Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Realizace systému řízení inteligentní budovy v prostředí cloudových služeb

Michal Budík

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Michal Budík

Systémové inženýrství a informatika
Informatika

Název práce

Realizace systému řízení inteligentní budovy v prostředí cloudových služeb

Název anglicky

Realization of intelligent building management system in Cloud environment services

---

Cíle práce

Cílem bakalářské práce je navrhnout a implementovat systém pro řízení inteligentní budovy s maximálním využitím služeb cloudového prostředí.

Práce bude obsahovat analýzu návrhu architektury aplikace a její implementaci v cloudovém prostředí. V analytické části bude vyhodnoceno jaké cloudové služby použít, jak budou řízeny a jakým způsobem si budou předávat data. Součástí práce bude také vytvoření aplikační logiky pro následné vyhodnocování dat. Při tvorbě systému i aplikační logiky využijeme nabízených funkcí z katalogu cloudu (např. vytvoření grafů z naměřených dat).

Vývoj a nasazení aplikace bude podporován službami DevOps pro automatizaci CD (Continuous development) a CI (Continuous integration).

Metodika

Na začátku je nutné vybrat vhodné cloudové prostředí, na kterém bude aplikace spuštěna. Rozhodujícími faktory budou například služby v rámci bezplatné licence, velikost datového uložiště cloudu a cena následného provozu.

Poté je nutné vytvořit lokální vývojové prostředí v počítači například s využitím aplikace MiniKube. V lokálním vývojovém prostředí budeme vyvíjet jednotlivé moduly aplikace. Následně hotovou aplikaci vložíme do cloudového prostředí a připojíme na něj sensory schopné posílat data do cloudu.

Sestavení aplikace bude probíhat automaticky procesy CD/CI.

Doporučený rozsah práce

35-40 stran

Klíčová slova

Cloud, inteligentní budova, DevOps

Doporučené zdroje informací

Autor: Bill Laberis, Název: ”What Is the Cloud?”, Vydavatel: O’Reilly Media, Inc., ISBN: 9781492052906
Autor: Duncan C. E. Winn, Název: ”Cloud Foundry: The Definitive Guide, 1st Edition”, Vydavatel: O’Reilly Media, Inc., ISBN: 9781491932438
Autor: John J. Geewax, Název: ”Google Cloud Platform in Action”, Vydavatel: Google Cloud Platform in Action, ISBN: 9781617293528
Autor: Marko Lukša, Název: ”Kubernetes in Action”, Vydavatel: Manning Publications, ISBN: 9781617293726
Autor: Sam Newman, Název: ”Building Microservices”, Vydavatel: O’Reilly Media, Inc., ISBN: 9781491950357
Autor: Victor Farcic, Název: ”The DevOps 2.1 Toolkit: Docker Swarm”, Vydavatel: Packt Publishing, ISBN: 9781787289703

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 23. 2. 2021

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 23. 2. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 13. 03. 2021

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Realizace systému řízení inteligentní budovy v prostředí cloudových služeb" jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.3.2021 _____

Poděkování

Rád bych touto cestou poděkoval svému vedoucímu bakalářské práce Ing. Jiří Brožkovi, Ph.D., za ochotnou pomoc a odborné vedení při zpracování práce. Dále bych chtěl poděkovat své rodině za velkou míru trpělivosti a podpory.

Realizace systému řízení inteligentní budovy v prostředí cloudových služeb

Abstrakt

Tato bakalářská práce se zabývá realizací ukázky systému inteligentní budovy s využitím služeb veřejného cloudu. V úvodní teoretické části je čtenář seznámen s problematikou cloudů, IoT a inteligentních budov.

Praktická část bakalářské práce se zabývá návrhem a realizací systémů inteligentní budovy. Jsou popsány postupy vytvoření jednotlivých modulů systému inteligentní budovy a jsou detailně popsány všechny důležité části projektu.

Klíčová slova: Cloud, IoT, Inteligentní budova, Django, Python

Realization of intelligent building management system in Cloud environment services

Abstract

The bachelor thesis deals with the implementation of demonstration systems of intelligent buildings using public cloud services. In the introductory theoretical part, the reader is acquainted with issues of clouds, IoT and intelligent buildings.

The practical part of the bachelor thesis deals with the design and implementation of intelligent building systems. Procedures for creating individual modules of an intelligent building system are described and all important parts of the project are described in detail.

Keywords: Cloud, IoT, Intelligent building, Django, Python

Obsah

Seznam obrázků

1 Úvod

Cloudové služby a cloudová řešení se dnes stala novým způsobem vývoje a provozování aplikací. Umožňují dynamický vývoj aplikací a jejich velmi rychlé přizpůsobení na aktuální požadavky uživatelů. Proto dnes aplikace tyto technologie velmi často používají.

V práci jsou ukázány možnosti využití služeb cloudového prostředí ve spojení se systémy inteligentní budovy. Spojení technologie IoT a cloudů přináší do problematiky inteligentních budov nové možnosti a výrazné úspory nákladů.

V práci jsou představeny klíčové technologie pro vývoj v cloudovém prostředí a je realizována ukázka jednoduchého systému inteligentní budovy, který využívá služeb veřejného cloudu.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem bakalářské práce je navrhnout a implementovat systém pro řízení inteligentní budovy s maximálním využitím služeb cloudového prostředí.

Práce obsahuje analýzu návrhu architektury aplikace a její implementaci v cloudovém prostředí. V analytické části se vyhodnocuje, jaké cloudové služby použít, jak jsou řízeny a jakým způsobem si předávají data. Součástí práce je také vytvoření aplikační logiky pro následné vyhodnocování dat. Při tvorbě systému i aplikační logiky jsou využívány nabízené funkce z katalogu cloudu (např. vytvoření grafů z naměřených dat).

Vývoj a nasazení aplikace je podporován službami DevOps pro automatizaci CI (Continuous Integration) a CD (Continuous Development).

2.2 Metodika

Na začátku je nutné vybrat vhodné cloudové prostředí, na kterém je aplikace spuštěna. Rozhodujícími faktory jsou např. služby v rámci bezplatné licence, velikost datového uložiště cloudu a cena následného provozu.

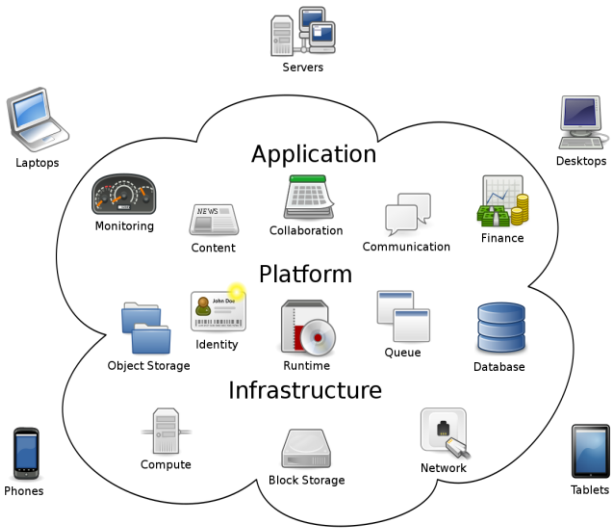
Po výběru cloudového prostředí je potřeba vytvořit lokální vývojové prostředí v počítači např. s využitím aplikace MiniKube. V lokálním vývojovém prostředí jsou vyvíjeny jednotlivé moduly aplikace. Následně je hotová aplikace vložena do cloudového prostředí a jsou na ni připojeny senzory schopné posílat data do cloudu.

Sestavení aplikace probíhá automaticky pomocí procesů CI/CD.

3 Teoretická východiska

3.1 Cloud

Cloud je synonymem slova internet, kterým je označena celosvětová síť serverů, na kterých běží služby. Tyto služby jsou poskytovány cloudovými poskytovateli jako Amazon, Google nebo Microsoft v rámci modelu Cloud computing. Uživatelé, tedy jednotlivci nebo firmy, se k těmto serverům mohou připojit téměř kdekoliv na světě a také pomocí mnoha různých zařízení, kterými může být např. počítač, tablet nebo mobilní telefon.



Obrázek 1 Cloud computing [1]

Tyto služby jsou poskytovány buď zdarma nebo jsou prodávány na měsíční bázi podle toho, jaké prostředky a jaké množství jich bylo uživatelem použito. Může se jednat o služby cloudového uložiště (např. Google Drive nebo Microsoft OneDrive), webové aplikace (např. Facebook), nebo specializované služby určené pro sdílení souborů (např. Instagram pro sdílení fotografií) a mnohé další. [2] [3]

Pro koncového uživatele není důležité vědět, jaký hardware používá konkrétní server, na kterém služba běží, jaký používá operační systém nebo kde se tento server nachází. Pro

koncového uživatele je důležitá jen informace, zda služba funguje podle očekávání. Tento základní koncept cloudových služeb je nazýván computing-as-a-service. [4]

3.1.1 Typy cloudů (obecně)

Cloudy jsou rozděleny dle jejich modelu nasazení a dle distribučního modelu.

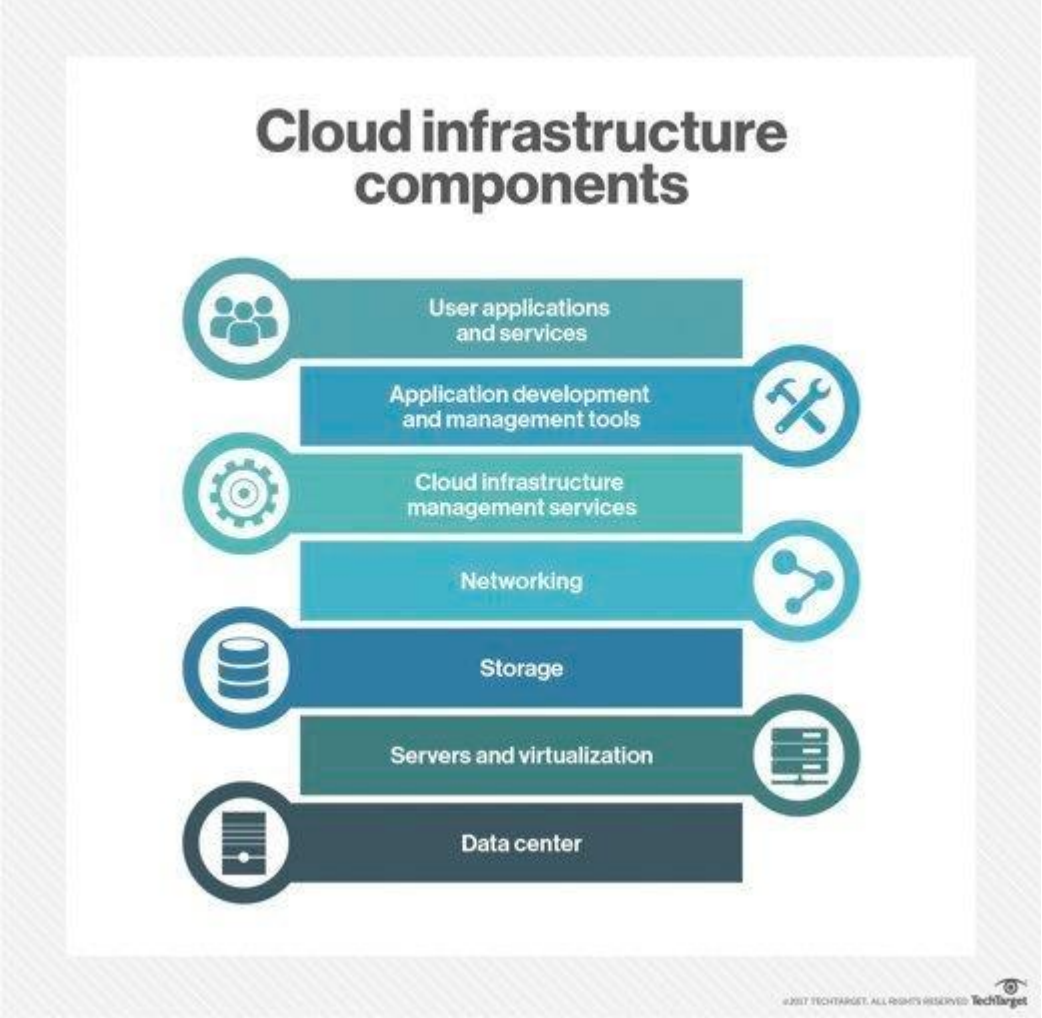
3.1.1.1 Model nasazení

Pomocí modelu nasazení je určeno, zda je cloud veřejný, privátní nebo hybridní.

3.1.1.1.1 Veřejné cloudy

Veřejné cloudy jsou cloudy, které jsou dostupné veřejně na internetu a jsou spravovány nějakou třetí stranou. V mnoha případech se jedná o nadnárodní soukromé společnosti jako Google, Amazon nebo Microsoft. Tyto velké společnosti vlastní servery a vysokorychlostní síťovou infrastrukturu rozmístěnou po celém světě, pomocí které je zajištěno, že jejich služby jsou celosvětově dostupné. [4] [5]

Tyto společnosti také vlastní téměř neomezené zdroje jako datové uložiště a výpočetní výkon, které jsou uživatelům pronajímány. Na jednom serveru tohoto cloudového poskytovatele najednou běží více softwarů různých uživatelů. Software a data různých uživatelů jsou od sebe vzájemně oddělena, ačkoliv sdílí např. stejnou databázi nebo stejný virtuální počítač. [4] [6]



Obrázek 2 Komponenty cloudové infrastruktury [7]

Mezi hlavní výhody veřejných cloudů patří jejich téměř neomezená škálovatelnost, adaptibilita a spolehlivost. Pro tyto společnosti není žádný problém navýšit tyto zdroje, pokud se po nich vyskytne vysoká poptávka. V případě, kdy je jedno z datových center vyřazeno např. působením přírodní katastrofy, je okamžitě nahrazeno jiným datovým centrem. Je tak omezeno riziko celkového selhání služby.

Mezi další výhody patří cenová dostupnost těchto cloudů. Malá začínající společnost si často nemůže dovolit vlastnit svoji serverovou infrastrukturu. Je pro ni výhodnější si tuto infrastrukturu pronajmout od poskytovatele, který zařídí údržbu serverů, jeho zabezpečení a umožní také této malé společnosti operovat mezinárodně, protože data společnosti umístěná na serverech cloudového poskytovatele jsou celosvětově dostupná. [3]

Jednou z největších a dlouhodobých nevýhod veřejných cloudů je bezpečnost dat, která jsou na nich uložena. Veřejné cloudy jsou jedním z největších cílů hackerů po celém světě. Cloudoví poskytovatelé investují biliony do vylepšování zabezpečení proti kyberútokům, a proto je velice složité pro hackery se do jejich systémů dostat. Společnost Gartner např. tvrdí, že nejvyšší hrozbou bezpečnosti pro cloudy není jejich poskytovatel, ale jejich uživatel. Pokud nejsou dostatečně zabezpečeny přihlašovací údaje uživatele, pak může hacker tuto slabinu využít. Je tedy z pohledu zabezpečení důležité, jak jsou cloudové služby uživatelem využívány. [8] [9]

Některé firmy a instituce ani z právního hlediska svoje data a aplikace umístit na veřejný cloud nemohou. Jde často o státní instituce (např. finanční úřad, registr zbraní). Pro tyto instituce je vhodné mít data uložena na vlastních serverech nebo privátním cloudu a přes zabezpečenou síťovou komunikaci tato zašifrovaná data posílat na front-end, který je vytvořen na veřejném cloudu.

3.1.1.1.2 Soukromé cloudy

Soukromé nebo také privátní cloudy jsou cloudy vytvářené pro účely jedné konkrétní firmy nebo instituce. Na rozdíl od veřejných cloudů má ale tato společnost veškerá svoje data umístěna na svých serverech nebo serverech, které jsou spravované třetí stranou. Tyto servery jsou vždy plně dedikovány pro danou firmu, která si je pronajala.

Účelem soukromého cloudu je eliminace problému s bezpečností dat na veřejných cloudech. Pro evropské firmy, které musí dodržovat zákony ohledně datového soukromý (GDPR) toto představuje optimální řešení, protože tyto firmy musí vědět, kde se jejich data nachází a jak jsou zabezpečená. [4] [10]

Společnost nebo instituce má také pod úplnou kontrolou, jaký software a hardware je na jejich cloudu používán. Není omezena tím, co jí cloudový poskytovatel nabídne. Je tak možné mnohem efektivněji nasazovat aplikace na soukromý cloud než na veřejný a jeho zabezpečení je také jednodušší, jelikož běží za jednotným firewallem, který je většinou umístěn u uživatele. [11]

Ačkoliv tyto cloudy nabízí mnohem vyšší bezpečnost a kontrolu, počáteční cena jejich zřízení je mnohem vyšší než u veřejných cloudů. Do ceny je započítána jak nákupní cena, tak cena za údržbu. [11]

3.1.1.1.3 Hybridní cloudy

Hybridní cloud je cloudové řešení, které umožňuje využít výhody veřejného cloudu, ale umožňuje zmírnit jeho nevýhody, hlavně z hlediska zabezpečení dat. Toto řešení spojuje veřejné a soukromé cloudy v jednu infrastrukturu s různými prostředími. Uživatel je tak umožněno např. ponechat svoje soukromá data na datových centrech soukromého cloudu, ale služby, které nepotřebuje mít takto zabezpečené, může migrovat na veřejný cloud a tím zlevnit své celkové náklady. [4] [10] [11]

Toto řešení je dnes používáno mnoha společnostmi, protože umožňuje využít flexibility veřejných cloudů a zároveň poskytuje zabezpečení dat na úrovni soukromých cloudů, u kterých jsou splněny bezpečnostní a právní předpisy.

V roce 2018 bylo pomocí dotazníku od firmy IDC zjištěno, že z 400 dotázaných firem jich cca 80 % migruje svoje služby z veřejného cloudu na soukromý cloud nebo na svoji vlastní infrastrukturu. [12]

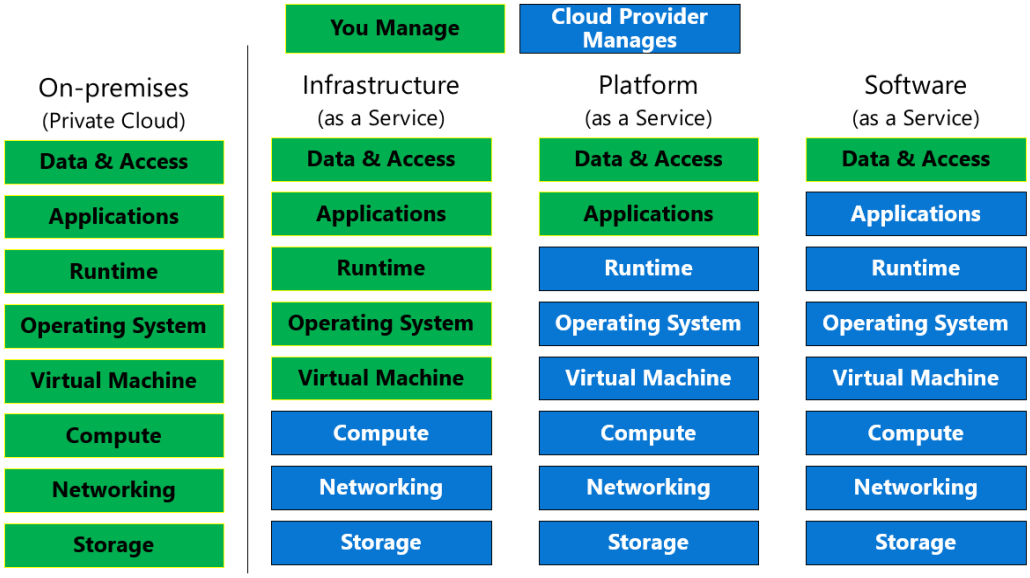
Samozřejmě toto řešení nenabízí tak vysokou bezpečnost a kontrolu jako soukromé cloudy a ani tak vysokou škálovatelnost jako u cloudů veřejných.

