TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Sběr a zpracování dat v prostředí internetu věcí

Bakalářská práce

Studijní program: B0714A270001 Mechatronika
Autor práce: **Jan Veverka**
Vedoucí práce: Ing. Jan Kraus, Ph.D.
Ústav mechatroniky a technické informatiky

Liberec 2022

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Sběr a zpracování dat v prostředí internetu věcí

Jméno a příjmení: **Jan Veverka**
Osobní číslo: M19000109
Studijní program: B0714A270001 Mechatronika
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2021/2022**

Zásady pro vypracování:

1. Seznamte se s komunikačními protokoly, formáty dat a hromadnými datovými sklady obvykle využívanými v prostředí internetu věcí (IoT) pro sběr a archivaci větších objemů procesních dat.
2. Vyberte vhodné varianty architektury pro takový systém, implementujte je do fáze funkčních ukázkových řešení a na vybraných detailech demonstrujte vhodnost či nevhodnost dané varianty.
3. Pro alespoň jednu alternativu implementujte i jednoduché vizualizační a analytické rozhraní v oblasti sběru a vyhodnocování dat měření energetické účinnosti budov.
4. V závěru stručně a přehledně shrňte dosažené výsledky a diskutujte možnosti dalšího využití či rozvoje vytvořené aplikace.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30–40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] MATHIAS, Selvine G.; SCHMIED, Sebastian; GROSSMANN, Daniel. An investigation on database connections in OPC UA applications. *Procedia Computer Science*, 2020, 170: 602-609.

[2] MISHRA, Biswajeeban; KERTESZ, Attila. The use of MQTT in M2M and IoT systems: A survey. *IEEE Access*, 2020, 8: 201071-201086.

[3] BAPTISTA, Gabriel; ABBRUZZESE, Francesco. *Software Architecture with C# 9 and. NET 5: Architecting software solutions using microservices, DevOps, and design patterns for Azure*. Packt Publishing Ltd, 2020.

[4] MARJANI, Mohsen, et al. Big IoT data analytics: architecture, opportunities, and open research challenges. *ieee access*, 2017, 5: 5247-5261.

[5] SOUSA, Tiago Boldt, et al. Engineering Software for the Cloud: Messaging Systems and Logging. In: *Proceedings of the 22Nd European Conference on Pattern Languages of Programs*. 2017. p. 1-14.

Vedoucí práce:	Ing. Jan Kraus, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	12. října 2021
Předpokládaný termín odevzdání:	16. května 2022

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Černohorský, Ph.D.
vedoucí ústavu

V Liberci dne 12. října 2021

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

13. května 2022 Jan Veverka

Sběr a zpracování dat v prostředí internetu věcí

Abstrakt

Tato práce se zabývá sběrem a zpracováním dat v prostředí internetu věcí. Konkrétněji se věnuje komunikačním protokolům, formátům pro větší objemy dat a používaným technologiím. Tyto poznatky ukazuje na realizovaných funkčních řešeních, která jsou použitelná pro vizualizaci takových dat.

Klíčová slova: IoT, internet věcí, sběr dat, zpracování dat, vizualizace dat

Abstract

This thesis deals with data collection and processing in the Internet of Things environment. More specifically, its focus is on communication protocols, formats for rather large volumes of data and technologies applied. These findings are demonstrated in implemented functional solutions utilisable for such data visualisation.

Keywords: IoT, Internet of Things, data collection, data processing, data visualisation

Poděkování

Rád bych poděkoval všem, kteří přispěli, přímo či nepřímo, ke vzniku této práce. Jmenovitě upřímně děkuji vedoucímu práce, bratrovi, rodině a kamarádům.

Obsah

Seznam obrázků 9
Seznam zkratek 10

1 Úvod **11**

2 Rešerše **12**
2.1 Internet věcí 12
2.2 Protokoly v IoT 12
2.2.1 MQTT protokol 14
2.2.2 AMQP - Advanced Message Queuing Protocol 16
2.2.3 CoAP 17
2.2.4 SigFox 17
2.2.5 LoRa 18
2.2.6 NB-IoT 19
2.2.7 OPC Unified Architecture 20
2.3 Databáze 21
2.3.1 Relační databáze 21
2.3.2 Nerelační databáze 21
2.3.3 Time series databáze 22
2.3.4 InfluxDB 22
2.3.5 TimescaleDB 23
2.3.6 Prometheus 23
2.3.7 Agregace dat 23
2.4 Senzory 25
2.5 Deployment 27

3 Implementace - Aplikace pro sběr a zpracování teplotních dat **29**
3.1 PostgreSQL implementace 29

3.1.1 Generování dat 29
3.1.2 Logger – záznam chyb při běhu programu 33
3.1.3 Dataloader 34
3.1.4 Grafana 36
3.2 Implementace v prostředí AWS 38
3.3 Implementace MongoDB 39
3.4 Zpracování energetických dat budovy 42

4 Porovnání výsledků **43**

5 Závěr **44**

6 Přílohy **46**

Použitá literatura **52**

Seznam obrázků

2.1 Grafana - heat mapy 24

3.1 Diagram architektury 30
3.2 MongoDB - tabulky 33
3.3 Grafana - Graf - Praha 36
3.4 Grafana - Graf - Liberec 37
3.5 Grafana - hlavní panel 38
3.6 Jednotlivé instance v AWS 39
3.7 MongoDB - graf 1 41
3.8 MongoDB - graf 2 41
3.9 Grafana - rozložení příkonu 42

6.1 LoRaWAN pokrytí 46
6.2 Grafana - podmínky výstrah 47
6.3 Grafana - napětí - detail 47
6.4 Grafana - napětí 47
6.5 TSV - kopírování z PostgreSQL 48
6.6 YAML soubor se souřadnicemi 48
6.7 Import dat do databáze MongoDB 48
6.8 MongoDB - import dat 48
6.9 Grafana - hodinový histogram 49

Seznam zkratek

TUL	Technická univerzita v Liberci
FM	Fakulta mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci
IoT	internet věcí („Internet of Things“)
AWS	webové služby Amazon („Amazon Web Services“)
MQTT	síťový protokol („MQ Telemetry Transport“)
AMQP	protokol aplikační vrstvy („Advanced Message Queuing Protocol“)
CoAP	protokol pro zařízení s omezenými prostředky („Constrained Application Protocol“)
DDS	služba distribuce dat („Data Distribution Service“)
TCP	protokol řízení přenosu („Transmission Control Protocol“)
BLE	Bluetooth s nízkou spotřebou energie („Bluetooth Low Energy“)
3GPP	dohoda o spolupráci v oblasti mobilních komunikací („The 3rd Generation Partnership Project“)
LTE	dlouhodobý vývoj 3GPP („3GPP Long Term Evolution“)
AES	standard pokročilého šifrování („Advanced Encryption Standard“)
QoS	příznak kvality služeb („Quality of Service“)
TLS	zabezpečení transportní vrstvy („Transport Layer Security“)
SASL	jednoduchá vrstva ověřování a zabezpečení („Simple Authentication and Security Layer“)
UDP	internetový protokol bez záruky doručení („User Datagram Protocol“)
HTTP	internetový protokol pro komunikaci s webovými servery („Hypertext Transfer Protocol“)
UNB	systémy s velmi úzkou šířkou pásma kanálu („Ultra Narrowband“)
GSM	standard pro telekomunikační normy („Groupe Spécial Mobile“)
SQL	strukturovaný dotazovací jazyk („Structured Query Language“)
JSON	JavaScriptový objektový zápis („JavaScript Object Notation“)
XML	rozšiřitelný značkovací jazyk („Extensible Markup Language“)
GCP	sada služeb Google cloud computingu („Google Cloud Platform“)
HVAC	topení, větrání a klimatizace („heating, ventilating, air-conditioning“)
YAML	formát pro serializaci dat („YAML Ain’t Markup Language“)
RSS	rodina XML formátů („Rich Site Summary“)
TSV	tabulátorem oddělené hodnoty („Tab-Separated Values“)
PDCA	systém plánuj-dělej-kontroluj-jednej („Plan-Do-Check-Act“)

1 Úvod

Tato bakalářská práce se věnuje problematice rozmachu zpracovávaných a sbíraných dat v prostředí internetu věcí (Internet of Things, IoT). Pro vhodné zpracování tématu se teoretická část práce věnuje převládajícím komunikačním protokolům, formátům dat a hromadnému skladování dat. Tyto teoretické poznatky jsou dále rozvíjeny pomocí praktických implementací, které navazují na dnešní trendy v oblasti sběru, zpracování a archivace velkého množství dat, a přibližují danou problematiku v textu praktické části této práce. Tyto aplikace jsou vhodné k demonstrování reálných postupů, řešení a problémů v této technické oblasti.

V prostředí internetu věcí je důležitá, ne-li nejdůležitější část výběru architektury v řešeném systému. Tato bakalářská práce tento výběr řeší, usnadňuje a shrnuje ideální řešení pro různé druhy aplikace pomocí jednotlivých implementací. Pro každou variantu je popsána její vhodnost použití, výhody i nevýhody.

Praktické realizace různých architektur aplikace a datových skladů jsou tři, první implementace (popsána v kapitole 3.1) je realizována pomocí vlastní aplikace napsané v programovacím jazyce Python, který je všeobecně vhodný pro práci s daty, spolu s relační databází PostgreSQL pro ukládání generovaných a zpracovávaných dat a vizualizační aplikací Grafana, druhá implementace využila korporátního cloudového prostředí MongoDB Atlas a MongoDB Compass. Třetí implementace se věnuje oblasti sběru a zpracování dat souvisejících s měřením energetické účinnosti budov. Této implementaci se práce věnuje v části 3.4. Dále je naznačena možnost implementace v prostředí AWS (zkratka z anglického „Amazon Web Services“, česky Webové služby Amazon) v kapitole 3.2.

V teoretické části se práce také věnuje protokolům používaným v prostředí internetu věcí, jako je MQTT, AMQP, SigFox, LoRa a další nejčastěji používané. Kapitola zabývající se hromadnými datovými sklady používanými v aplikacích pro IoT je kapitola Databáze (kapitola 2.3), která obsahuje rozdělení na databázové systémy relační a nerelační, a speciálně se zmiňuje o databázích typu time series.

2 Rešerše

2.1 Internet věcí

Pro pochopení cílů a obsahu této práce je potřeba definovat internet věcí jako takový. V článku [1] je internet věcí popsán následovně:

> „Síť fyzických zařízeních, vozidel, domácích spotřebičů a dalších zařízení, která jsou vybavena elektronikou, softwarem, senzory/čidly a hlavně síťovou konektivitou. Ta umožňuje těmto zařízením se navzájem propojit a vyměňovat si data.“

Vzhledem k počtu zmiňovaných zařízení je jasné, že internet věcí a infrastruktura kolem něj je a bude velmi rozvíjející se obor. V dnešní běžné domácnosti najdeme několik chytrých telefonů, chytré ledničky, chytré pračky, prakticky všechny spotřebiče se dají zakoupit doplněné o elektroniku, která z nich dělá chytré. Abychom mohli tyto doplňující vlastnosti efektivně využívat, potřebujeme k tomu určité komunikační protokoly. V dnešní době se komunikační protokoly pro IoT zařízení dělí do dvou skupin: datové a síťové protokoly.

2.2 Protokoly v IoT

Datové protokoly jsou protokoly, které používáme pro přenos dat z koncových zařízení, senzorů. Ve většině případů jsou tato zařízení s nízkými výkony nepřipojená přímo k internetu, ale připojují se k IoT bráně. IoT brána je zařízení sloužící ke sběru a odesílání dat z koncových bodů do cloudu. Nejčastější datové protokoly používané v IoT jsou:

- MQTT (zkratka z anglického „MQ Telemetry Transport“) - velmi jednoduchý, nenáročný protokol, postavený na protokolu řízení přenosu (TCP, zkratka z anglického „Transmission Control Protocol“), publisher-subscriber, centrální bod (broker)

- AMQP (zkratka z anglického „Advanced Message Queuing Protocol“) - podobný MQTT, datově náročnější, a tudíž není úplně ideální pro použití v některých IoT odvětvích, centrální bod (broker)
- CoAP (zkratka z anglického „Constrained Application Protocol“, česky protokol pro zařízení s omezenými prostředky) - podobné HTTP (zkratka z anglického „Hypertext Transfer Protocol“) - velmi snadný překlad, oproti HTTP má funkce navíc (např. multicast) pro internetovou komunikaci
- DDS (zkratka z anglického „Data Distribution Service“, česky služba distribuce dat) - aplikace v reálném čase, standard pro Object Management Group, zabezpečené, velký výkon

Informace o protokolech pochází ze zdrojů [2], [3], [4], [5] a [6].

Pro obor IoT je přenos dat zásadní, a proto je dobré definovat synchronní a asynchronní přenos dat. Přenos dat je synchronní tehdy, když má přijímač i vysílač jeden zdroj hodinového signálu. Jedná se o optimální způsob přenosu dat, ale znamená to, že musí být vysílač i přijímač součástí jednoho zařízení nebo musíme zajistit spolehlivý přenos hodinového signálu. Při použití asynchronního přenosu se vysílač a přijímač synchronizují po přenosu každého slova. Tento způsob přenosu je pomalejší, ale technicky jednodušší, jelikož nemusíme zajišťovat synchronizaci hodinového signálu.

Síťové protokoly jsou využívány při připojování jednotlivých zařízení na komunikační nebo transportní vrstvě. Nejčastější síťové protokoly jsou:

- 2G/3G - dlouhé vzdálenosti, spíše pro malé objemy dat, vysoká cena
- Bluetooth/BLE (zkratka z anglického „Bluetooth Low Energy“, česky Bluetooth s nízkou spotřebou energie) - wearables, propojení s telefonem, BLE = nízká spotřeba při zachování podobného dosahu
- IEEE 802.15.4 (zkratka z anglického „Institute of Electrical and Electronics Engineers“, česky Institut pro elektrotechnické a elektronické inženýrství) - bezdrátová technologie pro krátké vzdálenosti, definuje fyzickou a linkovou ISO/OSI vrstvu
- LoRa - minimální spotřeba při dlouhém dosahu, využívá tzv. bezlicenčních pásem, malý datový tok

- LTE (zkratka z anglického „3GPP Long Term Evolution“, dlouhodobý vývoj 3GPP) Cat 0/1 - velká spotřeba - snaha o zmenšení u Cat 0, velké pokrytí
- WiFi - velmi využívané telefony a PC = velmi dostupné, a tudíž často používané, většinou standard 802.11n - velké přenosy dat i spotřeba energie
- ZigBee - navrženo pro industriální prostředí, velmi populární, nízká spotřeba, možnost šifrování 128 bit AES (zkratka z anglického „Advanced Encryption Standard“, česky standard pokročilého šifrování)
- Z-Wave - zapojení do mesh sítě (způsob vzájemného propojení bezdrátových síťových prvků), časté využití pro automatizaci domácnosti, případně u bezpečnostních prvků

Porovnání těchto protokolů naleznete v tabulce 6.1.

