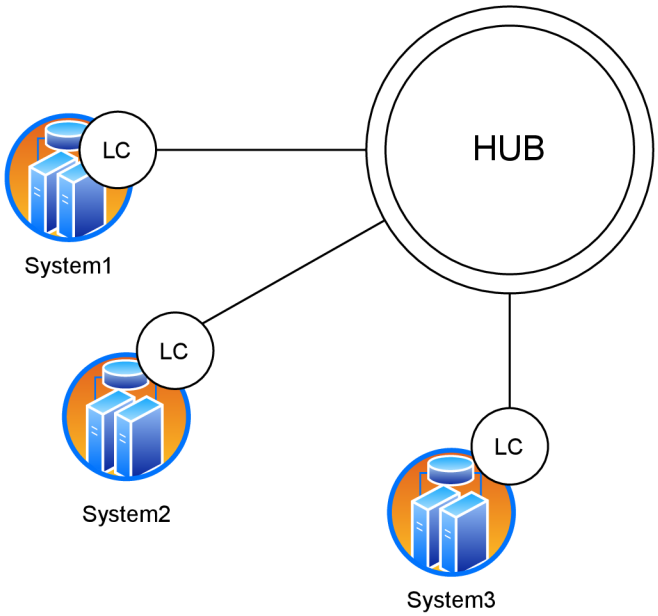
2.4.2 Integrace Hub-and-Spoke

Tato topologie se skládá z rozbočovače (centrální bod, anglicky hub), paprsků (systémy spojené s rozbočovačem) a jednoduchých konektorů (konektory, které je třeba zkonstruovat aby systém mohl komunikovat s rozbočovačem). Hlavní výhodou tohoto řešení je, že ve srovnání s point-to-point je potřeba mnohem méně konektorů pro připojení, v případě že je potřeba integrovat velké množství systémů. Logika integrace je v tomto případě umístěna do centra a lze ji centrálně spravovat a řídit. Když vezmeme v úvahu komunikační protokol, nebyl vytvořen žádný standard a každý vývojář integračních systémů využívá vlastní proprietární protokol. Toto jednání vede



Obr. 2.1: Topologie point-to-point.

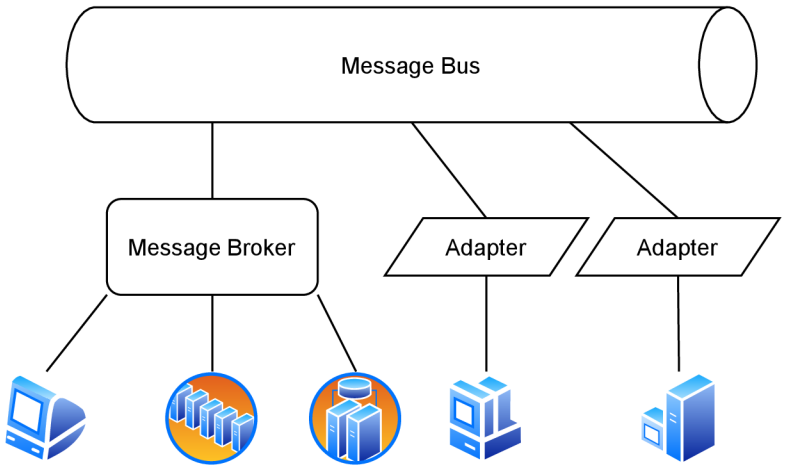
k zablokování dodavatele. Centralizovaná povaha tohoto stylu je také jeho hlavní nevýhodou - rozbočovač je jediným bodem selhání a překážkou. Když rozbočovač selže nebo se přetíží, selže celá topologie integrace, nebo rychle roste doba odezvy. Tento problém lze vyřešit přidáním podpory klastrování a implementací podpory převzetí služeb při selhání pro rozbočovače, to je v této topologii poměrně komplikovaný úkol [14].



Obr. 2.2: Topologie Hub-and-Spoke.

2.4.3 Enterprise Messagge Bus

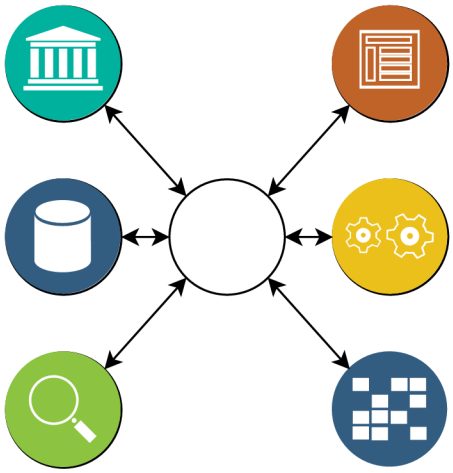
Postupem vývoje se Hub-and-Spoke vyvinula do řešení založeného na MOM (Middleware zaměřený na zasílání zpráv) s názvem Enterprise Message Bus, dále jen EMS. Centrální částí topologie je sběrnice zpráv. Sběrnice zpráv nabízí pouze podporu integrace a to tak, že nabízí systémové kanály pro ukládání zpráv. Jeden nebo více integračních brokerů, kteří poskytují integrační logiku (transformace, směrování atd.), je připojeno ke sběrnici zpráv. Protokol pro komunikaci je stále proprietární, stejně jako u topologie Hub-and-Spoke, což je jednou z nevýhod topologie EMS. Systémy lze připojit ke sběrnici pro zasílání zpráv buď přímo pomocí adaptéru (pokud je systém schopen komunikovat pomocí proprietárního formátu zprávy) nebo připojení zajišťuje integračního broker. Hlavních výhoda této topologie je to, že není potřeba přidávat konektory ke každému systému připojenému ke sběrnici. Komunikace může podporovat vytrvalost zpráv, možnosti opětovného doručení a další aspekty interakce MOM. Spolehlivost a propustnost topologie je také lepší než v topologii Hub-and-Spoke [14].



Obr. 2.3: Topologie EMS.

2.4.4 Enterprise Service Bus

Myšlenka Enterprise Message Bus inspirovala vznik topologie Enterprise Service Bus (ESB), která se dnes velmi běžně používá jako páteř pro komunikaci v podnikovém prostředí [14]. ESB je kolekce služeb middlewaru, která poskytuje možnosti integrace. Logika integrace je umístěna uvnitř sběrnice. Integrované systémy jsou připojeni k ESB prostřednictvím vstupních bodů. Připojení systému je znázorněno na obrázku 2.4.



Obr. 2.4: Topologie ESB.

3 Cloudové systémy

Cloud nebo také cloud computing je model, který poskytuje služby nebo programy jednoduše přístupné z internetu. Uživatelé mohou ke cloudovým službám přistupovat vzdáleně, například pomocí internetového prohlížeče. V případě, že je služba placená, uživatel neplatí za vlastní software, ale za jeho využívání [15].

Velký posun v odvětví cloudových systémech je motivován myšlenkou, že zpracování a ukládání dat může být efektivnější ve velkých výpočetních farmách a úložných systémech. Cloud computing umožňuje škálovatelnou výpočetní sílu a uživatelům lze účtovat pouze takový výkon, jaký si vyžádají [16]. Takové jednání je nákladově efektivní. Nákladovou efektivitu také zaručuje možnost multiplexování zdrojů. Data aplikací mohou být uložena blíže k místu, kde mají být využívána a to způsobem nezávislým na zařízení a poloze. Ve výsledku jde říci, že společnosti využívající cloud šetří své náklady absencí vlastních IT týmů a nepotřebným získáváním vlastního hardwaru a jeho údržbou [16].

3.1 Výhody a nevýhody cloudových systémů

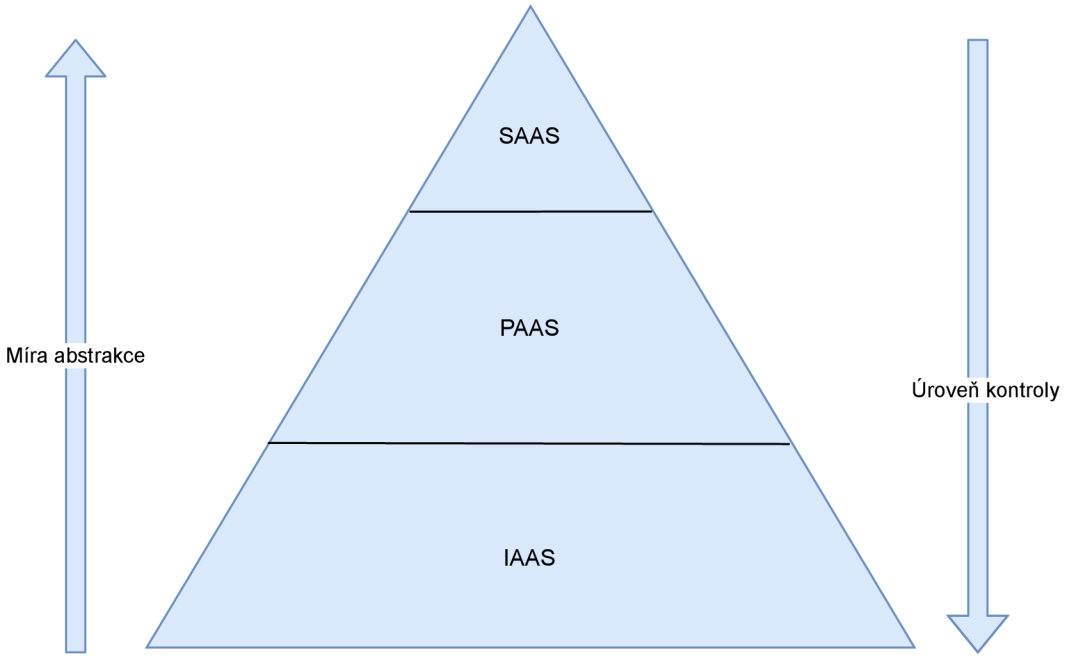
Hlavní výhodou cloudových systémů je využití výpočetní síly bez nutnosti správy vlastního technického vybavení a personálu. Další výhodou je také služba „pay as you go“. Jde o průběžný systém financování, ve kterém zákazník, nebo uživatel nepředplácí službu dopředu, ale platí pouze částku, která je vypočtená na základě využívaných služeb a zdrojů cloudu. V případě potřeby je možné dynamicky měnit využívaný výkon, nebo služby [16].

Většina běžných uživatelů využívá cloud ve formě e-mailu, nebo sdíleného uložiště dat (gmail, dropbox). Největší výhodou těchto služeb je dostupnost a to z jakehokoliv místa, kde je připojení k internetu.

Nevýhodou je malá kontrola nad vlastními daty. Data se ukládají na různých místech světa a nemůže být ovlivněno jejích případné zneužití. Člověk se stává závislý na poskytovatelích cloudových služeb [16].

3.2 Modely cloudových služeb

Cloudové prostředí se skládá ze tří koncepčních vrstev generalizovaného cloudového prostředí, které definuje základní služby poskytované poskytovatelem cloudu [17]. Není vždy pravidlem, že jedna služba využívá právě jednu vrstvu.



Obr. 3.1: Grafické zobrazení vrstev cloudového prostředí

3.2.1 Software jako služba

Software jako služba (SaaS – Software as a Service), jak název napovídá, je služba cloudové technologie, kde poskytovatel SaaS hostuje různé aplikace v cloudu a zpřístupňuje je zákazníkům přes internet. SaaS je také známý jako cloudové aplikační služby a je jednou z nejčastěji používaných cloudových služeb [17].

Poskytovatel SaaS, kromě zpřístupnění aplikace koncovým uživatelům je odpovědný za poskytování služeb, jako jsou správa dat klienta, úložiště a správa aktualizací aplikací. Většina aplikací SaaS funguje přímo ve webových prohlížečích. To znamená, že koncoví uživatelé potřebují pro přístup k aplikacím pouze počítač s internetovým připojením a prohlížeč. V důsledku toho nemusí obchodní organizace využívající model SaaS najímat samostatné pracovníky pro úkoly, jako jsou stahování, instalace, aktualizace a správa softwaru na každém počítači zvlášť. Zatímco se o tyto problémy stará poskytovatel SaaS, mohou se zaměstnanci soustředit na naléhavější a důležitější úkoly [17].