3.1.1.1.4 Komunitní cloudy

Komunitní cloud je méně známou variantou soukromého cloudu. Umožňuje více firmám nebo institucí s jednotným zájmem nebo bezpečnostní politikou sdílet jednu cloudovou infrastrukturu. Tato infrastruktura je spravována buď členem komunity nebo třetí stranou. Toto řešení je optimální např. pro státní instituce, které mohou využívat jedno cloudové prostředí placené státem a je tak umožněno snížit cenu infrastruktury, její údržby a zabezpečení. Dále by mohlo vést ke snížení byrokracie např. při vyřizovaní stavebního povolení by člověk nemusel zdlouhavě chodit mezi úřady, ale podal by jednu žádost a všechna potřebná data by byla získána z jednoho komunitního cloudu. [13] [14]

3.1.1.2 Distribuční model

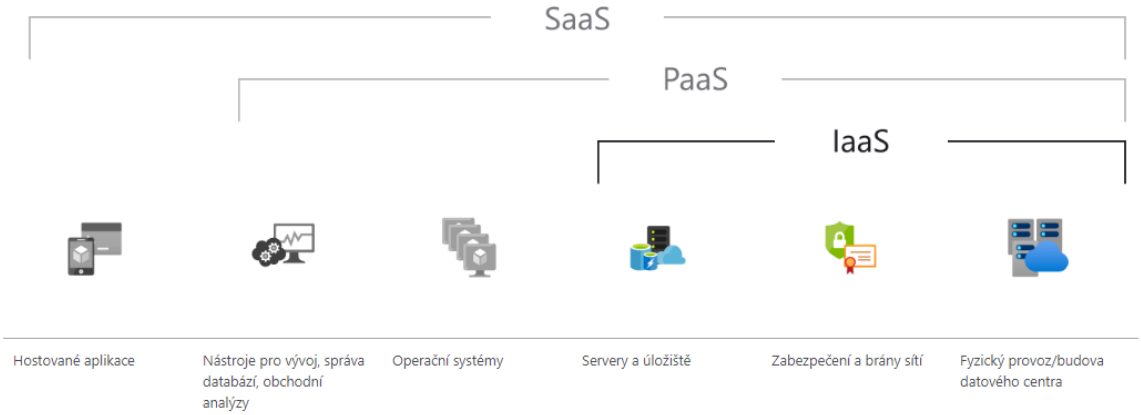
Pomocí distribučního modelu je určeno, jaké typy služeb (infrastrukturu, hardware a software) jsou cloudem nabízeny. Z *obrázku 3* je viditelné, že u soukromých cloudů jsou všechny vrstvy zajišťovány samotnou firmou, ale u dalších typů cloudů je poskytovatelem cloudu zajišťováno čím dál více vrstev.



Obrázek 3 Cloudové distribuční modely [15]

3.1.1.2.1 IaaS (Infrastructure as a service – Infrastruktura jako služba)

V IaaS je uživatelům poskytována infrastruktura cloudového prostředí jako fyzické/virtuální servery, bezpečnostní firewally, datová úložiště a procesory zpracovávající procesy uživatelových aplikací. Všechny tyto služby jsou uživateli pronajímány za určitý finanční obnos. [4] [16]



Obrázek 4 IaaS [16]

IaaS lze použít pro vývoj a testování softwaru, pro zálohování dat a vytvoření virtuálních počítačů. IaaS je nejčastěji používán společnostmi, pro jednotlivce nemusí být oproti PaaS nebo SaaS cenově optimální, protože mnoho částí cloudu je stále zajišťovány samotným uživatelem a jsou tak na něj kladeny vyšší nároky z pohledu technické expertízy. [16] [17]

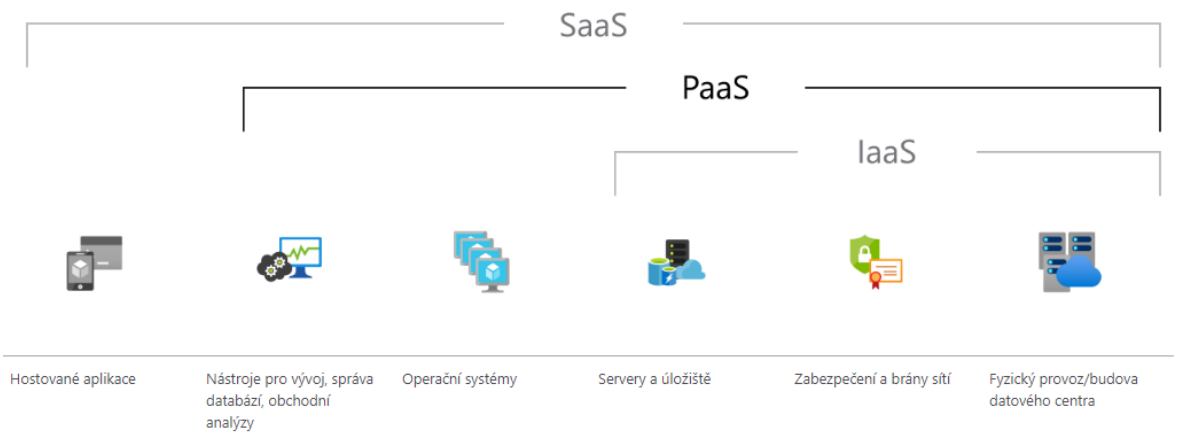
Mezi poskytovatele těchto cloudových služeb patří:

- Amazon AWS;
- Microsoft Azure;
- Google Cloud Platform (GCP);
- IBM cloud.

3.1.1.2.2 PaaS (Platform as a Service – Platforma jako služba)

PaaS je zatím nejmenší z cloudových modelů dle ročního příjmu, ačkoliv v knize „*What is Cloud*“ od Billa Laberise je zmíněno, že dle bezpečnostních expertů bude tento typ cloudu v budoucnosti velmi důležitý a velmi rychle rostoucí. [4]

V PaaS je poskytovatelem zajišťována infrastruktura a prostředí, ve kterém jsou aplikace vyvíjeny. Pro uživatele je tak přednastaven operační systém, databázové systémy a systémy jejich správy, nebo také vývojová prostředí. [3] [18]



Obrázek 5 PaaS [18]

Do konfigurace prostředí nemůže uživatel zasahovat. V PaaS jsou též omezeny typy prostředí, které je možné použít. Jednotlivé platformy tak nemusí např. podporovat některé programovací jazyky. Protože ale v PaaS nemusí být prostředí zajišťováno uživatelem, tak

je celkově oproti IaaS bezpečnější a cenově mnohem dostupnější, a proto je populární např. mezi vývojáři webových aplikací. [4] [17] [18]

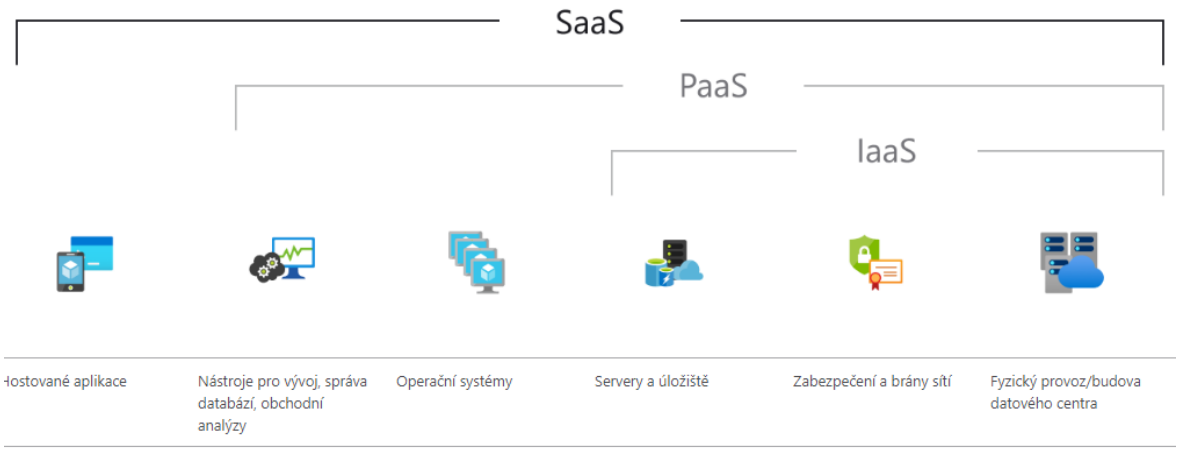
Mezi hlavní platformy PaaS patří:

- Amazon AWS Elastic Beanstalk;
- Microsoft Azure;
- Heroku;
- Google App Engine.

3.1.1.2.3 SaaS (Software as a Service – Software jako služba)

SaaS je dnes nejvíce používanou formou cloudové služby. Tyto cloudové služby zahrnují aplikace jako Gmail, Google Drive, Office 365 a mnohé další aplikace, které jsou používány miliony uživateli po celém světě. Např. aplikace Gmail byla v roce 2018 aktivně používána 1,5 miliony uživatelů. [5] [19] [20]

SaaS umožňuje uživateli používat již vytvořenou aplikaci, která celá běží na cloudovém prostředí. Veškerá infrastruktura, vývojové prostředí, data a kód aplikace je zajištěna poskytovatelem cloudu. Uživatel tyto aplikace jen používá většinou přes svůj webový prohlížeč. [4] [19]



Obrázek 6 SaaS [19]

V SaaS většinou nejsou na uživatele kladeny téměř žádné technické nároky k tomu, aby s aplikací pracoval. Použití těchto aplikací je velmi často bezplatné. Jsou také velmi dobře

zabezpečené, jelikož zabezpečení je zajištěno přímo cloudovým poskytovatelem. Samozřejmě ze strany uživatele není možné aplikace modifikovat. [17]

Tyto cloudové služby jsou používány jak jednotlivci, tak i společnostmi, pro které je mnohem levnější a rychlejší koupit tyto hotové produkty než za mnohem větší finanční obnos vyvíjet vlastní. Mnoho těchto aplikací je dnes možno spustit z jakéhokoliv běžně dostupného zařízení, ať už se jedná o mobil, počítač nebo tablet. [4]

3.1.2 Cloud storage vs Cloud computing

Ačkoliv jsou termíny Cloud storage a Cloud computing často zaměňovány, nemají stejný význam. Tyto dvě služby mají určité věci společné např. nutné připojení k internetu pro správnou funkci. Obě tyto služby jsou odolné vůči náhlým selháním, jelikož jsou obě zálohované ve více datových centrech a obě jsou nasazeny na systémech cloudového poskytovatele, který se stará o jejich údržbu. I přesto jsou v těchto službách rozdíly.

3.1.2.1 Cloud storage

Cloud storage je cloudová služba umožňující uživateli uložit svá data na internetu do cloudového uložiště. Po jejich uložení mohou být stažena jakýmkoliv uživatelem, který má správný přístup. Jedná se tedy o jakoukoliv službu, na kterou uložíme svá data a tato data nejsou nijak dál transformována.

Tyto služby jako Google Drive a Microsoft OneDrive jsou používány k zálohování důležitých souborů a v dnešní době začínají nahrazovat fyzické HDD a SSD. Oproti nim mají totiž cloudová uložiště téměř neomezený úložní prostor a jsou lépe zabezpečena před ztrátou, jelikož jsou zálohována v několika datových centrech. [21]

Pro cloud storage software je vyžadován malý výpočetní výkon a velký úložní prostor, který je ale dnes relativně levnou komoditou, a proto např. na Google Drive je každému uživateli nabízeno uložení 15 GB dat zadarmo.

3.1.2.2 Cloud computing

Cloud computing je jakákoliv služba cloudu, která dále transformuje data uložená na cloud storage. Může se jednat např. o aplikaci jedoucí na cloudovém superpočítači, do které je vložen vstupní datový soubor využitý pro výpočet nějakého složitého problému. Jedná se také např. o aplikaci Office 365, ve které je možné editovat dokumenty a tabulky. Gmail nebo Microsoft Outlook je také forma cloud computingu, jelikož jsou v nich editována data ve formě e-mailů. [22]

V Cloud computingu je využíván primárně výpočetní výkon procesorů. Cloud computing a cloud storage jsou tedy propojeny tím, že data potřebná pro výpočty a jiné operace jsou ukládána na cloud storage a samotné výpočty jsou prováděny na cloud computingu.

3.1.3 Cloud native

Cloud native je způsob vývoje softwaru, kdy je k jeho vývoji maximálně využit cloud computing model pro vytvoření vysoce škálovatelných systémů, které jsou primárně provozovány na rychle se měnících prostředích veřejných, soukromých nebo privátních cloudů. Díky cloudům není důležité, na jakém místě je aplikace nasazena, ale jak je vytvářena a nasazena. [23] [24]

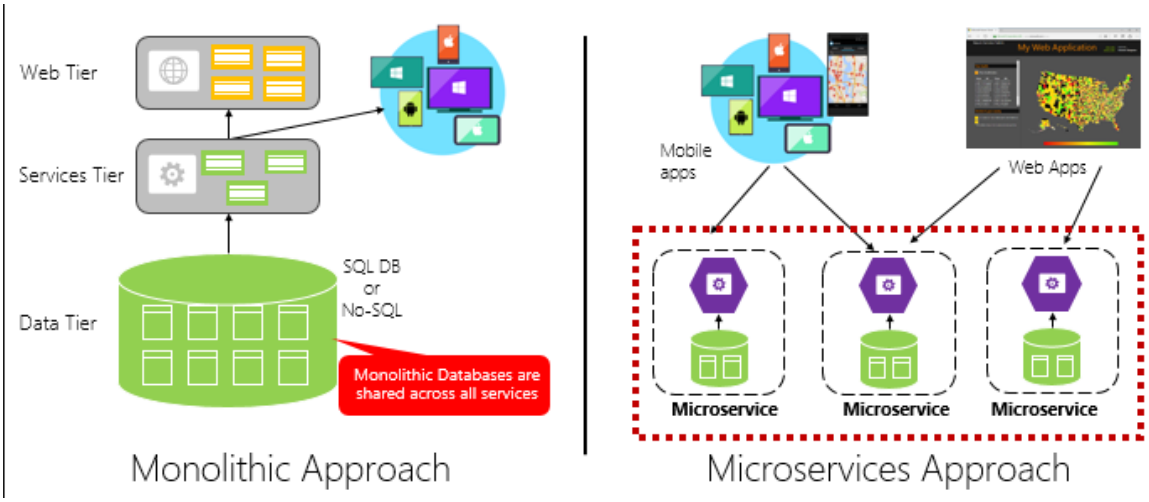
Cloud native ovlivňuje, jakým způsobem je vytvořen design systému a jeho implementace nasazením prvků jako DevOps, mikroslužby, kontejnery a kontejnerové řídící platformy.

3.1.3.1 Mikroslužby (*microservices)*

Aplikace jako taková je rozdělena na menší celky (mikroslužby), které tvoří jeden celkový systém aplikace. Každá mikroslužba je nějaká vrstva celého systému, kdy každá vrstva řeší určitou jeho část. Jedna mikroslužba může např. řešit GUI aplikace, druhá může řešit backend, třetí mikroslužba může řešit API systému, čtvrtá komunikaci s databází. Tyto jednotlivé vrstvy fungují nezávisle na sobě a mohou fungovat i po selhání jedné z vrstev, čímž dělají celkový systém vysoce odolný vůči případným selháním. [25] [26]

Každá mikroslužba je vytvářena v kontejnerech, ve kterých je zabalen a izolován kód aplikace, její konfigurace, knihovny a soubory od ostatních mikroslužeb. Kontejner je následně spouštěn ve virtualizovaném prostředí a stává se tak nezávislým na hardwaru, na

kterém běží. Oproti tzv. virtuálnímu stroji (*VM – virtual machine)* se kontejnery dají spouštět mnohem rychleji a jejich velikost je většinou mnohem menší (v řádech megabytů). Díky tomu je možné spustit mnoho instancí kontejnerů a není tolik plýtváno hardwarovými zdroji. [27] [28]



Obrázek 7 Mikroslužby [26]

Mikroslužby celého systému aplikace je možné vyvíjet nezávisle a s menším rizikem toho, že po změně jedné části systému přestane celý systém fungovat, což je u monolitických aplikací časté riziko.

Mikroslužby je možné škálovat jednotlivě. Pokud např. málo uživatelů používá API, je možné nechat běžet jen jednu mikroslužbu, která toto API provozuje. Pokud ale náhle API začne být více používáno, je možné spustit více instancí této mikroslužby, které náhlý nápor vybalancují. U monolitického systému jen nutné pro stejný efekt spustit více jeho instancí, a tak by bylo zbytečně využíváno více hardwarových zdrojů, než je potřeba. [26]

Monolitické aplikace se hodí pro jednodušší, méně komplexní systémy, jelikož je jednodušší vyvíjet, pokud jejich vývojáři používají určité standarty. Je také jednodušší je nasadit. Monolitické aplikace také lze často jednodušeji testovat. [29]

3.1.3.2 Řízení mikroslužeb

Pro aplikaci sestavenou z mikroslužeb musí existovat jiná aplikace nebo systém, který dokáže propojit jednotlivé mikroslužby s kontejnery do jednoho celku. Celkově tyto systémy slouží k automatizaci práce, která by jinak byla provedena uživatelem.

Mezi takové systémy patří např. kontejnerový orchestrační systém Kubernetes. Jejich účelem je automatizace nasazení, správa a škálování mikroslužeb, které jsou jimi řízeny. Obstarávají komunikaci mezi jednotlivými mikroslužbami a komunikaci mezi nimi a uživateli pomocí portů, na které jsou uživatelé připojeni. Umožňuje také balancovat přenos dat do mikroslužby a podle potřeby je škálovat. Mezi mikroslužbami jsou také balancovány hardwarové zdroje, aby fungovali co nejlépe. [26] [30]

Většinou se nepracuje s jedním zařízením, ale s clustery, ve kterých se nachází více zařízení. Mezi tyto zařízení jsou rozděleny jednotlivé mikroslužby a jejich kopie. Pokud tedy jedna mikroslužba selže, je okamžitě nahrazena jinou instancí stejné mikroslužby.[31]

Kuberentes je sice velice efektivní nástroj na řízení aplikací vytvořených z mikroslužeb, ale pro menší aplikace je lepší použít méně komplexní systémy jako např. Docker swarm.

Všechny tyto prvky jsou úzce propojeny s agilními metodikami, ve kterých je software vytvářen v cyklu sběru požadavků, analýzy, vývoje a následné zpětné vazby. [32]

3.1.4 Příklady využití cloudů

Netflix je jednou z firem, které cloudové služby využívají nejvíce. Dříve společnost dodávala DVD, v roce 2007 si uvědomila popularitu streamování videí po internetu. V roce 2006 začal Amazon investovat do serverové architektury, aby ji mohl následně pronajímat, a tak se stal prvním cloud providerem. Netflix si uvědomil výhody, které pronájem Amazon serverů nabízely, a začal do nich investovat.

Mnoho cloudových technologií a postupů bylo vyvinuto právě firmou Netflix, např. dnes využívá pronajatých systémů s mikroprocesory, na kterých běží umělá inteligence (AI), pomocí které jsou vyhodnocována data ze sledování seriálů a videí s využitím neurálních sítí a strojového učení. Ačkoliv je Netflix velmi bohatou firmou, není pro něj výhodné si

nutnou infrastrukturu pro toto AI budovat, a proto si ji pronajímá od firmy Amazon. [33] [34]

Cloudy je možné využít i na úrovni států v rámci jejich systému e-governmentu, ve kterém jsou zakomponovány všechny informační systémy daného státu. Dle souhrnné analytické zprávy vytvořené pracovní skupinou RVIS složené ze zástupců NÚKIB, Ministerstva vnitra a Ministerstva financí ČR by takový systém e-government cloud (eGC) byl postaven na několika cloudových řešení.