2.2.1 MQTT protokol

Zdroj [7] popisuje protokol MQTT následovně:

> „MQTT je standardní protokol OASIS k zasílání zpráv pro internet věcí. Je navržen jako extrémně lehký transport zpráv typu publish/subscribe, který je ideální pro připojení vzdálených zařízení s malou stopou kódu a minimální šířkou pásma sítě. Protokol MQTT se dnes používá v celé řadě průmyslových odvětví, například v automobilovém průmyslu, výrobě, telekomunikacích, ropném a plynárenském průmyslu atd.“

Ideální IoT systém funguje na principu snímač (senzor) - centrální bod - zařízení. Centrální bod v takovém systému figuruje proto, aby zpracování dat neprobíhalo v zařízeních, které k tomu nejsou určené, jako právě senzory nebo koncová zařízení. Díky tomu nemusí senzor posílat už vyhodnocenou zprávu různým zařízení, ale stačí, když pošle aktuální snímaný údaj do centrálního bodu, který je uzpůsoben pro zpracování a logické rozhodování na základě příchozích dat. Daný centrální bod funguje v protokolu MQTT jako tzv. broker. MQTT je lehký protokol, což je v IoT velká výhoda - ušetří výpočetní výkon a spotřebu baterie koncových zařízení. Díky lehkosti a efektivnosti je tento protokol velmi dobře škálovatelný. Tato vlastnost je velmi potřebná, ne-li zásadní pro použití v IoT kvůli počtu zařízení, která spolu musí komunikovat. Další potřebnou vlastností je spolehlivé odesílání, přenos a doručování

zpráv. Koncová zařízení se mohou vyskytovat v podmínkách, které nejsou ideální pro přenos zpráv. MQTT protokol obsahuje řešení takových problémů v podobě příznaku kvality služeb, tzv. QoS. Tento příznak nám dovoluje nastavit úroveň spolehlivosti doručení zprávy. Nejen kvůli přenosu soukromých dat je v MQTT možnost šifrování TLS (zkratka z anglického „Transport Layer Security“, česky zabezpečení transportní vrstvy) a autentizace klientů.

Protokol MQTT funguje na principu publikace a odběru zpráv z tématu. Máme tři různé role klienta-zařízení, každé má svoji úlohu v posílání dat. První role je publisher (klient), který produkuje zprávu, volí topic (téma) a posílá zprávu do brokera. Druhá role subscriber (klient), který pomocí zprávy subscribe a tématu se přihlašuje do brokera pro odebírání zpráv s daným tématem. Poslední role je broker - centrální bod MQTT architektury, připojují se k němu ostatní klienti, přijímá zprávy od klientů-publisherů, filtruje je podle témat a předává klientům-subscriberům.

Protokol definuje, jak má datová zpráva vypadat a co musí obsahovat. Zpráva je složena ze třech částí - fixní hlavička, povinná ve všech zprávách, nepovinná variabilní hlavička a nepovinná data, tzv. payload.

Fixní hlavičku nalezneme v každé MQTT zprávě, je to jediná povinná část zprávy. Fixní hlavička má velikost 2 byte, rozděluje se prakticky na 3 části. První 4 bity jsou vyhrazené pro typ zprávy, další 4 bity jsou pro příznaky a zbytek - druhý byte je používaný pro zbývající délku zprávy. Fixní hlavička tedy obsahuje následující náležitosti: typ zprávy - v protokolu MQTT je 15 různých druhů zpráv. 1. příznak je příznak DUP - duplikát, který se používá pro opakované odeslání paketu v některých typech zpráv. Další, 2. příznak je QoS, úroveň kvality služeb (0, 1, 2), zabírá 2 bity. Poslední, 3. příznak je příznak RETAIN. Tento příznak u zprávy je užitečný v případě, že se k odběru tématu přihlásí nový klient (subscriber). Ten hned po přihlášení k odběru dostane zprávu od brokera, která má příznak RETAIN. Tím pádem má klient alespoň nějaká data, než se objeví další zpráva pro dané téma.

V některých aplikacích IoT je potřeba, abychom odesílali a přijímali data pouze jednou, protože duplikace dat je nežádoucí. Kvůli tomu existuje příznak QoS, který nám určuje kvalitu služeb. Tento příznak může nabývat tří různých hodnot. QoS 0 - maximálně jednou, což znamená, že je zpráva publisherem odeslána jednou a poté je smazána. Další hodnota příznaku QoS je 1 - minimálně jednou, kdy broker po obdržení zprávy odešle zpět zprávu typu PUBACK, čímž potvrdí publikaci, a tím pádem víme, že broker zprávu obdržel. Nedá se ovšem zabránit duplikacím zpráv, protože pokud klient-publisher neobdrží zprávu PUBACK, tak odešle zprávu znovu

s příznakem DUP = 1. Poslední hodnota QoS je 2 - přesně jednou. Tato varianta je sice náročnější na síťový a výpočetní výkon, ale zaručuje neduplicitní dodání zprávy. Prakticky funguje na následujícím principu: klient-publisher odešle zprávu, broker ji potvrdí pomocí zprávy PUBREC s ID paketu, klient brokerovi potvrdí potvrzení pomocí zprávy PUBREL a broker kopii zprávy smaže po přijmutí zpráby PUBREL a dokončí transakci odesláním zprávy PUBCOMP klientovi.

Variabilní hlavička obsahuje další příznaky, informace o zprávě, které nejsou povinné, ale pro přehlednost je lepší ji ve zprávě mít. Příznaky ve variabilní hlavičce jsou následující: identifikátor paketu, název protokolu, verze protokolu, příznak „connect flags“, uživatelské jméno, heslo, Last Will retain - „poslední vůle“ (příznak používaný pro upozornění ostatních klientů při neočekávaném odpojení klienta, respektive při odpojení klienta bez odeslání zprávy DISCONNECT), Last Will QoS, clean session, session present, connect return code.

Protokol MQTT nemá jasně daná pravidla pro samotná data. Aplikace si samy řeší formát a obsah datových zpráv, velikost dat se dá snadno spočítat pomocí informace o velikosti zprávy obsažené ve fixní hlavičce. Použití protokolu MQTT v IoT nebo machine-to-machine (stroj komunikující se strojem) systémech popisuje zdroj [8], respektive řeší růst výzkumu MQTT v porovnání s jinými protokoly používanými v daných systémech. Dále také zjišťuje, že protokol MQTT je jedním z nejvíce používaných protokolů v oblasti IoT. Zmiňuje ale také nevýhodu, respektive prostor pro zlepšení, v oblasti bezpečnosti tohoto protokolu. MQTT umožňuje v rámci své nenáročné povahy protokolu odesílat data v nezabezpečené podobě. V některých implementacích MQTT brokera je tudíž kladen důraz na zlepšení této nevýhody. Zdroj informací o MQTT je také zdroj [9].

2.2.2 AMQP - Advanced Message Queuing Protocol

Při výběru architektury IoT systému se může zdát jako jeden ze žádaných protokolů protokol AMQP, protože funguje na velmi podobném principu jako výše rozebíraný protokol MQTT. Funguje totiž na modelu publisher - broker - subscriber. Proti protokolu MQTT má určité výhody. Bezpečnost přenosu má lepší AMQP, protože disponuje SASL (zkratka z anglického „Simple Authentication and Security Layer“, česky jednoduchá vrstva ověřování a zabezpečení) vrstvou (jednoduchá autentifikační bezpečnostní vrstva), což je bezpečnostní vrstva navíc oproti jednodušší autentifikaci na transportní vrstvě (TLS) u MQTT, která například nepodporuje delší

uživatelská jména a hesla. Dále mezi výhody patří spolehlivost doručení - oba dva protokoly mají příznak QoS, ale protokol AMQP má výhodu v tom, že umožňuje definovat adresy pro předání zpráv pro případ nečekaného ukončení spojení. MQTT podporuje pouze zprávy typu publish-subscribe, AMQP má v tomto ohledu mnohem více možností, například fronty zpráv, store-and-forward (ulož-a-přepošli) nebo i zmíněné publish-subscribe.

Oproti MQTT je protokol AMQP náročnější na spotřebu dat i energie. Také díky funkcím navíc má protokol AMQP větší jednotlivé zprávy. Teoreticky je implementace protokolu MQTT jednodušší, ale při použití správných open-source knihoven se tento rozdíl smazává. Zdroj informací o protokolech AMQP a MQTT je zdroj [10].

2.2.3 CoAP

Protokol CoAP je další z nejčastěji používaných protokolů. Je založený na jiném modelu než MQTT nebo AMQP, podobá se HTTP. Funguje na principu klient-server, kde klient odesílá požadavky podobně jako u webových stránek a server mu odpovídá žádanou odpovědí. CoAP protokol je založený na asynchronním přenosu dat a byl navržen pro použití v IoT aplikacích. CoAP využívá UDP (zkratka z anglického „User Datagram Protocol“) protokol. Informace o CoAP protokolu pochází ze zdroje [5].

Protože HTTP protokol, ze kterého tento protokol vychází, je paměťově velmi náročný, tak muselo proběhnout několik úprav. Protokol HTML obsahuje textovou hlavičku s velkým množstvím informací. To je zbytečné pro použití v IoT, a tudíž je tato textová hlavička nahrazena hlavičkou bitovou. Přenos dat jako takových probíhá pomocí krátkých zpráv. Mezi nevýhody tohoto protokolu patří nutnost znát IP (zkratka z anglického „Internet Protocol“) adresu IoT modulu.

2.2.4 SigFox

SigFox je bezdrátový komunikační systém fungující na frekvenci 868 MHz (v Evropě), který je energeticky nenáročný a vhodný pro přenos malých zpráv. Díky dosahu několika kilometrů a provozu v Česku na infrastruktuře firmy T-Mobile se jedná o spolehlivou síť s pokrytím údajně až 94 % ČR. Jde o síť vhodnou pro levné odesílání dat ze vzdálenějších senzorů, kde není potřeba odesílat mnoho dat v reálném čase.

O protokolu SigFox se na webu IoT Portál (viz [11]) uvádí:

„K přenosu SigFox komunikace se využívá tzv. UNB (zkratka z anglického „Ultra Narrowband“) pásmo pro vysílání jen krátkého pulsu dat s vysílacím výkonem omezeným na 100 mW a modulací pracující v 200kHz veřejném pásmu. Každá přenášená zpráva v době přenosu zabírá šířku pásma 100 Hz a je přenášena rychlostí 100 nebo 600 bitů/s (v závislosti na regionu). Toto řešení zajišťuje dlouhý dosah a velkou odolnost proti rušení. Vysílaná zpráva využívá tzv. DBPSK modulaci, které stačí pro rychlost přenosu 1 bit/s jen frekvenční pásmo 1 Hz. Tedy při deklarované přenosové rychlosti SigFox komunikace 100bitů/s se využívá již zmíněné šířka frekvenčního pásma 100 Hz. Nízká přenosová rychlost (bitrate) a úzké koncentrované přenosové pásmo DBPSK modulace tedy přináší velmi efektivní využití přenosového spektra, je velmi snadné ho implementovat a přijímač může demodulovat signály velmi blízko hladině šumu.“

2.2.5 LoRa

Pro aplikace, které vyžadují mobilní konektivitu, je vhodné použít již zavedené sítě mobilní konektivity. Pokud nechceme pro takové aplikace využít síť SigFox, tak máme možnost použít síť LoRaWAN, která funguje na základě protokolu LoRa.

LoRa používá k přenosu dat rádiovou komunikaci s velkým dosahem. Díky posílání malých zpráv mají koncová zařízení malou spotřebu, a tudíž velkou výdrž baterie. Posílání zpráv funguje na principu rozprostřeného spektra rádiových vln. Patent na tuto technologii vlastní firma Semtech. LoRa funguje na stejném frekvenčním pásmu jako technologie SigFox, 868 MHz (v Evropě).

Komunikace pomocí protokolu LoRa je velmi dobře zabezpečená, funguje na principu 128 bitových klíčů, kdy jeden klíč má provozovatel služby a druhý klíč má koncové zařízení, respektive zákazník. Zařízení se pomocí rádiových vln připojují k LoRa bráně - vysílací/přijímací stanici, která následně odesílá data na centrální server, který také řídí parametry rádiového vysílání, řeší monitoring stavu sítě a zařízení v ní a zajišťuje správné směrování dat v síti. Všechna čidla použitá v síti LoRaWAN podléhají certifikaci, do sítě jsou tedy připojena pouze certifikovaná čidla, což zvyšuje bezpečnost a spolehlivost celé sítě.

Technologii LoRa v Česku provozuje a rozšiřuje telekomunikační společnost České Radiokomunikace. Ačkoliv není pokrytí tak dobré, jako má síť SigFox, tak je i přesto na velmi dobré úrovni, např. v Praze je pokrytí 92 %, což s 110 vysílacími body stačí pro lokalizaci zařízení na nižší desítky metrů. Síť vznikla v roce 2015,

takže se na kompletním pokrytí ČR postupně pracuje. V roce 2017 bylo pokryto 70 % ČR. Pokrytí ČR v červnu 2020 je vidět na obrázku ze zdroje [12] v příloze 6.1.

Hlavním rozdílem technologií Sigfox a LoRa je jejich přenosová rychlost, SigFox má maximální přenosovou rychlost pouze 100 bit/s, LoRa na rozdíl od toho dosahuje rychlostí od 300 bit/s až do 50 000 bit/s. SigFox má ovšem větší pokrytí ČR, takže pokud existuje požadavek na co nejlepší spolehlivost v různých částech ČR, tak má SigFox výhodu.

2.2.6 NB-IoT

Narrow Band IoT je další z IoT technologií rozšířených v Česku. Mezi její hlavní výhody patří jednoduchá implementace do existujících mobilních sítí, protože není potřeba doinstalovávat či kupovat další hardware, vše funguje pomocí běžného LTE pásma, a tudíž stačí pouze softwarová změna pro použití stávajících vysílačů. Z tohoto důvodu má tedy technologie NB-IoT velmi dobré pokrytí v celé ČR, Vodafone dokonce udává pokrytí 100 %. Další z operátorů, kteří nabízejí NB-IoT služby, je O2 (Narrow Band LTE).

Narrow Band IoT je úzkopásmový protokol navržený pro přenos dat v internetu věcí, mezi jehož největší výhody patří možnost nasazení v již existujících GSM (zkratka z francouzského „Groupe Spécial Mobile“) a LTE pásmech. NB-IoT nabízí tři varianty implementace na straně vysílačů. Varianta „standalone“ obsahuje samostatný vysílací hardwarový prvek využitelný převážně v sítích, kde je provozováno GSM a LTE zároveň. Druhá varianta je varianta in-band, která pro vysílání využívá již existujících vysílačů LTE nebo GSM, výhodou je snadné nasazení bez nutnosti hardwarové instalace, stačí pouze softwarová úprava. Poslední varianta guard-band znamená, že vysílání funguje na principu tzv. ochranného pásma, NB-IoT má v tomto případě vyhrazenou část přenosového pásma již existujícího GSM nebo LTE vysílače.

