VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

SYSTÉM SBĚRU DAT V PRŮMYSLU
INDUSTRIAL DATA COLLECTION SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Lukáš Hvizdák

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. Zdeněk Bradáč, Ph.D.

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Lukáš Hvizdák ***ID:*** 177508

Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Systém sběru dat v průmyslu

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte systém bezpečného sběru dat z PLC do serveru.

1. Proveďte literární a internetovou rešerši.
2. Navrhněte koncepci komunikace a koncepci serverového systému tak, aby odpovídal zvolené koncepci. Zaměřte se především na licenční a cenovou politiku.
3. Navrhněte a implementujte vhodný bezpečný komunikační protokol. Implementujte vhodný databázový systém a zrealizujte programové vybavení PLC i serveru pro využití tohoto protokolu.
4. Implementujte SW pro vizualizaci dat. Popište implementaci a případnou úpravu vizualizace.
5. Popište a charakterizujte datové toky a možnosti systému sběru dat.
6. Zhodnoťte dosažené výsledky, programové vybavení a vizualizaci.

DOPORUČENÁ LITERATURA:

Pavel Herout: Učebnice jazyka C, KOPP, 2004, IV. přepracované vydání, ISBN 80-7232-220-6

Dle pokynů vedoucího práce.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: doc. Ing. Zdeněk Bradáč, Ph.D.

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

Abstrakt

Diplomová práce se zaměřuje na návrh a realizaci sběru dat z výroby pomocí PLC do SQL databáze umístněné v cloudu a následnou vizualizaci. Práce popisuje použitelné komunikační protokoly MQTT a OPC UA s tím, že byl vybrán protokol MQTT. Zabývá se zabezpečením přenosu dat z linky do cloudu pomocí TLS protokolu. Jsou zde popsány jednotlivé služby cloudu a jejich možnosti pro sběr dat. Práce se zabývá možnostmi vizualizace dat za použití existujících open source řešení a rozdíly mezi nimi. Popisuji možnosti úpravy prostředí open source projektu Grafany. Jsou prezentovány reálné dashboardy z výroby. Uvedený systém sběru dat byl nasazen do dvou provozů za účelem testování.

Klíčová slova

Sběr dat, PLC, Azure, IoT Hub, IoT Edge, Docker, S7 1200, Stream Analytics, Azure Functions, SQL, NonSQL, Kibana, Grafana.

Abstract

The master thesis focuses on the design and implementation of data collection from production using a PLC into an SQL database located in the cloud and subsequent visualization. The work describes the applicable communication protocols MQTT and OPC UA with the fact that the protocol MQTT was selected. It deals with securing data transfer from the line to the cloud using the TLS protocol. The individual cloud services and their possibilities for data collection are described here. The work deals with the possibilities of data visualization using existing open source solutions and the differences between them. I describe the possibilities of modifying the open source environment of the Grafany project. Real dashboards from production are presented. The data collection system was deployed in two plants for testing.

Keywords

Data Acquisition, PLC, Azure, IoT Hub, IoT Edge, Docker, S7 1200, Stream Analytics, Azure Functions, SQL, NonSQL, Kibana, Grafana.

Bibliografická citace

HVIZDÁK, Lukáš. *Systém sběru dat v průmyslu* [online]. Brno, 2020 [cit. 2020-06-01]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/127007. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Zdeněk Bradáč.

Prohlášení

Prohlašuji, že svoji diplomovou práci na téma „SYSTÉM SBĚRU DAT V PRŮMYSLU“ jsem vypracoval samostatně pod vedením vedoucího semestrálního projektu a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedeného diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení §11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne …….

Lukáš Hvizdák

Poděkování

Děkuji vedoucímu diplomové práce doc. Ing. Zdeňku Bradáčovi, Ph.D.za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

Dále chci poděkovat vedoucímu projektu Ing. Petrovi Pokornému za vedení projektu ve společnosti kde pracuji. Za jeho odborné rady a vedení práce.

V Brně dne: **31. května 2020**

……
podpis autora

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

S větší dostupností výpočetní techniky a většími požadavky na efektivitu výroby s postupným snižováním ceny výrobků je vyvíjen tlak na výrobní podniky, aby implementovaly tzv. Průmysl 4.0 do své výroby. Označení Průmysl 4.0 v sobě skrývá inovace a proměny výrobních procesů za použití digitalizace a kompletního propojení, automatizace veškerých výrobních procesů a také služeb s nimi spojených. Proto mluvíme o tzv. „chytrých továrnách“. Průmysl 4.0 se tak vyznačuje automatickým řízením opakujících se činností, ale také přístupem k veškerým informacím nutným k výrobě daného produktu a jeho aktuálním postupem výrobou. Tyto informace pak mohou sloužit k optimalizaci výroby, implementaci nových výrobních postupů—nebo k rychlejšímu zavedení nového výrobního procesu atd. Tento trend tak vyžaduje přístup k datům přímo z výrobních zařízení podílejících se na výrobě. Základem je tedy nejdříve data získat, vyhodnotit, a na základě toho provádět změny ve výrobním procesu.