Nejkritičtější infrastruktura státu by byla zakomponována do privátního cloudu spravovaného státem v rámci Orgánu státní správy nebo státním podnikem, aby byla zajištěna jeho maximální bezpečnost. Infrastruktura, která je méně kritická, by následně byla postavena ve veřejném IaaS cloudu. Následně pokud by existoval IS s částmi, které mají různé bezpečnostní úrovně a požadavky, bylo by využito hybridní řešení, kdy např. datová část (backend) by byla spuštěna v soukromém cloudu státu a komunikační část (frontend) by byla spuštěna ve veřejném IaaS cloudu. V České republice je dnes provozováno cca 7500 informačních systémů mnohdy se stejnou funkcionalitou a technologickou infrastrukturou. Zbytečně je tak plýtváno finančními prostředky státu na jejich údržbu a zabezpečování. Pomocí výše popsaného cloudové řešení by mohli být sníženy celkové náklady o 10 % až 50 %. Podobné cloudové řešení mají již např. v zemích jako Velká Británie na serveru GOV.UK. [35]

Cloudová řešení jsou také velmi využívána jednotlivci, ačkoliv si to ani mnohdy neuvědomují. Jak již bylo popsáno v sekci č. *3.1.1.2.3 SaaS*, služby jako Gmail, Google Drive a Office 365 jsou využívány miliony uživateli po celém světě díky tomu, že tyto aplikace fungují ve veřejných cloudech.

3.2 DevOps

DevOps je možné popsat jako množinu postupů, které slouží k automatizaci a integraci procesů mezi vývojem softwaru a IT týmy za účelem rychlejšího a častějšího sestavování, testování a vydávání softwaru. Jde tedy o snahu automatizovat procesy používané při vývoji softwaru a zefektivnit spolupráci a komunikaci jednotlivých týmů, které se na jeho vývoji podílí. Z důvodu špatné komunikace a lidské chyby totiž často dochází k

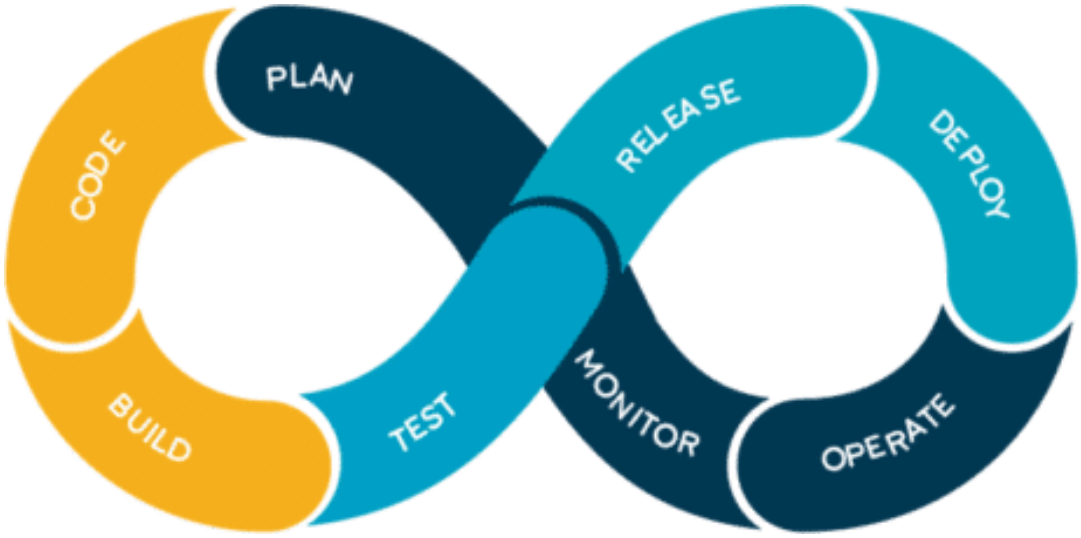
problémům, kterým lze právě s využitím automatizace a principů DevOps předejít. V posledních letech se DevOps stává standardem ve většině středních a velkých softwarových projektů. [36]

Dle článku od společnosti Amazon [37] mezi největší výhody zavedení DevOps patří:

- zrychlení vývoje;
- vyšší frekvence vydání nových verzí programu;
- vyšší spolehlivost kvůli automatizaci a automatickému testování;
- zefektivnění spolupráce mezi týmy.

3.2.1 Životní cyklus

Životní cyklus DevOps je grafická reprezentace způsobu, jakým funguje celý vývoj daného softwaru. Konkrétní životní cyklus je specifický pro daný software, existuje tak mnoho různých verzí.



Obrázek 8 Ukázka životního cyklu [38]

Obrázek 8 je velmi častou reprezentací životního cyklu pomocí diagramu, kde jednotlivé části jsou:

1. Code

- Vývojáři naprogramují funkcionalitu.
- Jde o první krok v životním cyklu.

2. Build
    - Vývojář zabalí novou verzi program do nasaditelného balíčku.
3. Test
    - Automatické spuštění testů nad novou verzí programu;
4. Release
    - Proces plánování nasazení nové verze programu.
5. Deploy
    - Příprava spustitelného balíčku nové verze programu a jeho nasazení na server.
6. Operate
    - Spuštění programu na serveru, kde je použit klienty.
7. Monitor
    - Fáze získávání informací, podle kterých je určována funkčnost programu při použití klienty.
8. Plan
    - Fáze zhodnocení získaných informací a naplánování změn pro vylepšení programu

3.2.2 Continuous Integration (CI)

Průběžná integrace (Continuous Integration) je praktika, jejíž cílem je urychlit a otestovat integraci změn, které jsou zavedeny v programu jednotlivými vývojáři. Změny jsou posílány do centrálního úložiště (např. Git), následně je aplikace automaticky sestavena a otestována. Kromě jednotkových a integračních testů lze spustit i např. testy kvality kódu. Účelem CI je tedy rychlé nalezení chyb vzniklých při vytváření změn, vylepšení kvality softwaru a snížení času, který je třeba pro validaci a nasazení softwarového updatu. Pokud dojde k chybám, je tým zodpovědný za jejich opravu upozorněn. [39]

3.2.3 Continuous Delivery/Deployment (CD)

Průběžné dodávání (Continuous Delivery) je praktika, ve které jsou změny v kódu automaticky připraveny pro nasazení do produkce. Jde o rozšíření nad CI, které umožňuje automaticky provést další typy testů (např. UI testy, další integrační testy…). Účelem CD je tedy rychle a spolehlivě vytvořit spustitelnou verzi upraveného programu.

Nejdříve musí být úspěšně provedeno CI, následně je pomocí CD nasazena změna na testovací prostředí. Zde dochází k automatickému testování a/nebo testování s pomocí konečných uživatelů. Při Continuous Deployment je po úspěšném otestování změna nasazena do produkce automaticky, v případě Continuous Delivery musí být nasazení do produkce povoleno. [40]

3.3 IoT (Internet věcí)

IoT je síť zařízení, které se propojí pomocí internetu a vzájemně přes něj komunikují. Tato zařízení můžou být malé jednoduché senzory, které zaznamenávají data jako např. vlhkost, teplota. Může se ale také jednat o velice složitá zařízení jako např. samořiditelná auta nebo celý systém inteligentních budov.

Základním cílem IoT je připojit ta zařízení, která ještě nejsou připojená k internetu, aby mohla komunikovat s lidmi a ostatními zařízeními. Ve 20. století počítače spoléhaly hlavně na uživatele, který musel zadat data pomocí klávesnice nebo čárového kódu do počítače, na základě nich následně počítač provedl své operace. Navíc spolu zařízení nedokázala výrazněji spolupracovat. S nástupem rychlejšího internetu dostupného pro každou domácnost a cloudových služeb mají nyní počítače možnost získávat data přímo ze senzorů, bez potřeby uživatele a následně pomocí těchto dat spolupracovat s ostatními zařízeními. [41]

Pomocí cloudů je možné sestavit síť zařízení s levným málo výkonným hardwarem vykonávající jednoduché operace, se kterými potom složitější více výkonný systém pracuje. Tato zařízení také umožňují sbírat data pro neurální sítě umělých inteligencí. Obecně se dají IoT zařízení dělit na 3 kategorie.

První kategorie jsou zařízení umožňující sbírat a odesílat data. Jedná se většinou o velice jednoduchá zařízení s malým mikroprocesorem a možností připojení k Wi-Fi, na která je napojeno např. teplotní čidlo. Zaznamenaná data ze zařízení jsou následně vyslána do cloudu, kde jsou informace zpracovány.

Druhá kategorie jsou zařízení umožňující na zaznamenaná data reagovat. Mezi taková zařízení patří např. kouřové senzory a jiné bezpečnostní detektory, které reagují na nějakou situaci a upozorní uživatele.

Třetí kategorií jsou zařízení schopná pracovat s externími zdroji informací. Např. zavlažovací systém, který s pomocí jiných zařízení připojených do jedné sítě dokáže zjistit, která část půdy je suchá a tu zavlaží. Může také získat předpověď počasí z internetu a zjistit tak, zda např. nebude pršet. [42]

3.3.1 Využití IoT

IoT může být využito v různých odvětví jako zdravotnictví, zemědělství, doprava, správa elektrické rozvodné sítě a další.

Asi nejvíce médii pokrývaným využitím IoT je automobilový průmysl v samořiditelných autech a celkově využití IoT v dopravní infrastruktuře. Jsou vyhodnocována data ze senzorů aut, data z GPS, o počasí a dopravě pomocí umělé inteligence. Na základě těchto dat je optimalizována trasa a řízení auta.

IoT zařízení jsou využívány v konceptu chytrých měst, např. při detekci volných míst na parkovištích. Senzory na jednotlivých parkovacích místech detekují, zda na nich jsou zaparkovaná auta a tyto informace vysílají do cloudového systému. Jakmile je parkoviště plné, může tento cloudový systém např. upozornit předem řidiče, který na něm chce zaparkovat.

Tato IoT zařízení komunikují buď pomocí kabelů nebo pomocí komunikačních protokolů jako ZigBee, Z-wave nebo MQTT. Při komunikaci pomocí těchto protokolů je používána nižší frekvence než u Wi-Fi, a tak je spotřeba energie nižší a komunikace je stabilnější.
Proto je protokol ZigBee používán v komerčních systémech pro inteligentní domácnost jako Apple HomeKit nebo Alexa od Amazonu. [43] [44]

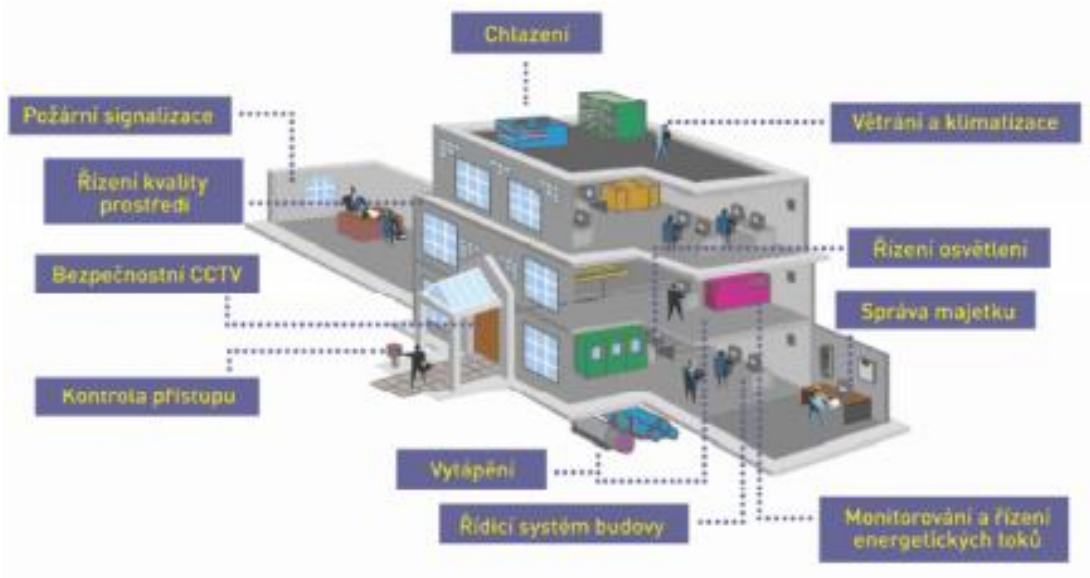
3.4 Inteligentní budova

Termín Inteligentní budova je nejednoznačný. Neexistuje všeobecně používaná definice. Obvykle tento pojem vyjadřuje jeden z možných pohledů na velice širokou problematiku.

Inteligentní budova je chápána především jako budova s pokročilým systémem řízení, regulace a monitoringu (umělá inteligence) bez nutnosti zásahů člověka spolu se systémovým řešením strojních zařízení budovy. Inteligentní budova zajišťuje optimální vnitřní prostředí pro komfort osob či výrobní produkci prostřednictvím stavební konstrukce, techniky prostředí, řídicích systémů, služeb a managementu. Je efektivní ekonomicky, energeticky i z hlediska působení na vnější prostředí a umožňuje víceúčelové použití a rekonfigurace. [45]

Hlavní oblastí, kde se koncepty inteligentních budov uplatňují, jsou z logiky věci především velké kancelářské komplexy, kde lze počítat s velkými úsporami provozních nákladů a kde si investor tento požadavek začlení už do zadání stavby.

V poslední době se koncept inteligentní budovy používá i v případě rodinných domů či bytů. Zde může být koncept inteligentní budovy aplikován v oblasti systémů větrání a zastiňování nebo v oblasti vytápění a alternativních zdrojů tepla (solární panely, tepelná čerpadla apod.), kdy inteligentní systémy pomáhají eliminovat případné výpadky těchto alternativních zdrojů např. v případě nepříznivých klimatických podmínek, extrémních mrazů atd. Kromě snahy o úsporu energií přichází i myšlenka, že by inteligentní budova měla reagovat nejen na změny vnějšího prostředí a počasí, ale třeba i na náladu a potřeby uživatelů. Principy inteligentní budovy lze také s úspěchem realizovat i dalších objektech. V hospodářských budovách, sklenících apod. [46]



Obrázek 9 Příklad inteligentní budovy [47]

3.4.1 Koncept inteligentní budovy

O inteligentní budově je možné mluvit tehdy, když jsou všechny systémy koncepčně provázány, aby maximálně sloužily uživatelům budovy. Ona „inteligence“ je fyzicky realizována technickými zařízeními instalovanými uvnitř budov a řídícím systémem budovy, který získává z těchto zařízení informace a dokáže je efektivně ovládat.

Technická zařízení budov (TZB) jsou:

- Instalace (vytápění, vzduchotechnika, klimatizace, chlazení, rozvody plynu, vody a kanalizace, centrální vysavače).
- Elektrotechnické rozvody (měření a regulace, elektrorozvody, zabezpečovací technika, řídicí systémy pro veškerá technická zařízení, hromosvody, telefonní rozvody, rozvody televizního signálu, počítačové sítě apod.).
- Další technická zařízení v budovách (osvětlení, zastiňování, výtahy apod.).

Řídicí systém budovy musí ze všech TZB získávat informace o jejich nastavení a aktuálním stavu. Část TZB dokáže řídící systém budovy přímo ovládat. S narůstajícím množstvím TZB co do druhů i počtu ale narůstá i celková složitost celého systému. Je nutné zajistit dostupnost komunikace TZB s řídícím systémem budovy. Protože by nárůst počtu TZB po jejich propojení s řídícím systémem budovy vedl k velkému množství kabelových rozvodů, nahrazují je často systémy bezdrátových komunikací. Proto se dnes TZB často doplňují technologiemi IoT.

Dostupnost je dalším aspektem, kterým je nutné se při realizaci řídícího systému budovy zabývat. Čím důmyslnější "schopnosti" má řídící systém budovy, tím vyšší nároky jsou na jeho dostupnost. Výpočetní systémy s vysokou dostupností jsou založeny obvykle na redundanci jednotlivých stavebních prvků a již z principu svého fungování jsou tedy velmi nákladné. S rozvojem poskytování cloudových služeb je možné tyto požadavky na dostupnost výpočetní infrastruktury přenést do prostředí veřejného cloudu a zakoupit je jako službu.

3.4.2 Možnosti inteligentní budovy

Možnosti inteligentní budovy jsou nepřeberné a neustále se vyvíjejí. Jedním z možných příkladů je ventilace řízená technikou. Jde o sofistikovaný systém přirozeného větrání, pomocí kterého je nahrazena klasická klimatizace. V řídícím systému budovy se shromažďují informace z mnoha čidel. Je z něj ovládáno otvírání a zavírání střešních oken a prosklených stěn na všech fasádách a rovněž jejich zastiňování.

Základem přirozené ventilace je tzv. inteligentní fasáda, která je vybavena funkčním systémem. Ten ovládá otevírání oken a stínění na jižní, východní a západní fasádě. Tento systém měří vnitřní i vnější klima – teplotu a rychlost větru. Přírodní ventilace funguje na základě principů pulzující ventilace a denní přírodní ventilace.

Pulzující ventilace znamená, že v průběhu dne se okna otevírají na krátké doby (např. 3 minuty) a tím vytváří dobrou vnitřní klimatickou atmosféru. Většinou je využívána v průběhu zimního období a v chladnějších měsících jara a podzimu. Teplotou ovládaná ventilace znamená, že okna jsou nepřetržitě ovládána na základě vnitřního a vnějšího prostředí. Většinou je využívána v průběhu letního období a v teplejších měsících jara a podzimu.

Denní přírodní ventilace je v létě doplněna i noční ventilací, pokud je nutné chlazení. Ovládací systém přirozené ventilace umožňuje efektivní noční chlazení budovy. Spočívá v tom, že termální masa budovy se v noci vychladí a celý následující teplý den působí jako pasivní chlazení. Čerstvý vzduch proudí dovnitř ze spodních pater a cirkuluje střešními okny.

Systém zajišťuje lepší vnitřní klima budovy (čerstvý vzduch, lepší rovnováhu mezi teplotou uvnitř a venku) a je výhodnější také s ohledem na dlouhodobou spotřebu energie (v rámci údržby, životnosti a využití zdrojů). [48]

4 Vlastní práce

V rámci vlastní práce je nejdříve analyzován systém inteligentní budovy a určena jeho požadovaná funkcionalita. Poté jsou jednotlivé funkcionality rozděleny podle modulů, ke kterým patří. Následně je v návrhu a implementaci navrženo cloudové řešení pro jednotlivé moduly, propojení mezi těmito moduly a popsán samotný vývoj aplikace.

4.1 Analýza

4.1.1 Analýza systémů inteligentní budovy

Navrhovaný systém inteligentní budovy získává informace ze senzorů umístěných v budově a mimo budovy. Dále může získávat informace z technických zařízení budovy nebo je ovládat. Senzory a aktivní prvky jsou tvořeny z jednoduchých IoT zařízení. Cílem je minimalizovat lokální IT infrastrukturu budovy jejím přenesením do prostředí cloudů. V cloudovém prostředí jsou ukládána data ze senzorů a je prováděna jejich analýza pro potřeby řízení technických zařízení budovy. Ovládání systému inteligentní budovy je prováděno prostřednictvím webové aplikace umístěné v cloudovém prostředí. Pro vytváření aplikací v cloudu jsou využity služby SaaS a PaaS.