Mezi výhody použití této technologie patří podpora vysokého počtu zařízení. Jelikož je tento protokol stavěný přímo pro použití v IoT, tak je implementace velkého množství zařízení snadná. Dále je to nízká míra rušení - na rozdíl od předchozích dvou technologií je NB-IoT používáno v licencovaném, a tudíž v méně rušeném pásmu. Koncová zařízení mohou být v případě potřeby velmi úsporná i díky použití úzkopásmové sítě. Neovlivňuje to ale případnou potřebu pro větší výkon zařízení, pokud nám nevadí větší spotřeba energie, tak koncová zařízení snadno zvládnou i ná-

ročnější oboustranné komunikace. Jako každá technologie má i Narrow Band IoT nevýhody: nutnost operátora - kvůli použití licencovaného komunikačního pásma je použití této technologie limitováno na omezený počet subjektů, je nutné se spoléhat na zavedené operátory, kteří mají příslušná licenční pásma pod kontrolou. Mezi další nevýhody patří nutnost použití SIM karet - koncová zařízení musí mít buď fyzickou SIM kartu, nebo musí podporovat eSIM. Cena koncových zařízení může být mírně vyšší, například z důvodu nutnosti použití SIM karty. Zdroj informací o Narrow Band IoT je zdroj [13].

Tabulka 2.1: Porovnání síťových protokolů

Technologie	Frekvence	Datový tok	Dosah	Spotřeba energie	Cena
2G/3G	Mobilní frekvence	10 Mbps	několik km	vysoká	vysoká
Bluetooth/BLE	2,4 GHz	1-24 Mbps / 1-2 Mbps	10-240 m	nízká	nízká
IEEE 802.15.4	<1 GHz, 2,4 GHz	40, 250 kbps	10 m	nízká	nízká
LoRa	<1 GHz (subGHz)	< 50 kbps	2-5 km	nízká	střední
LTE Cat 0/1	Mobilní frekvence	1-10 Mbps	několik km	střední	vysoká
WiFi	<1 GHz, 2,4 GHz, 5 GHz	0,1-54 Mbps	40-100 m (2,4 GHz)	střední	nízká
ZigBee	2,4 GHz	250 kbps	100 m	nízká	střední
Z-Wave	<1 GHz (subGHz)	40 kbps	30 m	nízká	střední

2.2.7 OPC Unified Architecture

OPC Unified Architecture je komunikační protokol vyvinut nadací OPC Foundation, který slouží pro přenost dat mezi jednotlivými stroji v průmyslu. S rostoucím výpočetním výkonem a konektivitou průmyslových vestavných zařízení se stává celá řada aplikací realizovatelná a dostupná. Trendem těchto aplikací internetu věcí je spolupráce rozdílných systémů, modularita a přenositelnost systémů a softwaru, takže lze užitečně propojit značně odlišné oblasti. Tento protokol lze využít v průmyslu jako jednoduchý nástroj pro přenos informací mezi digitálními objekty, jako jsou databáze a sítě. Tyto informace o protokolu OPC Unified Architecture pochází ze zdroje [14], který dále a hlouběji řeší použití tohoto protokolu a prozkoumává dynamiku agregace OPC UA sestávající se z více databázových serverů ze specifického prostředí, jako je výroba. Tento protokol by bylo vhodné implementovat do existujícího průmyslového prostředí pro evaluaci jeho přínosu v průmyslových systémech a architekturách.

2.3 Databáze

Při navrhování IoT architektury je důležitý výběr vhodného typu databáze, protože rychlost a spolehlivost přenosu dat je irelevantní v případě, že data nedokážeme srozumitelně, efektivně a spolehlivě ukládat do databáze. Nejběžnější rozdělení databází je na relační - SQL (zkratka z anglického „Structured Query Language“, česky strukturovaný dotazovací jazyk) a nerelační - NoSQL databáze. Jejich rozdíly jsou popsány s využitím informací ze zdroje [15] a [16] v následujícím porovnání.

2.3.1 Relační databáze

Relační databáze jsou nejpoužívanější databáze, to ovšem neznamená, že pro určité případy není vhodnější použít nerelační databázi. Relační databáze jsou založeny na tzv. relacích, což jsou jednotlivé databázové tabulky, které mohou být vzájemně propojené pomocí konkrétních klíčů. Relace obsahuje atributy a záznamy. Atributy označují jednotlivé sloupce a záznamy odpovídají jednotlivým řádkům s daty.

Atributy mohou plnit funkce tzv. klíčů. Jednotlivé klíče se dělí na kandidátní, primární, alternativní nebo cizí. Kandidátní klíče jednoznačně identifikují jednotlivé záznamy. Kandidátních klíčů může být mezi atributy více. Primární klíč, stejně jako kandidátní klíč, identifikuje konkrétní záznam, ale musí mít ještě další vlastnosti. Vždy musí nabývat nenulové hodnoty a neměl by se měnit. Alternativní klíče jsou klíče kandidátní, které se nestaly klíčem primárním. Cizí klíče jsou nezbytné k propojování relací (tabulek) v relačních databázích. Tabulky se propojují tak, že primární klíče jedné tabulky se shodují právě s cizími klíči jiné tabulky.

Pro operace s relačními databázemi používáme jazyk SQL. Jedná se o převážně dotazovací jazyk, umožňuje vyhledávat, třídit a seskupovat konkrétní žádaná data.

2.3.2 Nerelační databáze

Nerelační databáze se vyznačují hlavně absencí jednotlivých tabulek. Z toho vyplývá nejzásadnější výhoda těchto databází, čímž je možnost ukládání různorodých dat, které nemají pevně definovanou strukturu. Tento přístup je velmi efektivní v prostředí internetu věcí, kde nejsou předem přesně určené druhy zařízení, a tím pádem i datové struktury. Pokud tedy máme například v domě senzory teploty a chceme přidat senzory vlhkosti, nemusíme předělávat nebo přidávat tabulku jako u relační databáze.

Nerelační databáze se dále dělí na několik druhů. Liší se od sebe způsobem, kterým ukládají data. Jedním z nejčastěji používaných druhů je databáze typu klíč-hodnota (key-value). V této databázi se při ukládání hodnoty přidělí unikátní klíč. Místo číselné hodnoty může být ke klíči přiřazený odkaz na jiný pár klíč-hodnota. Tyto databáze jsou specifické pro svou vlastnost ukládání dat do nejednotné struktury. Další často používaný druh nerelační databáze je document-oriented databáze. Tyto databáze ukládají data nejčastěji ve formátu JSON (zkratka z anglického „JavaScript Object Notation“, česky JavaScriptový objektový zápis), binární JSON nebo XML (zkratka z anglického „Extensible Markup Language“, česky rozšiřitelný značkovací jazyk). Výhody jsou stejné jako u databáze klíč-hodnota, můžeme přidat další klíč do formátu dat beze nutnosti změny celé databáze. Mezi nejpoužívanější nerelační databáze patří MongoDB a Elasticsearch.

2.3.3 Time series databáze

Time series jsou takové databáze, které mají posloupnost dat seřazenou podle času. Pomocí takové posloupnosti dat v čase je v mé implementaci 3.1 realizováno ukládání teplotních hodnot. Pro vizualizaci daných dat se používá graf, který graficky znázorňuje průběh dané veličiny v čase. time series databáze většinou sdílejí další vlastnosti - data se přidávají na konec databáze v pravidelných intervalech (v mé implementaci se data přidávají každou minutu). Další ze společných vlastností je absence úprav již vložených hodnot. Time series databáze se dají použít k sledování a vyhodnocení trendu.

2.3.4 InfluxDB

InfluxDB je time series databáze typu open-source vyvíjená přímo k použití v oblastech IoT pro vizualizace, monitoring a analýzu dat. Používá programovací jazyk Go. Existuje také jako služba pro AWS, GCP (zkratka z anglického „Google Cloud Platform“) i Azure. Součástí je také API (zkratka z anglického „Application Programming Interface“, česky rozhraní pro programování aplikací) pro jednoduchou práci s informacemi, tzv. REST API. InfluxDB podporuje Flux, což je samostatný skriptovací a dotazovací jazyk, který zvyšuje produktivitu a opakované použití kódu. Tento jazyk například dokáže vyplnit chybějící data pomocí duplikace předchozí hodnoty nebo doplnění pomocí předem definované konstanty.

2.3.5 TimescaleDB

TimescaleDB je další z databází využitelných v architektuře pro sběr a zpracování velkého množství dat s časovou složkou. Jedná se o open-source rozšíření databáze PostgreSQL, které má údajně optimalizovanou část pro práci s daty typu time series a tím pádem dosahuje 10-100násobné rychlosti pro dotazy. Stejně jako InfluxDB i tato databázová služba je připravena k nasazení na službách AWS, GCP nebo Azure. Další z výhod použití této databáze je možnost snadného kombinování relačních a time series dat, což dává prostor pro více komplexní dotazy, a tím pádem i lépe využitelné vizualizace a analýzy.

2.3.6 Prometheus

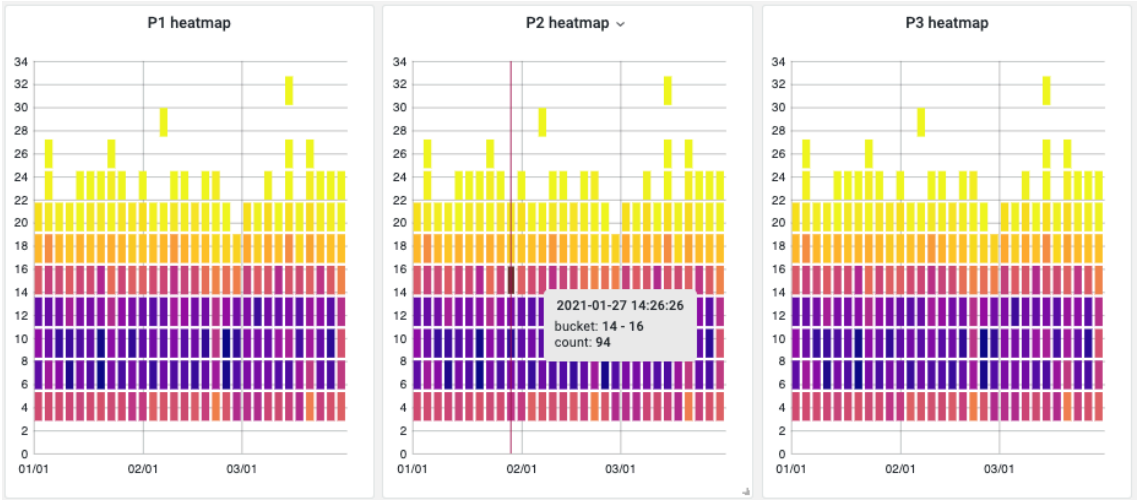
Prometheus je open-source monitorovací systém, který vytváří a spravuje výstrahy. Funguje na principu time series databáze, tj. informace o metrikách jsou uloženy s časovým razítkem a s volitelnými dvojicemi klíč-hodnota. Mezi hlavní funkce aplikace Prometheus patří vícerozměrný datový model s daty časových řad identifikovanými pomocí názvu metriky a dvojic klíč/hodnota, PromQL, což je flexibilní dotazovací jazyk pro využití vícerozměrného modelu. Dále neexistuje závislost na distribuovaném úložišti; jednotlivé uzly serveru jsou autonomní a shromažďování časových řad probíhá prostřednictvím modelu pull přes protokol HTTP. Prometheus podporuje více režimů tvorby grafů a dashboardů, což je vhodné pro vizualizaci výstrah a dostupných dat v monitorovacím systému.

2.3.7 Agregace dat

Agregace dat je důležitá součást analýzy dat. Využívá se pro úpravy dat a pro generování statistik. Kvůli množství generovaných dat v prostředí IoT je agregace dat velmi důležitá. Umožňuje nám vyčíst z velkého množství dat na první pohled nerozpoznatelné informace. Agregace dat může probíhat na databázové úrovni pomocí skriptu, který bude vybírat pomocí databázových dotazů konkrétní data a bude je definovaným způsobem zpracovávat. Tento princip se dá využít v aplikaci Grafana, která umí vybírat například dvě datové hodnoty za sebou a vzájemně je porovnávat (viz 3.1.4). Další z panelů použitých v aplikaci Grafana je panel heat mapy (tepelná mapa, obrázek 2.1), což je vizualizační reprezentace dat sloužící pro zobrazení rozložení hodnot v dvourozměrném poli. Implementaci heat mapy je možno vidět na

obrázku 2.1.

Kvůli velkému množství dat byl zaveden pojem Big Data, který popisuje takové množství dat, které není schopen pojmout a zpracovat běžný systém. Kolem zpracování, ukládání a analýzy dat je postavený celý analytický obor. Výhody prameníci ze zpracování takového objemu dat jsou například odhalování korelací v měřítku nepostřehnutelném člověkem, nečekaných vzorců v posloupnostech dat, tržní trendy, preference zákazníků a další užitečné obchodní informace. Kvůli velké náročnosti tradičního zpracování je vhodné použít nástroje pro agregaci dat, čímž vytváříme tzv. metadata, což jsou data, která poskytují informaci o jiných datech.



Obrázek 2.1: Grafana - heat mapy

Existují různé druhy analytických systémů a zdroj [17], ze kterého jsou převzaty informace o Big Data a jejich zpracování, rozděluje tyto systémy do 5 různých druhů. Mezi tyto druhy patří analýza v reálném čase, která je většinou prováděna na datech pocházejících ze senzorů kvůli rychlosti změn hodnot. Hlavní používané architektury pro tento druh analýzy jsou clustery pro paralelní zpracování využívající tradiční relační databáze a výpočetní platformy založené na pamětích. Další druh analýzy je offline analýza, která je vhodná pro použití v situacích, kdy nepožadujeme okamžitou odezvu. Například mnoho internetových podniků používá off-line analýzu, aby snížily náklady na konverzi formátu dat. Analýza na úrovni paměti je další z druhů analytického systému, který se používá v případě, že je velikost dat menší než paměť clusteru. Taková analýza je vhodná pro použití v reálném čase. Příkladem této architektury je MongoDB. Analýza Business Intelligence se používá se specifickou velikostí dat, které jsou následně importována do analytického prostředí.

Tento druh analýzy podporuje data řádově na úrovni TB. Poslední druh analýzy je masivní analýza, která využívá distribuovaný souborový systém Hadoop a je určená pro největší objemy dat, které jsou nevhodné pro předchozí druhy analýz.

2.4 Senzory

Celý obor IoT stojí na datech, která potřebujeme nejenom přenášet, ukládat a vizualizovat, ale i získávat z daného prostředí. K tomu nám slouží široká škála různých senzorů. Senzor je zařízení, které měří nějakou veličinu. Díky tomu, že se senzory začaly vyvíjet nezávisle na IoT, jsou jejich technologie značně pokročilé. Senzory máme tím pádem většinou velmi malé, levné a energeticky nenáročné, což se velmi hodí například při napájení IoT zařízení bateriemi. Jedno z možných dělení senzorů je podle druhu veličiny, kterou senzor měří. Následující senzory patří mezi nejčastěji používané.