V dnešní tobě se tak objevují nová řešení získávaní, reprezentace dat z výroby a jejích archivování. Často je tak vyvíjen tlak na integrátory ze strany výrobních podniků k implantaci systému pro sběr dat na nová, ale i stávající zařízení jako první krok k dosažení autonomie výroby a variability popisované v Průmyslu 4.0.

Integrátoři mají pak na výběr ze dvou možností, jak tomuto trendu vyhovět. Buď implementací řešení třetí strany anebo vyvinutí vlastního řešení, které můžou nabízet společně s novou implementací linky nebo zařízení.

V této práci se tak budu zabývat popisem řešení sběru dat za pomoci již hotových open source řešení a služeb poskytovaných cloudem, která umožňuji použití pro redistribuci a komerční účely. Popisu dva komunikační protokoly vhodné pro posílání dat do cloudu a implementaci jednoho z nich v řešení. Dále se věnuji knihovně s LMQTT pro PLC a aplikaci napsané .NET Core knihovně pro MQTT komunikaci. Práce zahrnuje konfigurace serverů a programového vybavení těchto serverů. Popisuji jednotlivé služby použité v řešení a případné rozdíly mezi službami, které se mohou navzájem svojí funkcionalitou zastoupit. Přiblížil jsem možnosti úpravy vizualizačního nástroje Grafana a řešil teoretické limity systému. Práce se zabývá i možnostmi jak celou komunikaci zabezpečit a zabránit tak zneužití dat.

V průběhu práce se z projektu stal produkt, který se nabízí zákazníkům a dál se vyvíjí. Proto zde nemohu odhalit celou práci se všemi zdrojovými kódy ani konkrétním popisem jednotlivých řešení.

2 Existující řešení

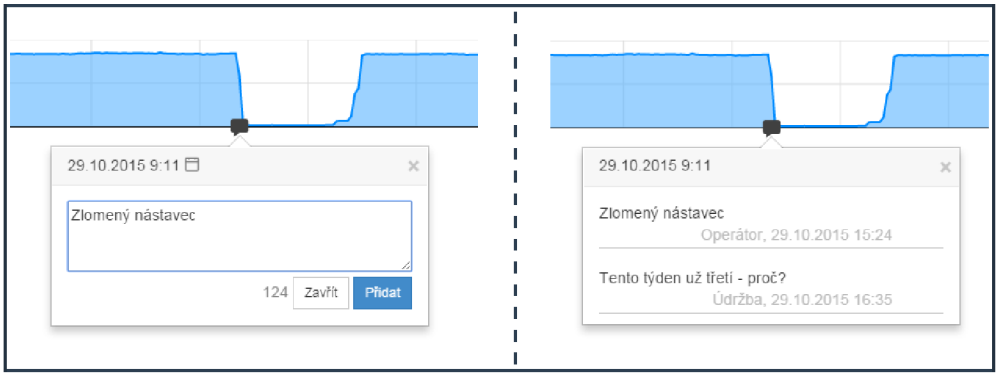
2.1 Plantyst

Plantys umožňuje monitorování technologií a jednoduchý shop floor managment. Data zobrazuje v lineárním grafu s možností vkládání textové anotace viz. obrázek 1. Společnost nabízí kompletní řešení tzn. jejich vlastní HW a SW včetně vlastního cloudového řešení pro ukládání dat.

Společnost Plantyst dodává vlastní hardware formou pronájmu. Plantys box je určený pro sběr dat přímo ze strojů zákazníka. Plantyst box je průmyslový mikropočítač umístěný v černé ocelové krabici, sloužící k monitorování výrobních zařízení. Monitorování probíhá skrze digitální vstupy a měří se počet tiků stroje. Jeden tik by tak měl reprezentovat jeden vyrobený kus.

Data z výroby zasílá po síti přes HTTPS protokol do vzdálených serverů Plantystu, kde jsou data-archivována.