4.1.2 Funkční požadavky systému

A. Měření teploty, tlaku a vlhkosti a intenzity osvětlení
    a. Senzor měří v pravidelných intervalech fyzikální veličiny teplotu, tlak, vlhkost a intenzity osvětlení v jednotlivých místnostech. Naměřené hodnoty jsou odesílány do databáze umístěné v prostředí veřejného cloudu. V případě krátkodobých výpadků internetového připojení nesmí dojít ke ztrátě naměřených dat.
B. Měření informací o provozu vytápění budovy
    a. Senzor měří v pravidelných intervalech teplotu topné vody, teplotu vody pro ohřev teplé užitkové vody a vlastní teplotu užitkové vody. V případě krátkodobých výpadků internetového připojení nesmí dojít ke ztrátě naměřených dat.
C. Nastavení stavu technických zařízení budovy
    a. Aktivní prvky zjišťují své nastavení v pravidelných intervalech z řídící aplikace umístěné v prostředí veřejného cloudu.

b. Jsou simulovány aktivní prvky pro ovládání žaluzií a termostatických hlavic u radiátorů pomocí skriptu.

D. Systém umí pracovat s historií dat ze senzorů
   a. Systém umožňuje vytvoření grafů jednotlivých naměřených dat ze senzorů v nastavitelné časovém rozpětí. Je možné vytvářet tabulky s grafy s nastavitelnou velikostí a tvarem.

E. Systém je řízen aplikací ve webové prostředí
   a. Aplikace umožňuje spárovat jednotlivé řídící prvky a senzory s místnostmi. Následně pro tyto místnosti je možné nastavit hodnoty řídících prvků a zobrazit poslední naměřená data fyzikálních veličin ze senzorů, která se v místnostech nachází.

4.1.3 Nefunkční požadavky systému

A. Senzory jsou realizovány ve vývojovém prostředí Raspberry PI.
B. Databáze pro ukládání dat ze senzorů je realizována v rámci cloudového prostředí s využitím služeb SaaS nebo PaaS.
C. Ovládací aplikace je realizováno v rámci cloudového prostředí s využitím služeb SaaS nebo PaaS.
D. Grafická prezentace dat je realizována v rámci cloudového prostředí s využitím služeb SaaS nebo PaaS.

4.1.4 Moduly systému

Systém inteligentní budovy se skládá ze dvou funkčních celků, z lokálních zařízení IoT umístěných v budově a centrálního řídícího systému umístěného v prostředí veřejného cloudu.

Lokální zařízení IoT jsou např. senzory měřící teplotu, tlak, vlhkost a intenzitu osvětlení. Následně tyto senzory odesílají naměřená data do databáze umístěné ve veřejném cloudu. Dále mezi lokální zařízení patří simulované aktivní prvky, které v pravidelných intervalech získávají informace o tom, v jakém stavu se mají nacházet.

Centrální řídící systém se skládá z databáze s daty ze senzorů a z aplikace pro grafickou prezentaci, která zobrazí data z databáze pomocí grafů. Systém má také řídící webovou

aplikaci, která umožňuje vytvářet místnosti. U těchto vytvořených místností umožňuje přidat senzory, které v aplikaci ukazují poslední naměřené hodnoty. K místnostem je také možné přidat řídící prvky, pro které lze následně nastavovat hodnoty, se kterými tyto řídící prvky pracují.

4.2 Implementace

Po analýze požadavků systému inteligentní budovy přichází na řadu její implementace. V této části jsou popsány všechny kroky nutné pro rozběhnutí jednotlivých modulů, které dohromady vytvářejí systém inteligentní budovy.

4.2.1 Cloudová databáze

Z dostupných cloudových řešení pro ukládání dat je vybrána služba SaaS InfluxDB cloud od InfluxData, která používá databázi InfluxDB ve verzi 2.0. Tato cloudová databáze časových řad (*time series database*) umožňuje ukládat naměřená data ve formátu tzv. měření, který je specificky vytvořen pro IoT zařízení. Je také kompatibilní s řadou jiných programovacích jazyků, které umožňují manipulovat a ukládat data do databáze. Mezi ně patří programovací jazyky jako Python, GO, Java, PHP, a další, které následně používají klientské knihovny vytvořené pro práci s API InfluxDB.

Služba je hostovaná na cloudech Amazonu, Microsoftu nebo Google a je dostupná na oficiálních stránkách firmy InfluxData ve výhodné bezplatné licenci. Tato licence umožňuje shromažďovat data po dobu 30 dní v jedné tabulce. Omezuje také velikost databázového přenosu do databáze a mimo databázi. [49]

Datový formát InfluxDB 2.0 zapouzdřuje jednotlivá naměřená data, k nim uložené značky a časové razítko, které identifikuje, kdy měření bylo uloženo do databáze. Toto časové razítko může být specifikováno uživatelem. Syntaxe tohoto formátu je následující:

```
// Syntaxe

<measurement>,[<tag_key>=<tag_value>,<tag_key>=<tag_value>]
<field_key>=<field_value>,<field_key>=<field_value>] [<timestamp>]


// Příklad

myMeasurement,tag1=”value1”,tag2=”value2” fieldKey="fieldValue" 1556813561098000000
```

- *measurement_name*
    - Určuje název měření, pod kterým budou ukládána následující data.
    - Musí být vyplněn pro správné naměřených dat.
- *tag_key*
    - Určuje název značek, kterými lze rozlišovat jednotlivá měření.
    - Tyto značky jsou nepovinné.
    - Jako hodnotu značky lze ukládat jen text.
    - Jednotlivé značky jsou od sebe odděleny čárkou.
    - Značky a naměřené hodnoty jsou odděleny mezerou.
- *field_key*
    - Specifikuje název pole, pod kterým je uložena naměřená hodnota.
    - Musí být vyplněno alespoň jednou pro správné uložení.
    - Může ukládat hodnoty ve formě textu, celých čísel, čísel s desetinou čárkou nebo boolean hodnoty.
    - Jednotlivá pole jsou oddělena čárkou.
- *timestamp*
    - Specifikuje hodnotu časového razítka.
    - Pokud není tato hodnota specifikována, je automaticky uložen čas, kdy bylo měření uloženo do databáze.

Všechna měření jsou následně ukládána do tabulek shromažďujících jednotlivá měření, které tento databázový systém nazývá „*bucket*“. Tyto buckety následně vlastní určitá organizace, do které uživatel patří.

Data z těchto bucketů jsou získána pomocí datového skriptovacího jazyka Flux, který slouží jako ekvivalent k dotazovacímu jazyku SQL a umožňuje zapisovat a získávat data z databáze. Syntaxe tohoto jazyka je velmi podobná skriptovacímu jazyku Javascript. Příklad této syntaxe je vidět na následujícím příkladě:

```
from(bucket: "Sensor_data")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r.measurement == "boiler_radiator_temp")
  |> filter(fn: (r) => r._field == "tmp_in" or r._field == "tmp_out")
```

Každý skript jazyka Flux začíná příkazem *from*, kterým deklarujeme název bucketu, ze kterého jsou data získávána. Následně musí být specifikován časový rámec, ve kterém mají být tato data získávána pomocí příkazu *range*. Příkaz range pracuje buď se specifikováním času od začátku po konec, nebo je specifikováno, o kolik minut, hodin nebo dní nazpátek mají být data získána z databáze např.:

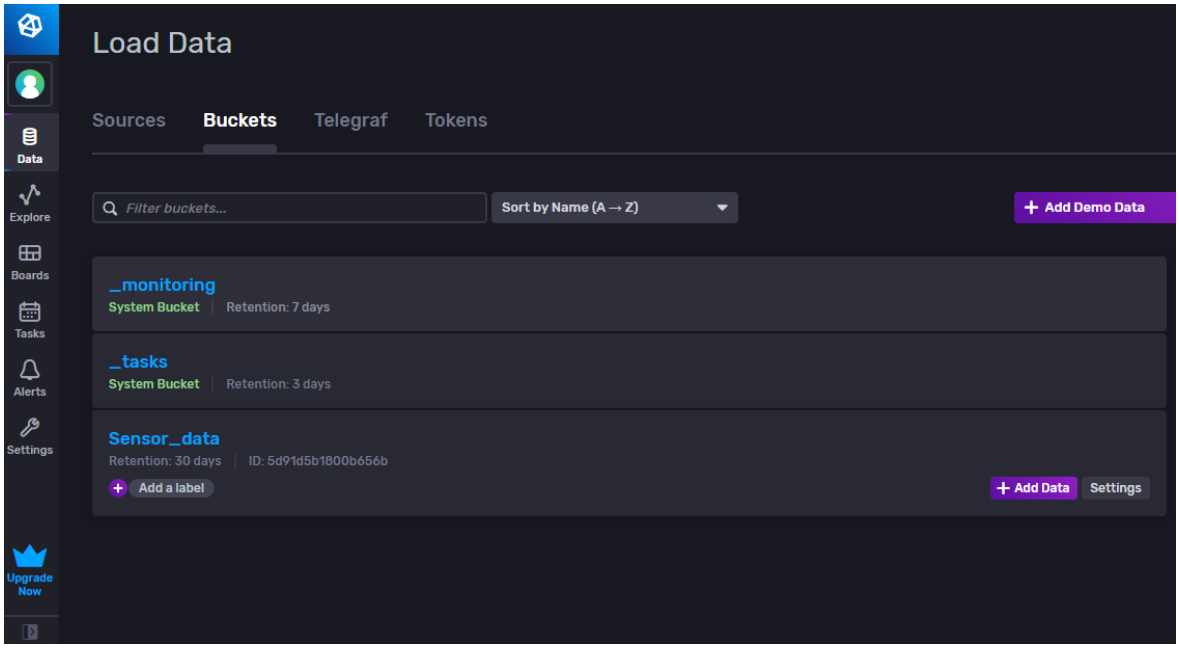
```
from(bucket: "Sensor_data")
  |> range(start: -1h) # získána data zapsaná před 1 hodinou
  |> filter(fn: (r) => r.measurement == "boiler_radiator_temp") # filtrování dle názvu měření
  |> filter(fn: (r) => r.host == "rpiboiler") # filtrování dle názvu tagu
  |> filter(fn: (r) => r._field == "tmp_in" or r._field == "tmp_out") # filtrování dle názvu pole
```

Každý příkaz má také před sebou značku „|>“, která přenáší výsledky z jednoho příkazu na druhý. Po získání všech dat v daném časovém rozmezí následuje příkaz *filter*, pomocí kterého jsou filtrovány získané hodnoty dle zadaných vlastností.

4.2.1.1 Vytvoření databáze v cloudu

Pro samotné vytvoření této cloudové databáze je nutné provést registraci na oficiálních stránkách. Registraci lze provést s jakoukoliv emailovou adresou. Pro registraci lze také použít účet Google nebo Microsoft.

Po dokončení registrace musí být zvoleno jméno organizace, ve které jsou vytvářeny buckety a jméno bucketu, do kterého jsou ukládána data. Po jejich zvolení je vytvořena cloudová databáze s jednou tabulkou.



Obrázek 10 UI InfluxDB cloudu (zdroj: autor)

Na *obrázku 10* je vidět GUI aplikace InfluxDB cloud, ve které je vytvořen uživatelův bucket s názvem „Sensor_data“. Do tohoto bucketu jsou ukládána data uživatele. Pro jejich uložení musí být vytvořen bezpečnostní token, který je dál použit v aplikacích napojených na tuto databázi. Token lze jednoduše vytvořit kliknutím na záložku *Tokens* a vygenerováním bezpečnostního tokenu pro bucket *Sensor_data*. Po vytvoření tokenu lze do databáze zapisovat data.

4.2.2 Senzory

4.2.2.1 Hardware senzorů

Dle funkčních požadavků systému jsou vytvořeny senzory pro měření teploty, tlaku, vlhkosti a intenzity světla uvnitř pokojů a senzor pro měření teploty topné vody, teploty vody pro ohřev teplé užitkové vody a vlastní teplotu užitkové vody přímo z kotle budovy.

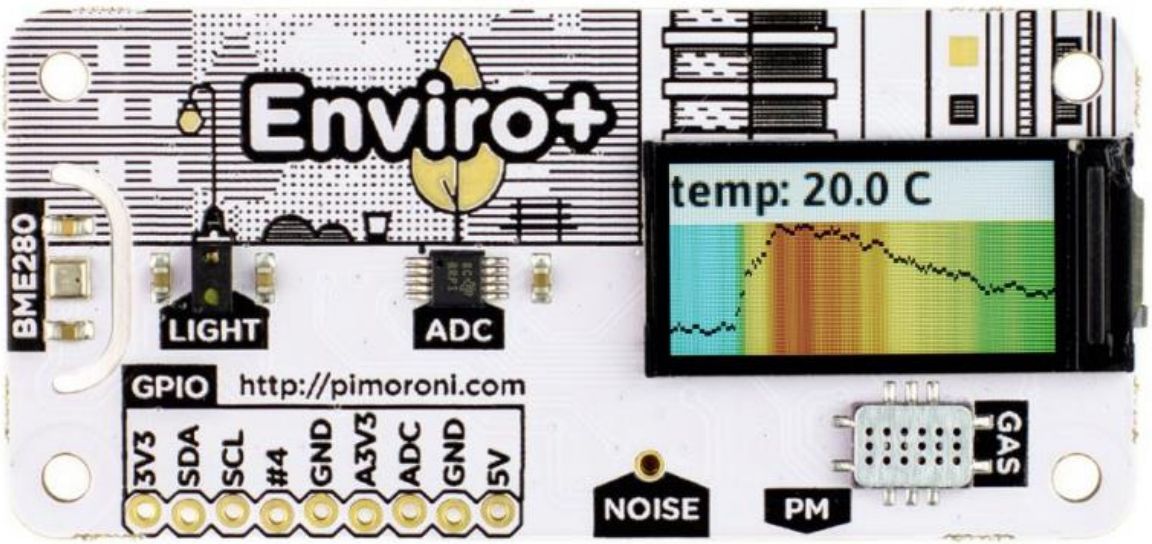
Senzory sbírající data uvnitř pokojů budovy jsou realizovány pomocí mikropočítačů Raspberry Pi Zero W (RPI Zero) s připojenými senzory pro měření fyzikálních veličin teplota, tlak, vlhkost a světlo. Do těchto malých a levných mikropočítačů za cca 300 Kč lze zasunout microSD kartu, na kterou je následně nahrán operační systém Raspberry Pi OS, který je odvozený z Linuxové distribuce Debian.

RPI Zero typu W také obsahuje Wi-Fi adaptér, přes který je RPI Zero možné připojit na lokální síť. Je tak možné RPI Zero umístit kamkoliv, kam dosáhne signál Wi-Fi a kde je připojení do elektrické sítě budovy. [50]

Řešení pomocí Wi-Fi není optimální pro širší využití, protože je mnohem více energeticky náročná a méně stabilní, pokud je připojen větší počet zařízení. Pro tuto práci je propojení pomocí Wi-Fi dostatečné, pokud by byl ale připojen vyšší počet zařízení, bylo by lepší použít např. protokol MQTT. Tento protokol lze používat na jakémkoliv Raspberry Pi, které má Wi-Fi adaptér.

Na RPI Zero je připojena senzorovou desku Enviro Plus od společnosti Pimoroni. Tato senzorová deska je vybrána díky velkému množství senzorů, které obsahuje:

- BME280 pro měření teploty, vlhkosti a tlaku;
- LTR-559 pro měření intenzity světla;
- MICS6814 analogový senzor pro měření plynu a analogový převodník ADS1015;
- K desce lze také připojit snímač částic PMS5003, se kterým lze sledovat kvalitu ovzduší.



Obrázek 11 Senzorová deska Enviro Plus [51]

Tato senzorová deska se svojí cenou cca 1300 Kč patří mezi jedny z dražších. Jako levnější variantu je možné pořídit např. levnější, starší variantu Enviro Plus zvanou *Enviro*. Toto zařízení stojí cca 830 Kč a oproti Enviru Plus neobsahuje senzor pro měření plynu a analogový převodník. Pro nejlevnější možnou variantu lze pořídit Enviro pHAT, které stojí cca 450 Kč a oproti Enviru plus obsahuje jen senzory pro měření teploty, tlaku a intenzity osvětlení. [51] [52] [53]

Samozřejmě existují i varianty od jiných firem. Senzorové desky Enviro ale mají tu výhodu, že lze k nim nalézt knihovny v programovacím jazyce Python, které značně ulehčují získávání dat a jejich případnou manipulaci.

Senzor, který získává data o provozu vytápění z kotle budovy používá stejný mikropočítač RPI Zero. Místo senzorové desky Enviro Plus ale používá konvertor I2C/one-wire Modulowo MOD-35, do kterého jsou zapojeny 4 vodotěsné teplotní senzory DS18B20. Tyto následně zaznamenávají teplotu z kotle.

4.2.2.2 Software senzorů

Na mikropočítač RPI Zero je nainstalován kromě operačního systému Rasbian také program Telegraf. Tento serverový agent napsaný v jazyce Go funguje na základně vstupních a výstupních pluginů. Pomocí vstupních pluginů lze získávat data z datových zdrojů a následně pomocí výstupních pluginů jsou tato data odesílána např. do databáze. Umožňuje také uchovávat záznamy ze vstupů lokálně na tzv. bufferu. V případě ztráty internetového připojení tak data nejsou ztracena a lze je odeslat po opětovném připojení.

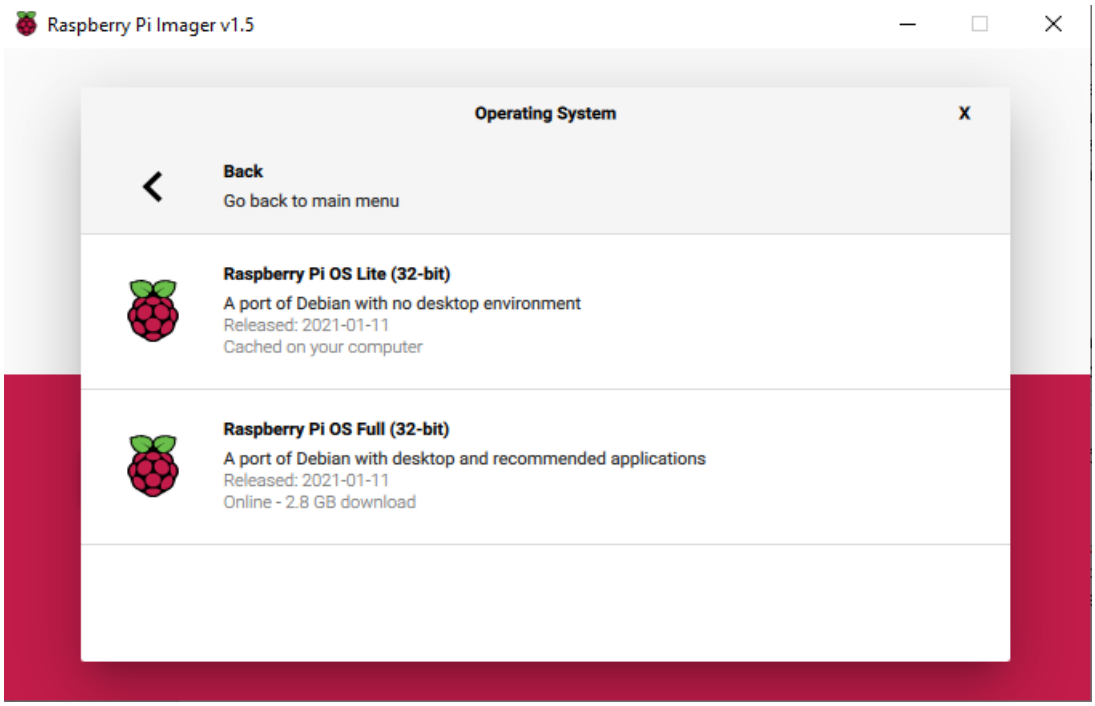
Telegraf je možné modifikovat pomocí textového konfiguračního souboru, ve kterém lze specifikovat hodnoty jako interval zápisu do databáze, velikost bufferu, jednotlivé vstupní a výstupní pluginy a hodnoty, se kterými tyto pluginy pracují např. adresa cloudové databáze.