Doručovací model SaaS funguje následujícím způsobem – Poskytovatel hostuje aplikace na jednom centrálním místě. Klient zaplatí poskytovateli všechny služby spojené s aplikacemi. V doručovacím modelu SaaS poskytovatel poskytne klientovi síťový přístup ke kopii aplikace. Všichni koncoví uživatelé si pak stáhnou kopie této aplikace na své stroje. V tomto případě mají všichni koncoví uživatelé stejný zdrojový kód aplikace. Pokud je třeba zavést jakékoli změny, nebo aktualizace, bude to pro poskytovatele SaaS i pro klienta snadné.

V závislosti na dohodě mezi poskytovatelem SaaS a klientem mohou být data klienta uložena buďto na počítači, v cloudu nebo na obou místech. Prostřednictvím SaaS mohou organizace také použít pomoc poskytovatele k migraci vlastních aplikací do cloudového systému, odkud bude poskytovatel hostovat a pokračovat v údržbě aplikací.

3.2.2 Platforma jako služba

Platforma jako služba (PaaS – Platform as a service) je službou cloudové technologie, kde poskytovatel služby poskytuje platformu klientovi nebo koncovému uživateli za účelem vytvoření softwaru. Poskytovatel PaaS poskytne komponenty infrastruktury související s platformou, jako jsou úložiště, servery, operační systémy a síťová zařízení. Poskytovatel je rovněž odpovědný za konfiguraci a údržbu všech výše uvedených komponent. Poskytovatelé navíc klientovi poskytují služby, jako je podpora programovacích jazyků, systémy správy databází, knihovny a podobně. [17].

Poskytovatel PaaS v zásadě vytváří framework, který pak vývojáři používají k vývoji a údržbě aplikací. Poskytovatel PaaS je pověřen odpovědností za poskytování různých hardwarových a softwarových nástrojů vývojářům, kteří jsou hostováni v cloudu. Architektura PaaS funguje způsobem, že infrastruktura zůstane skrytá před vývojáři a uživateli. Poskytovatel PaaS vytváří optimalizované prostředí a platformu pro uživatele k instalaci a spouštění jejich aplikací.

PaaS může najít uplatnění v určitých scénářích, které zahrnují [17]:

- Návrh, vývoj, testování a nasazení aplikací,
- integrace databází,
- integrace pomocí webové služby.

3.2.3 Infrastruktura jako služba

Infrastruktura jako služba (IaaS – Infrastructure as a Service) je cloudová technologická služba, kde poskytovatelé IaaS zpřístupňují infrastrukturní služby. Poskytovatel například prostřednictvím virtualizační technologie hostuje součásti infrastruktury, jako jsou virtuální stroje, virtuální sítě LAN, nebo úložiště.

Poskytovatel IaaS poskytuje svému klientovi buď řídicí panel nebo API, které přináší koncovému uživateli přístup ke všem službám. Takto má koncový uživatel úplnou kontrolu nad celou infrastrukturou [17].

Spolu s infrastrukturními službami poskytovatel poskytuje doprovodné služby. Některé příklady těchto služeb jsou podrobná fakturace, vyvažování zátěže a zvyšování spolehlivosti. Poskytovatel IaaS poskytuje také služby související s ukládáním dat, jako je zálohování dat, replikace dat a obnova dat [17].

IaaS lze přirovnat k tradičnímu datovému centru v tom smyslu, že poskytne koncovému uživateli veškerou infrastrukturu, technologii a schopnosti datového centra. Rozdíl však spočívá ve skutečnosti, že klient nemusí utrácet obrovské množství prostředků na nákup a údržbu celé infrastruktury.

IaaS se stará o nastavení virtuálního datového centra a jeho údržbu pro klienta. Klient je zase zcela odpovědný za své vlastní aplikace, operační systémy, middleware a data. Klient nebo kupující pronajímá prostor v cloudu prostřednictvím poskytovatele IaaS. Poskytovatel IaaS zase poskytuje klientovi virtuální servery a virtuální úložiště. Poskytovatel pomáhá klientovi při budování sítě kolem těchto služeb. Koncoví uživatelé používají své vlastní operační systémy a aplikace na finální platformě, čímž snižují další náklady. IaaS je využíváno v následujících scénářích [17]:

- Vývojové prostředí,
- testovací prostředí,
- datové uložiště,
- datová analýza,
- webové stránky a webové aplikace s uživatelskými interakcemi.

3.3 Modely nasazení

Cloudové systémy se dělí podle modelu nasazení na čtyři základní typy, které nejsou vzájemně slučitelné [18].

3.3.1 Veřejný

Veřejný cloud je také označován jako klasický cloud. Je to model, ve kterém je výpočtová služba poskytnuta široké veřejnosti. Může být vlastněna, spravována a provozována obchodní, akademickou nebo vládní organizací nebo jejich kombinací. Existuje v prostorách poskytovatele cloudu [18].

3.3.2 Soukromý

Cloudový systém využívající soukromý model je zřízen pro výhradní použití jednou organizací zahrnující více spotřebitelů. Může být vlastněna, spravována a provozována organizací, třetí stranou nebo jejich kombinací a může existovat v prostorách i mimo ně [18].

3.3.3 Komunitní

Jedná se o cloudový model pro výhradní použití konkrétní komunitou spotřebitelů z organizací, které jsou spojené některými zásadami (požadavky na zabezpečení,

zásady a dodržování předpisů). Může být vlastněna, spravována a provozována jednou nebo více organizacemi v komunitě, třetí stranou nebo jejich kombinací a může existovat v prostorách i mimo ně [18].

3.3.4 Hybridní

Infrastruktura hybridního cloudu se skládá z dvou nebo více cloudových systémů (soukromých, veřejných, nebo komunitních), které zůstávají jedinečnými entitami, ale jsou spojeny standardizovanou, nebo dedikovanou technologií, která umožňuje přenositelnost dat a aplikací [18].

3.4 Bezpečnost cloudu

Informace uložené v cloudu se často považují za cenné. Může existovat spousta potencionálních útočníků, kteří chtějí data uložená na cloudu využít se zlým úmyslem. Existuje také spoustu osobních informací, které lidé ukládají do svých počítačů. Tato data se nyní přenášejí do cloudu. Cloud může přinést bezpečnostní rizika a výzvy pro správu IT, což může být pro organizaci náročnější, i když vezmeme v úvahu úsporu nákladů dosaženou přechodem do cloudu. Proto je pro organizace velmi důležité pochopit jejich požadavky dříve, než se rozhodnou pro různé modely nasazení dostupné v cloudu [19].

3.4.1 Referenční model cloudové bezpečnosti

Abychom měli spravedlivou představu o tom na co se zaměřit, pokud jde o bezpečnost v cloudu, diskutuje referenční model zabezpečení cloudové aliance o důležitosti vědění [19].

- Způsob nasazení cloudových služeb – patří sem čtyři výše popsané modely nasazení cloudu (privátní cloud, veřejný cloud, komunitní cloud a hybridní cloud).
- Způsob spotřebovávání cloudových služeb – různé cloudové služby (SaaS, PaaS, Iaas) mají související bezpečnostní riziko. Je důležité pochopit, kde leží hranice zabezpečení, pokud jde o cloud computing. To je často popsáno ve vztahu k umístění obvodu správy nebo zabezpečení organizace. (obvykle definováno přítomností brány firewall).
- Opětovná permietrie a narušení hranic důvěry – vysoká úroveň vzájemného propojení a výměny informací v důsledku využívání obchodních příležitostí vystavuje podnikání různým bezpečnostním rizikům. Tato rizika však nejsou dostatečně zmírněna tradičními kontrolami statické bezpečnosti.

Schopnost mapovat cloudové modely na řídicí rámce je klíčem k zahájení, implementaci, udržování a zlepšování bezpečnosti informací v organizaci.

Zavedení bezpečnostních kontrol pomáhá zmírnit všechna bezpečnostní rizika dříve, než vyústí v narušení zabezpečení. K tomu je důležité klasifikovat cloudovou službu proti modelu cloudové architektury, aby bylo možné mapovat architekturu zabezpečení a také obchodní, regulační a další požadavky na dodržování předpisů. Informační aktivum lze poté důkladně chránit před výsledky.

Pochopení toho, jak se architektura, technologie, procesy a požadavky na lidský kapitál mění nebo zůstávají stejné při nasazování služeb cloud computing, je zásadní. Bez jasného pochopení architektonických implikací na vyšší úrovni je nemožné racionálně řešit podrobnější problémy.

Zabezpečení v cloudu přímo souvisí s modely nasazení v cloudu (veřejné, soukromé, hybridní a komunitní) a modely cloudových služeb (SaaS, PaaS a IaaS). Modely nasazení v cloudu a modely služeb poskytují základ pro referenční model cloudového zabezpečení, který vysvětluje odpovědnost poskytovatelů služeb i uživatelů za zajištění zabezpečení dat v cloudu [20].

3.5 IBM Cloud

Společnost IBM se zaměřuje na poskytování služeb jiným společnostem a součásti portfolia je IBM cloud. IBM Cloud je dostupný i pro veřejnost, ale není veřejností tak často využíván. IBM Cloud je kombinace PaaS, SaaS a IaaS, které jsou popsané v části 3.2. IBM Cloud nabízí také *Function-as-a-Service* neboli FaaS. IBM Cloud udržuje 60 datových center po celém světě, což umožňuje dobrou globální škálovatelnost. Cloudová platforma IBM se skládá z několika komponent, které společně vytvářejí konzistentní cloudové prostředí. Komponenty cloudové platformy jsou:

- **Console** – komponenta nabízející front-end pro řízení zdrojů uživatele.
- **Identity and Access management** – řídí autentikaci a kontrolu přístupu.
- **Catalog** – katalog služeb nabízených společností IBM.
- **Account and Billing management** – poskytuje cenové plány pro jednotlivé služby.

3.5.1 Katalog služeb IBM Cloud

IBM Cloud katalog je seznam všech služeb a funkcí poskytovaných společností IBM, případně společnostmi třetích stran. V současné době se v katalogu nachází stovky dostupných služeb a funkcí, které jsou dále rozdělovány do kategorií a podkategorií. V následují části je popsáno jen několik kategorií, zejména ty, které byly využity ke splnění cílů této práce.

Kontejnery

Služba kontejnery (Containers) je založena na systému Docker, který umožňuje vybudování prostředí pro spuštění programu podle instrukcí uložených spolu se zdrojovým kódem. Docker slouží ke kontejnerovým operacím.

Uživatel IBM Cloudu má možnost si založit vlastní registr kontejnerů přímo v prostředí IBM Cloud, kde může také svoje kontejnery spravovat. Tato služba se nazývá Container Registry. Součástí Container Registry je Vulnerability Advisor, který slouží pro kontrolu obrazů v registru a zabraňuje spouštění škodlivého kódu.

IBM Cloud poskytuje také řídící nástroj pro vytváření clusterů s názvem Kubernetes. Kubernetes slouží pro plánování kontejnerů na clusteru výpočtových strojů. Tato služba je využívána vývojáři pro vývoj aplikací s vlastnostmi kontejnerů. Mezi základní možnosti Kubernetes clusteru patří monitorování, automatické škálování a ochrana infrastruktury uživatelova clusteru. Kubernetes cluster také podporuje interakci mezi clusterem a dalšími nabízenými službami IBM cloudu.

Uložiště

Kategorie uložiště (Storage) nabízí uživateli výběr ze tří typů uložiště.

- **Block Storage** – zvyšuje kapacitu virtuálních a fyzických serverů na IBM Cloud.
- **File Storage** – je systém založený na *Network File System*, který pomáhá vytvářet sdílený soubor a minimalizovat cenu ukládaných dat.
- **Cloud Object Storage** – je dobře škálovatelné cloudové uložiště poskytující trvanlivost, odolnost a bezpečnost. Poskytuje jednoduchou správu uložených dat a propojení s jinými aplikacemi v IBM Cloud.