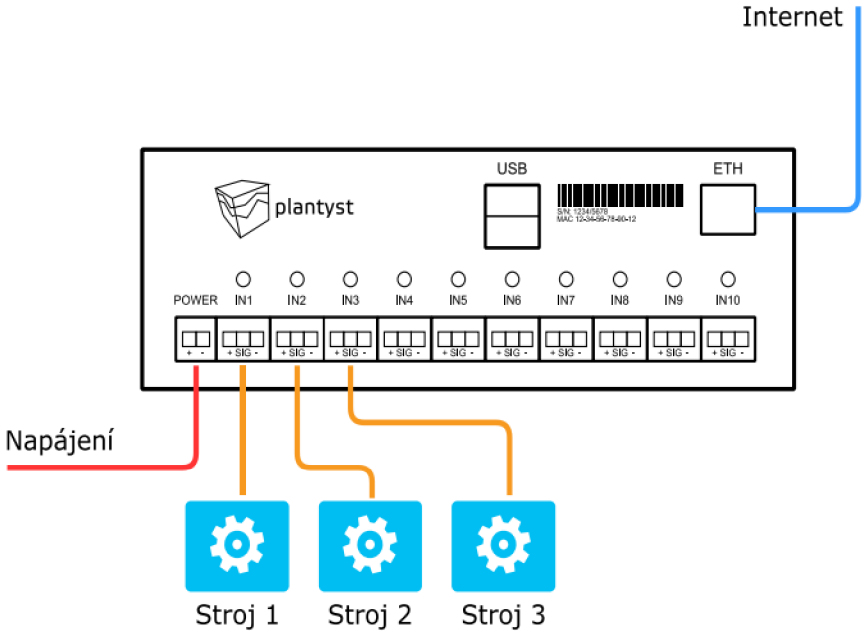
Zobrazení dat je možné skrze webové rozhraní s přihlášením v zákaznické sekci.



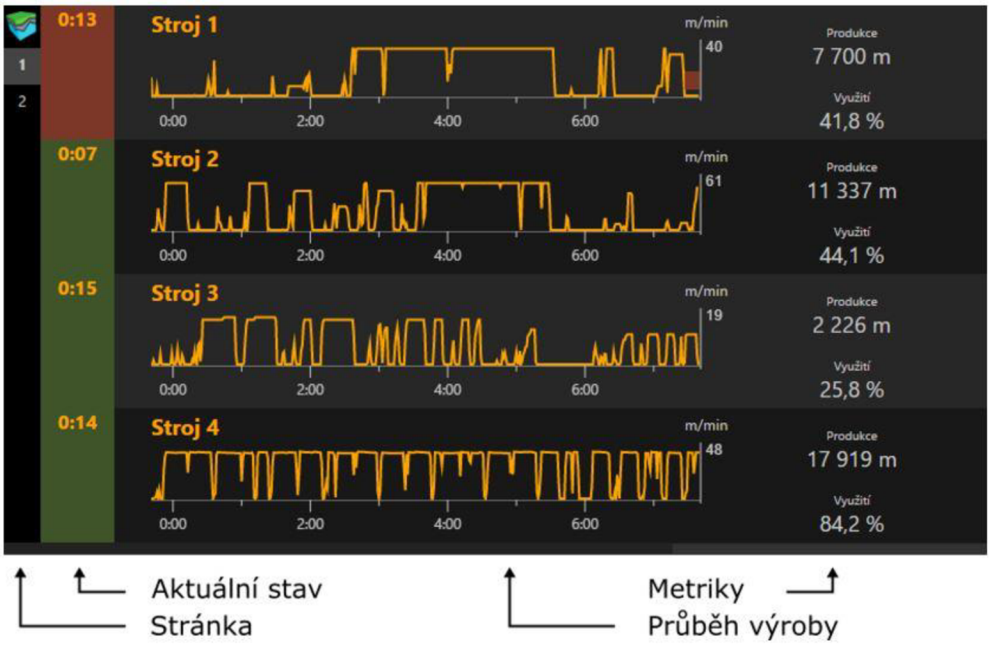
Obrázek 2.1-1 Graf a anotace plantys

Na obrázku 2-1 je vidět detail grafu z Plantyst webového rozhraní, které poskytuje svým zákazníkům na základě měsíčního poplatku. Obrázek 2-2 zobrazuje schématické napojení Plnatyst boxu do výroby. Jedná se o vcelku jednoduchou instalaci do výroby. S jednoduchostí instalace je ale spojena i jednoduchost a zároveň omezenost zobrazovaných dat.

Řešení od společnosti Plantyst spíše slouží částečně jako směr pro vývoj vizualizace, ale zároveň je nutné se vyhnout omezenosti systému kvůli těžko odhadnutelnému směru vývoje technologií v příštích letech, které budou potřebné do budoucna implementovat.