Teplotní senzory a senzory vlhkosti - Tyto dva druhy senzorů jsou velmi rozšířené a často jsou prodávány společně v jednom modulu, protože chceme ve většině případů monitorovat obě veličiny zároveň. Teplotní senzory jsou v průmyslu využívány pro kontrolu teploty strojů kvůli optimalizaci spotřeby, chlazení, opotřebení a s tím souvisejících provozních nákladů a kvality výrobků. Dále jsou tyto druhy senzorů používané v potravinářském průmyslu pro kontrolu teploty potravin převážně při skladování a převozu. Senzory teploty jsou využívány i ve zdravotnictví, skladování a systémech HVAC (zkratka z anglického „heating, ventilating, air-conditioning“, česky topení, větrání a klimatizace).

Senzory tlaku - Senzory tlaku měří tlak v kapalinách a plynech. Jsou používané převážně v průmyslu – například rafinérie, energetika a údržba. Barometrické senzory tlaku najdeme v meteorologických stanicích pro měření atmosferického tlaku.

Senzory pohybu - Používají se k detekci pohybu v daném prostředí. Bývají jedním z bezpečnostních prvků v chytrých domácnostech. Můžou být doplněné například o skript, který v případě zachycení pohybu zapne nahrávání bezpečnostních kamer a upozorní majitele domácnosti.

Senzory hladiny - Jsou využívány nejen v automobilovém průmyslu. Informace o výšce hladiny je užitečná v mnoha odvětvích průmyslu, jako je například chemický, potravinářský nebo ropný průmysl.

Obrazové senzory - Obrazové senzory zachycují obraz a převádí ho do digitální podoby pro následné zpracování. Tyto senzory jsou užitečné pro technologie čtení

poznávacích značek u aut nebo pro automatické průmyslové linky, například jako kontrola kvality. V dnešní době se také rozvíjí systémy založené na strojovém učení, umělé inteligenci.

Senzory přiblížení - Tyto senzory detekují přítomnost objektu pomocí různých technologií bez dotyku daného objektu. Mezi tyto technologie patří indukční senzory na principu elektromagnetické indukce, vhodné pro kovové materiály. Kapacitní senzory fungují díky jiné relativní permitivitě objektu oproti vzduchu. Senzory fotoelektrické měří část světla odraženou od objektu. Ultrazvukové senzory fungují na principu odrážení ultrazvukových vln od objektu.

Senzory kvality vody - V dnešní civilizaci je kontrola kvality vody nesmírně důležitá a pro její kontrolu je velmi vhodné využít dostupné IoT technologie. Tyto senzory měří hlavně teplotu, kalnost, hodnotu pH, vodivost a koncentraci rozpuštěného kyslíku.

Chemické senzory a senzory plynů - Chemické senzory a senzory plynů mají velkou roli v chemickém průmyslu. Hlavní využití je detekce nebezpečných chemických látek a plynů ve vzduchu - dokáží detekovat únik nějaké skladované, potenciálně nebezpečné látky a plynů.

Senzory kouře - Senzory kouře jsou důležité nejen v domácnostech, ale i v průmyslových halách, kde je potencionální nebezpečí ohně. Tyto senzory fungují většinou na principu ionizace nebo na principu optického senzoru.

Infračervené teplotní senzory měří infračervenou energii vyzařovanou z objektu za účelem určení teploty objektu. Jsou to bezkontaktní zařízení užitečná v různých aplikacích, kde není možné nebo žádoucí mít senzor v kontaktu s měřicí plochou. Popis IR senzorů pochází ze zdroje [18].

Senzory akcelerace - akcelerometry - „Je to zařízení, které zajišťuje možnost měření a analýzy lineárního a úhlového zrychlení. Tato funkce je nezbytná v mnoha základních zařízeních a systémech používaných v téměř každé oblasti života – jak v domácích zařízeních pro každodenní použití, tak i v profesionálních průmyslových nebo výzkumně-vývojových aplikacích.“

Gyroskopy - Jsou využívány pro udržování a měření úhlové rychlosti a orientace objektu. Na rozdíl od akcelerometru dokáží měřit i náklon a boční orientaci. Hodí se pro určení orientace objektu.

Optické senzory - Používají se v průmyslu běžně. Fungují na principu přerušení paprsku světla v případě, že se objekt dostane mezi přijímač a vysílač. Informace o senzorech jsou převzaté ze zdrojů [19], [20] a [21].

2.5 Deployment

Docker je open-source platforma používaná pro jednoduché nasazování aplikací. Funguje na principu kontejnerů, což je prakticky izolovaná vrstva nad Dockerem. Existuje podobnost s virtualizací, která má nad vrstvou operačního systému tzv. hypervizor, na kterém jsou spuštěné jednotlivé operační systémy se svými knihovnami a aplikacemi. Systémy založené na platformě Docker mají oproti tomu operační systém, nad kterým běží Docker aplikace, která vytváří jednotlivé kontejnery s definovanými knihovnami a spouští v nich jednotlivé aplikace. Tím pádem odpadá vrstva hypervizoru a problém s několika operačními systémy běžícími na jednom stroji. Mezi hlavní výhody patří jednoduché nasazení Docker aplikací v cloudu, zvýšená bezpečnost a jednoduchá implementace do již vytvořené aplikace, která se jako taková neupravuje, pouze se pro ni nastaví parametry daného Docker kontejneru.

Článek [22] uvádí jako hlavní výhody použití Dockeru rychlost, přenositelnost, škálovatelnost, rychlé nasazování a využití systémových prostředků. Rychlost Dockeru je dána především používáním malých kontejnerů, které obsahují jednu aplikaci, a tudíž se rychle vytvoří a spustí. Díky rozsáhlému repozitáři se také velmi snadno implementuje kontejner do produkčního serveru. Přenositelnost spočívá v jednoduchém přenosu jednotlivých kontejnerů a jejich obsahu. Škálovatelnost je velmi dobrá díky jednoduchým přenosům a úpravám kontejnerů.

Další varianta nasazení aplikace je nasazení v prostředí AWS, této problematice se věnuje kapitola 3.2.

Mezi korporátní řešení patří také nasazování IoT aplikací v prostředí Azure. Podle dotazníku organizace Eclipse foundation (viz zdroj [23]) je Azure druhá nejžádanější platforma pro IoT aplikace realizované v cloudu. Realizace IoT v prostředí Azure IoT Central je velmi jednoduchá, uživatel si může vybrat z přednastavených vzorů a jednoduše je použít. Jedna z hlavních výhod použití hotového korporátního řešení je také bezpečnost, Azure ji má na velmi dobré úrovni.

Google Cloud Platform je stejně jako AWS či Microsoft Azure další korporátní řešení pro realizaci aplikací v cloudovém prostředí. Tato platforma je realizována na stejné infrastruktuře jako některé služby společnosti Google, například Google Vyhledávání, Gmail nebo Google Disk. Google Cloud Platform má velkou škálu produktů, od výpočetních serverů přes databázové služby po produkty založené na umělé inteligenci či produkty přímo zaměřené na nasazování v prostředí IoT. Obdoby většiny těchto produktů je možné nalézt v nabídce konkurenčních společností,

které se zabývají provozem služeb v cloudovém prostředí. Pokud jde o aplikace internetu věcí (dle zdroje [24]), je zřejmé, že Google musí pokročit, aby mohl konkurovat AWS a Microsoft Azure. To neznamená, že pro určité aplikace nemůže být GCP nejvhodnější varianta. Pro porovnání platforem je využit článek [24], který porovnává tyto platformy z pohledu uživatele.

3 Implementace - Aplikace pro sběr a zpracování teplotních dat

3.1 PostgreSQL implementace

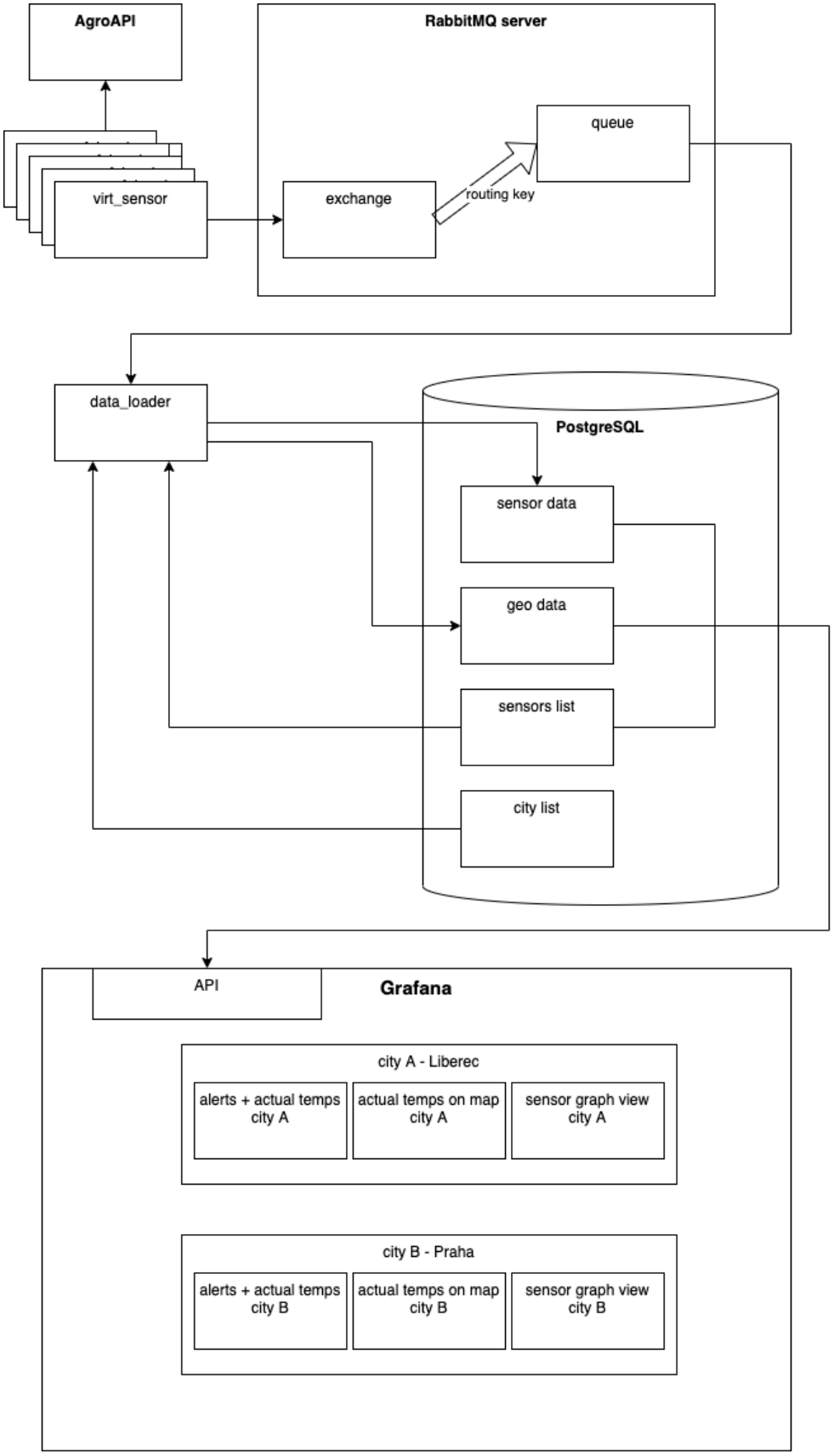
Tato aplikace byla vyvinuta jako ukázkové řešení implementace open-source technologií pro řešení sběru a zpracování dat v oblasti internetu věcí. Řešení je plně funkční a vhodné pro ukázku výhod a nevýhod dané architektury.

Aplikace je založena na přenosu dat ze simulovaných senzorů do databáze a následně do uživatelského rozhraní, kde koncový uživatel může monitorovat a analyzovat naměřená data.

Aplikace funguje na bázi přenosu dat ze simulovaných senzorů do relační databáze PostgreSQL pomocí protokolu MQTT a jeho brokera RabbitMQ. Mezi MQTT brokerem a databází PostgreSQL je implementován program dataloader, který má na starost routing dat, verifikaci senzorů a vytváření vstupních skriptů do databáze s příchozími daty.

3.1.1 Generování dat

Kvůli absenci vlastní měřicí techniky bylo potřeba realizovat měření pomocí softwarové varianty. Pro účely tohoto měření jsem využil volně dostupné rozhraní Agriculture API, což je rozhraní, kde lze získat data relevantní nejen v zemědělství. Patří mezi ně například satelitní snímky, historické, aktuální i předpovídané hodnoty o teplotě, vlhkosti nebo tlaku. Díky svému přístupovému API klíči jsem byl schopný realizovat odečet teploty pro libovolnou lokalitu, v tomto případě města Liberec a Praha. Tato data se generují pomocí aplikace data_generator, která obsahuje mimo jiné dvě důležité funkce, a to funkci pro generování teploty a funkci pro odesílání dat do RabbitMQ. Funkce generate_temperature data v ideálním případě přímo negeneruje, ale získává je pomocí dotazu na Agriculture API. Jedná se o si-



Obrázek 3.1: Diagram architektury

mulaci teplotního senzoru. Hlavní úkol této funkce je tedy vytvořit správnou adresu, na kterou se bude následně dotazovat. Součástí dotazovací adresy daného prostředí je hlavně hodnota zeměpisné šířky a zeměpisné délky. Dále dotaz obsahuje kromě samotné adresy rozhraní i osobní klíč, který byl uživateli rozhraní přidělen. Tento klíč je stejně jako přihlašovací údaje uložen v systémových proměnných. Klíč je složený z 32 hexadecimálních znaků. Reálné senzory nejsou vždy bezchybné, a z tohoto důvodu je zavedená umělá chyba i ve funkci pro sběr hodnot. Tato chyba je realizována pomocí případného selhání získání údaje z rozhraní pro sběr dat a následným generováním náhodné hodnoty. S touto chybou je tedy nutné se vypořádat při zpracování dat, což se děje v tomto konkrétním případě v aplikaci Grafana při zpracování zobrazovaných dat pro město Praha. Následuje kód funkce generate_temperature.

```
# Funkce generate_temperature
    def generate_temperature(logger):
    position = 'lat=' + LAT + '&lon=' + LON
    url = OWM_API_URL + position + '&appid=' + OWM_API_KEY
    try:
        response = requests.request(method='GET', url=url)
        if response.status_code == 200:
            payload = json.loads(response.text)
            temp = round(payload['main']['temp'] - 273.15, 2)
            logger.info('Agro API request ok - ' + str(temp))
        else:
            logger.info('Agro API request failed')
            temp = round(random.uniform(0.00, 33.33), 2)
            logger.debug('Generated temp: ' + str(temp))
        return temp
    except Exception as error:
        logger.error('temperature generation failed')
        logger.debug(error)
        return 0
```

Přenos samotných dat probíhá pomocí odesílání generovaných dat aplikací data_generator do MQTT brokeru RabbitMQ, kde data_generator slouží jako klient-publisher. Odesílá zprávy do tzv. exchange, která slouží pro rozdělování zpráv do

jednotlivých specifických front (queues). Samotné rozdělování zpráv - routing probíhá v případě této aplikace na základě tématu (topic). Tento princip funguje na základě uloženého routovacího klíče v samotné zprávě, pomocí kterého RabbitMQ broker určí, na základě regulárních výrazů v jednotlivých frontách, do které fronty zprávu odešle.