4.2.2.3 Vytvoření senzoru pro pokoje

Před samotnou instalací softwaru je nutné správně nasadit senzory na piny RPI Zero. U senzorové desky Enviro Plus stačí konektor této desky nasadit na piny mikropočítače.

Aby bylo možné mikropočítač spustit, je nejdříve potřeba vytvořit microSD kartu s operačním systémem Raspberry Pi OS. Pro vytvoření je využit oficiální nástroj od firmy Raspberry Pi Foundation s názvem Raspberry Pi Imager. Tento program, který lze nainstalovat na operační systémy Linux, Mac a Windows, automaticky stáhne a vytvoří microSD kartu s operačním systémem Raspberry Pi OS. Stačí jen zvolit operační systém a kartu, na kterou má být systém nainstalován.

Na microSD kartu je nainstalována jednodušší varianta operačního systému Raspberry Pi Lite, která neobsahuje grafické uživatelské rozhraní a může být ovládána jen přes terminál. Tato varianta zabírá méně paměti na microSD kartě a celkově lépe funguje na hardwaru RPI Zero.



Obrázek 12 Zvolení operačního systému pro RPI Zero (zdroj: autor)

Obrázek 13 Obraz aplikace Raspberry Pi imager před vytvořením OS (zdroj: autor)

Po vytvoření microSD karty s operačním systémem je potřeba vytvořit ve svazku souborů *rootfs* (*ext4*) ve složce */etc/wpa_supplicant* soubor *wpa_supplicant.conf*, ve které je uloženo zašifrované heslo k Wi-Fi připojení.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=CZ
network={
    ssid="<Wi-fi ID>"
    psk=<WPA passphrase>
    key_mgmt=WPA-PSK
}
```

Je potřeba také zprovoznit SSH pro přístup do zařízení ze vzdáleného počítače. SSH je zpřístupněno vytvořením prázdného souboru *ssh* v adresáři */boot* ve svazku souborů *rootfs*. Jakmile jsou tyto soubory vytvořeny, je možné microSD kartu zasunout do RPI Zero a připojit zařízení ke zdroji napětí. RPI Zero se následně automaticky připojí k Wi-Fi a lze se k němu připojit ze vzdáleného počítače přes SSH.

Po připojení do RPI Zero je potřeba provést aktualizaci softwaru zařízení pomocí:

```
$ sudo apt update && sudo apt upgrade
```

Po jeho dokončení je také dobré změnit výchozí heslo RPI Zero na jiné pomocí příkazu:

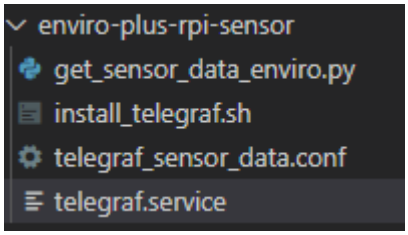
```
$ passwd
```

Jakmile je změněno heslo a jsou nainstalovány poslední aktualizace, je nutné nainstalovat knihovnu pro používání senzoru Enviro Plus. Oficiální GitHub úložiště tohoto senzoru nabízí jednoduchý instalační skript, který je možné získat přímo z internetu pomocí příkazu:

```
$ curl -sSL https://get.pimoroni.com/enviroplus | bash
```

Tento jednoduchý příkaz získá kód ze stránek výrobce senzoru, který nainstaluje všechny potřebné knihovny pro senzor Enviro Plus. Pokud tento příkaz nefunguje, je možné použít alternativní metody instalace, které jsou uvedeny na stránkách GitHub uložiště.

Po provedení těchto kroků již RPI Zero dokáže číst data ze senzorů této senzorové desky. Dále musí být vytvořeny soubory, které jsou potřeba pro správnou instalaci Telegrafu.



Obrázek 14 Soubory pro instalaci Telegrafu na RPI Zero (zdroj: autor)

Jako první je vytvořen konfigurační soubor pro aplikaci Telegraf s názvem „*telegraf_senzor_data.conf*“. Tento konfigurační soubor specifikuje hodnoty jako interval získávání hodnot ze vstupních pluginů, limit bufferu output pluginů a časovou hodnotu mezi sbíráním jednotlivých dat ze vstupních pluginů. Je v něm také specifikována cesta k textovému souboru, do kterého Telegraf zapisuje svůj log.

Nejdůležitější částí, kterou soubor specifikuje, jsou jednotlivé vstupní a výstupní pluginy a hodnoty, které tyto pluginy používají. Tuto část lze vidět na následující ukázce:

```
[[outputs.influxdb_v2]]
## The URLs of the InfluxDB cluster nodes.
##
## Multiple URLs can be specified for a single cluster, only ONE of the
## urls will be written to each interval.
## urls exp: http://127.0.0.1:9999
urls = ["<URL databáze>"]
## Token for authentication.
token = "<Bezpečnostní token pro zápis do databáze>"
## Organization is the name of the organization you wish to write to
## Must exist.
organization = "Organizace která vlastní bucket"
## Destination bucket to write into.
bucket = "Bucket organizace do kterého budou data zapsána"
[[inputs.exec]]
commands = ["/etc/telegraf/get_sensor_data_enviro.py"]
data_format = "influx"
```

Na této ukázce je viditelná část konfiguračního souboru, ve kterém je specifikována konfigurace výstupního pluginu *outputs.influxdb_v2* a vstupního pluginu *inputs.exec*. Vstupní a výstupní plugin jsou vnitřní funkce aplikace Telegraf.

Vstupní plugin vyžaduje specifikovat python skript, ve kterém je konkrétní implementace získávání dat ze senzorové desky. Data jsou následně předána do aplikace Telegraf.

```
def main():
    i = 0
    # First data from enviro sensor is flawed.
    # This gets data after 3 cycles each with 1 second waiting time
    while i <= 2:
        try:
            temperature = bme280.get_temperature()
            temperature_compensated = get_compensated_temperature()
            pressure = bme280.get_pressure()
            humidity = bme280.get_humidity()
            lux = ltr559.get_lux()
            prox = ltr559.get_proximity()
            i += 1
            time.sleep(1)
        except:
            print(traceback.format_exc())
    sensor_id = "raspi-" + get_serial_number()
    sensor_type = "Enviro-plus" # TODO Make dynamic
    # mymeasurement,tag1=tag1,tag2=tag2 fieldA="aaa",fieldB="bbb
    print("sensor_temperature,sensor_id={},sensor_type={} temperature={:.2f},t
emperature_compensated={:.2f}".format(
        sensor_id, sensor_type, temperature, temperature_compensated))
    print("sensor_pressure,sensor_id={},sensor_type={} pressure={:.2f}".format
(sensor_id, sensor_type, pressure))
    print("sensor_humidity,sensor_id={},sensor_type={} humidity={:.2f}".format
(sensor_id, sensor_type, humidity))
    print("sensor_light,sensor_id={},sensor_type={} light={:.2f},proximity={:.
2f}".format(sensor_id, sensor_type, lux, prox))

if __name__ == "__main__":
    main()
```

V této části je viditelný kód skriptu *get_sensor_data_enviro.py*, který získá tlak, vlhkost a intenzitu světla ze senzorové desky Enviro Plus. Tyto hodnoty jsou následně uloženy do jednotlivých měření, kterými jsou:

- *sensor_temperature* měření teploty v pokoji;
- *sensor_pressure* měření tlaku v pokoji;
- *sensor_humidity* měření vlhkosti v pokoji;
- *sensor_light* měření intenzity světla v pokoji.

Společně s jednotlivými naměřenými hodnotami je do cloudu také uloženo sériové číslo RPI Zero sloužící jako unikátní identifikátor senzoru a typ senzoru, pokud by v budoucnu byl používán jiný typ zařízení než RPI Zero.

U měření „*sensor_temperature*“ jsou uloženy dva druhy teplot. První druh teploty je čistá teplota bez jakékoliv kompenzace. Druhý je druh teploty, který se snaží kompenzovat teplo z procesoru mikropočítače RPI Zero. Může totiž dojít ke zkreslení teplotních hodnot ze senzorové desky v případech, kdy senzorová deska a RPI Zero jsou těsně u sebe.

Na níže uvedeném příkladě je zobrazen formát InfluxDB, který je používán pro ukládání dat do bufferu aplikace Telegraf.

```
sensor_temperature,sensor_id={sensor_id},sensor_type={sensor_type}
temperature={temperature},temperature_compensated={temperature_compensated}
```

- Tag *sensor_temperature* je název měření.
- Tag *sensor_id* je sériové číslo RPI Zero.
- Tag *sensor_type* značí, zda jde o Raspberry PI nebo senzor jiného typu.
- Následuje skupina hodnot, které jsou měřeny. Zde je měřena teplota okolního prostředí a teplota okolního prostředí kompenzovaná o teplotu z procesoru RPI Zero.

Následně tyto data skript vypíše. Telegraf dokáže tento výpis zachytit a následně ho uložit do bufferu output pluginů. Hodnoty z tohoto bufferu následně pošle pomocí output pluginu do cloudové databáze.

Konfigurační soubor „*telegraf.service*“ služby *systemctl* popisuje, jak spravovat službu, která spouští aplikaci InfluxDB na RPI Zero. To zahrnuje způsob, jakým je služba spuštěna a zastavena, za jakých okolností má být automaticky spuštěna a informace o softwaru, na kterém je aplikace závislá. Také popisuje, pod jakým uživatelem má být služba spuštěna a jaký konfigurační soubor má služba použít. Tato služba používá výchozí složku */etc/telegraf*, kterou program Telegraf používá, pokud je spuštěn bez parametrů.

```
[Unit]
Description=The plugin-driven server agent for reporting metrics into InfluxDB
Documentation=https://github.com/influxdata/telegraf
After=network.target
[Service]
EnvironmentFile=-/etc/default/telegraf
User=pi
ExecStart=/usr/bin/telegraf -config /etc/telegraf/telegraf.conf -config-
directory /etc/telegraf/telegraf.d $TELEGRAF_OPTS
ExecReload=/bin/kill -HUP $MAINPID
Restart=on-failure
RestartForceExitStatus=SIGPIPE
KillMode=control-group
[Install]
WantedBy=multi-user.target
```

Konfigurační soubor služby Telegraf

Skript „*install_telegraf.sh*“ nainstaluje aplikaci Telegraf a zkopíruje konfigurační soubor „*telegraf.service*“ do složky „*/etc/systemd/system/multi-user.target.wants/*“, aby byla vytvořena služba, která po každém restartu RPI Zero aplikaci Telegraf spustí. Je také vytvořen soubor „`/var/log/telegraf/telegraf.log`“, do kterého následně aplikace zapisuje veškeré události.

Samotný konfigurační soubor aplikace Telegraf a python skript jsou zkopírovány do složky „*/etc/telegraf*“. Toto je výchozí složka pro aplikaci Telegraf, ve které hledá konfigurační soubory a skripty v případě, kdy nejsou zadány žádné parametry. Skript také změní u každého souboru práva tak, aby mohl být skript spuštěn uživatelem „*pi*“, pod kterým se tento skript bude automaticky spouštět.

Tento instalační skript musí být spuštěn s oprávněními uživatele *root* pomocí příkazu:

```
$ sudo ./install_telegraf.sh
```

Následně musí být RPI Zero restartováno a po novém SSH připojení zadán příkaz:

```
$ sudo systemctl restart telegraf
```

Tento příkaz aktivuje službu, která spustí Telegraf, začne zaznamenávat hodnoty ze senzorů a odesílá je do InfluxDB cloudu. Tato služba bude automaticky spuštěna po každém restartování zařízení.

4.2.2.4 Vytvoření senzoru pro vytápění budovy

Před instalací je propojena sběrnice RPI Zero s I2C sběrnicí Modulowo MOD-35, na kterou jsou napojeny teplotní senzory.

Senzor pro měření hodnot na kotli budovy používá stejný mikropočítač se stejným operačním systémem, ale s jinými knihovnami. Předpokládá se tedy, že uživatel se již připojil do mikropočítače přes SSH a nainstaloval aktuální operační systém.

Aby bylo možné knihovny *one-wire* nainstalovat, je potřeba nejdříve v konfigurační aplikaci RPI Zero povolit jejich rozhraní. Do *raspi-config* se lze dostat pomocí příkazu:

```
sudo raspi-config
```

V aplikaci je vybrána položku nastavení rozhraní (*interface options*), kde musí být povoleno rozhraní I2C a one-wire. Po jejich povolení jsou uloženy změny konfigurace a následně nainstalovány knihovny pro I2C a one-wire.

Instalace knihoven I2C:

```
sudo apt-get install -y python-smbus
sudo apt-get install -y i2c-tools
```

Instalace knihoven one-wire:

```
sudo mkdir /mnt/1wire
sudo apt-get install owfs
sudo apt-get install owfs-doc
sudo apt-get install ow-shell
```

```
sudo apt-get install python-ow
```

Po instalaci těchto knihoven musí být upraveny jejich konfigurační soubory. Ve složce „*/etc/owfs.conf*“ je potřeba nahradit řádek s textem „*server: FAKE = DS18S20,DS2405*“ za následující:

```
device = /dev/i2c-0
mountpoint = /mnt/1wire
Celsius
allow_other
error_print = 0
error_level = 0
```

Po restartu tato konfigurace nastaví sběrnici one-wire, následně lze příkazem „*owread*“ číst parametry z jednotlivých čidel.

Po nainstalování knihovny pro práci se senzorem je možné nainstalovat program Telegraf, jehož instalace je pro tento senzor stejná jako pro Enviro Plus (viz. sekce *4.2.2.3 Vytvoření senzoru pro pokoje)*. Jediné, čím se liší, je python skript, který je používán vstupním pluginem Telegrafu, jelikož používá jiný senzor a knihovnu. Tento python je vidět v následující ukázce:

```
#!/usr/bin/env python3

import subprocess

def get_serial_number():
    with open('/proc/cpuinfo', 'r') as f:
        for line in f:
            if line[0:6] == 'Serial':
                return line.split(":")[1].strip()


def main():
    # vstupní teplota vody do radiátorů
    to_inp = float(subprocess.check_output(
        ['owread', '/28.9C873E1B1901/temperature']).decode().strip())
    # vstupní teplota vody do radiátorů
    to_out = float(subprocess.check_output(
        ['owread', '/28.41DA211C1901/temperature']).decode().strip())
    # vstupní teplota vody do radiátorů
    dhw_tmp = float(subprocess.check_output(
        ['owread', '/28.F6775F070000/temperature']).decode().strip())
    # vstupní teplota vody do radiátorů
    dhw_coil_temp = float(subprocess.check_output(
        ['owread', '/28.4618C1070000/temperature']).decode().strip())

    sensor_id = "raspi-" + get_serial_number()
    sensor_type = "1wire"  # TODO Make dynamic
    print("boiler_radiator_temp,sensor_id={},sensor_type={} tmp_in={:.2f},tmp_
out={:.2f}".format(sensor_id, sensor_type, to_inp, to_out))
    print("boiler_dhw,sensor_id={},sensor_type={} dhw_tmp={:.2f},dhw_coil_temp
={:.2f}".format(sensor_id, sensor_type, dhw_tmp, dhw_coil_temp))