Všechny tři typy jsou rychlostně a výkonově nastavitelné podle potřeb uživatele.

Databáze

Důležitou vlastností cloudu je spolehlivá úschova dat. K tomu slouží nástroj Databases. Pro ukládání JSON dokumentů je uživateli nabízena jednoduše škálovatelná služba Cloudant. Správu cloudantu plně zabezpečuje IBM. Další nabízené databázové služby od IBM jsou například:

- PostgreSQL,
- Redis,
- Elasticsearch,
- MongoDB,
- Db2.

Nástroje pro vývojáře

Kategorie Developer Tools, v překladu nástroje pro vývojáře, poskytuje nástroje k ulehčení vývoje, nasazení a interakci s aplikacemi, infrastrukturami, nebo kontejnery. Tyto nástroje pomáhají zkracovat čas nutný pro vývoj a zvyšovat spolehlivost vyvíjeného softwaru. V kategorii Developer Tools jsou například následující nástroje:

- **Auto-scaling** – automaticky zvyšuje anebo snižuje výpočtovou kapacitu a počet aplikací na základě potřeb uživatele.
- **Availability Monitoring** – provádí pravidelnou kontrolu dostupnosti z různých lokací.
- **LogDNA** – tento nástroj se dělí na dvě kategorie. IBM CloudActivity tracker a IBM Log Analysis. První z nich pomáhá sledovat aktivity na uživatelu IBM účtu, monitoruje ukládání událostí a ukládání dat do IBM Cloud Object Storage. IBM Log Analysis umožnuje datakovat a spravovat všechny druhy logu.
- **Event Management** – pomáhá vývojářům rychle reagovat na vzniklé provozní události, případně je úplně automatizovat, čím zvyšují kvalitu poskytované služby zákazníkovi.

Bezpečnost a identita

Kategorie Security and Identity (bezpečnost a identita) obsahuje sadu služeb, které zabezpečují ochranu uživatelských aplikací. Za správu veškerých služeb poskytující ochranu platformy slouží jednotná konzole Security Advisor. V kategorii bezpečnost a identita nalezneme například následující služby:

- **App ID** – Přidává autentikaci na aplikace pomocí e-mailové adresy a hesla, případně pomocí účtů jiných organizací jako je facebook čí google. Informace se ukládají do škálovatelného registru uživatelovi dané aplikace.
- **Certificate Manager** – zajišťuje správu SSL certifikátů pro aplikace v prostředí IBM Cloud. SSL certifikáty zabezpečují šifrovaný přenos údajů od uživatele na server a naopak. Certifikáty se využívají zejmena při zadávání citlivých dat jako jsou bankovní či osobní citlivé údaje.
- **Key Protect** – zabezpečuje generování a správu šifrovacích klíčů používaných v aplikacích. Je využíváno *IBM Cloud Hardware Security* (HSM). HSM je kryptoprocesor určený na ochranu šifrovacích klíčů po čas jejich existence. Zajišťuje správu v odolném hardwarovém zařízení, čím poskytuje maximální ochranu.
- **Contrast Security** – posuzuje zranitelnost a monitoruje případné útoky na software, čímž pomáhá identifikovat a opravovat chyby při vývoji.

- **FusionAuth** – je platforma na správu zabezpečení uživatelů. Poskytuje služby jako jsou lokalizace, správa hesel, hashování nebo řízení přístupu. Umožňuje také migraci uživatelů bez nutnosti změny jejich hesel.

4 Vývoj aplikace

V této části je popsána praktická část práce. Cílem diplomové práce je návrh a implementace funkční a plně konfigurovatelné integrační aplikace. Integrace bude probíhat vždy mezi dvěma ITSM nástroji. Práce je zaměřena na architektonická rozhodnutí pro dané případy užití. Výsledná aplikace bude nasazená na IBM Cloud.

4.1 Architektonický návrh

Design aplikace je velmi důležitým prvkem v architektuře a to jak z estetických, tak fyzických důvodů. Architektura aplikací se velmi podobá architektuře budov v reálném životě. Architektonické řešení může mít vážné důsledky ve spolehlivosti a robustnosti výsledné aplikace v závislosti na určitém využití. Ve stavebnictví nemohou být použity technologie pro výstavbu rodinného domu k výstavbě mnohem větší stavby, jako je výšková budova nebo sportovní aréna. Stejným způsobem musí být zvolen architektonický návrhový vzor pro požadovanou aplikaci, pracovní zátěž a očekávanou úroveň použití.

Pro implementaci integrační aplikace v této diplomové práci byl zvolen architektonický vzor založený na mikroslužbách.

4.1.1 Mikroslužby

Pojem „Microservice Architecture“, v překladu architektura založená na mikroslužbách, popisuje konkrétní způsob návrhu softwarových aplikací jako sadu nezávisle nasaditelných služeb [21]. I když neexistuje přesná definice tohoto architektonického stylu, existuje určitý počet charakteristických vlastností. Zjednodušeně se jedná o dekompozici celého projektu na malé samostatné aplikace představující služby. Každá služba by měla obstarávat konkrétní oblast systému. Vzájemná komunikace mezi jednotlivými službami probíhá prostřednictvím kombinace rozhraní REST API, event streaming, message brokers nebo podobných [22].

Pro vývoj aplikace v diplomové práci byla zvolena architektura založená na mikroslužbách. Hlavní výhodou oproti jiným možným řešením, jako je například monolitická aplikace, je velmi dobrá škálovatelnost. V architektuře založené na mikroslužbách lze jednotlivé služby škálovat individuálně. Mikroslužby vyžadují menší infrastrukturu než monolitické aplikace, protože umožňují přesné škálování pouze těch komponent, u kterých je to potřebné. Jeden z problémů při škálování monolitických aplikací je jejich příliš velká granularita a z toho plynoucí větší náklady na hardware či na prostředky dostupné v cloudu.

Architektura založená na mikroslužbách přináší také výhodu v organizaci práce, obzvlášť ve velké organizaci, jako je IBM. Přínosem využití mikroslužeb ve velké organizaci je to, že mikroslužby lépe odrážejí způsob jakým vedoucí organizace chce strukturovat a provozovat své týmy a vývojové procesy [21]. Dalšími výhodami mikroslužeb oproti monolitické aplikaci jsou:

- Nezávislá implementace,
- použití optimální technologie.

Nezávislá implementace

Jednou z důležitých vlastností mikroslužeb je to, že není potřeba obrovského aktu ke změně jednoho řádku kódu v aplikaci. Důvodem jsou relativně malé služby a fakt, že jsou nezávisle nasaditelné. Jednoduchá oprava chyb ale není jedinou hodnotou rozdělení do nezávisle implementovaných služeb.

Model založený na mikroslužbách umožňuje vytvářet malé týmy vývojářů napříč jednotlivými službami. Tyto týmy mohou pracovat nezávisle na sobě. Malá velikost služeb v kombinaci s jejich jasnými hranicemi a komunikačními vzory usnadňuje novým členům týmu pochopit základ kódu a rychle k němu přispět. To je jasná výhoda z hlediska rychlosti i morálky zaměstnanců organizace.

Použití optimální technologie

Monolitické aplikace bývají napsané v jednom programovacím jazyku. Výhodou architektury založené na mikroslužbách je jejich nezávislost. Každá mikroslužba může být napsána v libovolném jazyku nebo každé může využívat jiný framework. Technologie se neustále mění a systém složený z několika menších služeb je mnohem snazší a levnější vyvíjet s modernějšími technologiemi. Nejmodernější technologie mohou být velmi rychle implementovány hned, jakmile jsou k dispozici.

Nevýhody mikroslužeb

Hlavní nevýhody mikroslužeb oproti monolitickým aplikacím jsou.

- **Správa více samostatných entit** – správa více samostatných entit vyžaduje důsledný monitoring jednotlivých mikroslužeb. Velkou nevýhodou oproti jedné monolitické aplikaci je správa verzí samotných mikroslužeb. Jednotliví verze nemusí být kompatibilní se zbytkem systému.
- **Složitější proces nasazení** – proces nasazení vyžaduje sestavení všech mikroslužeb a zajištění komunikace mezi nimi. Toto je časově i procesně náročný úkon. V případě diplomové práce je tento proces automatizován a je popsán v části 4.6.

4.2 Výběr programovacího jazyku

V dnešní době existuje spoustu technologií pro vytváření cloudových řešení, které zvládnou značné pracovní vytížení, mají velkou uživatelskou základnu a poskytují vysokou míru zabezpečení. Historicky jsou nejznámější Java, .NET a PHP. Ve všech případech se jedná o vyspělé technologie vydané zhruba před 20 lety. Pro tvorbu webových aplikací byly tyto programovací jazyky rozšířeny pomocí technologií jako jsou JavaEE nebo ASP.NET [23]. Naproti tomu Node.js, s prvním vydaním v roce 2009, byl od samého začátku vytvořen jako čistě webová technologie. Této technologii se podařilo velmi rychle zaujmout místo jako jedna z primárních možností pro nové produkty SaaS a podnikové responzivní webové aplikace pro interní použití. Integrační aplikace v této diplomové práci je zhotovena v Node.js.

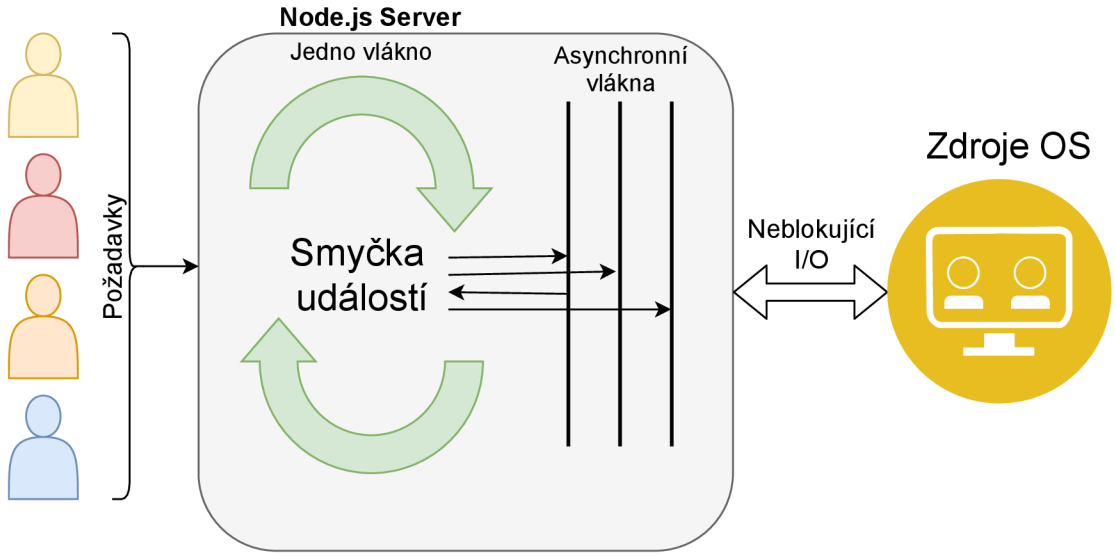
4.2.1 Node.js

Node.js je prostředí umožňující spouštět JavaScript aplikace mimo webový prohlížeč. Tato platforma je postavena na Chrome V8 JavaScript enginu. Je to stejný engine, který využívá webový prohlížeč Google Chrome. Prostředí Node.js lze využít k vývoji aplikace na straně serveru [24]. Node.js je v dnešní době hodně populární, naznačuje tomu fakt, že je využíván společnostmi jako je Microsoft nebo Yahoo [25].

Jádro Node.js je sestaveno z asynchronních funkcí a jedno-vláknové smyčky událostí (single-thread event-driven architecture). Celé to funguje tak, že do smyčky událostí vstupují požadavky od uživatelů (události), které jsou postupně přidělovány nezávislým funkcím. Asynchronní neboli nezávislé funkce manipulují se zdroji operačního systému. Manipulace se zdroji je prováděna pomocí událost a jejich volání je opět zařazeno do hlavní smyčky událostí. Takový mechanismus se nazývá asynchronní neblokující vstupně/výstupní mechanismus [24]. Celý tento proces znázorňuje obr. 4.1.