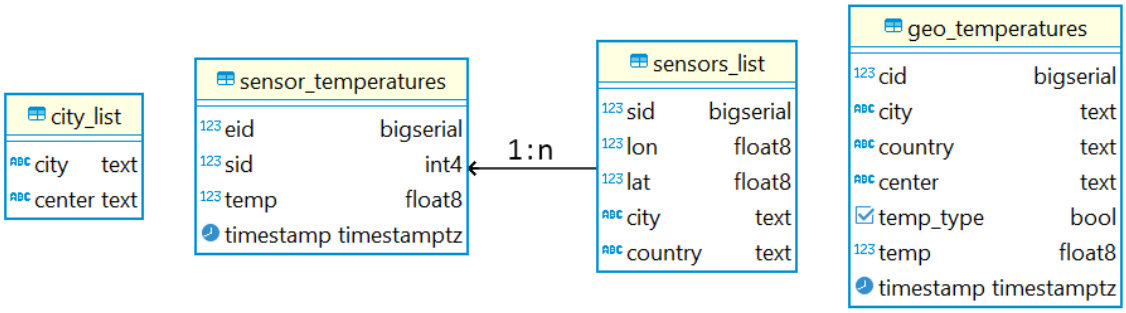
Pro zápis dat na serverové straně aplikace potřebujeme tzv. konzumovat data ze své fronty na straně RabbitMQ. Toho docílíme díky implementaci Python knihovny Pika. Jedná se o knihovnu, která pracuje na základě AMQP protokolu a je vhodná k vytvoření konzumenta pro data ve frontách brokera. Dále aplikace dataloader zapisuje data do databáze PostgreSQL.

Přenos dat do aplikace Grafana je jednoduchý, protože probíhá pomocí přímé komunikace API Grafany s databázovým serverem PostgreSQL. Celá konfigurace týkající se tohoto přenosu je definována v grafickém prostředí Grafany.

Při návrhu databáze byl hlavní účel spolehlivost a jednoduchá implementace. Vzhledem ke článku [15], který porovnává relační a nerelační databáze vzhledem k užití v prostředí internetu věcí, jsem vybral databázi relační. Z článku plynou jisté výhody a nevýhody obou typů databáze. Pro tuto konkrétní aplikaci nebyla tolik zásadní rychlost zápisu a čtení z databáze, což je výhoda nerelačních databází, díky relativně dlouhým časovým úsekům mezi jednotlivými čteními ze simulovaných senzorů. Proto jsem raději zvolil cestu subjektivně jednodušší implementace a rozhodl jsem se přiklonit se k spolehlivější architektuře, čímž je databáze relační. Nerelační databáze jsou lépe využitelné v různorodém IoT prostředí, kde data pocházejí z různých druhů zdrojů, protože u nich není třeba řešit jednotlivé tabulky a schémata.

Při výběru konkrétní databáze jsem vycházel z článku [25], kde autoři hodnotí výkonnost open-source relačních databází, mezi něž patří MongoDB, MySQL a PostgreSQL. Z tohoto porovnání vychází pro čtení menšího počtu dat nejlépe PostgreSQL. MySQL je rychlejší než PostgreSQL při více než 20 000 záznamech, ale PostgreSQL databáze stále poráží MongoDB. Při čtení malého množství dat je nejlepší MongoDB a pro zápis většího počtu dat vychází nejlépe PostgreSQL. Protože je v mé aplikaci vhodnější rychlý zápis dat do databáze a na rychlost čtení není kladen takový důraz, tak jsem zvolil databázi PostgreSQL. Tato databáze má také výhodu ve snadné implementaci v prostředí Grafany, což by bylo obtížnější pro databázi MongoDB. Jednotlivé tabulky v databázi jsou zobrazeny na obrázku 3.2.

Stejně jako ostatní části aplikace je databáze spuštěna na CentOS serveru.



Obrázek 3.2: MongoDB - tabulky

3.1.2 Logger – záznam chyb při běhu programu

Celá aplikace je uzpůsobena pro vývoj a případné hledání chyb v celém procesu díky implementaci tzv. loggeru. Ten nám dává informaci o běhu programu napříč celou aplikací. Inicializace loggeru a parametr pro nastavení úrovně informování jsou v každé části aplikace.
Inicializace loggeru:

```
logging.basicConfig(format=FORMAT)
    logger = logging.getLogger('JV_data_loader')
    logger.setLevel(args.loglvl)
    wait_for_rabbit(logger)
    logger.info('Started')
```

Vstup pro nastavení úrovně logování:

```
    parser = argparse.ArgumentParser(description='Loads
        temperature messages')
    parser.add_argument('--loglvl', type=str,
        default='INFO', help=('level of logging.
        Possible values are: DEBUG, INFO, WARN'))
```

Logger má v tomto případě tři různé úrovně zpráv. Tyto úrovně se definují pomocí argumentu použitého při spouštění jakékoliv části aplikace. Úrovně jsou definovány následovně:

- DEBUG - varianta, které se používá hlavně při vývoji; jedná se o nejnižší úroveň informování, kde nám programy hlásí nejvíce možných informací. Užitečnost této úrovně je tedy hlavně při hledání důvodu nefunkčnosti aplikace nebo její části.

- INFO - základní varianta. Pokud nedefinujeme jinou, tak se použije tato; informuje méně detailně o běhu programu, máme tedy všechny důležité informace o běhu programu; snadno se dá při této úrovni vyznat v informacích o průběhu.
- WARN - úroveň s nejmenším počtem informací; na výstupu se objevují pouze závažné informace o běhu programu.

3.1.3 Dataloader

Program dataloader je další zásadní komponentou aplikace. Jeho účel spočívá v odběru dat ze žádaného tématu z MQTT brokera RabbitMQ a jejich zápisu do relační databáze PostgreSQL (viz podkapitola 2.3.1). Odběr zpráv z daného tématu probíhá pomocí knihovny Pika. Pro úspěšné propojení aplikace dataloader a MQTT brokera RabbitMQ je potřeba přihlašovací jméno, heslo, IP adresa serveru s RabbitMQ, port, název fronty, ze které chceme odebírat data, a routovací klíč. Po spuštění čeká aplikace na příchozí data a po přijetí dat se spouští funkce callback, která má na starost kontrolu formátu dat, verifikaci identifikačního čísla senzoru a spuštění funkcí na vložení dat do databáze. Aplikace má možnost zápisu do jednodušší či složitější tabulky, záleží, jaké informace potřebujeme. Popis těchto tabulek naleznete v příloze 3.2.

YAML (zkratka z anglického „YAML Ain't Markup Language“) je datový formát pro serializaci strukturovaných dat. Díky snadné čitelnosti počítačem i člověkem je vhodný pro předávání a ukládání parametrů aplikace. V mé open-source implementaci jsem jazyk YAML použil pro předávání zeměpisných souřadnic aplikaci data_generator, která tyto souřadnice následně použije při volání API s žádanými hodnotami v dané lokaci. Soubor pos.yaml použitý pro tento účel je na obrázku 6.6.

JSON je stejně jako YAML datový formát snadno čitelný člověkem i strojem. Je to výpočetně nenáročný protokol založený na podmnožině programovacího jazyka JavaScript. JSON je textový, na jazyce zcela nezávislý formát, využívající však konvence dobře známé programátorům jazyků rodiny C (C, C++, C#, Java, JavaScript, Perl, Python a dalších). Díky tomu je JSON pro výměnu dat ideálním jazykem.

JSON je založen na dvou strukturách: Jedna z nich je kolekce párů název/hodnota. Ta bývá v rozličných jazycích realizována jako objekt, záznam (record), struktura (struct), slovník (dictionary), hash tabulka, klíčový seznam (keyed list) nebo asociativní pole. Druhá struktura je seznam seřazený hodnot. Ten je ve větši-

ně jazyků realizován jako pole, vektor, seznam (list) nebo posloupnost (sequence). Jedná se o univerzální datové struktury a v podstatě všechny moderní programovací jazyky je v nějaké formě podporují. Je tedy logické, aby na nich byl založen i na jazyce nezávislý výměnný formát.

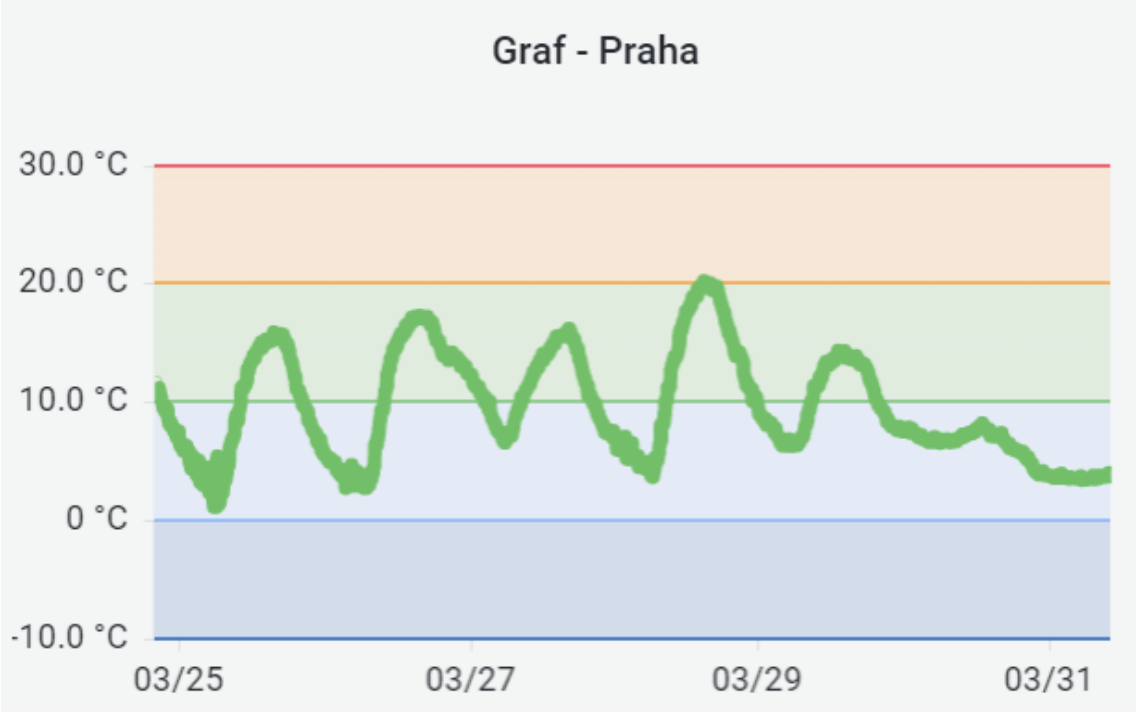
Zdroj informací o jazyce JSON je [26]. V mé aplikaci je JSON využíván pro přenos dat v aplikaci dataloader. Vybral jsem si tento formát kvůli jednoduché práci s ním v jazyce Python pomocí knihovny json. Následující část kódu z aplikace dataloader popisuje načtení JSON dat do proměnné data, ze které následně ověřujeme, zda senzor, ze kterého přišla data, existuje.

```
if method.routing_key == 'sensors.data':
    try:
        data = json.loads(body)
        logger.debug(data['temperature'])
        logger.debug(data['sensor_id'])
        if validate_sensor(data['sensor_id']):
            basic_insert(logger, data)
            geo_insert(logger, data)
        else:
            logger.error('Unknown sensor id')
    except Exception as error:
        logger.debug(error)
        logger.error('Data arent JSON')
```

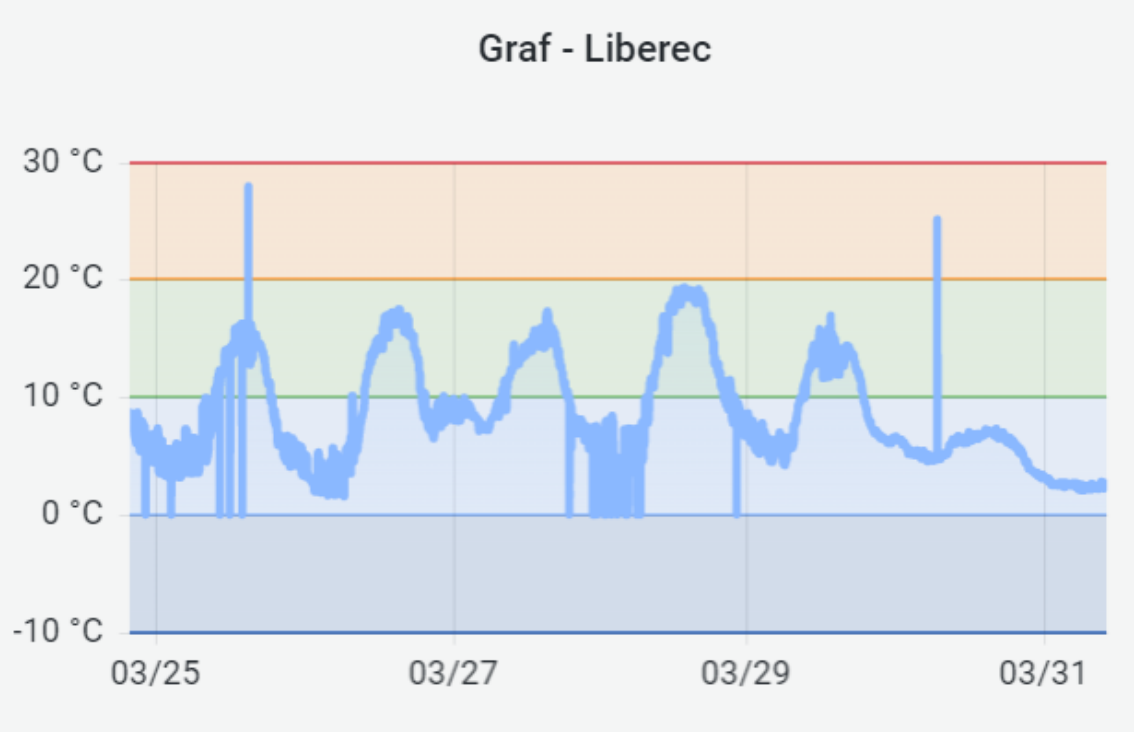
Geohash je unikátní klíč, pomocí kterého lze identifikovat jakékoliv místo na Zemi. Spočívá v rozdělení povrchu Země na regiony, jejichž velikost je definovaná počtem použitých bitů - čím méně bitů Geohash má, tím větší plocha je jím specifikovaná. Geohash podporuje formu zápisu pomocí binárních čísel nebo i pomocí alfanumerických znaků, což je běžnější varianta zápisu. Jeho přesnost záleží na počtu použitých bitů, při použití 9 znaků - 45 bitů je specifikovatelná oblast o velikosti 22,92 metru čtverečních. V aplikaci používám Geohash o délce 11 znaků pro přesnější zaměření. Hlavním účelem použití Geohash v mé implementaci je snadné zobrazení dat na panelu WorldMap v aplikaci Grafana (3.1.4).