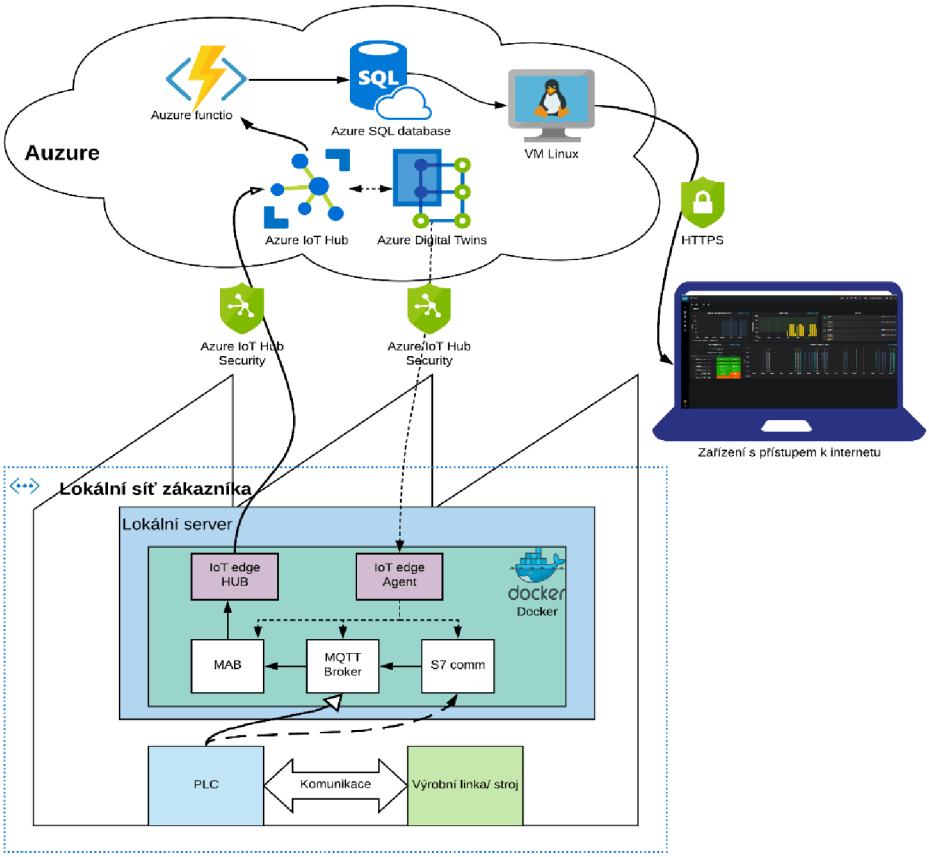
Obrázek 2.1-2 Plnatyst box - schéma komunikace



Obrázek 2.1-3 Grafické rozhraní plantyst

3 Návrh řešení

Sběr dat byl při návrhu rozdělený do několika úrovní podle hierarchie dat a provozu, kdy za nejnižší úroveň je považovaná linka a její řízení. To znamená kolekce dat přímo z linky nebo stroje a posílání do vyšších vrstev. Další vrstvou je kolekce dat z vícero linek a strojů a jejich jednotné zaslání do cloudu nebo do lokální databáze. Třetí vrstva je tak samotná databáze a manipulace dat v rámci databáze. Poslední vrstvou je zobrazení nasbíraných dat. Rozhodnutí pro cloud Azure od Microsoftu bylo v podstatě podmíněno pouze doporučením pracovníkem analytické společnosti, se kterou se zvažovala v budoucnu spolupráce. Mně osobně u Azure oslovilo množství návodů pro jednotlivé služby, které poskytují, ale i 101 řešení sběru dat pro domácnost s IoT snímač za pomoci služeb na cloudu. Hlavní rozdíl sběru dat z průmyslu vůči domácnosti je v samotném zdroji dat a jak se daty nakládáno po stránce bezpečnostní. Azure nabízí velké množství služeb, které se můžou v budoucnu zapojit do řešení a zároveň mají dobře zpracovanou dokumentaci dostupnou zadarmo online. V návaznosti na projekt sběru dat bude projekt prediktivní údržba pomocí umělé inteligence nebo inteligentní plánovaní výroby s využitím MES systému již integrovaných u zákazníka. Na cloudovém uložišti Azure jsou tak nejen data zákazníka ale budou i tam virtuální servery služby spojené se strojovým učením jako je služba Machine learning.