Z obr. 4.1 je zřejmé, že celý proces řídí jedna smyčka události. Toto je jednoduché, ale efektivní řešení, které dělá z Node.js výkonnou technologii a to zejména v oblasti webových aplikací s velkým počtem uživatelských volání. Je to zároveň i případ řešeného problému v této diplomové práci. Mezi ITSM nástroji může docházet k nahodile velkému počtu volání, které je snaha zpracovat bez nutnosti využití velkého výpočetního výkonu. Klíčovými ukazateli výkonu webových aplikací jsou tři aspekty:

- Škálovatelnost,
- latence,
- propustnost.



Obr. 4.1: Node.js smyčka událostí.

Node.js díky neblokujícímu mechanismu dosahuje nízké latence a velké propustnosti. V programovacím jazyce využívajícím blokující mechanismus, server pro každou příchozí žádost vytvoří nový podproces spuštění, nebo rozvětví nový proces pro zpracování žádosti a odeslání odpovědi [23]. Z koncepčního hlediska to dává dokonalý smysl, ale v praxi to vede k velké režii. Přestože jednotlivá vlákna mají menší požadavek na paměť a CPU, při vytvoření velkého počtu vláken se stávají neefektivní. Při vytvoření velkého počtu vláken, systém vynakládá spoustu zdrojů pro plánování vláken a přepínání kontextu a to vede ke zvyšování latence a snižování propustnosti [25].

Node.js má jiný přístup. Spustí smyčku událostí s jedním vláknem zaregistrovanou v systému pro zpracování příchozích žádostí (událostí) a každé nové připojení způsobí spuštění funkce zpětného volání JavaScriptu. Funkce zpětného volání zpracovává požadavky s výše popsaným neblokujícím mechanismem. Přístup uzlu k škálování pomocí funkcí zpětného volání vyžaduje méně paměti pro zpracování více příchozích žádostí než většina konkurenčních technologií, které se škálují vytvářením vláken. Do těchto technologií patří například Java [24].

Zásadním důvodem využití Node.js v aplikaci tvořené v této diplomové práci je využití pouze jednoho vlákna a využití neblokujících volání. ITSM nástroje jsou aplikace, které nevyžadují real-time synchronizaci a tím pádem jsou předností menší náklady na zdroje na úkor pomalejšího zpracování velkého počtu transakcí.

Existují i další důvody, kvůli kterým je Node.js populární mezi vývojáři a to:

- Pro mnoho vývojářů je pohodlné využívat JavaScript (ECMAScript) na straně serveru i na straně klienta. Není tedy nutné učit se více technologií nebo jazyků.

Následkem je také snížení nákladů na vývojáře.

- Dalším důvodem je dobrá podpora komunity. To má významný vliv na rychlost vývoje a rychlost učení.

Balíčkovací systém npm

Spolu s vývojem Node.js existují nástroje pro správu závislostí. Takzvaný balíčkovací systém. Balíčkovacích systémů existuje více, nejznámější systémy spojované s Node.js jdou npm a Yarn. V této diplomové práci je využíváno npm. Npm je přímo součástí instalace Node.js a ovládá se pomocí příkazové řádky [26].

Pomocí npm jde obecně instalovat i spravovat závislosti ve svých projektech, ale také sdílet a distribuovat svůj JS kód. Npm využívá veřejný registr balíčků, který patří k těm největším vůbec. Jeho obsahem jsou jednoduché knihovny i celé frameworky jako je AngulerJS nebo React [26].

4.2.2 TypeScript

TypeScript byl vytvořen firmou Microsoft v roce 2012 a je vydáván jako open-source. Jedná se o nadstavbu jazyka JavaScript, který ho rozšiřuje o statické typování, třídy, rozhraní a další věci známé z Objektově orientovaného programování. TypeScript je kompilován do jazyka JavaScript a tím pádem lze říct, že každý platný JavaScriptový program je také platný TypeScript [27]. Na následujících příkladech lze pozorovat čitelnost kódu psaného v programovacím jazyku JavaScript a TypeScript.

```
var greeter = (function () {
  function Greeter(message) {
    this.greeting = message;
  }
  Greeter.prototype.greet = function () {
    return 'Hello, ' + this.greeting;
  }
  return Greeter
});
```

```
class Greeter {
  constructor(private greeting: string) {
  }
  greet() {
    return 'Hello ' + this.greet;
  }
}
```

Psaní kódu v JavaScriptu se díky dynamickému typování zdá být časově výhodnější a jednodušší. Při pohledu na předchozí příklad, TypeSript je mnohem čitelnější. TypeScript má i další výhody proč ho využívat a to zejména na větších projektech a pří týmové práci. TypeScript umožňuje najít chyby ještě před samotným spuštěním programu. Jestliže člen týmu vytvoří část kódu, může okamžitě zjistit, jestli je kompatibilní se zbytkem projektu a případně tuto kompatibilitu vyřešit. To šetří spoustu času při ladění programu. Nejběžnější chyby na které kompilátor jazyku TypeScript narazí jsou:

- objekt není definován,
- objekt neobsahuje potřebné atributy,
- požadovaný typ není kompatibilní s návratovým typem.

Ve výsledku využitá jazyku TypeScript zrychluje celý vývoj a zmenšuje chybovost výsledného programu.

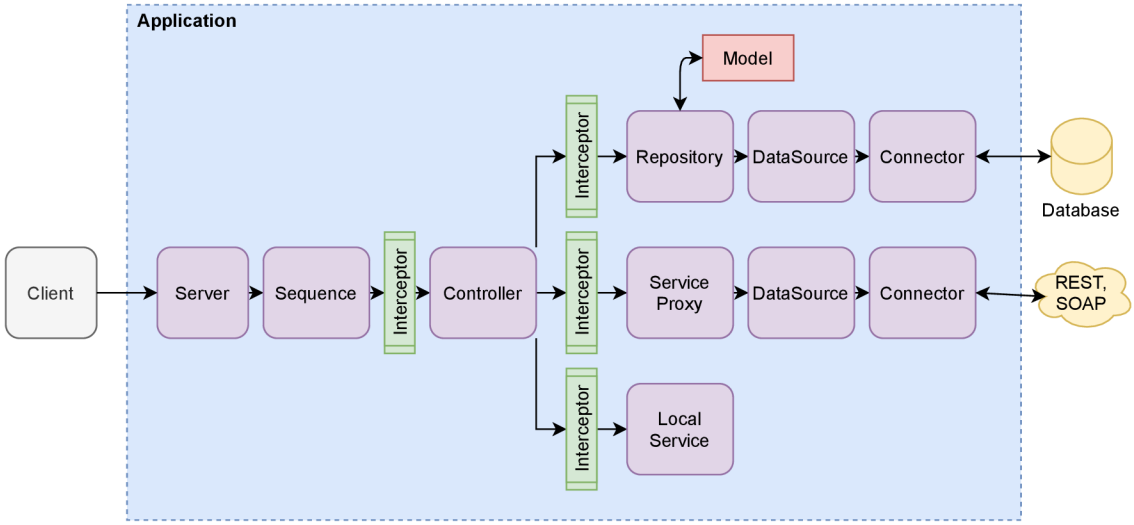
TypeScript je vhodný pro tuto diplomovou práci, protože kódová základna projektu se stále rozrůstá a samotná implementace není zhotovena pouze jedním člověkem.

4.2.3 Loopback framework

LoopBack je vysoce rozšiřitelný open-source framework, který je napsaný pro Node.js [28]. Implementace aplikace v diplomové práci je vyvíjena za pomocí Loopback frameworku verze 4. Tato verze je napsaná celá v TypeScriptu a je založena na frameworku Express. Loopback 4 umožňuje rychle vytvářet API a mikroslužeb složených z back-endových systémů, jako jsou databáze a webové služby používající SOAP nebo REST [28]. V mnoha ohledech může být využití frameworku přítěží pro specifická zadání, jako jsou integrační aplikace. Hlavním důvodem využití frameworku Loopback 4 je využití implementovaných principů *Inversion of Controll* a *dependency injection*, které jsou popsané níže v textu. Principy IoC a DI jsou užitečné zejména pro vytvoření konfigurovatelné aplikace.

Koncept frameworku LoopBack 4

LoopBack 4 definuje klíčové bloky, které jsou stavebními kameny celého frameworku. Každý blok má určitou roli. V následující části budou popsány hlavní bloky frameworku Loopback 4 [28].



Obr. 4.2: LoopBack4 koncept

- **Application** – je centrální třída pro nastavení všech komponent, kontrolerů, serverů a závislostí dané aplikace. Třída *Application* rozšiřuje třídu *Context* o ovládací prvky pro spouštění a zastavování všech serverů.
- **Server** – je třída implementující logiku pro zpracování příchozích transportů, jako jsou REST, gRPC, graphQL. Obvykle poslouchá na konkrétním koncovém bodě. Zpracovává příchozí požadavky a poté vrací příslušnou odpověď.
- **Sequence** – je třída poskytující bezstavové seskupení akcí, které určují způsob, jakým server reaguje na požadavky.
- **Interceptor** – zprostředkovává vyvolání statické funkce, která je prováděna při volání metody nebo instance.
- **Controller** – je třída, ve které je implementována obchodní logika aplikace. Funguje jako most mezi HTTP/ REST API a modely databáze.
- **Service** – je v třída, která odkazuje na objekt s metodami pro provádění místních nebo vzdálených operací. Tyto objekty jsou zapisovány do contextu aplikace. V rámci LoopBack 4 existují 3 typy služeb: proxy, class a provider.
- **Repository** – je typ služby, která představuje datovou kolekci v rámci datového zdroje (DataSource).
- **Model** – je definice objektu k datovému zdroji. Poskytuje speciální dekorátory pro přidávání metadat k objektu. Využívá se pro definici datové jednotky

spojené s Repository a DataSource.

- **DataSource** – je pojmenovaná konfigurace pro instanci konektoru, která představuje data v externím systému.
- **Context** – je jádro aplikace. Představuje abstrakci stavů a závislostí v celé aplikaci. Je to také lokální registr pro všechny konstrukční části frameworku LoopBack 4.
- **Component** – je balíček, který umožňuje rozšiřitelnost LoopBack 4 aplikace.

Inversion of Controll a Dependency injection

Inversion of Controll, dále jen IoC, je obecný princip, který upřednostňuje kontrolu objektů z vnější strany nad situací, kdy si objekt sám říká o věci v rámci svého kódu [29]. Výsledkem je dobrá kontrola nad objekty a tento návrh zároveň umožňuje dobrou znovupoužitelnost částí kódu. Framework Loopback 4 implementuje podporu pro IoC podle návrhového vzoru Dependency Injection, dále jen DI. DI umožňuje vkládat závislosti tříd tak, aby jedna třída mohla používat druhou, aniž by na ní měla v době sestavování programu určenou referenci. Ve třídě tedy nemusí být vytvářena konkrétní instance využívané třídy. O životní cyklus třídy se stará framework [29], [28].

Ve frameworku Loopback 4 je využívána komponenta *Context* ke sledování všech závislostí. Existuje více způsobů, jak uložit závislost do komponenty *Context* [28]. Následující příklad ukazuje způsob vložení hodnoty do contextu.

```
1 app.bind(key).to(value)
```

kde:

- **app** – je instance komponenty Context,
- **key** – je klíč, pod kterým je hodnota uložena,
- **value** – je uložená hodnota.

Do komponenty Context jdou ukládat i celé třídy. To je možné následujícím způsobem.

```
1 app.bind(key).toClass(value)
```

Závislosti jdou vkládat do libovolné třídy pomocí klíčového slova *@inject(key)*. Podpora IoC a DI, ve frameworku Loopback 4, byly hlavní důvody, proč je využíván v implementaci aplikace a to z důvodu snadné práce s konfiguračními soubory a objekty.