3.1.4 Grafana

Grafana je front-endový software používaný pro analýzu, monitorování a zobrazování dat. Funguje na principu jednotlivých panelů, které lze jednoduše implementovat a upravit podle požadavků. Dále podporuje použití externích rozšíření, konkrétně v mé implementaci se jedná o použití WorldMap panelu pro zobrazení polohy jednotlivých simulovaných senzorů (viz spodní část obr. 3.5). Grafana je v mé realizaci propojena s databází a následně v jednotlivých panelech je upravován SQL dotaz na danou databázi pro výběr požadovaných dat. Grafana umožňuje také jednoduché zpracování dat na straně Grafany, například pro vyfiltrování a vyřazení chyb používám v grafu pro Prahu (viz obrázek 3.3) derivaci a následné vyřazení hodnot s větší derivací, než je 2. Tato hodnota byla zvolena na základě typické rychlosti změn venkovní teploty. Díky malému intervalu sběru teplot (1 minuta) je skoková změna strmější než f(x)=2x považována za chybnou. Tímto způsobem detekuji a odstraňuji simulované chybné hodnoty, které vykazují velkou skokovou změnu, která je v případě teplotních rozdílů nereálná, respektive velmi málo pravděpodobná. Takto neodfiltrovaná data jsou zobrazena pro graf - Liberec (viz obrázek 3.4).



Obrázek 3.3: Grafana - Graf - Praha

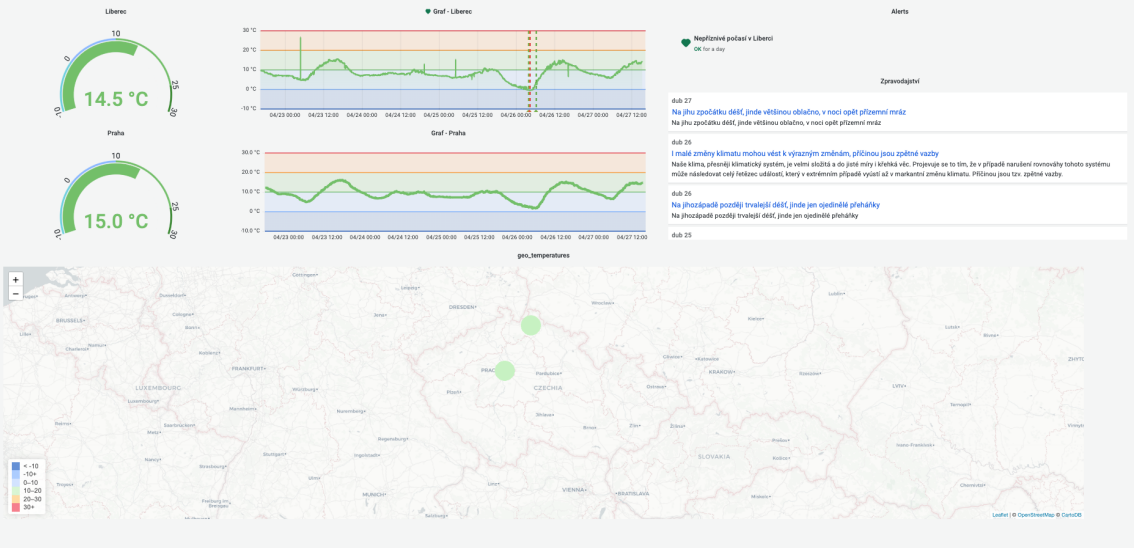


Obrázek 3.4: Grafana - Graf - Liberec

Uživatelské rozhraní dále obsahuje dvě kapitoly, které ukazují teplotu z posledního odečtu pro dané místo. Všechny teplotní rozsahy jsou označeny stejnou barvou v každém z panelů. Tato barevná spektra řeší z větší části Grafana sama, u WorldMap panelu bylo potřeba spektrum posunout, aby odpovídalo i zbytku panelů.

V aplikaci Grafana lze nastavit výstrahy, tzv. Alert list. Tyto výstrahy lze nastavit pomocí pravidel, která mají vstup z panelu grafu. Pomocí regulárních výrazů a matematických operací lze nastavit podmínky pro spuštění výstrahy. Příklad těchto podmínek v mé implementaci je v příloze 6.2. Při těchto podmínkách se výstraha spustí v případě, že teplota je menší než 0 nebo větší než 30 stupňů Celsia. Dané výstrahy se zobrazují i v grafu jako svislé přerušované čáry. K výstraze je možno přiřadit e-mailovou adresu kam se odešle v případě aktivování výstrahy upozornění. Tato část Grafany se také dá propojit s monitorovacím softwarem Prometheus, který je popsaný v podkapitole 2.3.6.

Mezi další panely patří panel Zpravodajství, který je založený na odběru RSS (zkratka z anglického „Rich Site Summary“) kanálu z webu in-pocasi.cz. Jedná se o panel News. Četnost příchozích zpráv je zhruba 1–2 zprávy denně. Pro mou aplikaci se tento panel hodí na získávání dodatečných informací o počasí v ČR.



Obrázek 3.5: Grafana - hlavní panel

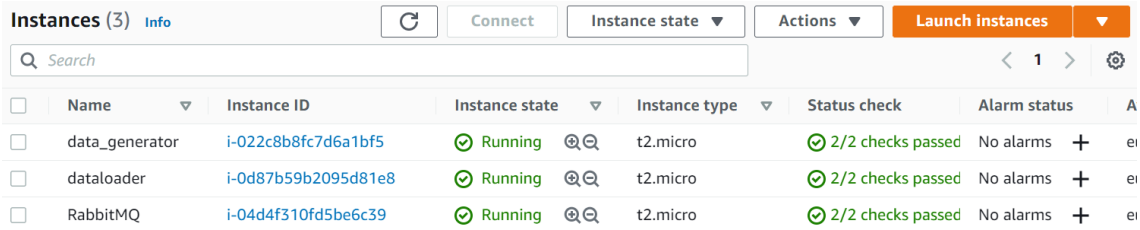
Timeseries databáze (viz 2.3.3) se v prostředí Grafany velmi jednoduše vizualizují. Pro taková data existuje panel Timeseries, který rovnou realizuje časovou osu a grafické znázornění hodnot. Samotnou vizualizaci dat je možno nastavit jako přímku mezi body, respektive Grafana podporuje i interpolaci datových bodů křivkou. Další možnost zobrazování je tzv. step-before a step-after, viz vizualizace.

Určité části této implementace vyžadují verifikaci pomocí přihlašovacího jména a hesla. Aby se tato jména a hesla nemusela opakovaně zadávat, tak jsou uložena v systémových (environment) proměnných, což jsou proměnné, které jsou nastaveny přímo v systému. Tyto proměnné si následně aplikace přečte a přihlašuje se pomocí nich.

3.2 Implementace v prostředí AWS

Jako druhou možnou implementaci aplikace je možno zvolit komerční prostředí Amazon Web Services. Díky modulům EC2 (zkratka z anglického „Amazon Elastic Compute Cloud“) a RDS (zkratka z anglického „Amazon Relational Database Service“) je možno aplikaci realizovat v korporátním prostředí, které má své výhody a nevýhody. Toto řešení eliminuje problémy s pokročilejší správou samotných datových serverů, usnadňuje monitoring a existuje technická podpora v případě řešení problémů. Mezi nevýhody patří cena, při reálném nasazení v nejnižší konfiguraci stojí databáze 1,284 amerického dolaru za hodinu. Jednotlivé služby, které jsou v open-

-source řešení realizované pomocí kontejnerů, jsou v prostředí AWS implementované jako jednotlivé instance, virtuální stroje. Další nevýhodou je špatný přehled o financích. Pokud je IoT aplikace realizovaná na soukromém serveru, tak nemusí docházet k nečekaným výdajům.



Obrázek 3.6: Jednotlivé instance v AWS

3.3 Implementace MongoDB

Pro určité porovnání je další implementace zaměřená na výhody a nevýhody využití nerelační databáze (2.3.2). Pro některá využití v prostředí IoT je výhodnější použít nerelační databázi. Konkrétní výhoda v případě mé implementace může být například jednoduché přidání další měřené veličiny, jako je vlhkost či atmosferický tlak. To by v případě relační databáze byl problém, který by se řešil přidáním další tabulky nebo předěláním tabulky původní. Kompatibilitu databáze s příchozími daty u nerelační databáze nemusíme řešit.

Pro implementaci databáze MongoDB jsem nainstaloval na svém serveru MongoDB server, na který jsem následně vložil všechna doposud naměřená data. Docílil jsem toho pomocí jednoduchého exportu dat z databáze PostgreSQL díky kombinaci příkazů COPY a SELECT, kde jsem v rámci příkazu SELECT určil požadované sloupce a jejich datové typy a následně je vykopíroval pomocí příkazu COPY ven z databáze do souboru TSV (tab-separated values). Druhá možnost migrace dat by byla při použití formátu JSON, který podporuje komplexnější (hierarchická) schémata databáze. To není případ mé aplikace, tudíž jsem zvolil formát TSV (zkratka z anglického „Tab-Separated Values“, česky tabulátorem oddělené hodnoty). Samotný příkaz je v příloze 6.5. Podobné příkazy jsem vytvořil i pro zbývající tři tabulky a vykopíroval jimi obsah celé databáze. Pro následný import do databáze MongoDB existuje příkaz mongoimport, díky kterému jsem snadno importoval data ze souboru do databáze (viz příloha 6.7 a 6.8).

Pro práci s MongoDB databází v prostředí aplikace Grafana je potřeba instalace oficiálního pluginu MongoDB, který ale není zdarma a jeho implementace není jednoduchá, což byla pro mne nečekaná překážka pro zprovoznění MongoDB jako zdroje dat v Grafaně. Z tohoto důvodu jsem se rozhodl zprovoznit MongoDB Atlas v kombinaci se službou MongoDB Compass.

MongoDB Atlas je databázová služba realizovaná v cloudu. Jedná se o tzv DBaaS, což znamená databáze jako služba, díky které nemusíme sami řešit komplexnost nasazování a udržování databáze, softwarové aktualizace, škálování a monitoring databáze. Služba MongoDB Atlas je realizovatelná na více možných poskytovatelích cloudového prostoru, jako je například AWS (3.2), Azure (2.5), Google Cloud Platform a další. Výhoda použití této služby je bezpečnost, která je řešena na úrovni poskytovatele služby, jednoduchá vizualizace dat a databází, přenos a práce s daty. MongoDB Atlas umožňuje tři různé uskupení databáze, tzv. clustery. Cluster je systém, který je založený na propojení více serverů, které se chovají jako jeden velký server s možností efektivního rozdělování výpočetních prostředků.

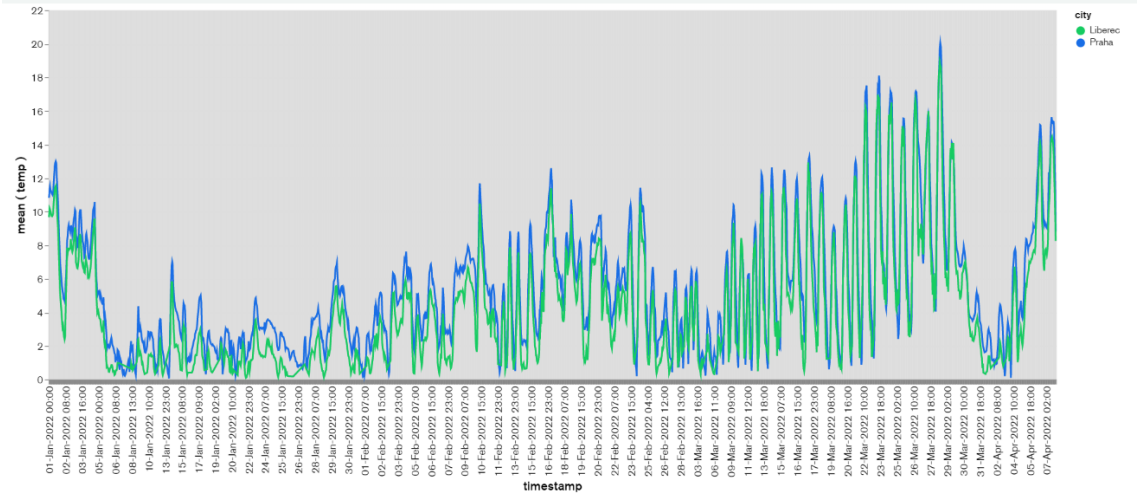
V nejzákladnější a nejlevnější variantě se jedná o sdílený cluster, který je určený hlavně pro vývoj a pro malé aplikace. Jeho nevýhoda spočívá v malém úložišti, které se pohybuje v rozmezí od 512 MB do 5 GB, což je při větším objemu dat nedostačující. Mnou importovaná data do této databáze měla velikost 21,6 MB při cca 340 000 záznamech. Na to stačí použití nejnižší konfigurace, ale také se pořád jedná o databázi testovacího účelu. Sdílené clustery nemají svou vlastní dedikovanou RAM ani vlastní virtuální procesor. Tyto prostředky jsou tedy sdílené s ostatními uživateli clusterů. Ceny této varianty produktu jsou dle požadavků na úložný prostor od verze zdarma s 512 MB úložištěm až po verzi za 25 amerických dolarů měsíčně za verzi s 5 GB úložištěm.

Dedikované clustery mají, jak vyplývá z názvu, vlastní dedikované výpočetní prostředky. Jsou tím pádem vhodnější pro nasazování reálných aplikací, pokud nepotřebujeme velký výpočetní výkon nebo úložiště. Existují dvě úrovně těchto clusterů, jedna má dedikované 2 GB RAM, 10 GB úložného prostoru a 1 virtuální procesor. Provoz této varianty stojí od 0,08 amerického dolaru za hodinu (1,79 českých korun v roce 2022). Druhá varianta pro náročnější aplikace má k dispozici 4 GB RAM, 20 GB úložiště a 2 virtuální procesory. Druhá varianta stojí 0,20 amerického dolaru za hodinu provozu (cca 4,5 českých korun v roce 2022).