Obrázek 2.1-1 Návrh řešení sběru dat

4 Cesta dat ze zařízení do Cloudu

Tvorba návrhu probíhala od nejnižší vrstvy systému směrem nahoru. To znamená nejdříve se řešili možnosti, jak získat data z linky a poté jakým způsobem je dostat do databáze. Důraz je kladený na flexibilitu řešení v podobě možnosti implementace na stávající linky ale i na nová zařízení dodávané společnosti kde pracuji. Podle toho se vybíralo PLC pro nultou (testovací) verzi. Rozhodovalo se především mezi PLC od Siemens nebo B&R kvůli nativní podpoře MQTT klienta na těchto zařízeních. Vzhledem k přihlédnutí k ekonomické stránce a firemnímu standardu bylo rozhodnuto pro PLC Siemens řady S7 1200 nebo S7 1500. Do budoucna se bude rozšiřovat podpora na ostatní značky PLC a podpoře dalších komunikačních standardů jako je OPC UA.

Scénáře implementace:

- Zákazník má zdrojový kód k PLC řady S7 1200 nebo 1500.
  Implementuje se LMQTT modul a data se posílají přímo ze stávajícího PLC.
- Zákazník nemá zdrojový kód.
  Nutná implementace vlastního PLC. Data jsou sbírána pomoci digitálních a analogových vstupů PLC.
- Zákazník má starší PLC řady S7 se zdrojovým kódem.
  Bude do budoucna upravena aplikace tak aby běžela jako kontejner pod Dockerem. Aplikace překládá komunikační protokol S7 na MQTT.
- Zákazník sbírá data ukládá je do vlastní databáze. V tomto případě se řeší most mezi zákazníkem a cloudem kde se navazuje řešení sběru dat.

4.1 OPC UA

OPC UA (Open Platform Communications – Unified Architecture) je otevřená komunikační platforma sloužící jak pro komunikaci mezi stroji, tak hlavně pro vertikální komunikaci stroje do cloudu. OPC UA je takzvaně na platformě neutrální komunikační standard. Znamená to, že OPC UA může využít stávající komunikační protokoly definující způsob přenosu dat jako je například PROFINET, TCP/IP a další. Komunikace v standardu OPC UA je chápana jako síťová komunikace proto musí obsahovat mechanismy zaručující bezpečnost přenosu. Je použito autentizace[1], autorizace[2] a šifrování s použitím certifikátu. [1]

[1] Autentizace – Proces ověření totožnosti pomocí přihlášení uživatele.
[2] Autorizace – Proces kontroly, zda daný autentizovaný uživatel má oprávnění něco udělat.

OPC UA má 4 úrovně zabezpečení komunikace:

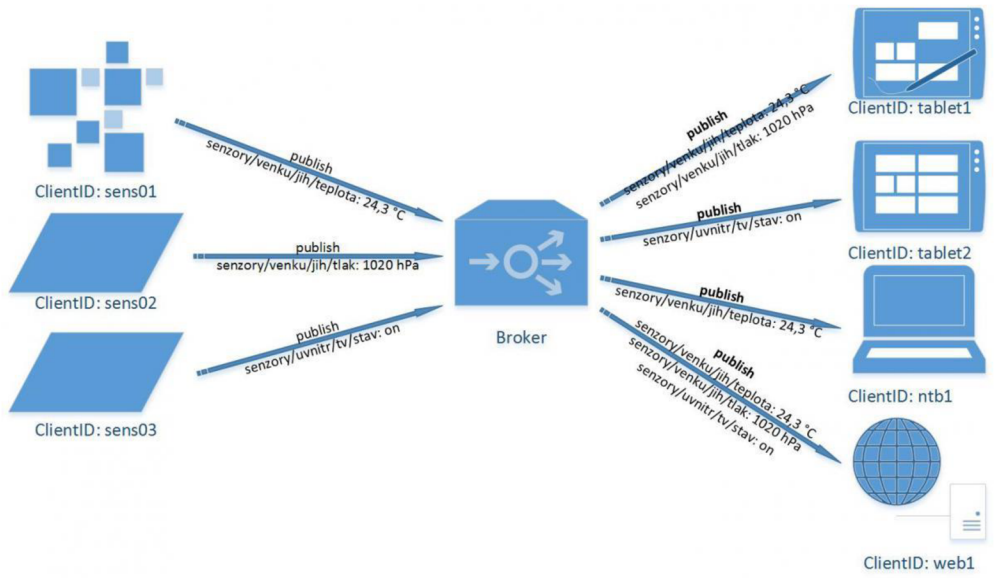
- Úroveň 1 – bez autentizace. Klient i server umožňuji komunikovat s kýmkoliv. Všechny platné certifikáty jsou tak považovány za důvěryhodné. To znamená, že obě strany komunikace automatický uznávají platné certifikáty za důvěryhodné i když nejsou zařazeny na seznam platných certifikátu. [1]
- Úroveň 2 – serverová autentizace. Server dovolí připojení jakéhokoliv klienta a ověření se provádí pouze pomoci jména a hesla. Každý klient musí důvěřovat serverovému certifikátu. Pokud není certifikát v seznamu důvěryhodných serverů je porovnán DNS jméno certifikátu s DNS jménem počítače na, kterém server běží. [1]
- Úroveň 3 – klientská autentizace. Klient se může připojit k libovolnému serveru, ale server umožní připojení pouze ověřeným klientům. Postup je stejný jako u úrovně 2 pouze jej provádí server a ne klient. To znamená, že server kontroluje u sebe, zda má klient certifikát ze seznamu důvěryhodných zdrojů. [1]
- Úroveň 4 – oboustranná autentizace. Server umožní připojení pouze důvěryhodných partnerů. Jedná se o nejbezpečnější variantu, ale je nutné nastavení důvěryhodných certifikátů na obou stranách. Pokud by nebyl server explicitně důvěryhodný může jej klient ověřit jako u úrovně 2. [1]