4.3 Komunikace mezi službami

Jednou z nejvíce se vyskytujících překážek při přizpůsobování architektuře mikroslužeb je způsob komunikace mezi službami [30]. V této části práce bude popsáno, jaké způsoby komunikace jsou možné a proč bylo v této diplomové práci využito asynchronních služeb.

4.3.1 Synchronní komunikace

V relativně malých monolitických aplikacích je využití synchronní komunikace zcela běžnou věcí, protože jde o velmi jednoduchý koncept. Klient odešle požadavek na server a server mu na něj odpoví. Při použití architektury založené na mikroslužbách existuje spousta různých a nezávislých služeb, které po celou dobu vzájemně komunikují. V takovém případě synchronní komunikace přináší určité potíže. Když jedna služba volá službu druhou a tato služba nepošle odpověď včas, nebo vůbec, volající služba také selže. To může způsobit řetězovou reakci selhávání ostatních služeb, což způsobí poruchu celého systému. I když předpokládáme, že volaná služba bude odpovídat včas, vlákno volající služby je blokováno, dokud nedostane odpověď. To může způsobit, že celý systém bude pomalý a nebude reagovat. Při využité synchronní komunikace v integrační aplikaci by se stal systém jedné společnosti závislý na rychlosti systému druhé společnosti a v potřebě poslat větší množství změn by to vedlo k zahlcení. Existují způsoby, jak se s těmito problémy vypořádat, například implementací takzvaných jističů. Tato implementace však vyžaduje značné úsilí. Výhodou synchronní komunikace je, že volající služba vždy obdrží potvrzení, že byl přijat požadavek a byla provedena odpovídající akce.

4.3.2 Asynchronní komunikace

Při použití asynchronní komunikace volající služba nečeká na odpověď od volané služby. To má samozřejmě velkou výhodu, že volací služba není závislá na volaných službách. Pokud selžou, volající služba bude nadále fungovat. Další výhodou je samozřejmě to, že vlákna volajících služeb již nejsou blokována čekáním na odpověď. Asynchronní komunikace umožňuje možnost komunikace One-To-Many, kdy klient může odeslat zprávu více službám najednou. Zatímco u synchronních komunikačních metod bude klient muset odesílat zprávy každé jiné službě samostatně. Existují různé přístupy, které používají asynchronní formu komunikace, každý s vlastními výhodami a nevýhodami [31].

- Notifikace,
- požadavek/ asynchronní odpověď,
- komunikace založena na zprávách.

V této diplomové práci byla využita forma komunikace založena na zprávách, anglicky message-based. Výhodou této metody je, že přichází s vysokým oddělením mezi službami. Služba pouze publikuje zprávu a neví, které služby zprávu zpracují. To je přesně to, co je většinou vyžadováno u mikroslužeb. Služba by si většinou neměla být vědoma ostatních služeb a toho, co dělají se svými zprávami.

Další výhodou oproti jiným komunikačním metodám je ukládání zpráv do vyrovnávací paměti. Pokud je spotřebitel kanálu z nějakého důvodu nedostupný, zprávy se zařadí do fronty, dokud nebudou spotřebovány. Při synchronní komunikaci by obě služby musely být vždy dostupné, jinak by došlo ke ztrátě komunikačního provozu [31].

4.3.3 Apache Kafka

Pro implementaci asynchronní metody komunikace mezi službami bylo využito služby IBM Event Streams z IBM Cloud Catalogu. IBM Event Streams je platforma pro streamování událostí založena na Apache Kafka, která zahrnuje ověřenou technologii *Strimzi*. Jedná se o systém publikování a odběrů zpráv navržený tak, aby byl odolný vůči chybám a poskytoval platformu s vysokou propustností a nízkou latencí. Na obrázku 4.3 je zobrazen koncept Apache Kafka [32].

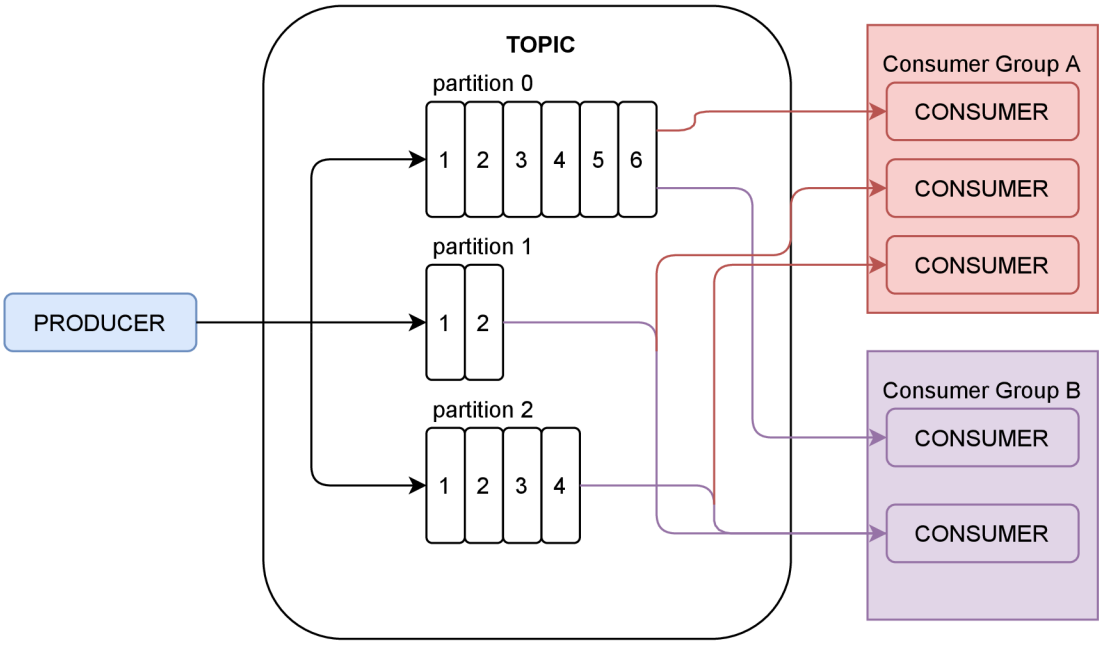
Díky službě postavené na Apache Kafka, IBM Event Streams přímo těží ze všech inovací v komunitě a podporuje klientská rozhraní API Kafka, Kafka Streams a Kafka Connect. Kafka umožňuje vytvářet škálovatelné datové kanály pro mnoho případů využití včetně připojení mikroslužeb, provádění agregace protokolů, získávání událostí a zpracování proudu [32].

V streamech událostí aplikace odesílají data vytvořením zprávy a jejím odesláním do topicu. Přijímání zpráv se provádí přihlášením k odběru daného topicu. Zprávy mohou být přijímány (konzumovány), nebo sdíleny mezi jednotlivými topicy. Zprávy jsou udržovány v seřazeném pořadí. Následující pojmy definují koncepty Apache Kafka [32].

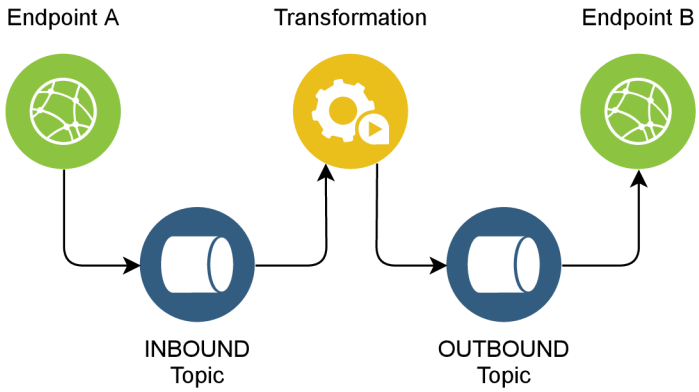
- **Broker** – Kafka cluster se skládá ze sady brokerů. Každý cluster má minimálně 3 brokery.
- **Message** – Message, česky zpráva, je datovou jednotkou v Apache Kafka. Každá zpráva je reprezentována jako záznam, který obsahuje dvě části: klíč a hodnotu. Klíč se běžně používá pro metadata o zprávě a hodnota je tělo zprávy.
- **Topic a partition** – topic je pojmenovaný proud zpráv. Topic je tvořen jedním nebo více oddíly (partition). Zprávy v oddílu jsou seřazeny podle čísla, které se nazývá offset. Pokud téma obsahuje více než jeden oddíl, umožňuje paralelní předávání dat. Toto se využívá pro zvýšení propustnosti oddílů v klastru. Počet

oddílů také ovlivňuje vyrovnávání pracovní zátěže mezi spotřebiteli.

- **Producer** – Producer odesílá zprávy do jednoho, nebo více topiců. Je možno volitelně vybrat oddíl, do kterého budou zasílány zprávy. Je možné taky nakonfigurovat producera, aby upřednosňoval rychlost nebo spolehlivost výběrem úrovně potvrzení, které obdrží z odeslané zprávy.
- **Consumer** – Consumer čte zprávy z jednoho nebo více topiců a zpracovává je. Rozdíl mezi aktuální pozicí consumera a offsetem je novější zprávy je nazýván offset lag. Pokud se offset lag v průběhu času zvyšuje, je to známka toho, že consumer není schopen držet krok. Z krátkodobého hlediska to není problém, ale v dlouhodobém horizontu by mohlo dojít k překročení doby uchování zprávy a ty by mohli být ztraceny.
- **Consumer Group** – Skupina consumerů obsahuje jednoho nebo více consumerů spolupracujících na zpracování zpráv. Zprávy z jednoho oddílu zpracovává vždy jen jeden consumer v každé skupině. Pokud je ve skupině více oddílu než consumerů, tak některý consumer zpracovává zprávy z více oddílů. Pokud ale existuje více consumerů než oddílů, tak některý consumer nezpracovává žádné zprávy.



Obr. 4.3: Koncept Apache Kafka.



Obr. 4.4: Jeden směr integrace mezi ITSM tooly

4.3.4 Využití front v aplikaci

Jak je na obr. 4.4 vidět, v jednom směru komunikace jsou využity 2 topicy. Po autentizaci a přijmutí požadavku od odesilatele je téměř hned odeslána zpráva do INBOUND topicu. Před odesláním zprávy do fronty se neprovádí žádné připojování do databáze či volání vzdálených služeb. To zaručuje velice rychlou odezvu odesilateli.

Do dalšího topicu jsou odesílány zprávy po transformaci, odkud jsou postupně konzumovány a po jedné jsou odesílány do ITSM nástroje příjemce. Takovéto uspořádání umožňuje ponechat transformované zprávy ve frontě v případě, že nastane výpadek na straně příjemce a jeho ITSM nástroj je nedostupný.