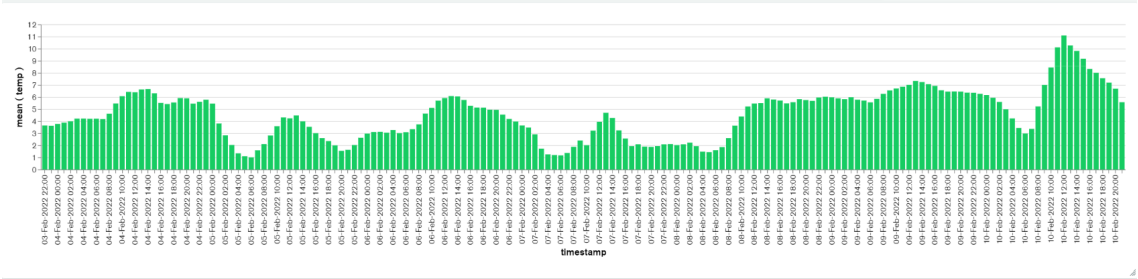
Nejlepší varianty clusterů jsou určené pro aplikace s vysokým provozem, nabízejí velikost RAM 8-32 GB, 40-160 GB úložného prostoru a 2-8 virtuálních procesorů.

Jejich cena se pohybuje od 0,54 amerického dolaru do 2 amerických dolarů za hodinu (12,1 až 44,7 českých korun v roce 2022). Tyto varianty jsou koncipované pro větší firmy nebo pro sběr velkého množství dat a jejich následné náročné zpracování.

MongoDB cloudových služeb existuje více, s databází realizovanou pomocí MongoDB Atlas komunikuje další z cloudových služeb MongoDB Charts. Pomocí této služby vizualizuji data z MongoDB podobně jako v Grafaně. Timeseries databáze je implementovaná v celém MongoDB Cloudu velmi dobře, podporuje ji jak MongoDB Atlas, tak i Charts a Compass. Možnosti vizualizace dat nejsou tak rozsáhlé jako při implementaci v Grafaně, ale vizualizace funguje dobře a jednoduše se nastavuje. Příklady vizualizací mých dat jsou na obrázku 3.7 a 3.8. Data jsou v těchto vizualizacích agregovaná průměrem za jednu hodinu.



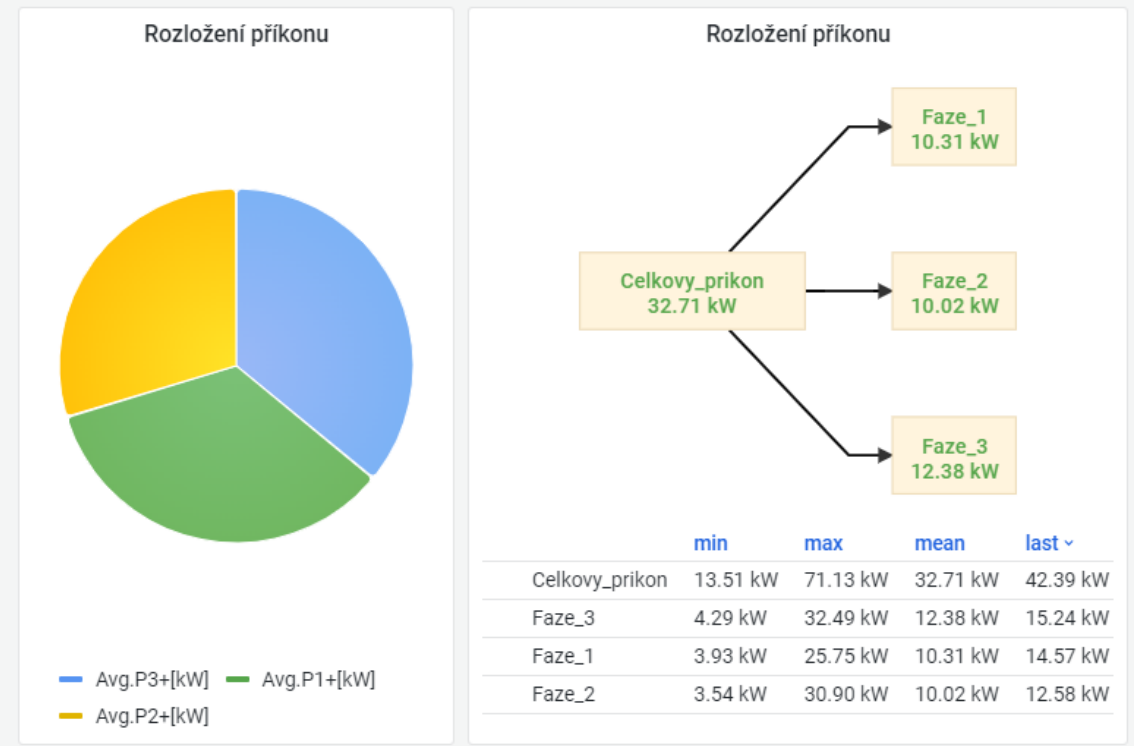
Obrázek 3.7: MongoDB - graf 1



Obrázek 3.8: MongoDB - graf 2

3.4 Zpracování energetických dat budovy

Možnosti zkombinování aplikace Grafana a databáze PostgreSQL jsou ukázány díky souboru s energetickými daty, který mi poskytl vedoucí této práce. Jedná se o data z třífázového elektroměru blíže neurčené budovy. Mezi nejdůležitější data v tomto souboru patří napětí a proudy v jednotlivých fázích, výkony a příkon. Rozložení napětí je zobrazeno v grafu v příloze 6.4 a 6.3. Další vizualizace je realizována pomocí heat mapy, která je popsána v podkapitole 2.3.7, a přímo zobrazovaný panel je na obrázku 2.1. Příkon a jeho rozložení v jednotlivých fázích je zobrazeno na obrázku 3.9. Samotná data jsou uložená v databázi PostgreSQL.



Obrázek 3.9: Grafana - rozložení příkonu

4 Porovnání výsledků

Všechny implementace popisované v předchozí kapitole jsou ve funkční fázi, a tudíž je možné popsat jejich rozdíly a porovnat výsledky plynoucí z části této práce. Tyto implementace jsou prakticky rozdělitelné na dvě varianty. První z nich je architektura postavená na relační databázi v kombinaci s vizualizační aplikací, v těchto případech konkrétně vizualizační aplikací Grafana. Druhá varianta je implementována pro porovnání jak různých typů databáze, tak i jiného vizualizačního rozhraní. Konkrétně je k tomu využito nerelační databáze MongoDB a s tím spojené vizualizační rozhraní v aplikaci MongoDB Atlas, respektive MongoDB Compass.

Hlavní rozdíl mezi relační a nerelační databází je v absenci pevně dané struktury (schématu) u nerelační databáze. Tak lze navrhnout architekturu systému i bez přesné informace o struktuře požadovaných a zpracovávaných dat. Dále tato varianta umožňuje snadné přidávání dalších datových zdrojů i s rozdílným typem dat. Při implementaci pomocí nerelační databáze byla použita data z databáze relační, tento přechod byl téměř bezproblémový díky výhodám nerelačních databází popsaných v kapitole 2.3.2. Jediný problém nastal se sloupcem obsahujícím časový údaj, timestamp, protože MongoDB používá jiný standard, než byl použitý v time series relační databázi PostgreSQL. Opačný přenos dat, z nerelační do relační databáze, by byl z podstaty relačních databází (viz 2.3.1) složitější kvůli nutnosti vytvoření struktury z existujících dat.

Mezi vizualizačními aplikacemi jsou rozdíly převážně v grafickém prostředí, systémově fungují prakticky stejně - přidá se databáze jako zdroj dat a poté se na hlavní obrazovce mohou přidávat jednotlivé vizualizační panely s vybranými daty podle sloupců nebo v případě nerelační databáze podle datových polí.

5 Závěr

Cílem této bakalářské práce bylo seznámit se s principy, protokoly a technologiemi používanými v prostředí internetu věcí a implementovat tyto poznatky v aplikacích využitelných v IoT. Princip internetu věcí a důvody pro zavedení tohoto pojmu jsou rozebírány v kapitole 2.1. V tematice spojené s internetem věcí je potřeba rozebírat nejen snímání samotných dat jako v kapitole 2.4, ale i protokoly pro přenos daných dat a datové sklady pro ukládání velkého množství záznamů. Pro účely lepšího porovnání jsou protokoly v této práci rozděleny do dvou skupin, na datové a přenosové protokoly. Nejdůležitější datový protokol pro použití v internetu věcí je subjektivně MQTT, jehož popis lze najít v části 2.2.1. Kromě MQTT je dalším podobným a srovnatelně významným protokolem AMQP, o kterém lze najít více informací v kapitole 2.2.2. Další skupina protokolů, kterým se tato práce věnuje, je skupina síťových protokolů, čímž jsou myšleny protokoly pro odesílání dat z koncových zařízení do serverové části pro zpracování a následně prováděné operace. Tomuto druhu protokolů je věnována druhá část kapitoly 2.2.

Ukládání dat se věnuje kapitola 2.3, která vymezuje rozdělení databází do hlavních skupin, relační databáze jsou popsány v kapitole 2.3.1 a nerelační databáze v kapitole 2.3.2. Mimo tyto dvě databáze obsahuje kapitola o databázích i databázi time series, což je velmi rozšířený formát databáze v IoT aplikacích, zejména kvůli časové posloupnosti snímaných dat v mnoha IoT odvětvích.

Nedílná součást IoT aplikací je nasazování softwaru, tzv. deployment. Tato část se věnuje různým možnostem nasazení, ať už v prostředí vlastního serveru za využití převážně volně dostupného softwaru, čemuž se věnuje má hlavní IoT aplikace pro sběr a zpracování dat (viz 3.1), nebo v prostředí korporátních komerčních projektů, jako je služba Amazon Web Services, Microsoft Azure, Google Cloud Platform nebo MongoDB Atlas.

Výsledkem této práce jsou dvě různé realizace sběru a zpracování dat použité na dvou různých platformách - architekturách, jedna na mém osobním běžném linu-

xovém serveru a druhá na korporátním řešení, které nabízí MongoDB - MongoDB Cloud. První aplikace se zabývá sběrem, přenosem a zpracováním dat z teplotních senzorů, které jsou simulované pomocí veřejné služby Agromonitoring API, o které je více napsáno v části 3.1.1. Tato aplikace používá služby RabbitMQ pro přenos MQTT zpráv, databázi PostgreSQL pro ukládání dat a aplikaci Grafana pro vizualizaci dat. Popis této aplikace a použitých technologií je v části 3.1 Druhá aplikace se týká analýzy a grafické reprezentace teplotních časově posloupných dat v korporátním rozhraní MongoDB Cloud. Cílem této aplikace bylo prozkoumat možnosti nerelační databáze a vyhodnotit rozdíly s databází relační.

Dále byla prozkoumána možnost implementace v AWS, která nebyla realizována jako funkční aplikace kvůli chybě v nastavování instancí a následné velké potenciální peněžní ztrátě, která naštěstí byla ze strany podpory AWS prominuta. Tato část je popsána v kapitole 3.2.

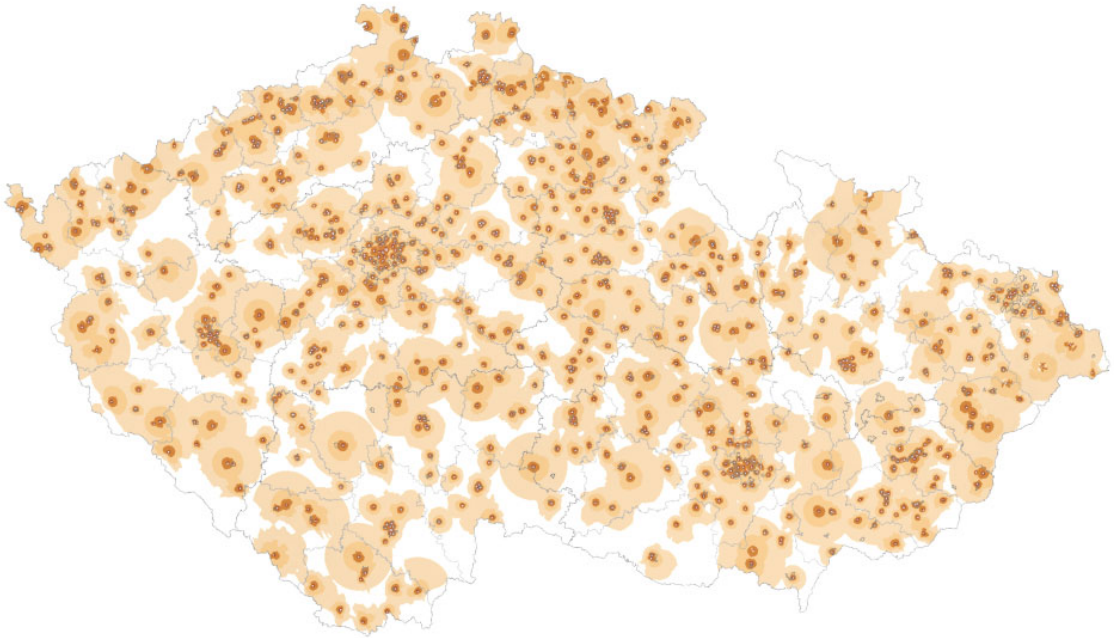
Pro zpracování dat z energetiky a v oblasti energetické účinnosti budov byla vedoucím dodána data pro analýzu příkonu, respektive jeho rozdělení v jednotlivých fázích, teplotní mapy (tzv. heatmapy) jednotlivých příkonů (viz podkapitola 2.3.7), průběhy fázových napětí pro třífázový přívod a histogram, který zobrazuje průměrný příkon pro jednu fázi za hodinu v průběhu měsíce ledna (viz 6.9). V případě, že by byla známá poloha zdroje těchto dat, tak by bylo možné propojení mé aplikace pro sběr teplotních hodnot a porovnávání například venkovní teploty s příkonem. Předpokládaný objekt je, díky rozložení průběhu příkonu během dne, obytný dům. V případě nižších teplot by se tedy dal očekávat větší příkon kvůli vytápění, osvětlení a trávení více času ve vnitřních prostorách, čímž roste čas využití elektrických spotřebičů. Mezinárodní standard ISO 50001 zabývající se energetický management rozděluje energetický management na čtyři části podle cyklu PDCA (zkratka z anglického „Plan-Do-Check-Act“, česky plánuj-dělej-kontroluj-jednej). Mnou realizovaná část je část kontrolní, která spočívá ve vizualizaci žádaných dat. V aplikaci Grafana je možné zobrazovat žádané informace, díky kterým lze energetický systém zlepšovat a optimalizovat. Rozšíření této kontrolní části a potencionální propojení s nějakou další částí PDCA cyklu je jedna z možností dalšího rozvoje této bakalářské práce. Zdroj informací o ISO 50001 je [27].

Další možnost následného rozvoje výsledků této bakalářské práce je převážně v rozšíření aplikace pro sběr a zpracování teplotních hodnot. V případě korektního nasazení je možné sbírat a uchovávat historická data. Tuto aplikaci by bylo vhodné dále rozvinout zejména v části pro ukládání a zpracování dat.