4.2 MQTT

MQTT (Message Queuing Telemetry Transport). Jedná se o nenáročný komunikační protokol původně vyvinutý IBM, ale později převzaty konsorciem Eclipse foundation. Splňuje standart OASIS. Protokol je založený na principu vydavatel (publisher) zpráv / odběratel (subscribers) zpráv. Toto předávání zpráv mezi klienty zajišťuje server v roly prostředníka (broker). Jeden server ve funkci brokeru může mít několik vydavatelů i odběratelů, kteří pak vystupují jako MQTT klienti. Teoreticky každý klient může být současně vydavatel i odběratel ale v reálu jsou tyto role odděleny. [2]

Zprávy jsou tříděny a rozpoznávány pomocí témat (topics). Každý vydavatel (publisher) vydá zprávu pod nějakým tématem a odběratel (subscriber) musí toto téma dopředu znát, aby byl schopný zprávu vyzvednout. Odběratel se v podstatě přihlásí k odběru daného tématu a server (broker) mu tyto zprávy přeposílá. Odběratel tak nemá informaci o IP adrese vydavatele nebo jeho umístnění. Jediné, co odběratel musí znát je IP adresa brokeru. [2]

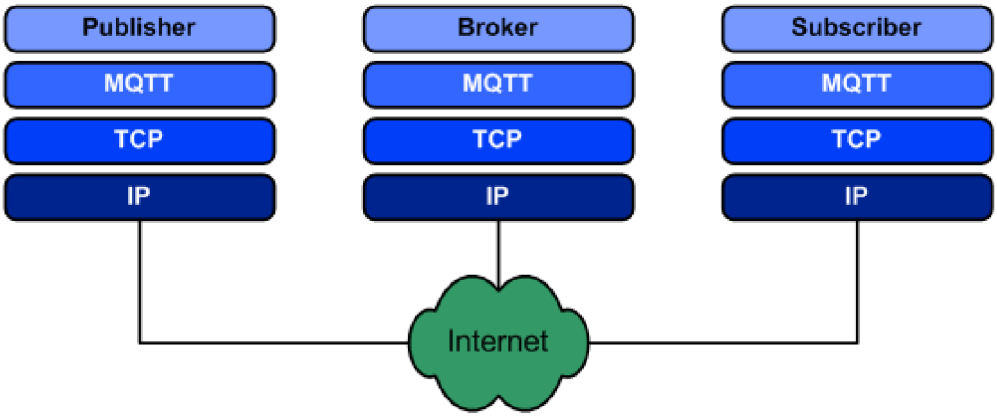
Témata jsou rozdělována lomítkem, které je dělí hierarchicky. Například: „společnostXY/linka99/zona00/stanice000/senzorXY“. Hierarchie není pevně daná a záleží na každém programátorovi, jak ji navrhne. Témata jsou kódovaná pomocí UTF-8. [2]



Obrázek 4.2-1 Příklad přenosu informace v různých topicích. [2]

Zpráva a její formát je čistě záležitostí vydavatele. Nejčastěji se požívá JSON (JavaScript Object Notation), BSON (Binární JSON) nebo textová zpráva. Broker serveru je tak jedno v jakém formátu zpráva v reálu je, jelikož s ní nějak neoperuje. Pouze ji uschovává a poté opět odešle. Je tak čistě na odběrateli, aby zprávě porozuměl. Velikost zprávy je aktuálně omezena na 256 MB. Vzhledem k použití pro přenos metrik ze snímačů a strojů je více než postačující. [2]

MQTT protokol nedefinuje způsob přenosu pouze popisuje strukturu zprávy. K definici způsobu přenosu se používá klasický TCP/IP protokol, který je využívaný u lokálních LAN ethernetových sítí i WAP internetovou sítí. MQTT tvoří aplikační hladinu OSI[3] modelu. Toto činí tento protokol snadno implementovatelný i do zařízení s malým procesore. [2]