if __name__ == "__main__":
    main()
```

Tento skript načte data ze senzorů připojených ke konvertoru a uloží je do proměnných, které jsou následně použity pro uložení měření do InfluxDB cloudu stejně jako python skript získávající data z Envira Plus. Mezi tato měření patří:

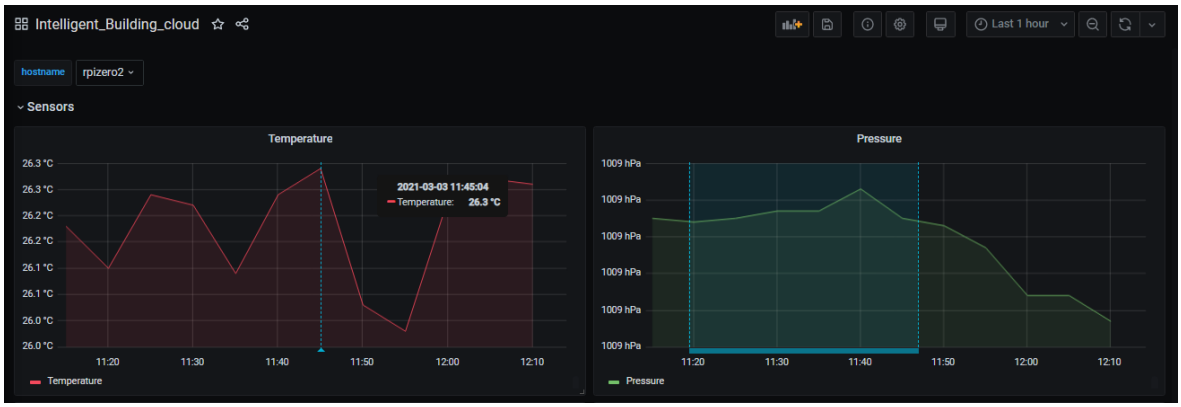
- *boiler_radiator_temp* měření teploty topné vody (teplota vody pro nahřívání systému radiátorů a teplota vody, která se vrací ze systému radiátorů);
- *boiler_dhw* teplota ohřevného tělesa teplé užitkové vody a vlastní teplota užitkové vody.

4.2.3 Grafická prezentace dat

Dle funkčních požadavků je pro vytvoření tabulek s grafy použita služba SaaS Grafana Cloud vytvořená firmou Grafana Labs. Stejně jako InfluxDB cloud tato aplikace je nabízena také ve výhodné bezplatné licenci s různými omezeními. [54]

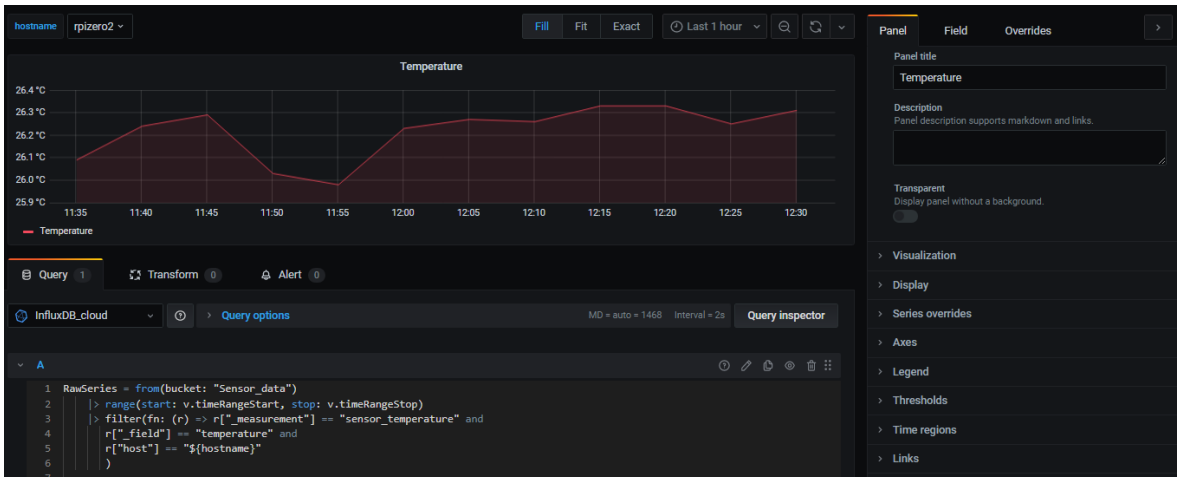
Tato aplikace umožňuje zobrazovat data z různých zdrojů pomocí tzv. „*dashboard*“ (přístrojová deska) s panely. Nejčastěji je používána s databázemi časových řad jako Prometheus, Graphite nebo InfluxDB, u kterých jsou data z těchto zdrojů zobrazena ve formě grafů. Aplikace podporuje i další druhy databází, mezi které patří MySQL, PostgreSQL, Oracle SQL, Microsoft SQL server, MongoDB a mnohé další.

V jednotlivých deskách jsou následně vytvářeny panely zobrazující jednotlivé grafy. Na *obrázku 15* je zobrazena tabulka vytvořená pro zobrazování dat z cloudové databáze InfluxDB cloud (viz. sekce *4.2.1 Cloudové databáze*). Do této databáze jsou ukládány data ze senzorů, které uživatel vytvořil (viz. sekce *4.2.2.3 Vytvoření senzoru pro pokoje)*.



Obrázek 15 Tabulka programu Grafana Cloud (zdroj: autor)

U jednotlivých panelů je možné měnit různé vlastnosti, které upravují jejich vzhled, velikost, jednotky na časových osách a mnohé další. Také lze hodnoty v grafu označovat pomocí anotací, které jsou viditelné pro všechny přihlášené uživatele. Menu pro editaci vlastností jednotlivých panelů je ukázáno na *obrázku 16*.



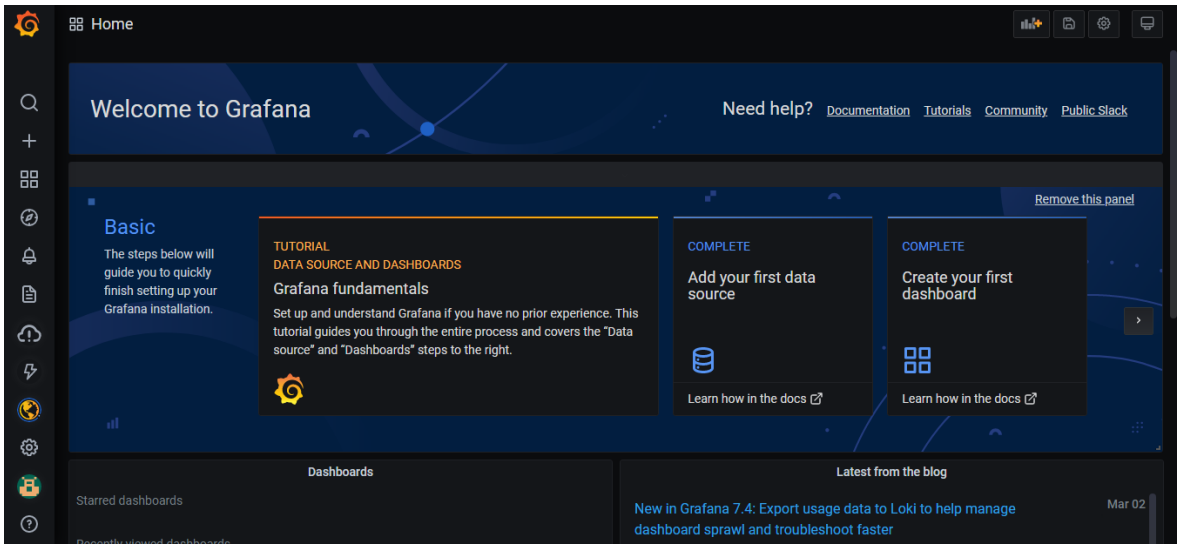
Obrázek 16 Editace grafu v aplikaci Grafana (zdroj: autor)

Kromě editace vlastností musí každý panel obsahovat příkaz pro získání dat z databáze. Na *obrázku 16* je ukázán použitý skriptovací jazyk Flux, který je používán pro získávání dat z databáze InfluxDB 2.0. Grafana umožňuje také vytvářet proměnné, které lze následně použít v příkazech. V tomto případě je vytvořena proměnná *hostname*, která umožňuje přepínat mezi jednotlivými senzory, a proměnná „*v*“, která umožňuje měnit časové rozmezí dat.

4.2.3.1 Vytvoření programu Grafana v cloudu

Pro vytvoření této cloudové služby se musí uživatel registrovat na oficiálních stránkách. Pro registraci je možné použít email, Amazon, GitHub, Google nebo Microsoft účet.

Po registraci musí být zadán název prostoru pro uložení dat aplikace a jméno organizace, pod kterou je aplikace spravována. Po vytvoření prostoru se může uživatel přihlásit do aplikace Grafana, kde se mu zobrazí její GUI.



Obrázek 17 GUI aplikace Grafana (zdroj: autor)

Následně je potřeba službu připojit k datovému zdroji, ze kterého mají být získávána data. V tomto případě se jedná o databázi InfluxDB hostovanou na InfluxDB cloudu. Musí být specifikována URL adresa cloudové databáze, název organizace a název bucketu databáze, ze které mají být data získána. Jakmile jsou hodnoty uloženy a test připojení proběhne bez problémů, může být vytvořena deska s panely viz. *obrázek 16*.

4.2.3.2 Vytvoření grafů v Grafana Cloud

Pro jednotlivé senzory umístěné v místnostech je vytvořena deska se čtyřmi panely pro teplotu, tlak, vlhkost a intenzitu světla. Do těchto panelů je vložen příkaz ve skriptovacím jazyce Flux, který vypíše odpovídající hodnoty. Kód pro získávání teploty pokojů vypadá následovně:

```
from(bucket: "Sensor_data")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "sensor_temperature" and
    r["_field"] == "temperature" and
    r["host"] == "${hostname}")
```

Typ hodnoty, která je zobrazena, je filtrován pomocí filtru pro měření (*_measurement*) a pole (*_field*). Pro vytvoření dalších panelů tedy stačí přepsat tyto hodnoty na:

```
Tlak -> r["_measurement"] == "sensor_pressure" and r["_field"] == "pressure"
Vlhkost -> r["_measurement"] == "sensor_humidity" and r["_field"] == "humidity"
Intenzita světla -> r["_measurement"] == "sensor_light" and r["_field"] == "light"
```

Kromě názvu měření a názvu pole jsou také jednotlivá měření filtrována dle proměnné „*hostname*“. Tato proměnná umožňuje přepínat mezi senzory v jednotlivých pokojích. Může být vytvořena v nastavení desek, kde je specifikováno její jméno a jako typ je zvolena hodnota „*Query*“, která mu umožní vložit následující příkaz v jazyce Flux.

```
import "influxdata/influxdb/v1"
v1.tagValues(
  bucket: v.bucket,
  tag: "host",
  predicate: (r) => r["host"] != "rpiboiler",
  start: -30d
)
```

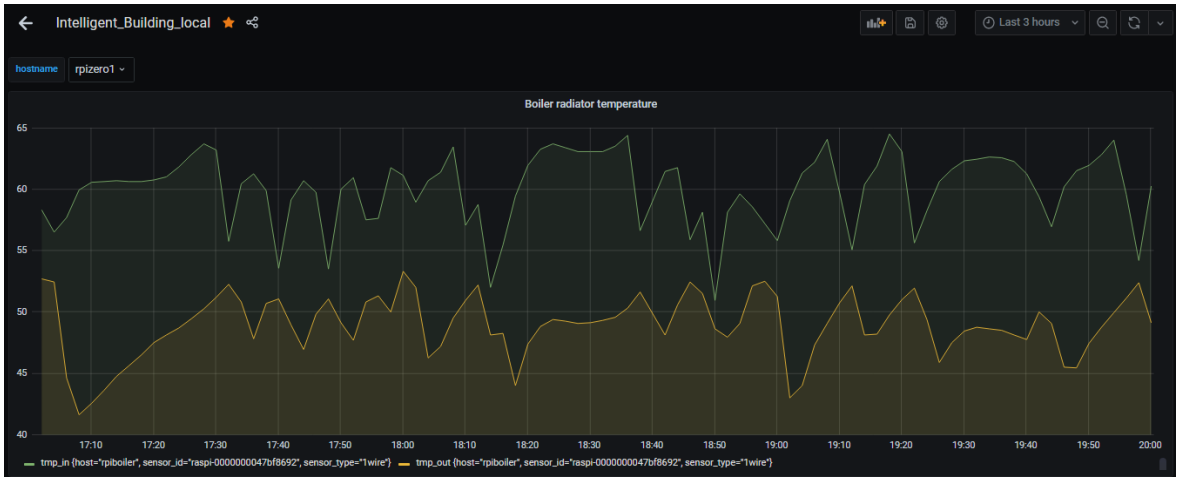
Tento příkaz získá z databáze všechny hodnoty tagu „*host*“ kromě hodnoty „*rpiboiler*“, která náleží senzoru sbírající data z kotle. Následně jsou hodnoty uloženy do proměnné programu Grafana, v tomto případě do proměnné „*hostname*“. Poté je možné tuto proměnou zavolat ve skriptu pomocí „*${hostname}*“.

Pro hodnoty získávané ze senzoru připojeného na kotel budovy je potřeba vytvořit další tři panely.

První panel zobrazuje teplotu topné vody. V rámci měření teploty topné vody se měří teplota vody pro nahřívání systému radiátorů a teplota vody, která se vrací ze systému radiátorů. Skript tohoto panelu vypadá následovně:

```
from(bucket: "Sensor_data")
  |> range(start: v timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "boiler_radiator_temp")
  |> filter(fn: (r) => r["_field"] == "tmp_in"  or r["_field"] == "tmp_out")
```

Výsledný graf lze vidět na *obrázku 18*:

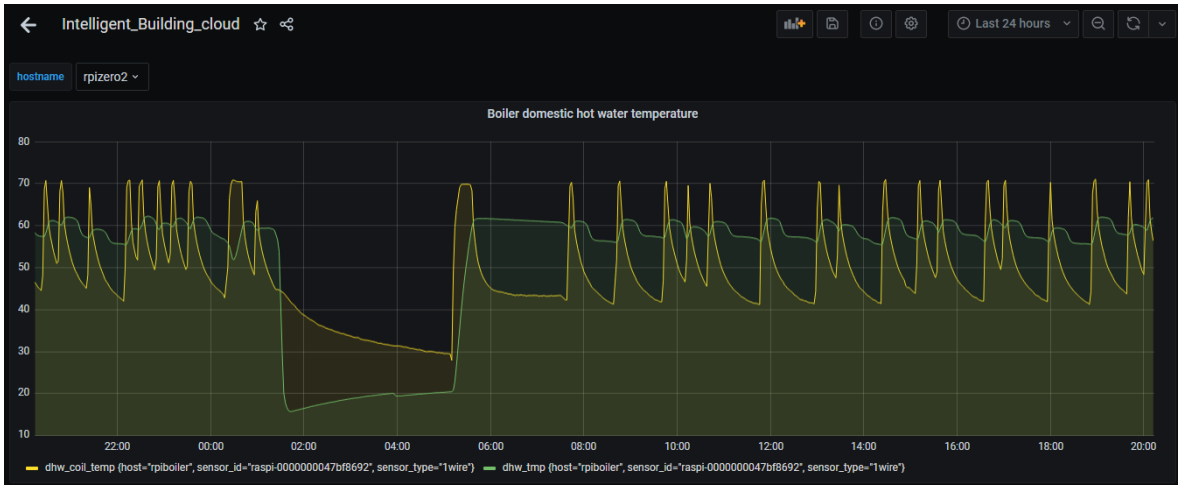


Obrázek 18 Graf teploty topné vody

Druhý panel zobrazuje ohřev teplé užitkové vody a vlastní teplotu užitkové vody. Její skript vypadá následovně:

```
from(bucket: "Sensor_data")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "boiler_dhw")
  |> filter(fn: (r) => r["_field"] == "dhw_coil_temp" or r["_field"] == "dhw_tmp")
```

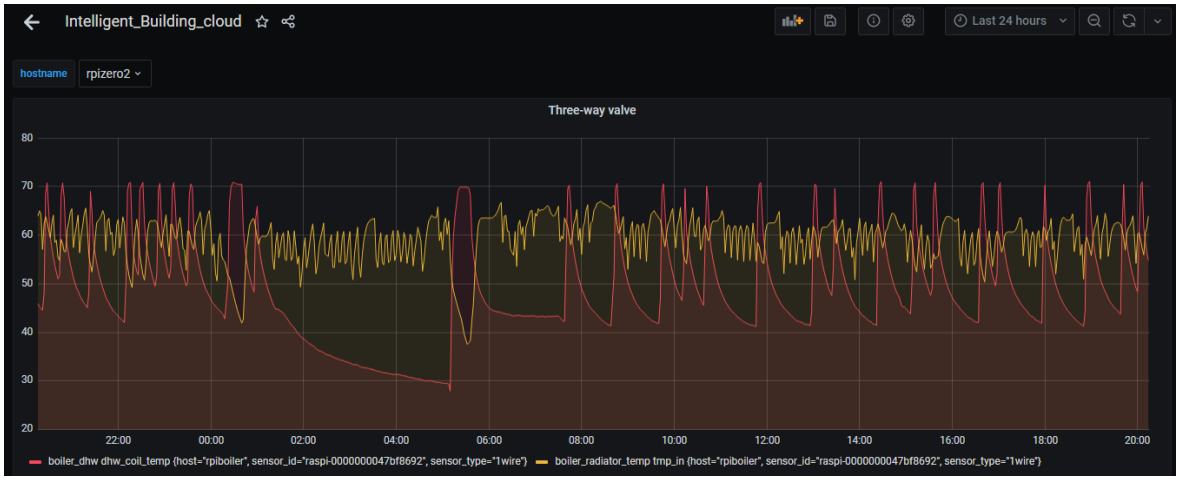
Výsledný graf lze vidět na *obrázku 19*:



Obrázek 19 Graf užitkové vody

Poslední panel kombinuje hodnotu teploty vody pro nahřívání systému radiátorů a ohřevu teplé užitkové vody v jeden graf. Z něj lze vyčíst, zda správně funguje trojcestný ventil kotle. Skript pro vytvoření grafu je následovný:

```
from(bucket: "Sensor_data")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_field"] == "tmp_in"  or r["_field"] == "dhw_coil_temp")
```



Obrázek 20 Graf trojcestného ventilu

4.2.4 Vytvoření řídící aplikace pomocí Django frameworku

Řídící aplikace inteligentní budovy je realizována pomocí webové aplikace, která je spuštěna v cloudovém prostředí Heroku nabízející služby PaaS. Cloudové prostředí Heroku zajišťuje prostředí, ve kterém je možné spouštět webové aplikace. Tyto aplikace můžou být napsané v mnoha programovacích jazycích jako Python, NodeJS, Ruby, Go a další. Nabízí také nástroje, pomocí kterých je možné tyto aplikace přesunout z lokálního vývojového prostředí do Heroku cloudu. [55]

Řídící aplikace je realizována pomocí webového frameworku Django. Tento framework napsaný v Pythonu slouží pro vytváření dynamických webových stránek. Takto vytvořená webová aplikace se skládá z jednoho Django projektu, který rozděluje jednotlivé funkce webové aplikace do Django aplikací. Mezi hlavní obsah těchto Django aplikací patří:

- HTML šablony udává základní tvar jednotlivých stránek webové aplikace.
- Soubor *views.py* zpracovává hodnoty na pozadí aplikace a následně tyto výsledné hodnoty zobrazuje v HTML šablonách.

- Soubor *models.py* vytváří datový model definující strukturu dat uložených v databázi.
- Soubor *forms.py* vytváří formuláře na základě datových modelů. Tyto formuláře jsou vyplněny uživatelem, následně jsou vyplněné hodnoty uloženy do databáze.

Django project obsahuje také složku s nastavením webové aplikace. Mezi hlavní soubory této složky patří:

- Soubor *settings.py* obsahující veškerá nastavení webové aplikace.
- Soubor *urls.py* definuje URL cesty k jednotlivým HTML šablonám, kopie tohoto souboru je většinou také vytvořena v Django aplikacích a poté importována zpět do hlavního souboru.

Django projekt také může obsahovat složku *templates* se základními šablonami. Tyto šablony obsahují prvky, které jsou jednotné pro všechny Django aplikace jako např. navigační menu. Pro formátování HTML pomocí CSS je možné v Django projektu vytvořit složku *static/css,* ve které jsou CSS s vlastním formátováním vzhledu aplikace.

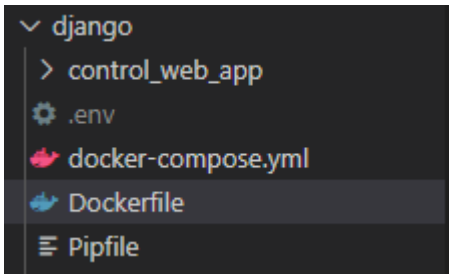
Data o uživatelských účtech, senzorech, místnostech a zařízeních jsou ukládána do PostgreSQL databáze. O veškeré operace s databází a jejími daty jsou realizovány přes Django framework. [56]

Pro vytváření vzhledu aplikace použijeme Bootstrap 5.0 CSS framework, který značně zjednodušuje vytváření vzhledu webové aplikace. [57]

4.2.4.1 Příprava vývojového prostředí

Řídící aplikace je vyvíjena pomocí Docker kontejnerů propojených pomocí Docker-compose, aby byla jednoduše nasaditelná na Heroku. V prvním kontejneru s Python image je spuštěna řídící aplikace založená na Django frameworku, v druhém je spuštěna PostgreSQL databáze, kterou tato řídící aplikace používá.

Pro vytvoření Docker kontejneru řídící aplikace je potřeba vytvořit soubory, které tento kontejner vytvoří. Všechny tyto soubory jsou ukázány na *obrázku 21*.



Obrázek 21 Soubory lokálního vývojového prostředí (zdroj: autor)

Nejdříve je založen soubor Pipfile, který deklaruje, jaké knihovny a jaké verze těchto knihoven řídící aplikace používá. Také je deklarována minimální verze Pythonu nutná pro používání těchto knihoven. Tento soubor je používán balíčkovacím nástrojem Pipenv. Soubor Pipfile je vidět na následující ukázce:

```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
django = "3.1.7"
influxdb-client = "1.15.0"
psycopg2-binary = "*"
dj-database-url = "0.5.0"

[dev-packages]
[requires]
python_version = "3.8"
```

Kromě Django frameworku mezi další používané knihovny patří:

- *influxdb-client* umožňující získat data z InfluxDB cloudu;
- *psycopg2-binary* sloužící pro komunikaci mezi řídící aplikací a PostgreSQL databází;

- *dj-database-url* sloužící pro připojení aplikace k databázi běžící v cloudovém prostředí Heroku.

U každé knihovny je specifikována verze, která má být stažena. V případě *psycoph2-binary* je použita nejnovější verze.

Po založení souboru Pipfile musí být vytvořena složka *control_web_app* obsahující soubory, které umožňují spustit webovou aplikaci. Celá tato složka je vytvořena pomocí příkazů:

```
# instalace knihovny Django, pokud ji nemáme nainstalovanou na lokálním prostředí mimo Docker kontejner
pip install django
# vytvoření souborů framework Django
django-admin startproject control_web_app
```

Tento příkaz vytvoří Django projekt ve složce *control_web_app*, která obsahuje soubor *manage.py*. Tento soubor slouží pro spouštění webové aplikace v Docker kontejneru a dává přístup k příkazům Django frameworku.

Příkaz *django-admin* také vytvoří složku se základním nastavením aplikace. V této složce musí být změněn soubor *settings.py*, ve kterém jsou změněny přihlašovací údaje o databázi, kterou řídící aplikace používá pro ukládání hodnot. Hodnoty v proměnné *DATABASES* jsou změněny následovně:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'postgres',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': 'db',
        'PORT': 5432,
    }
}
```

Následně musí být vytvořen soubor Dockerfile, který udává způsob, jakým bude Docker kontejner vytvořen. Obsah tohoto souboru je následující:

```
1. FROM python:3
2. ENV PYTHONUNBUFFERED=1
3. WORKDIR /app
4. ADD Pipfile /app
5. ADD Pipfile.lock /app
6. ADD ./control_web_app /app/
7. RUN pip install pipenv
8. RUN pipenv install --system --skip-lock
9. CMD python manage.py runserver 0.0.0.0:$PORT
```

Docker kontejner je dle Dockerfile založen na Docker image Python3. Následně na řádku 2 je nastavena proměnná *PYTHONUNBUFFERED*, která zajišťuje výpis výstupu v terminálu. Na řádku 3 je vytvořena pracovní složka, do které jsou na řádcích 4-6 zkopírován soubor *Pipfile* a složka *control_web_app* z hostovacího zařízení. Na řádku 7 je následně nainstalován nástroj Pipenv, pomocí kterého jsou na řádku 8 nainstalovány knihovny specifikované v souboru *Pipfile*. Poslední řádek udává, jakým příkazem je tento Docker kontejner spuštěn.