6 Přílohy

Tabulka 6.1: Síťové protokoly

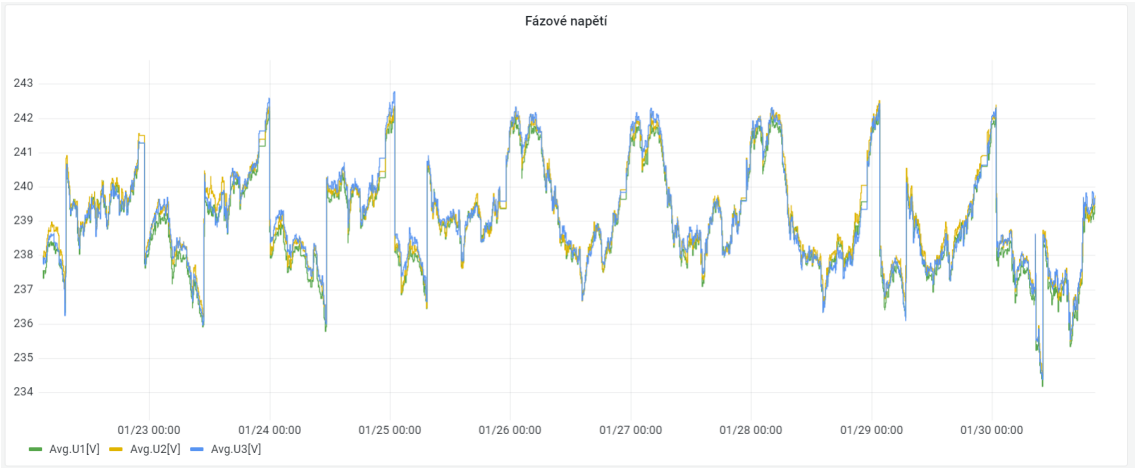
Technologie	Frekvence	Datový tok	Dosah	Spotřeba energie	Cena
2G/3G	Mobilní frekvence	10 Mbps	několik km	vysoká	vysoká
Bluetooth/BLE	2,4 GHz	1-24 Mbps / 1-2 Mbps	10-240 m	nízká	nízká
IEEE 802.15.4	<1 GHz, 2,4 GHz	40, 250 kbps	10 m	nízká	nízká
LoRa	<1 GHz (subGHz)	< 50 kbps	2-5 km	nízká	střední
LTE Cat 0/1	Mobilní frekvence	1-10 Mbps	několik km	střední	vysoká
WiFi	<1 GHz, 2,4 GHz, 5 GHz	0,1-54 Mbps	40-100 m (2,4 GHz)	střední	nízká
ZigBee	2,4 GHz	250 kbps	100 m	nízká	střední
Z-Wave	<1 GHz (subGHz)	40 kbps	30 m	nízká	střední



Obrázek 6.1: LoRaWAN pokrytí

Obrázek 6.2: Grafana - podmínky výstrah

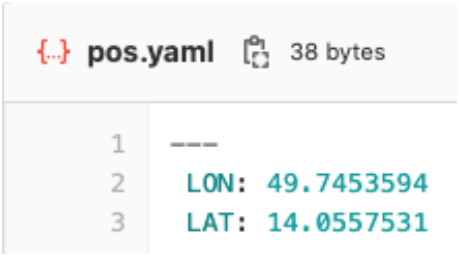
Obrázek 6.3: Grafana - napětí - detail



Obrázek 6.4: Grafana - napětí

```
COPY (SELECT
      cid AS "cid.auto()",
      city AS "city.auto()",
      country AS "country.auto()",
      center AS "center.auto()",
      temp_type AS "temp_type.bool()",
      temp AS "temp.double()",
      timestamp AS "timestamp.auto()",
    FROM
        geo_temperatures
) TO '/tmp/geo_temperatures.tsv' WITH (FORMAT CSV, HEADER TRUE, DELIMITER E'\t');
```

Obrázek 6.5: TSV - kopírování z PostgreSQL



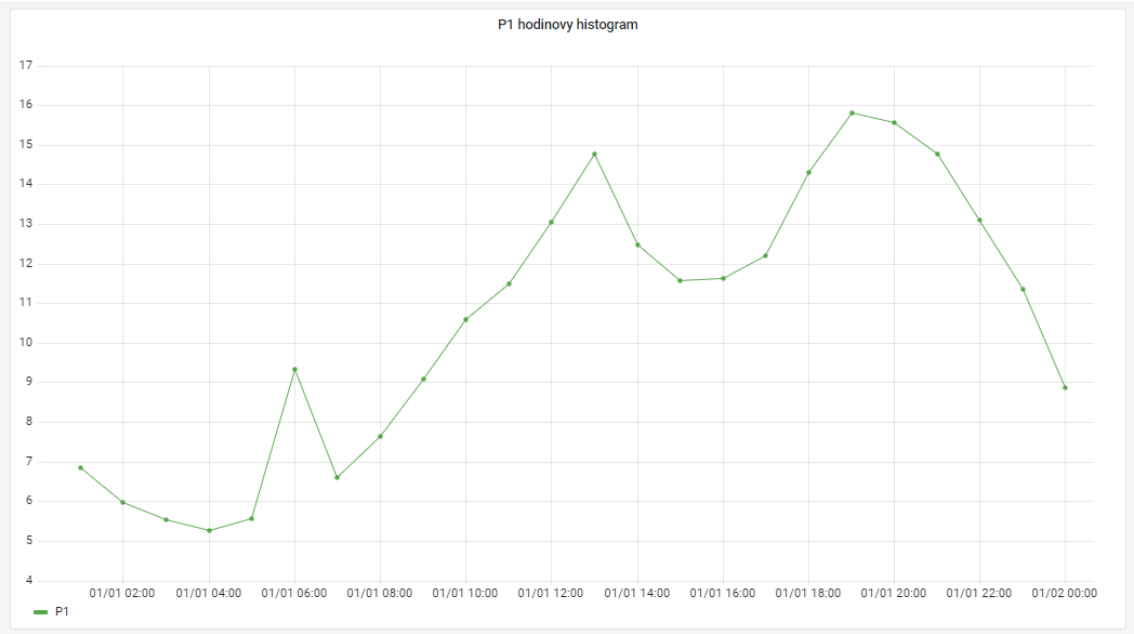
Obrázek 6.6: YAML soubor se souřadnicemi

```
mongoimport --host=localhost --port 27017 --username mongoAdmin
--password heslo --authenticationDatabase admin
--type tsv --file /tmp/sensor_temperatures.tsv
--headerline --columnsHaveTypes --db mongodb
```

Obrázek 6.7: Import dat do databáze MongoDB

```
2022-04-05T11:02:12.594+0200    no collection specified
2022-04-05T11:02:12.594+0200    using filename 'geo_temperatures' as collection
2022-04-05T11:02:12.640+0200    connected to: mongodb://localhost:27017/
2022-04-05T11:02:15.644+0200    [#####...................] mongodb.geo_temperatures     5.17MB/21.6MB (23.9%)
2022-04-05T11:02:18.642+0200    [###########.............] mongodb.geo_temperatures     10.6MB/21.6MB (48.9%)
2022-04-05T11:02:21.649+0200    [############............] mongodb.geo_temperatures     11.8MB/21.6MB (54.4%)
2022-04-05T11:02:24.647+0200    [#############...........] mongodb.geo_temperatures     12.6MB/21.6MB (58.3%)
2022-04-05T11:02:27.643+0200    [################........] mongodb.geo_temperatures     13.6MB/21.6MB (62.7%)
2022-04-05T11:02:30.642+0200    [################........] mongodb.geo_temperatures     14.6MB/21.6MB (67.5%)
2022-04-05T11:02:33.642+0200    [###################....] mongodb.geo_temperatures     18.6MB/21.6MB (86.1%)
2022-04-05T11:02:35.827+0200    [########################] mongodb.geo_temperatures     21.6MB/21.6MB (100.0%)
2022-04-05T11:02:35.827+0200    340018 document(s) imported successfully. 0 document(s) failed to import.
```

Obrázek 6.8: MongoDB - import dat



Obrázek 6.9: Grafana - hodinový histogram

Použitá literatura

[1] *Co to je IoT?* [Online]. Praha: České Radiokomunikace, 2020 [cit. 2022-05-02]. Dostupné z: https://www.iotport.cz/iot-novinky/ostatni-clanky-o-iot/co-to-je-iot.

[2] *Top IoT Communication Protocols Updated 2021 [ZigBee, NFC, And More]* [online]. Noida (Indie): Hashstudioz, 2021 [cit. 2022-05-02]. Dostupné z: https://hashstudioz.com/blog/top-iot-communication-protocols-2020/.

[3] GREGERSEN, CARSTEN. *A Complete Guide to IoT Protocols & Standards In 2021* [online]. Aarhus (Dánsko): Nabto, 2020 [cit. 2022-05-02]. Dostupné z: https://www.nabto.com/guide-iot-protocols-standards/.

[4] MALÝ, Martin. *Protokol MQTT: komunikační standard pro IoT* [online]. Praha: Internet Info, 2016 [cit. 2022-05-02]. Dostupné z: https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/.

[5] SHELBY, Zach, Klaus HARTKE a Carsten BORMANN. *The Constrained Application Protocol (CoAP)* [online]. Brémy: Universitaet Bremen TZI, 2014 [cit. 2022-05-02]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc7252.html.

[6] SOUSA, Tiago et al. Engineering Software for the Cloud: Messaging Systems and Logging. In: 2017, s. 1–14. Dostupné z DOI: 10.1145/3147704.3147720.

[7] *MQTT: The Standard for IoT Messaging* [online]. -: MQTT.org, - [cit. 2022-05-10]. Dostupné z: https://mqtt.org/.

[8] MISHRA, Biswajeeban a Attila KERTESZ. The Use of MQTT in M2M and IoT Systems: A Survey. *IEEE Access*. 2020, roč. 8, s. 201071–201086. Dostupné z DOI: 10.1109/ACCESS.2020.3035849.

[9] *Co je MQTT a k čemu slouží ve IIoT? Popis protokolu MQTT* [online]. Praha: IPC2U, - [cit. 2022-05-03]. Dostupné z: https://ipc2u.cz/blogs/news/mqtt-protokol?gclid=Cj0KCQiAuvOPBhDXARIsAKzLQ8HzMVshpLn5s9nCRXyaasvtNKXa9LlfFb083_iCLgj2OLsQt6Sqv_AaAnMBEALw_wcB.

[10] CRESSLER, Cosette. *AMQP vs MQTT: Comparing Instant Messaging Protocols* [online]. Denver: CometChat, 2021 [cit. 2022-05-02]. Dostupné z: https://www.cometchat.com/blog/amqp-vs-mqtt-comparing-instant-messaging-protocols.

[11] *SIGFOX – princip, struktura, protokol, použití* [online]. -: IoT portál, 2017 [cit. 2022-05-02]. Dostupné z: https://www.iot-portal.cz/2017/05/29/sigfox-princip-struktura-protokol-pouziti/.

[12] *Mapy pokrytí* [online]. -: IoT portál, 2020 [cit. 2022-05-02]. Dostupné z: https://www.iot-portal.cz/mapa-pokryti/.

[13] *NarrowBand IoT* [online]. -: IoT portál, 2016 [cit. 2022-05-02]. Dostupné z: https://www.iot-portal.cz/2016/04/30/narrowband-iot/.

[14] MATHIAS, Selvine G., Sebastian SCHMIED a Daniel GROSSMANN. An Investigation on Database Connections in OPC UA Applications. *Procedia Computer Science* [online]. 2020, roč. 170, s. 602–609 [cit. 2022-05-03]. ISSN 18770509. Dostupné z DOI: 10.1016/j.procs.2020.03.132.

[15] RAUTMARE, Sharvari a D. M. BHALERAO. MySQL and NoSQL database comparison for IoT application. In: *2016 IEEE International Conference on Advances in Computer Applications (ICACA)*. 2016, s. 235–238. Dostupné z DOI: 10.1109/ICACA.2016.7887957.

[16] MŮČKA, Jan. *Relační databáze vs. nerelační databáze: jaké jsou mezi nimi rozdíly?* [Online]. Brno: MasterDC, 2021 [cit. 2022-05-02]. Dostupné z: https://www.master.cz/blog/relacni-databaze-nerelacni-databaze-jake-jsou-rozdily/.

[17] MARJANI, Mohsen et al. Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges. *IEEE Access*. 2017, roč. 5, s. 5247–5261. Dostupné z DOI: 10.1109/ACCESS.2017.2689040.

[18] *Infrared Temperature Sensors* [online]. Brookfield (USA): IOThrifty, - [cit. 2022-05-02]. Dostupné z: https://www.iothrifty.com/collections/ir-sensors.

[19] SEHRAWAT, Deepti a Nasib Singh GILL. Smart Sensors: Analysis of Different Types of IoT Sensors. In: *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. 2019, s. 523–528. Dostupné z DOI: 10.1109/ICOEI.2019.8862778.

[20] EDWARDS, Ed. *Different Types of Internet of Things (IoT) Sensors* [online]. New York: Thomas Publishing Company [cit. 2022-05-02]. Dostupné z: https://www.thomasnet.com/articles/instruments-controls/types-of-internet-of-things-iot-sensors/.

[21] PELAEZ, AGUSTIN. *12 IoT Sensor Types To Keep An Eye On [With Examples]* [online]. Doral (USA): Ubidots, 2020 [cit. 2022-05-02]. Dostupné z: https://ubidots.com/blog/iot-sensor-types/.

[22] BASHARI RAD, Babak, Harrison BHATTI a Mohammad AHMADI. An Introduction to Docker and Analysis of its Performance. *IJCSNS International Journal of Computer Science and Network Security*. 2017, roč. 173, s. 8.

[23] MILINKOVICH, Mike. *The 2019 IoT Developer Survey Results are Live* [online]. -: Eclipse foundation, 2019 [cit. 2022-05-03]. Dostupné z: https://eclipse-foundation.blog/2019/04/17/2019-iot-developer-survey/.

[24] MUHAMMED, Aina'u Shehu a Derya UCUZ. Comparison of the IoT Platform Vendors, Microsoft Azure, Amazon Web Services, and Google Cloud, from Users' Perspectives. In: *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*. 2020, s. 1–4. Dostupné z DOI: 10.1109/ISDFS49300.2020.9116254.

[25] ASIMINIDIS, Christodoulos, George KOKKONIS a S. KONTOGIANNIS. Database Systems Performance Evaluation for IoT Applications. *SSRN Electronic Journal*. 2018, roč. 10. Dostupné z DOI: 10.2139/ssrn.3360886.

[26] *Úvod do JSON* [online]. -: JSON, - [cit. 2022-05-02]. Dostupné z: https://www.json.org/json-cz.html.

[27] *ISO 50001 - ENERGETICKÝ MANAGEMENT* [online]. Zlín: Institut pro testování a certifikaci, - [cit. 2022-05-02]. Dostupné z: https://www.itczlin.cz/certifikace-systemu-rizeni-itc/iso-50001-energeticky-management.