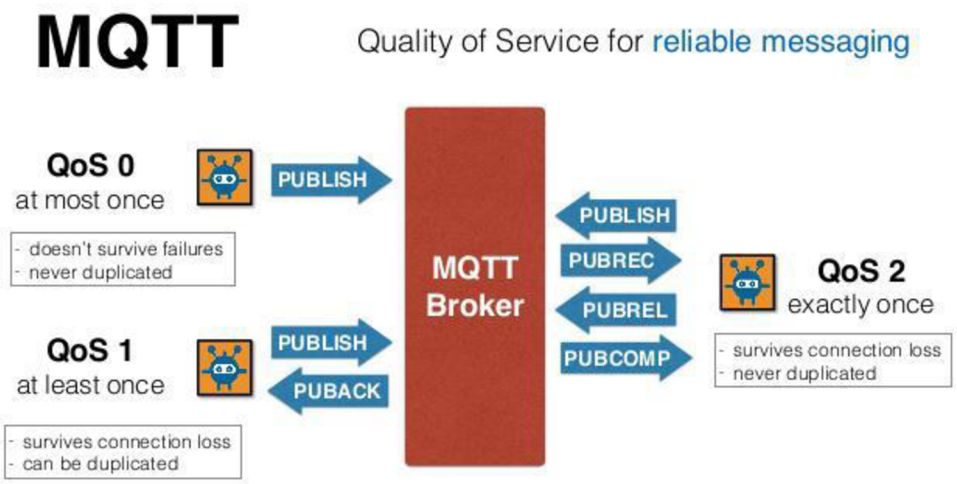
Obrázek 4.2-2 OSI model MQTT. [2]

[3] OSI model - (Open Systems Interconnection model) je koncepční model, který charakterizuje a standardizuje komunikační funkce telekomunikačního nebo výpočetního systému bez ohledu na jeho základní vnitřní strukturu a technologii.

4.2.1 Úrovně potvrzení zpráv při přenosu

Protokol definuje tří základní úrovně potvrzení přijetí zpráv. Takzvané QoS (quality of service) [2]

- QoS 0 – je nejnižší úroveň a znamená, že zpráva je odeslaná bez zpětné vazby o doručení. Zpráva je tak poslaná právě jednou. [2]
- QoS 1 – Zpráva je doručena alespoň jednou. [2]
- QoS 2 – Každá zpráva je doručena přesně jednou. Nedochází ke ztrátě nebo duplicitě zprávy. [2]



Obrázek 4.2-3 Popis nastavení doručení QoS . [1]

4.2.2 Zabezpečení protokolu

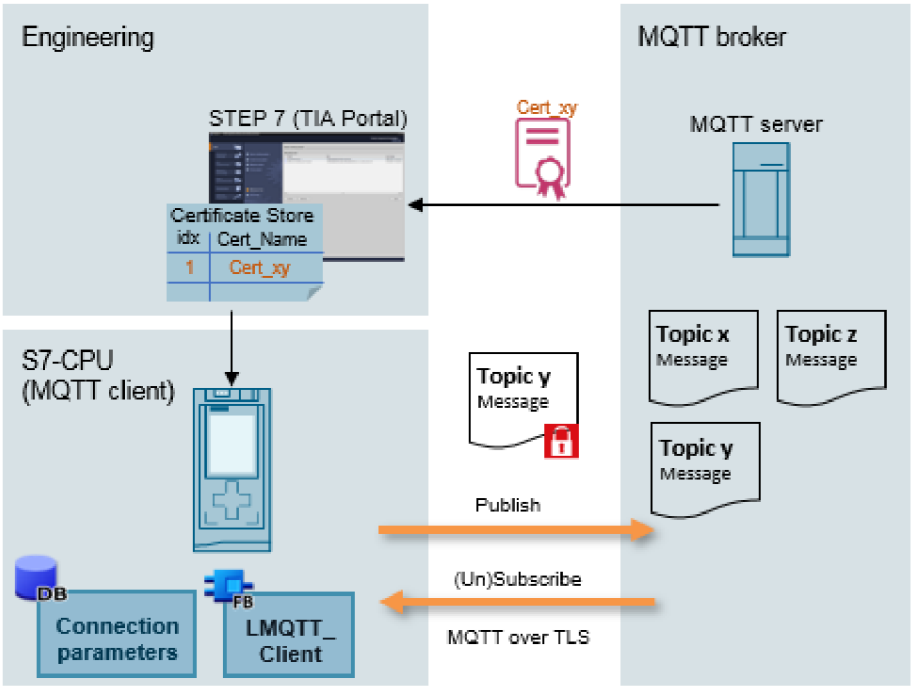
Protokol umožňuje několik možností zabezpečení. Každá komunikace mezi klientem začíná přihlášením klienta k brokeru (connect). Při přihlašování klienta se využívá identifikace klienta pomocí ClientID a volitelně pomocí uživatelského jména (username) a hesla (password). MQTT dále podporuje SSL/TLS šifrovanou komunikac. Je nutné si uvědomit, že zprávy jsou přenášené v textové podobě a kdokoliv je tak může číst na cestě mezi klientem a brokerem. MQTT podporuje TLS v 1.0, 1.1, 1.2 s x509 certifikáty. Podle konfigurace zabezpečení pak komunikace probíhá těmito TCP porty: 1883 je nešifrovaná komunikace, 8883 šifrovaná komunikace pomocí SSL/TLS protokolu zároveň je vyžadována podpora klienta, 8884 šifrovaná komunikace s využitím certifikátu klienta, který je potřeba pro potvrzení autencity klienta. Tento druh šifrovaní však poskytuje málo brokeru. Jeden z nich je mosquitto, který poskytuje test.mosquitto.org certifikát. [1]