Po vytvoření souboru Dockerfile musí být vytvořen soubor *docker-compose.yml*, který slouží ke spuštění aplikací skládajících se z více než jednoho Docker kontejneru. V tomto

souboru jsou definovány jednotlivé kontejnery aplikace a jejich vlastnosti. Obsah tohoto souboru je následovný:

```
version: "3.9"
services:
  db:
    image: postgres
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - PGDATA=/var/lib/postgresql/data/pgdata
      - POSTGRES_PORT=5432
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data/pgdata
  web:
    image: michbud98/django-control-web-app
    build: .
    command: python manage.py runserver 0.0.0.0:$PORT
    env_file: .env
    volumes:
      - ./control_web_app:/app
    ports:
      - "$PORT:$PORT"
    depends_on:
      - db

volumes:
  postgres_data:
```

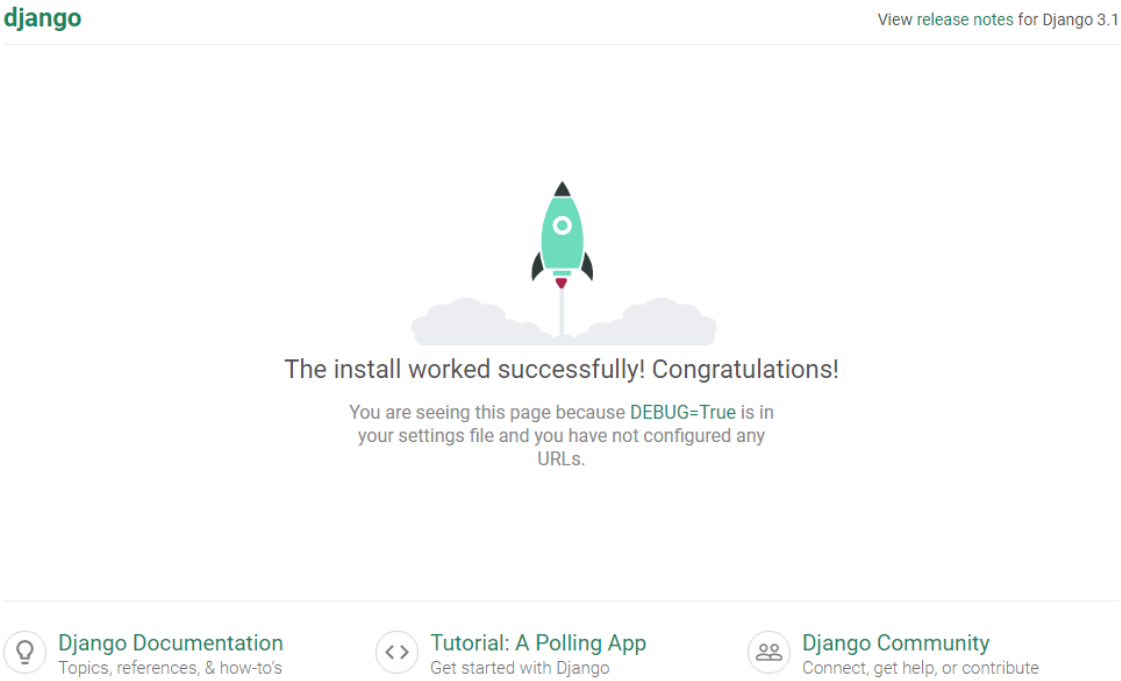
Docker-compose definuje službu s názvem *db*, která spustí Docker kontejner s PostgreSQL databází. V části *environment* je definováno jméno databáze, přihlašovací údaje uživatele, heslo uživatele, kam jsou uložena data databáze v kontejneru a port, na kterém databáze naslouchá. Tyto údaje jsou shodné s nastavením v souboru *settings.py* a umožňují tak řídící aplikaci připojení se k této databázi. V části *ports* je následně port 5432 otevřen pro komunikaci mimo kontejner. Jako poslední je definována složka na hostujícím zařízení, do které jsou ukládána data z tohoto kontejneru.

Druhá služba s názvem *web* spouští kontejner řídící aplikace, který je sestaven pomocí Dockerfile. Naslouchá na portu, který je definován proměnnou *PORT* vytvořenou v souboru „*.env*“.

Po vytvoření souboru Docker-compose lze aplikaci spustit pomocí příkazu:

```
docker-compose up
```

Po spuštění je tato aplikace dostupná na adrese *localhost:<PORT>*. Ukázka výchozí stránky řídící aplikace je vidět na *obrázku 22*.



Obrázek 22 Výchozí stránka řídící aplikace (zdroj: autor)

Nakonec je nutné migrovat data z řídící aplikace do databáze PostgreSQL a vytvořit superuživatele, pomocí kterého lze tuto databázi spravovat. Spuštění migrace a vytvoření superuživatele je vykonáno následujícími příkazy:

```
# Migrace dat do PostgreSQL databáze
docker-compose run web python manage.py makemigrations
docker-compose run web python manage.py migrate
# Vytvoření superuživatele, zadáno jméno uživatele, email a heslo
docker-compose run web python manage.py createsuperuser
```

4.2.4.2 Vytvoření řídící aplikace

Není popsána každá část aplikace, protože to není možné z důvodu rozsahu práce, ale jsou názorně představeny všechny části důležité pro projekt. Funkcionalita řídící aplikace je rozdělena do Django aplikací. Tyto Django aplikace obsahují HTML šablony, které jsou odkazovány základní šablonou *base.html*. Tato základní šablona vypadá následovně:

```
{% load static %}
<!DOCTYPE html>

<html lang="en">
    <head>
        <title>Building control app</title>
        <meta charset="utf-8">
        <meta name="author" content="Michal Budik">
        <!-- Bootstrap css -->
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-
BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl" crossorigi
n="anonymous">
        <link rel="stylesheet" type="text/css" href="{% static '/css/main.cs
s' %}">
    </head>
    <!-- Bootstrap Javascript -->


    <header>
        {% include "navbar.html" %}
    </header>
    <body>
        {% block content %}
        replace me
        {% endblock %}
    </body>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-
b5kHyXgcpbZJO/tY9Ul7kGkf1S0CWuKcCD38l8YkeH8z8QjE0GmW1gYU5S9FOnJ0" crossorigi
n="anonymous">
    </script>
</html>
```

Tato šablona obsahuje ve značce *head* metadata pro používání frameworku Bootstrap a odkaz na Javascript tohoto frameworku. Šablona také obsahuje ve značce *header* odkaz na soubor s navigačním menu *navbar.html*, které je zobrazeno v hlavičce řídící aplikace. Ve značce *body* je následně specifikován odkaz na část, která bude nahrazena šablonami Django aplikací.

Některé Django aplikace pracující s hodnotami senzorů inteligentní budovy se připojují do InfluxDB cloud databáze, která obsahuje data o senzorech. Toto připojení je realizováno pomocí python skriptu *get_influx_data.py*, který obsahuje kromě údajů pro připojení do databáze také funkce pro získání dat z cloudové databáze. Jednotlivé Django aplikace následně tyto funkce volají ve svých python skriptech. Na následujícím příkladě je znázorněna funkce získávající teplotu ze senzoru:

```
def query_last_sensor_temp(sensor_id):
    query = "from(bucket: \"{}\")\
        |> range(start: -1h)\
        |> filter(fn: (r) => r._measurement == \"sensor_temperature\")\
        |> filter(fn: (r) => r._field == \"temperature\")\
        |> filter(fn: (r) => r.sensor_id == \"{}\")\
        |> last()".format(get_bucket(), sensor_id)
    result = query_field_val_from_db(query)
    if not result:
        return None
    else:
        return result[0][1]
```

Vstupní hodnotou této funkce je sériové číslo senzoru. V proměnné *query* je uložen příkaz v jazyce Flux, který je použit ve funkci *query_field_val_from_db* ze souboru *get_influx_data.py*. Tato funkce následně uloží poslední naměřenou teplotu ze senzoru, která je funkcí vrácena.

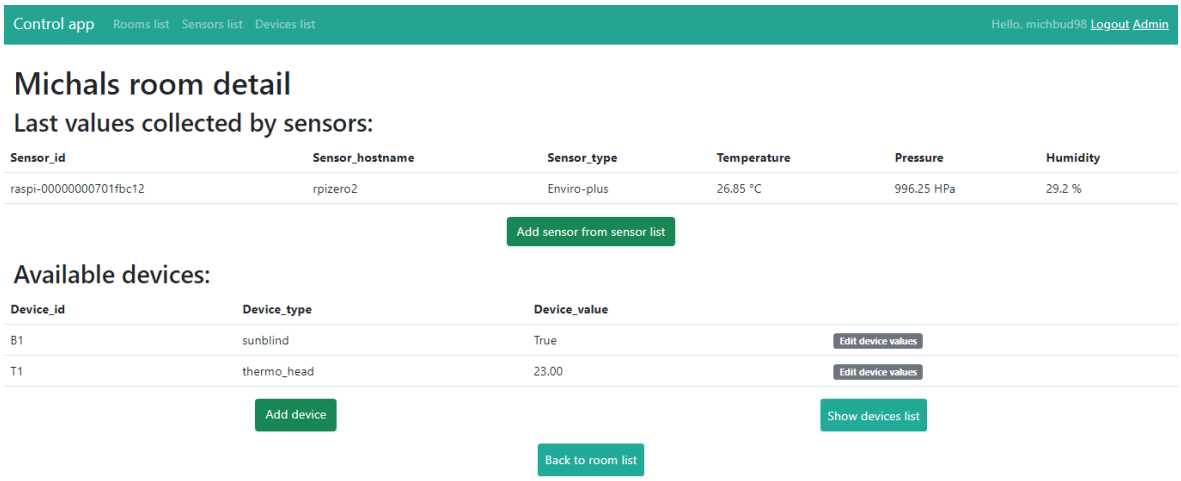
Funkcionalita řídící aplikace bude rozdělena do několika Django aplikací s názvy:

- *main_menu;*
- *room;*
- *sensor;*
- *device*;

- *users*.

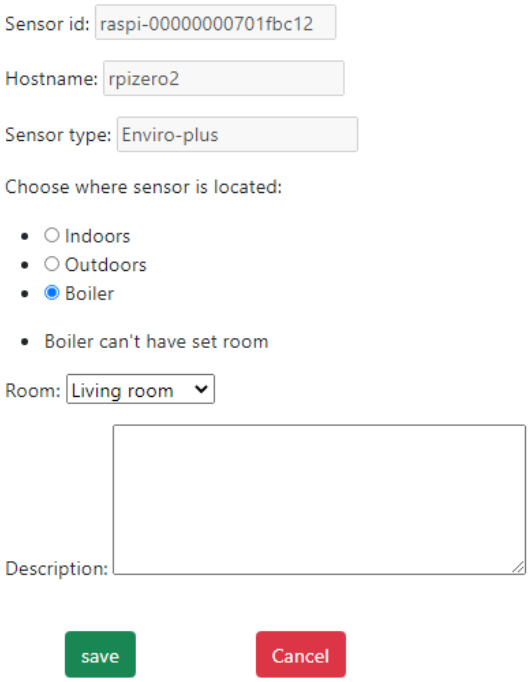
Django aplikace *main_menu* zobrazuje hlavní menu řídící aplikace, ve kterém je možné vidět průměr posledních naměřených hodnot teploty, tlaku a vlhkosti, intenzity světla uvnitř a mimo budovu a také hodnoty naměřené z kotle budovy. Odkazuje také na další Django aplikace.

Django aplikace *room* umožňuje spravovat jednotlivé místnosti budovy uvnitř řídící aplikace. Jednotlivé místnosti lze vytvořit pomocí HTML formuláře, který obsahuje název nově vytvořené místnosti. Název této místnosti je následně uložen do databáze PostgreSQL. U každé místnosti je možné zobrazovat naměřené hodnoty ze senzorů propojených s místností. Lze také ovládat aktivní prvky jako žaluzie a termostatické hlavice, které jsou s místností propojeny. Je možné upravovat názvy místností nebo je z databáze úplně vymazat. Na *obrázku 23* je možné vidět senzory a aktivní prvky připojené k místnosti *Michal's room*.



Obrázek 23 List senzorů a aktivní prvků propojených s místností (zdroj: autor)

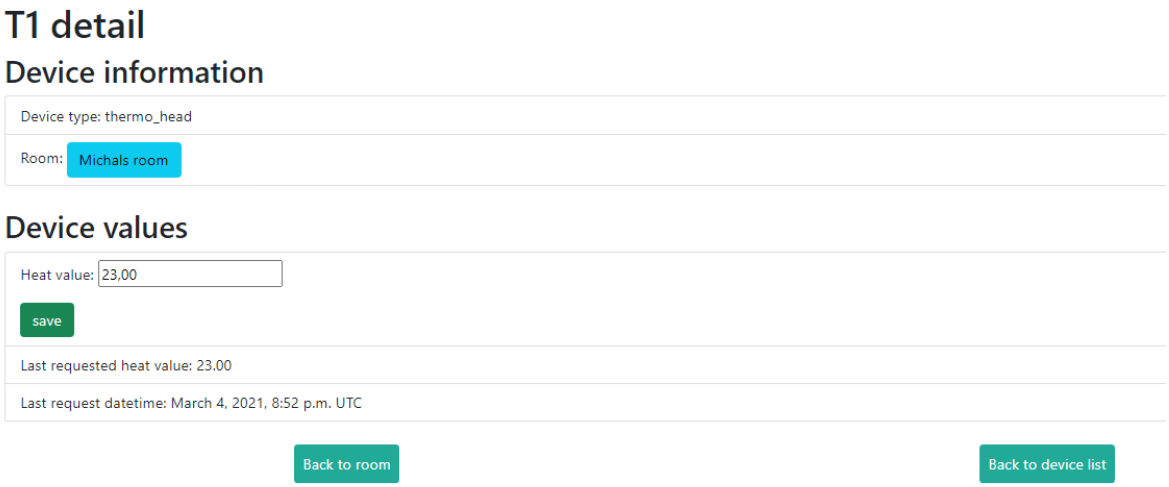
Django aplikace *sensor* slouží k propojování senzorů z cloudové databáze s lokacemi a místnostmi budovy. Na *obrázku 24* je vidět formulář, pomocí kterého jsou senzory přiřazeny k lokaci a místnosti.



Obrázek 24 Propojení senzoru s místností (zdroj: autor)

Tento formulář je zobrazován pomocí šablony *sensor_create.html*. Správné vyplnění této šablony je ověřováno v souboru *forms.py*, který je uložen v Django aplikaci *sensor*. V tomto souboru je např. definováno, že místnost může být nastavena jen v případě, kdy senzor má nastavenou lokaci uvnitř budovy (*indoors*). Na *obrázku 24* je vidět příklad špatně vyplněného formuláře. Po úspěšném vyplnění formuláře jsou tyto senzory uloženy do PostgreSQL databáze.

Django aplikace *device* umožňuje vytvářet aktivní prvky budovy jako žaluzie a termostatické hlavice a následně jim přiřazovat hodnoty. Všechny tyto aktivní prvky jsou zobrazeny v celkovém listu aktivních prvků nebo v listu aktivních prvků pro každou místnost. Tato aplikace také obsahuje funkce pro nastavování hodnot jednotlivých aktivních prvků. Na *obrázku 25* je vidět formulář pro nastavení hodnotu teploty u termostatické hlavice T1.



Obrázek 25 Nastavení hodnot u aktivních prvků (zdroj: autor)

Tato hodnota je následně uložena do PostgreSQL databáze. Aktivní prvek se sériovým číslem T1 následně může získat tuto hodnotu pomocí HTTP GET požadavku, který tato Django aplikace zpracuje.

Všechny tyto Django aplikace, kromě *main_menu*, jsou z bezpečnostních důvodů uzavřené. Uživatel se musí nejdříve před jejich použitím přihlásit. Poslední Django aplikace zvaná *users* zpracovává přihlašování a odhlašování uživatelů. Má-li tento uživatel administrátorská práva, tak tato Django aplikace poskytuje navíc odkaz na administrátorskou Django aplikaci, ve které je možné vytvářet další uživatele a spravovat databázi PostgreSQL.

4.2.4.3 Nasazení na Heroku

Řídící aplikace je nasazena na Heroku cloudu. Její testování a nasazování je automatizováno pomocí postupů DevOps:

1. Continuous Integration
    - Každá nová verze aplikace je po nasazení na GitHub otestována pomocí služby Travis CI.
2. Continuous Deployment
    - Nová verze aplikace je automaticky nasazena na Heroku.
    - Pokud část Continuous Integration skončí chybou, tak Continuous Deployment nenastane.

Protože jsou použity Docker kontejnery, je potřeba se přihlásit do Heroku cloudu a Heroku container registry pomocí příkazů:

```
heroku login
heroku container:login
```

Následující příkazy vytvoří Heroku aplikaci a nahrají do ní Docker kontejner řídící aplikace:

```
heroku create <název aplikace>
heroku container:push web
```

Docker kontejner PostgreSQL nemůže být nahrán do Heroku cloudu. Databáze musí být vytvořena jako přídavek k Heroku aplikaci pomocí příkazu:

```
heroku addons:create heroku-postgresql:hobby-dev
```

Tento příkaz vytvoří v Heroku malou bezplatnou PostgreSQL databázi, kterou řídící aplikace používá místo Docker kontejneru nasazeného v lokálním vývojovém prostředí. Řídící aplikace se v cloudu k této databázi připojí pomocí knihovny *dj_database_url*, která je použita v souboru *settings.py*.

Následně je nutné nastavit hodnoty systémových proměnných, ve kterých je uložen např. bezpečnostní token, pomocí kterého se aplikace připojí do InfluxDB cloudu. Tyto proměnné je možné vytvořit přímo v Heroku nebo pomocí příkazů Heroku CLI:

```
heroku config:set INFLUX_TOKEN=<Influx token>
heroku config:set SECRET_KEY=<Secret key>
heroku config:set DEBUG=False
heroku config:set LOCAL=False
```

Připravenou aplikaci lze nyní spustit a otevřít ve webovém prohlížeči pomocí příkazů:

```
# Spustí aplikaci na heroku cloudu
heroku container:release web
# Otevře aplikaci ve webovém prohlížeči
heroku open
```

4.2.5 Aktivní prvky

Aktivní prvky jako žaluzie nebo termostatické hlavice u radiátorů jsou simulovány pomocí python skriptu s knihovnou *requests*, který umožňuje skriptu komunikovat pomocí protokolu HTTP. Tato simulovaná zařízení zasílají HTTP požadavky na řídící systém, který jim vrací hodnoty:

- Pro termostatické hlavice je vracena teplota, na kterou mají být nastaveny.
- Pro žaluzie je vracen boolean, který říká, zda má být žaluzie otevřená nebo zavřená.