Implementace Kafka consumer

V předešlých částech práce bylo zmíněno, že implementace aplikace je realizována pomocí frameworku LoopBack 4. Tento framework ale zatím nepodporuje blok Server, popsaný v části 4.2.3, využívající Kafka consumer. Z toho důvodu byl zhotoven consumer jako služba, která volá příslušný koncový bod. Pro implementaci bylo využito knihovny *Kafkajs* z npm registry. Instalace je prováděna v kořenovém adresáři komponenty pomocí příkazu *npm install kafkajs*. Následující kód ukazuje implementaci Kafka consumer v jazyce TypeScript:

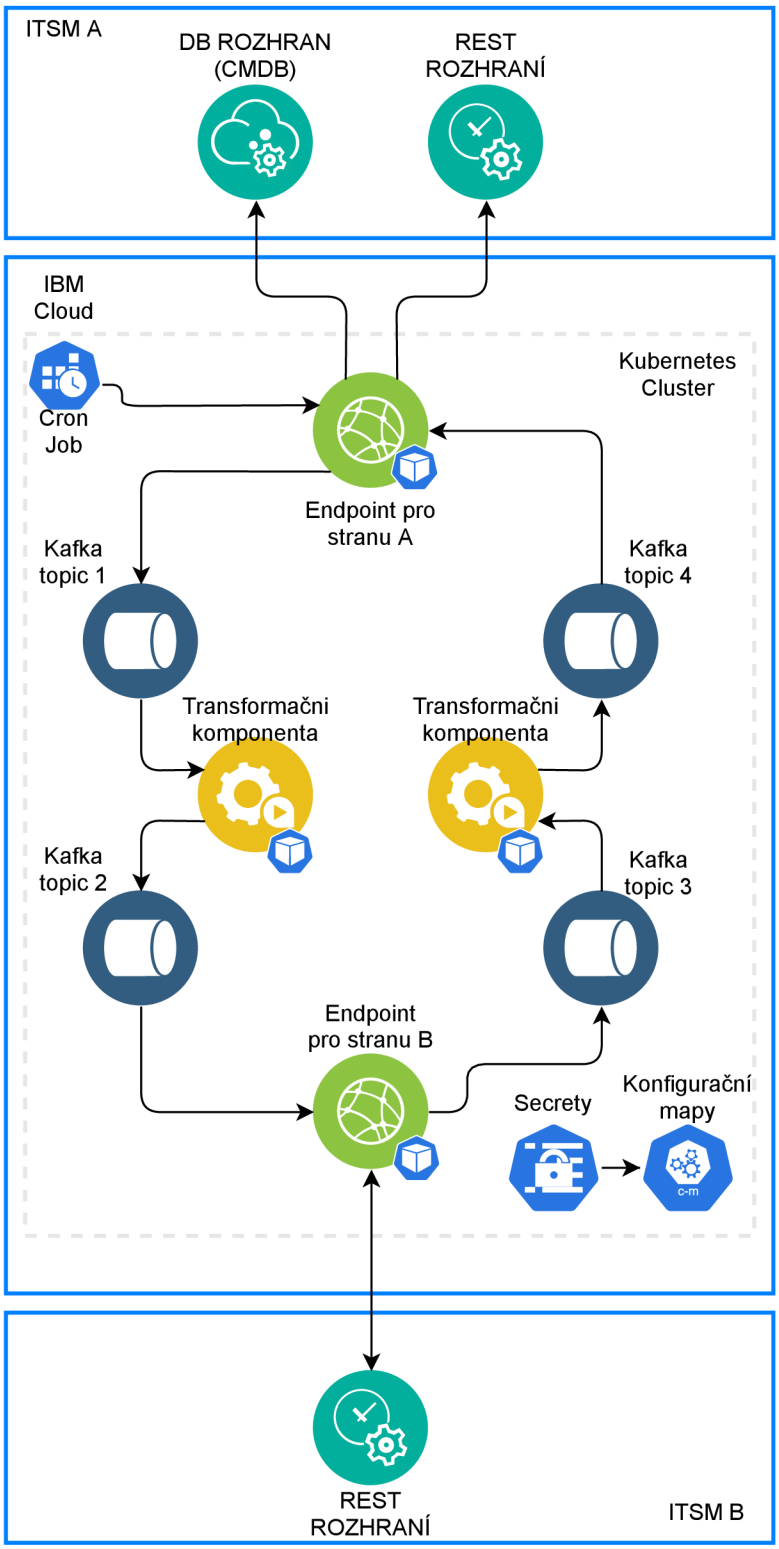
```
1    import {Kafka, Consumer, EachMessagePayload} from 'kafkajs';
2
3    export class KafkaJsConsumer {
4    private consumer: Consumer;
5    constructor(private config: ConsumerConfig,) {
6        const kafka = new Kafka(config);
```

```
7           this.consumer = kafka.consumer({ groupId: config.GROUPID });
8       }
9
10      public async consume(callRest: function, topic: string) {
11          if (this.consumer !== null) {
12              await this.consumer.connect();
13              await this.consumer.subscribe({ topic: topic});
14              await this.consumer.run({
15                  eachMessage: async (message: EachMessagePayload) => {
16                  callRest();
17                  })
18              });
19          }
20      }
```

Implementace Kafka producer

Kafka producer je zhotoven také jako služba. Je ale jednodušší, nemusí volat žádný další koncový bod. Implementace služby Kafka producer může vypadat následovně:

```
1  import {Kafka, Producer, RecordMetadata} from 'kafkajs';
2
3  export class KafkaJsProducer {
4    private producer: Producer;
5    constructor(private config: ProducerConfig) {
6      const kafka = new Kafka(config);
7      this.producer = kafka.producer();
8    }
9
10   public async send(message: object, topic: string) {
11     const messageInString = JSON.stringify(message);
12     await this.producer.connect();
13     const responseOfMessage = await this.producer.send({
14       topic: topic,
15       messages: [{ value: messageInString }],
16     });
17     return responseOfMessage;
18   }
19 }
```



Obr. 4.5: Příklad použití celé integrační aplikace dvou ITSM nástrojů

4.4 Návrh integrační aplikace

Celý integrační systém může být jednosměrný i obousměrný. Výhodou plně konfigurovatelné aplikace je její variabilita. Aplikace může přijímat zprávy z několika vstupů, ale i přes její rozšiřitelnost se stále jedná o Point-to-Point integraci, takže komunikuje vždy strana jednoho zákazníka se stranou IBM a naopak. Teoreticky by bylo možné nakonfigurovat vstup pro více zákazníku, to ale nebylo požadavkem zadavatele.

Aby bylo možné aplikaci volitelně rozšiřovat, je sestavena z klíčových komponent, neboli z mikroslužeb. Klíčové komponenty jsou:

- **Endpoint component** – mikroslužba, která se stará o vstupně výstupní operace. Dále se v této komponentě nachází obchodní logika, upravená na míru požadavkům konkrétních zákazníků.
- **Transformation component** – mikroslužba, která se stará o veškeré požadované transformace. Pomocí konfiguračního souboru lze nastavit transformaci přímou, přidání fixních hodnot, mapování hodnot z databáze, nebo vyčítání hodnoty z komunikujícího ITSM nástroje.

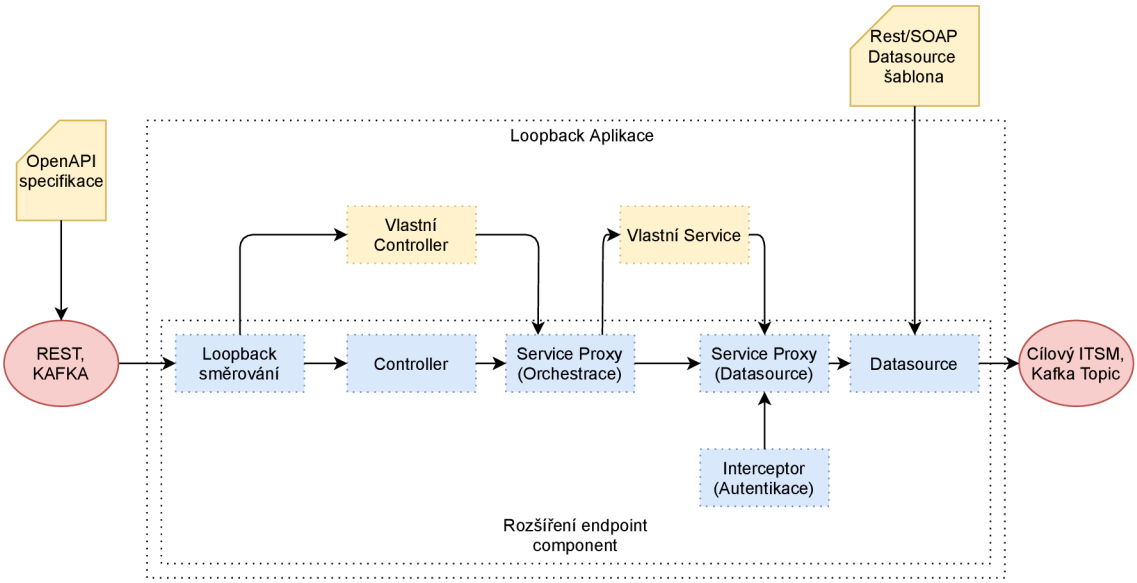
Na 4.5 je zobrazena integrační aplikace mezi dvěma ITSM nástroji. Zobrazená aplikace synchronizuje konfigurační databázi ITSM nástroje jedná strany s nástrojem strany druhé. Integrační aplikace se dotazuje v předem nastaveném intervalu databáze na provedené změny. Po vyčtení změn jsou odeslány všechny záznamy do kafka topic. Transformační komponenta odebírá z fronty jednotlivé záznamy, provede transformační změny podle přiloženého konfiguračního souboru a odesílá do fronty náležící pro transformované záznamy. Odtud jsou odebírány další Endpoint komponentou, která zajišťuje uložení změn v ITSM nástroji druhé strany. V opačném směru ITSM nástroj strany B odesílá záznamy na Endpoint komponentu a není zde potřeba v intervalech číst změny z databáze.

Jak je z obrázku znázorněno, celá aplikace je nasazena v Kubernetes clusteru. Kubernetes cluster je součástí IBM Cloud Kubernetes Service, která obsahuje mnoho nástrojů pro automatizaci, izolaci správu a monitoring kontejnerových aplikací.

4.4.1 Endpoint component

Jedná se o loopback aplikaci, která je rozšířena Loopback komponentou s názvem Endpoint Component. Vstup do komponenty je definovaný podle OpenAPI specifikace. V té je přesně nadefinováno název koncového bodu a také jaká metoda v jakém controlleru má být zavolána. Mohou být využity controllery nadefinované přímo v rozšiřující komponentě, nebo je možno vložit vlastní controller, který provádí proprietární logiku dle požadavků zákazníka. Controller může využívat vlastní

Service například pro volání služeb třetí strany, nebo provádění nestandardní logiky. Nakonec může příchozí zprávu odeslat do Event Stream fronty, nebo do cílového ITSM systému. Pro komunikaci s externímy systémy je implementován interceptor zajišťující oAuth2 autentikaci.



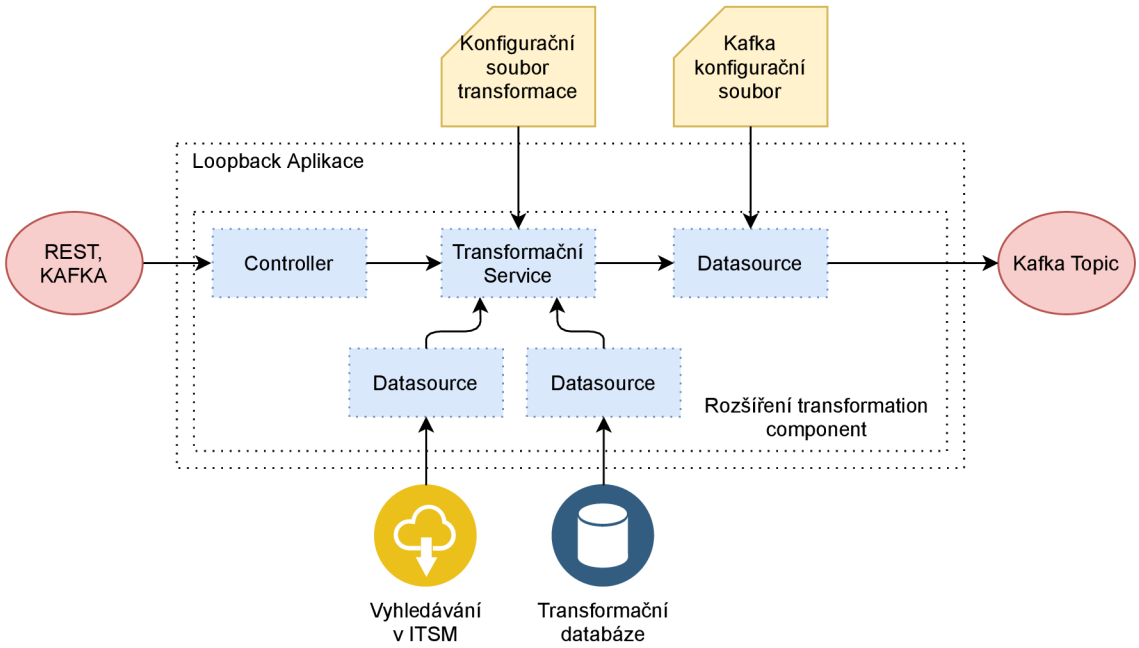
Obr. 4.6: Schéma mikroslužby Endpoint Component.