4.3 Testování

Je testován pouze modul řídící aplikace vytvořený pomocí Django frameworku. Ostatní moduly z větší části využívají externí knihovny, a tak je nemá smysl testovat, protože by pak spíš šlo o testování této knihovny.

Django framework má zabudované speciální unit testy, které jsou založené na testovacím frameworku *unittest*. Pomocí těchto testů lze:

- testovat interní chování kódu jako u standardních unit testů;
- testovat interakci uživatele s view (HTML stránkou) pomocí testovacího klienta.

V následujícím kódu je ukázka testu, který používá testovacího klienta a testuje view:

```
from django.test import TestCase, SimpleTestCase
from django.urls import reverse
from django.contrib.auth.models import User
from sensor.models import Sensor
from room.models import Room

class TestViews(TestCase):
    def setUp(self):
        user = User.objects.create(username='testuser')
        user.set_password('12345')
        user.save()
        self.new_room = Room.objects.create(id=1,room_name="test_room")

    def test_sensor_list_view(self):
        self.client.login(username='testuser', password='12345')
        response = self.client.get(reverse("sensor_list"))
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, "sensor_list.html")

    def test_sensor_create_view(self):
        self.client.login(username='testuser', password='12345')
        url = reverse("sensor_create", args=["S3", "sensor3", "STH22"])
        response = self.client.post(url, {
            "location": "indoors",
            "room": 1
        })

        sensor = Sensor.objects.get(sensor_id="S3")
        self.assertEquals(str(sensor), "S3")
        self.assertEquals(sensor.location, "indoors")
        self.assertEquals(sensor.room.id, self.new_room.id)
```

Metoda *setUp* je automaticky volána před každým testem. Slouží pro přípravu prostředí před jednotlivými testy, vytvoří dočasnou databázi a testovací objekty typu *User* a *Room*.

Metoda *test_sensor_create_view* je jeden z unit testů, jehož účelem otestovat, zda je vytvořen nový senzor po zavolání příslušné cesty a zda obsahuje všechny požadované náležitosti. Protože pro použití této cesty je potřeba se v Django aplikaci přihlásit, musí se přihlásit i testovací klient.

Metoda *test_sensor_list_view* je další unit test, jehož účelem je otestovat, zda se po provedení požadavku objeví správná HTML stránka.

5 Výsledky a diskuse

Výsledkem práce je funkční systém inteligentní budovy dle zadaných požadavků. Bylo ověřeno, že systém lze vybudovat s minimální lokální infrastrukturou s IoT prvky rozmístěnými v rámci budovy a s hlavním řídícím systémem nasazeným do cloudového prostředí. Funkcionalitu celého systému je tak možné rozdělit do modulů, které lze realizovat pomocí cloudových služeb.

Prvky IoT jsou v této práci realizovány pomocí mikropočítačů RPI Zero, které jsou vhodnou platformou pro úvodní vývoj a testování IoT prvků. Pro realizaci systému lze najít levnější a bezpečnější řešení. Vývoj IoT se zaměřuje také na využití vhodnějších komunikačních protokolů.

Hlavní řídící systém je realizován v cloudových prostředích Heroku, InfluxDB a Grafana Cloud. Tato cloudová prostředí oproti ostatním nabízejí výhodné podmínky (např. bezplatnou licenci) pro vývoj a praktické nasazení v malých projektech.

Testování a nasazování modulu řídící aplikace inteligentní budovy je automatizováno pomocí postupů DevOps zvaných Continuous Integration a Continuous Deployment. Pro realizaci této automatizace je použita služba Travis CI, která aplikaci otestuje a případně sama nasadí na Heroku cloud.

5.1 Možná vylepšení

V dalším vývoji by bylo vhodné zaměřit se na zlepšení bezpečnosti a funkcionality systému. Pro zlepšení bezpečnosti by místo mikropočítačů RPI Zero bylo možné využít mikrokontrolery jako např. NodeMCU, které nemají plný operační systém a jsou méně náchylné na útoky zvenčí. Dále pro snížení zátěže na lokální internetovou sít a snížení spotřeby energie by bylo dobré místo Wi-Fi využít vhodnější protokol pro komunikaci na lokální síti jako např. MQTT nebo ZigBee. Dále pro lepší bezpečnost by bylo možné omezit počet dotazů na databázi řídící aplikace např. dotaz pro získání dat pro jeden senzor by bylo možné poslat jednou za určité časové rozmezí.

Také by bylo v budoucnu dobré vytvořit funkci, která upozorní uživatele, pokud např. senzor dlouho nepošle data do databáze. Dále by bylo vhodné pro zlepšení funkcionality systému přenést grafickou reprezentaci dat přímo do řídící aplikace pro vyšší přehlednost.

6 Závěr

Cílem práce bylo navrhnout a implementovat systém pro řízení inteligentní budovy s maximálním využitím služeb cloudového prostředí. Realizovaná ukázka systému inteligentní budovy s využitím služeb veřejného cloudu tento cíl naplnila.

V rámci teoretických východisek byli čtenáři seznámeni s problematikou cloudů, kde byly ukázány typy cloudů, jejich využití a způsob vývoje softwaru pro cloudové prostředí. Byl také popsán koncept IoT jako řešení pro realizaci senzorů fyzikálních veličin. Následně byl popsán koncept inteligentní budovy a příklady její implementace.

Vlastní práce se zabývala analýzou funkcí inteligentní budovy a na jejím základě byl systém rozdělen do dvou funkčních celků. Jeden funkční celek byl tvořen lokálními zařízeními IoT umístěnými v budově a druhý funkční celek byl tvořen hlavním řídícím systémem umístěným v prostředí veřejného cloudu. Řídící systém byl následně rozdělen na moduly, které byly realizovány cloudovými službami.

Modul řídící aplikace inteligentní budovy byl realizován v jazyce Python pomocí webového frameworku Django. Tento webový framework se osvědčil pro svou robustnost a velmi kvalitní dokumentaci. Testování a nasazování tohoto modulu na cloud bylo automatizováno pomocí postupů DevOps.

7 Seznam použitých zdrojů

[1] JOHNSTON, Sam. *Diagram showing overview of cloud computing* [online]. 3. březen 2009 [vid. 2021-02-21]. Dostupné z: https://commons.wikimedia.org/wiki/File:Cloud_computing.svg

[2] GOODWILL COMMUNITY FOUNDATION. Computer Basics: Understanding the Cloud. *GCFGlobal.org* [online]. 1998-2020 [vid. 2021-02-21]. Dostupné z: https://edu.gcfglobal.org/en/computerbasics/understanding-the-cloud/1/

[3] CLOUDFLARE, INC. What is the Cloud? | Cloud Definition. *Cloudflare* [online]. 2021 [vid. 2021-02-21]. Dostupné z: https://www.cloudflare.com/learning/cloud/what-is-the-cloud/

[4] LABERIS, Bill. *What Is the Cloud?* [online]. 2019 [vid. 2021-02-22]. Dostupné z: https://www.oreilly.com/library/view/what-is-the/9781492052913/

[5] FRANKENFIELD, Jake. How Cloud Computing Works. *Investopedia* [online]. 28. červenec 2020 [vid. 2021-02-22]. Dostupné z: https://www.investopedia.com/terms/c/cloud-computing.asp

[6] PATRIZIO, Andy. What is Multi-Tenant Architecture? *Datamation* [online]. 12. únor 2021 [vid. 2021-02-22]. Dostupné z: https://www.datamation.com/cloud/what-is-multi-tenant-architecture/

[7] *The components of a cloud infrastructure* [online]. 2021 2010. Dostupné z: https://cdn.ttgtmedia.com/rms/onlineImages/cloud_infra_components.jpg

[8] STOČES, Jan. *Je cloud bezpečný? | AIMagazine* [online]. 4. duben 2019 [vid. 2021-02-22]. Dostupné z: https://www.aimtecglobal.com/aimagazine/je-cloud-bezpecny/

[9] PENNETA, Kasey. *Is the Cloud Secure?* [online]. 10. září 2019 [vid. 2021-02-22]. Dostupné z: //www.gartner.com/smarterwithgartner/is-the-cloud-secure/

[10] MICROSOFT. *Veřejný cloud versus privátní cloud versus hybridní cloud | Microsoft Azure* [online]. 2021 [vid. 2021-02-22]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-are-private-public-hybrid-clouds/

[11] IBM CLOUD EDUCATION. *What is Private Cloud?* [online]. 10. září 2020 [vid. 2021-02-22]. Dostupné z: https://www.ibm.com/cloud/learn/introduction-to-private-cloud

[12] HARANAS, Mark. Businesses Moving From Public Cloud Due To Security, Says IDC Survey. *CRN* [online]. 13. srpen 2018 [vid. 2021-02-22]. Dostupné z: https://www.crn.com/businesses-moving-from-public-cloud-due-to-security-says-idc-survey

[13] TUTORIALS POINT INDIA LIMITED. *Community Cloud Model - Tutorialspoint* [online]. 2021 [vid. 2021-02-23]. Dostupné z: https://www.tutorialspoint.com/cloud_computing/cloud_computing_community_cloud_model.htm

[14] TELRAJA, Manjula. Emerging Cloud Models: Community Cloud. *Cisco Blogs* [online]. 16. květen 2014 [vid. 2021-02-23]. Dostupné z: https://blogs.cisco.com/datacenter/emerging-cloud-models-community-cloud

[15] CHEN, Jiadong. Azure Fundamental: IaaS, PaaS, SaaS. *Medium* [online]. 19. červenec 2020 [vid. 2021-02-23]. Dostupné z: https://medium.com/chenjd-xyz/azure-fundamental-iaas-paas-saas-973e0c406de7

[16] MICROSOFT. *Co je IaaS? Infrastruktura jako služba | Microsoft Azure* [online]. 2021 [vid. 2021-02-23]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-iaas/

[17] STEDMAN, Josh. *Difference between SaaS Paas and IaaS | UnderCtrl IT* [online]. 26. květen 2017 [vid. 2021-02-23]. Dostupné z: https://underctrl.com.au/it-managed-services/difference-between-saas-paas-iaas/

[18] MICROSOFT. *Co je PaaS? Platforma jako služba | Microsoft Azure* [online]. 2021 [vid. 2021-02-23]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-paas/

[19] MICROSOFT. *Co je SaaS? Software jako služba | Microsoft Azure* [online]. 2021 [vid. 2021-02-23]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-saas/

[20] TANKOVSKA, H. Number of Gmail active users 2018. *Statista* [online]. 25. leden 2021 [vid. 2021-02-23]. Dostupné z: https://www.statista.com/statistics/432390/active-gmail-users/

[21] CLOUD CARIB. *Cloud Storage vs. Cloud Computing: What's the Difference?* [online]. 2018 [vid. 2021-02-22]. Dostupné z: https://info.cloudcarib.com/blog/cloud-storage-vs.-cloud-computing-whats-the-difference

[22] HERMAN. The Difference Between Cloud Storage and Cloud Computing. *computertech* [online]. 23. srpen 2020 [vid. 2021-02-23]. Dostupné z: https://computertech.com/blog/difference-cloud-storage-computing

[23] CNFC. CNFC Cloud Native Definition v1.0. *GitHub* [online]. 11. červen 2018 [vid. 2021-02-24]. Dostupné z: https://github.com/cncf/toc

[24] PATRIZIO, Andy. What is cloud-native? The modern way to develop applications. *InfoWorld* [online]. 14. červen 2018 [vid. 2021-02-24]. Dostupné z: https://www.infoworld.com/article/3281046/what-is-cloud-native-the-modern-way-to-develop-software.html

[25] VMWARE, Inc or its affiliates. *What are Cloud Native Applications?* [online]. 9. březen 2020 [vid. 2021-02-24]. Dostupné z: https://tanzu.vmware.com/cloud-native

[26] MICROSOFT. *Defining Cloud Native* [online]. 2021 [vid. 2021-02-24]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/definition

[27] JANSSEN, Thorben. What is Cloud-Native? Is It Hype or the Future of Software Development? *Stackify* [online]. 7. únor 2018 [vid. 2021-02-24]. Dostupné z: https://stackify.com/cloud-native/

[28] DOCKER. *What is a Container? | App Containerization | Docker* [online]. 2021 [vid. 2021-02-24]. Dostupné z: https://www.docker.com/resources/what-container

[29] GNATYK, Romana. *Microservices vs Monolith: which architecture is the best choice for your business?* [online]. 3. říjen 2018 [vid. 2021-02-24]. Dostupné z: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/

[30] THE LINUX FOUNDATION. What is Kubernetes? *Kubernetes* [online]. 2021 [vid. 2021-02-24]. Dostupné z: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[31] REDHAT. *What is a Kubernetes cluster?* [online]. 2021 [vid. 2021-02-24]. Dostupné z: https://www.redhat.com/en/topics/containers/what-is-a-kubernetes-cluster

[32] LABERIS, Bill. *What Is Cloud Native* [online]. B.m.: O'Reilly Media, Inc., 2019. ISBN 978-1-4920-5512-9. Dostupné z: https://www.oreilly.com/library/view/what-is-cloud/9781492055136/

[33] CLOUDSINE PTE LTD. Case Study: How Netflix uses Cloud for Innovation, Agility and Scalability. *Cloudsine* [online]. 2020 [vid. 2021-02-25]. Dostupné z: https://www.cloudsine.tech/news-trends/case-study-how-netflix-uses-cloud-for-innovation-agility-and-scalability/

[34] HEMEL, Zef. How Netflix Deploys Code. *InfoQ* [online]. 13. červen 2013 [vid. 2021-02-25]. Dostupné z: https://www.infoq.com/news/2013/06/netflix/

[35] RVIS. *Projekt Příprava vybudování EGovernment cloudu* [online]. 2018 [vid. 2021-02-24]. Dostupné z: https://www.mvcr.cz/soubor/souhrnna-analyticka-zprava-projektu-egovernment-cloud.aspx

[36] ATLASSIAN. What is DevOps? *Atlassian* [online]. 2021 [vid. 2021-03-04]. Dostupné z: https://www.atlassian.com/devops

[37] AMAZON WEB SERVICES. What is DevOps? - Amazon Web Services (AWS). *Amazon Web Services, Inc.* [online]. 2021 [vid. 2021-03-04]. Dostupné z: https://aws.amazon.com/devops/what-is-devops/

[38] INTELLIPAAT.COM. *DevOps Tutorial – Learn DevOps from Experts – Intellipaat* [online]. 2011-2021 [vid. 2021-03-04]. Dostupné z: https://intellipaat.com/blog/tutorial/devops-tutorial/

[39] AMAZON WEB SERVICES. What is Continuous Integration? – Amazon Web Services. *Amazon Web Services, Inc.* [online]. 2021 [vid. 2021-03-04]. Dostupné z: https://aws.amazon.com/devops/continuous-integration/

[40] AMAZON WEB SERVICES. What is Continuous Delivery? – Amazon Web Services. *Amazon Web Services, Inc.* [online]. 2021 [vid. 2021-03-04]. Dostupné z: https://aws.amazon.com/devops/continuous-delivery/

[41] HANES, David, Gonzalo SALGUEIRO a Rob BARTON. *IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things [Book]* [online]. 2017 [vid. 2021-02-26]. ISBN 978-0-13-430709-1. Dostupné z: https://www.oreilly.com/library/view/iot-fundamentals-networking/9780134307091/

[42] PIXMAN, s.r.o. *Co je to IoT? Jak funguje, kde se využívá a jak se vyvíjel?* [online]. 27. leden 2020 [vid. 2021-02-26]. Dostupné z: https://www.iotport.cz/iot-novinky/ostatni-clanky-o-iot/co-to-je-iot

[43] IMMAX WPB CZ, s.r.o. *ZigBee vs. WiFi při tvorbě chytré domácnosti | Immax.cz* [online]. 2021 [vid. 2021-02-28]. Dostupné z: https://www.immax.cz/clanky/detail/zigbee-vs-wifi-pri-tvorbe-chytre-domacnosti.htm

[44] ALZA, a.s. ZigBee. *Alza* [online]. 2021 [vid. 2021-02-28]. Dostupné z: https://www.alza.cz/zigbee

[45] A.S, MEDIA FACTORY Czech Republic. *Inteligentní budovy - Studijní programy - Inženýr - Zájemce o studium | FSv ČVUT* [online]. 2021 [vid. 2021-03-04]. Dostupné z: https://web.fsv.cvut.cz/zajemce-o-studium/inzenyr/programy-a-obory/inteligentni-budovy/

[46] PRAVDA, Ivan. *Inteligentní budovy* [online]. 2015 [vid. 2021-03-04]. Dostupné z: https://publi.cz/books/239/01.html

[47] VOŠ A SPŠ KUTNÁ HORA. *Inteligentní budovy* [online]. 2021 [vid. 2021-03-04]. Dostupné z: http://www.edumat.cz/texty/Rizeni_budov6.pdf

[48] JAGA MEDIA, S.R.O. Inteligentní budovy využívají moderní technologie. *ASB Portal* [online]. 19. srpen 2008 [vid. 2021-03-04]. Dostupné z: https://www.asb-portal.cz/stavebnictvi/technicka-zarizeni-budov/vetrani-a-klimatizace/inteligentni-budovy-vyuzivaji-moderni-technologie

[49] INFLUXDATA. InfluxDB Cloud — the most powerful time series database as a service. *InfluxData* [online]. 2021 [vid. 2021-03-02]. Dostupné z: https://www.influxdata.com/products/influxdb-cloud/

[50] THE RASPBERRY PI, Foundation. Buy a Raspberry Pi Zero W. *Raspberry Pi* [online]. 2021 [vid. 2021-02-28]. Dostupné z: https://www.raspberrypi.org/products/raspberry-pi-zero-w/

[51] RPISHOP.CZ. *Enviro+ pro Raspberry Pi* [online]. 2021 [vid. 2021-02-28]. Dostupné z: https://rpishop.cz/hat/1895-enviro-pro-raspberry-pi.html

[52] RPISHOP.CZ. Enviro pro Raspberry Pi. *RPishop.cz* [online]. 2021 [vid. 2021-02-28]. Dostupné z: https://rpishop.cz/hat/3222-enviro-pro-raspberry-pi.html

[53] RPISHOP.CZ. Enviro pHAT. *RPishop.cz* [online]. 2021 [vid. 2021-02-28]. Dostupné z: https://rpishop.cz/hat/734-enviro-phat.html

[54] GRAFANA LABS. *Grafana Cloud* [online]. 2021 [vid. 2021-03-03]. Dostupné z: https://grafana.com/products/cloud/

[55] SALESFORCE.COM. *What is Heroku | Heroku* [online]. 2021 [vid. 2021-03-05]. Dostupné z: https://www.heroku.com/what

[56] DJANGO SOFTWARE FOUNDATION. *The Web framework for perfectionists with deadlines | Django* [online]. 2015-2021 [vid. 2021-03-05]. Dostupné z: https://www.djangoproject.com/

[57] OTTO, Mark, Jacob THORNTON a BOOTSTRAP. *Bootstrap* [online]. 2021 [vid. 2021-03-05]. Dostupné z: https://getbootstrap.com/

8 Přílohy

Aplikace se zdrojovými kódy je dostupná v přiloženém zip souboru.