4.4.2 Transformation component

Transformační komponenta je velice jednoduchou mikroslužbou, která má za úkol pouze vykonání transformací nakonfigurovaných v přikládaném konfiguračním souboru. Konfigurační soubor je popsán v části 4.4.3. Transformační komponenta rozpoznává několik typů transformace. Každý typ má svoji vlastní funkcionalitu. Typy transformace jsou.

- **DIRECT** – Transformace měnící název klíče, pod kterým je uložena nějaká hodnota. V konfiguraci se udává název klíče ve vstupní zprávě a název klíče pro výstupní zprávu.
- **FIXED** – Transformace přidávající pevnou kompinaci klíče a hodnoty do odchazí zprávy.
- **LOOKUP** – Transformace provádějící volání do jiného systému, pro získání potrebných dat.
- **MAPPING** – Transformace volající transformační databázi pro získání transformačních dat.

V Endpoint Component, popsané v 4.4.1, je příchozí zpráva obohacena o manipulační vlastnosti. Jedna z vlastností důležitých pro transformaci je *scope*, podle které je určováno, která transformační operace má být prováděna pro danou zprávu.



Obr. 4.7: Schéma mikroslužby Transformation Component.

4.4.3 Konfigurační soubory

Znovupoužitelnost komponent pro různé potřeby a požadavky zákazníků zajišťují přikládané konfigurační soubory. Konfigurační soubory jsou psány v *YAML* (formát pro serializaci dat textových souborů) a přidávají se ke každé komponentě zvlášť. Asi nejdůležitější konfigurační soubory pro endpoint komponenty jsou pro nastavení serverů a popis API ITSM nástroje, s kterým daná komponenta komunikuje. Příklad konfigurace pro komunikaci komponenty s REST API:

```
        name: example-rest
        connector: rest
        crud: false
        debug: true
        options:
            strictSSL: true
        baseURL: '{ref:secrets.customer.baseUrl}'
        operations:
            - template:
                method: 'GET'
                options:
                headers:
                    Authorization: '{auth}'
                    accept: 'application/json'
                body: '{body}'
                url: '/example'
            functions:
                getExample: [body, auth]
            - template:
            method: 'POST'
            options:
                headers:
                Authorization: '{auth}'
                accept: 'application/json'
                body: '{body}'
                url: '/example/{id}'
            functions:
                sendExample: [id, body, auth]
```

4.4.4 Testování aplikace

S narůstající velikostí aplikace narůstá riziko chybovosti. Je možno tvrdit, že aplikace není kompletně dokončena, dokud není řádně otestována. Za test může být považován obyčejný výpis do konzole, aby bylo potvrzeno, že aplikace funguje podle plánovaného návrhu. Ruční testování je zdlouhavé a náchylné na chyby provedené člověkem. Mnohem přínosnější je automatické testování aplikace. Automatické testování zahrnuje psaní logiky k otestování kódu. Není třeba procházet jednotlivé funkce nebo třídy. Existuje několik fází testování aplikace. Fáze aplikace jsou.

- **Jednotkové testy** – jednotkové testy, nazývané jako unit testy, se používají k otestování funkčnosti konkrétních částí kódu. Testovány mohou být celé třídy, nebo jejich metody a funkce.
- **Integrační testy** – integrační testy se využívají k detekci chyb v rozhraních a mezi integrovanými komponentami. Tato fáze testování se využívá tam, kde k integraci dochází. Aplikace, která sama osobě pracuje dobře, ale nedokáže se domluvit s ostatními systémy, které pro svoje použití potřebuje, je pochopitelně těžko použitelná. Integrace přitom musí být správně navržena už při vytváření analýzy, podle které je aplikace vyvíjena.
- **Akceptační testy** – většinou se jedná o finální fázi testování. Akceptační testy by měli být sestavovány na základě požadavků klienta a využívají se k ověření ním určených požadavků na aplikaci.

K verifikaci funkčnosti rozšíření Endpoint Component a Transformation component jsou využity převážně jednotkové testy pro zajištění a ověření požadované obchodní logiky jednotlivých komponent. Pro ověření funkčnosti spojení s Apache Kafka a spojení s databází Cloudant jsou definované integrační testy. Pokrytí komponent testy není stoprocentní, ale je na dobré úrovni. Úroveň pokrytí pro jednotlivé třídy v Endpoint Component je na obrázku 4.8.

4.5 Kubernetes

Kubernetes je přenosná, rozšiřitelná open-source platforma pro správu kontejnerových úloh a služeb, která usnadňuje deklarativní konfiguraci i automatizaci. Kontejnery jsou podobné virtuálním počítačům, ale na rozdíl od virtuálních počítačů, sdílí jeden operační systém. Proto jsou kontejnery považovány za lehké a mají menší nároky na hardware. Podobně jako virtuální počítač má kontejner svůj vlastní souborový systém, podíl CPU, paměti, procesního prostoru a další. Protože jsou odděleny od základní infrastruktury, jsou přenosné přes různé typy cloudů a distribuce OS. V produkčním prostředí je důležitá správa kontejnerů. Například, pokud kontejner selže, je třeba ho opětovně spustit nebo spustit jiný kontejner. A právě k tomu je využíván Kubernetes, o spuštění nového kontejneru se postará automaticky. Zvládne i mnohem více věcí, jako je monitoring, škálování a mnoho dalšího [33].

Vždy, když je nasazena aplikace, je nasazena do clusteru. Kubernetes Cluster se skládá ze sady pracovních strojů, které nazýváme uzel (Node). Uzly spouštějí kontejnerové aplikace. Každý cluster má alespoň jeden pracovní uzel. Uzly jsou hostiteli základních jednotek Kubernetes známé pod názvem Pod. Pody jsou komponenty pracovní zátěže. V naší aplikaci si pod pojmem Pod můžeme představit jednu mikroslužbu (například Endpoint componenta). Uzel si zase můžeme představit, jako celou jednu aplikaci. V praxi se většinou vytváří repliky uzlu. Jestliže v nějakém uzlu

File	% Stmts	% Branch	% Funcs	% Lines
All files	89.82	81.62	87.21	89.57
src	95.19	66.67	100	95.1
component.ts	95	60	100	94.9
keys.ts	100	100	100	100
src/authentication/strategy	94.12	100	66.67	93.33
appId.ts	94.12	100	66.67	93.33
src/controllers/external	72.22	60	62.5	73.58
entrypoint.controller.ts	100	100	100	100
external-test.controller.ts	69.39	60	57.14	70.83
index.ts	100	100	100	100
src/controllers/internal	83.87	100	83.33	82.76
entrypoint.controller.ts	66.67	100	0	66.67
index.ts	100	100	100	100
internal-test.controller.ts	100	100	100	100
polling.controller.ts	63.64	100	100	63.64
src/datasources	88.06	85.59	83.33	88.15
attachment-downloader-interface.datasource.ts	100	100	100	100
axios.datasource.ts	88.37	79.1	75	88.24
axios.template.ts	96.18	90.78	100	96.73
cloud-object-storage.datasource.ts	65.08	50	73.33	64.52
index.ts	100	100	100	100
kafka-producer.datasource.ts	87.5	83.33	100	87.5
polling.datasource.ts	100	100	100	100
snow-rest.datasource.ts	100	100	100	100
src/errors	100	100	100	100
inbound-error.ts	100	100	100	100
index.ts	100	100	100	100
outbound-error.ts	100	100	100	100
src/interceptors	84.44	72	93.75	83.33
auth.interceptor.ts	67.57	57.14	87.5	65.71
ignore-scope.interceptor.ts	95	80	100	94.44
index.ts	100	100	100	100
message-enhancer.interceptor.ts	96.67	75	100	96.43
src/middlewares	98.88	93.55	100	98.8
external-endpoint-object-router.middleware.ts	100	91.67	100	100
index.ts	100	100	100	100
internal-endpoint-object-router.middleware.ts	97.06	95.45	100	96.88
route-restriction.middleware.ts	100	100	100	100
uniqid.middleware.ts	100	92.31	100	100

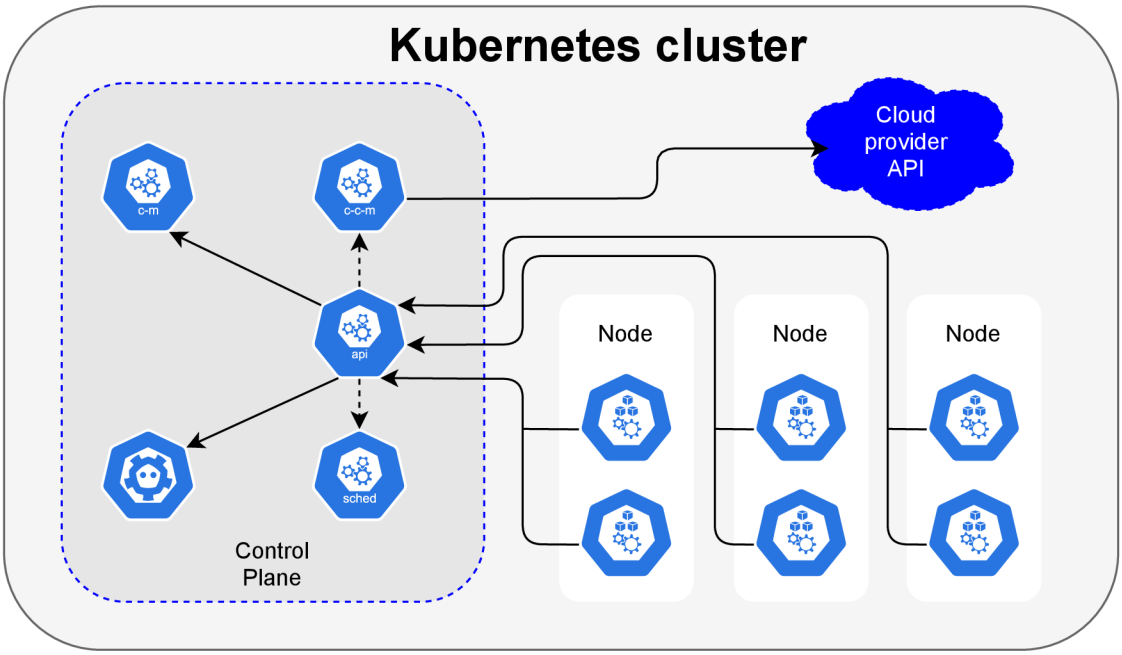
Obr. 4.8: Pokrytí kódu testy v Endpoint Component.

nastane chyba, je vytvořen uzel nový. Ostatní repliky vykrývají dobu nečinnosti při vytváření nového uzlu. Tyto procesy jsou řízeny komponentou Control Plane. Na 4.9 je diagram Kubernetes Clusteru se všemi komponentami [33].

4.6 Kontinuální integrace a nasazení

V závěrečné části práce je popsán princip kontinuální integrace a nasazení, které jsou využívány. Kontinuální integraci (CI) si můžeme představit jako sadu nástrojů a služeb, které slouží k vyhledání problematických, nebo chybných míst ve zdrojovém kódu aplikace. Typicky se v rámci CI spouští větší množství analytických nástrojů, v diplomové práci jsou v rámci CI spouštěny jednotkové testy.

V práci je využíváno platformy Travis CI. Travis CI je snadno použitelný s verzovacím systémem GitHub. Pro používání systému Travis CI je zapotřebí registrace



Obr. 4.9: Kubernetes cluster diagram.

na stránce *https://travis-ci.org* a následně nakonfigurovat propojení s GitHub repozitářem [33]. V rámci IBM je využíváno enterprise verze, ale je možno využívat i bezplatnou verzi s omezenou funkcionalitou. Pro spuštění úloh v Travis CI musí být v adresáři se zdrojovým kódem umístěn konfigurační soubor s názvem **.travis.yml**.

```
language: node_js
node_js:
  - "14"

sudo: required

services:
  - docker

git:
  submodules: false

jobs:
  include:
    - stage: "Tests"
      name: "Environment variables check"
      script: bash ./travis/test.sh
```

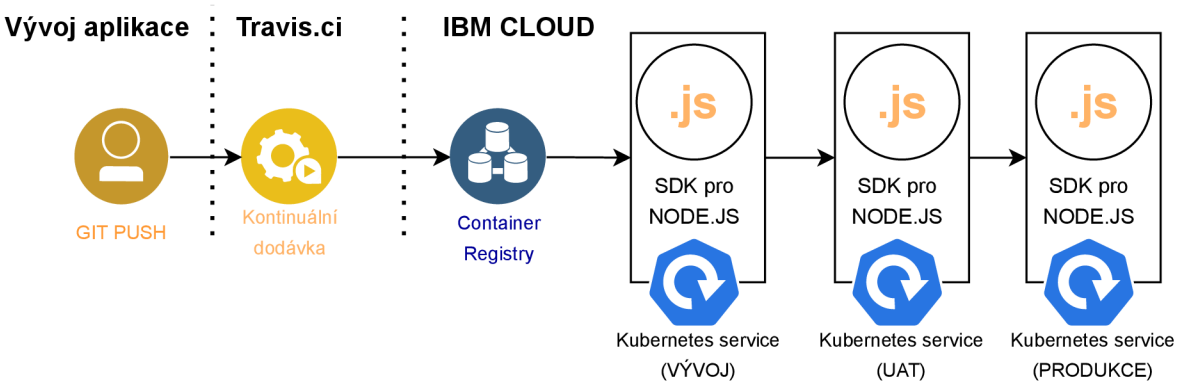
```
18
19      - stage: "DeployToCr"
20        script: bash ./travis/deploy_dev.sh
21
22  stages:
23    - name: Tests
24      if: type = pull_request AND branch = master
25    - name: Deploy_to_DEV_cr_namespace
26      if: branch = master AND tag IS present
```

Takto vytvořený konfigurační soubor zajišťuje, že po každé určité změně (nastavení pod klíčovým slovem *stages*) se provede odpovídající úloha. Úlohy jsou deklarovány pod klíčovým slovem *jobs* a jsou prováděny postupně. Pokud nějaká úloha skončí chybou, další úlohy už se nespustí. První úloha se jmenuje **Tests** a spouští testy zdrojového kódu. Jelikož je využíváno balíčkovacího systému npm, nastavení spouštěných testů je definováno v *Package.json* v kořenovém adresáři zdrojového kódu. Déle můžeme v konfiguračním souboru vidět, že testy jsou spouštěny vždy v případě, když je vytvořen „Pull Request“ a kořenová větev v systému GitHub se jmenuje *master*.

Druhá úloha s názvem **DeployToCr** má na starosti nahrát sestavený obraz kontejneru do *IBM CLOUD Container Registry*. Jedná se o uložiště obrazů, které jsou následně využívány službou *IBM CLOUD Kubernetes service*. Druhá úloha je spuštěna v případě vytvoření nové verze v systému GitHub na větvi se jménem **master**.

Výše popsaný konfigurační soubor je pouze ukázkový příklad, který zahrnuje klíčové části konfigurace pro Travis CI. Celý CI/CD proces je znázorněn na 4.10. Uložené obrazy aplikace mohou být postupně nasazeny do tří jmenných prostorů. Jmenné prostory jsou:

- **Vývojové prostředí**.
- **Prostředí pro uživatelské testování (UAT)**.
- **Produkční prostředí**.



Obr. 4.10: Grafické znázornění kontinuální integrace a dodávky

Závěr

Cílem diplomové práce bylo navrhnout vhodnou softwarovou architekturu včetně její implementace s ohledem na možnosti integrace ITSM řešení na základě široké škály zákaznických požadavků.

V první části práce byl popsán proces řízení IT služeb a praxí ověřené postupy z knihovny ITIL 4. Dále byly představeny příklady ITSM nástrojů, které výrazně zjednodušují procesy pro řízení IT služeb. V dnešní době mají organizace možnost výběru z mnoha již hotových nástrojů pro řízení IT služeb. Pro dobré poskytování IT služeb je nezbytná komunikace mezi více nástroji, kterou zajišťují integrace.

V druhé části práce bylo popsáno více možných řešení integrací mezi aplikacemi. K tomu bylo vhodné zmínit existující návrhové vzory pro řešení integrací. Byly popsány návrhové vzory Point-to-point, Hub-and-Spoke, Enterprise Messagge Bus a Enterprise Service Bus.

V rámci diplomové práce byla navrhnuta a implementována softwarová architektura umožňující integraci ITSM nástrojů. Návrh aplikace vyžadoval nastudování architektonických stylů a možných technologií. Během architektonických rozhodnutí byla zohledněna znovupoužitelnost systému a snadná konfigurovatelnost a rozšiřitelnost systému. Z důvodu nerovnoměrného zatížení systému bylo počítáno s potřebou škálování. Po zvážení kladů a záporů byl navržen systém založený na architektuře mikroslužeb. V rámci návrhu řešení bylo vybíráno z možných technologií, kde nakonec byly vybrány technologie Node.js a framework Loopback verze 4. V rámci diplomové práce byly také řešeny typy komunikace mezi jednotlivými mikroslužbami. Výsledná aplikace měla řešit nezávislost integrovaných ITSM nástrojů a řešit případné výpadky jednoho z nástrojů. Z toho důvodu bylo rozhodnuto o využití asynchronní komunikace. K implementaci asynchronní komunikace bylo využito IBM Event Streams z katalogu služeb IBM Cloud. IBM Event Streams je velmi výkonná platforma pro streamování událostí založena na Apache Kafka.

Zadavatel žádal nasazení aplikace na IBM Cloud. V diplomové práci bylo popsáno, co jsou cloudové systémy a fungování IBM Cloud. Pro IBM Cloud byly zmíněny pouze služby použité ve výsledném řešení.

Pro nasazení aplikace do IBM Cloud byla využita služba TravisCI. V rámci IBM je tato služba využívána ve verzi enterprise. V závěrečné části práce je popsán celý proces kontinuální integrace a nasazení. Celý systém byl nahrán na IBM GitHub a díky službě TravisCI je umožněno spouštět implementované testy a nasazení do vývojového, testovacího a produkčního prostředí.

V současné době je řešení diplomové práce využíváno pro integraci ITSM nástrojů ve firmě IBM a tím je současně ověřena funkčnost celého systému.

Literatura

[1] Rafael Piñeros and L. L. Gómez. How can information and communication technologies (ict) improve decisions of renewal of products and services and quest and selection of new suppliers. 2017.

[2] AXELOS. *ITIL Foundation ITIL 4 Edition*. TSO (The Stationery Office), Swindon, GBR, 2019.

[3] Stuart D. Galup and Ronald Dattero. A five-step method to tune your itsm processes. *Information Systems Management*, 27(2):156–167, 2010. doi:10.1080/10580531003685220.

[4] Stephen Mann. What is itsm? it service management explained. [online], 11 2017. URL: <https://itsm.tools/what-is-itsm/>.

[5] Ernest Brewster, John Sansbury, and Aidan Lawes. *IT Service Management - A Guide for ITIL V3 Foundation Exam Candidates*. BCS Learning Development Ltd., Swindon, GBR, 2010.

[6] Dr.MaryAnne Winniford, Sue Conger, and Lisa Erickson-Harris. Confusion in the ranks: It service management practice and terminology. *Information Systems Management*, 26(2):153–163, 2009.

[7] Elena Orta and Mercedes Ruiz. Met4itil: A process management and simulation-based method for implementing itil. *omputer Standards Interfaces*, 61(2):1–19, 2019. doi:https://doi.org/10.1016/j.csi.2018.01.006.

[8] David Cannon. *ITIL Service Strategy (ITIL v3 Service Lifecycle)*. The Stationery Office, London, 2011.

[9] Cabinet Office. *ITIL Continual Service Improvement 2011 Edition*. The Stationery Office, London, 2011.

[10] ServiceNow. Products and solutions. [online], 10 2015. URL: <http://www.servicenow.com/products/productsby-category.html>.

[11] ServiceNow. Service management. [online], 10 2015. URL: <http://www.servicenow.com/solutions/servicemanagement.html>.

[12] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging, Solutions*. Addison-Wesley Longman Publishing Co., Inc., USA, 2003.

[13] O. Zimmermann, C. Pautasso, G. Hohpe, and B. Woolf. A decade of enterprise integration patterns: A conversation with the authors. *IEEE Software*, 33(1):13–19, 2016. doi:10.1109/MS.2016.11.

[14] Binildas C. A. *Service Oriented Java Business Integration*. Packt Publishing, Birmingham, England, 2008.

[15] A. J. Coulson and R. G. Vaughan. Sp 800-145. the nist definition of cloud computing. Technical report, National Institute of Standards Technology, Gaithersburg, MD, USA, June 2011.

[16] Dan Marinescu. *Cloud Computing 2nd Edition*. Createspace Independent Publishing Platform, North Charleston, SC, 2017.

[17] Toby Velte, Anthony Velte, and Robert C Elsenpeter. *Cloud computing: A practical approach*. McGraw-Hill Education, USA, 2009.

[18] Vic (J.R.) Winkler. Chapter 2 - cloud computing architecture. In *Securing the Cloud*, pages 29–53, Boston, 2011. Syngress.

[19] Tim Mather, Subra Kumaraswamy, and Shahed Laitf. *Cloud security and privacy: An enterprise perspective on risks and compliance*. O'Reilly Media, Sebastopol, CA, 2009.

[20] X. Tan and B. Ai. The issues of cloud computing security in high-speed railway. In *Proceedings of 2011 International Conference on Electronic Mechanical Engineering and Information Technology*, pages 4358–4363, Harbin, China, 2011. IEEE.

[21] Alan Sill. The design and architecture of microservices. *IEEE Cloud Computing*, 3(5):76–80, 2016. doi:10.1109/MCC.2016.111.

[22] Manuel Mazzara and Bertrand Meyer. *Present and ulterior software engineering*. Springer International Publishing, Basel, Switzerland, 2017.

[23] Kristiāns Kronis and Marina Uhanova. Performance comparison of java ee and asp.net core technologies for web api development. *Applied Computer Systems*, 23(1):37–44, 2018. doi:10.2478/acss-2018-0005.

[24] Stefan Tilkov and Steve Vinoski. Node.js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83, 2010. doi:10.1109/MIC.2010.145.

[25] L. P. Chitra and R. Satapathy. 2017 international conference on algorithms, methodology, models and applications in emerging technologies (icammaet). In *Performance comparison and evaluation of Node.js and traditional web server (IIS)*, pages 1–4. IEEE, 2017.

[26] Azat Mardan. *Practical node.Js: Building real-world scalable web apps.* APRESS, New York, NY, 2014.

[27] Dan Maharry. *TypeScript (revealed).* APRESS, New York, NY, 2013.

[28] IBM / StrongLoop. LoopBack 4. [online], 4 2021. URL: <https://loopback.io/doc/en/lb4/apidocs.index.html>.

[29] Mark Clow. *Angular 5 projects: Learn to build single page web applications.* APRESS, New York, NY, 2018.

[30] Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonça, James Lewis, and Stefan Tilkov. Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3):24–35, 2018. doi:10.1109/MS.2018.2141039.

[31] Meenakshi Jhala and Rahul Menon. Examining the impact of an asynchronous communication platform versus existing communication methods: an observational study. *BMJ Innovations*, 7(1):68–74, 2021. doi:10.1136/bmjinnov-2019-000409.

[32] Bhole Rahul Hiraman, Chapte Viresh M., and Karve Abhijeet C. A study of apache kafka in big data stream processing. In *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*, pages 1–3, Pune, India, 2018. IEEE.

[33] The Kubernetes Authors. What is kubernetes? [online], Feb 2021. URL: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>.