4.3 LMQTT pro S7 plc

Protokol byl vybrán na základě snadné implementace a nenáročnosti na síť. Zároveň je široce podporován mezi zařízeními a samotný IoT Edge komunikuji s IoT Hub na Azure i pomocí MQTT.

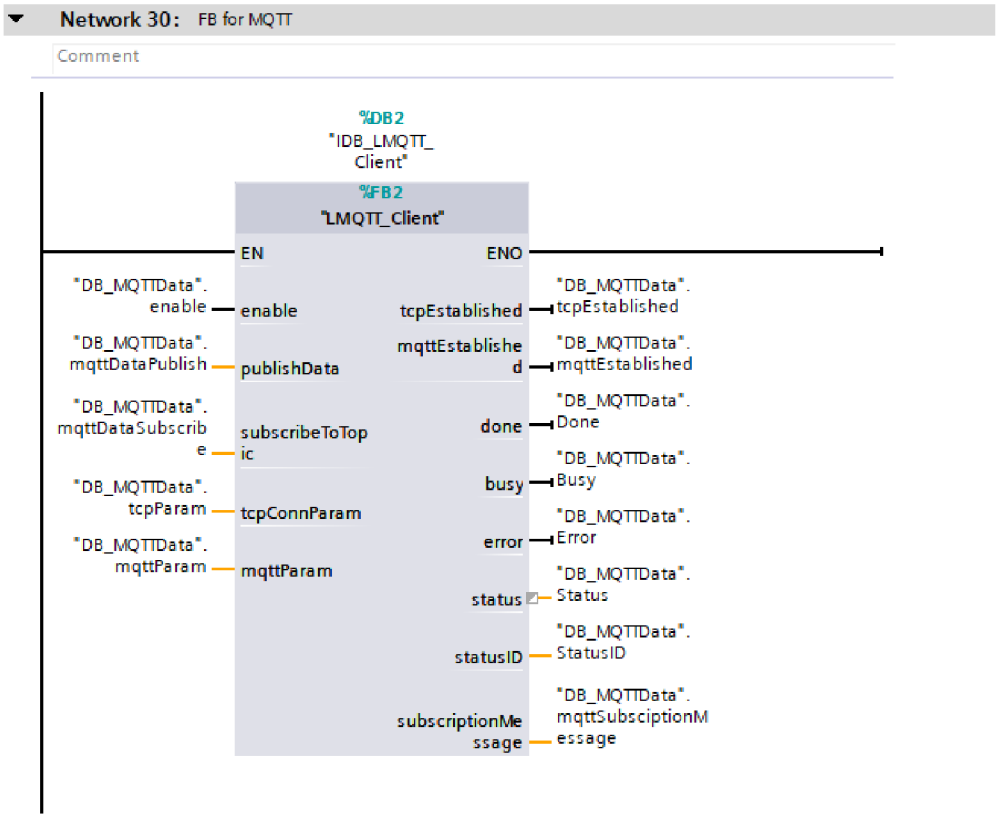
„Message Queue Telemetry Transport“ je jednoduše strukturovaný binární protokol Publish / Subscribe na úrovni TCP / IP. Je vhodný pro zasílání zpráv mezi zařízeními s minimální funkčností a pro přenos prostřednictvím nespolehlivých sítí s nízkou šířkou pásma a vysokou latencí. Díky těmto vlastnostem hraje MQTT zásadní roli pro IoT a v komunikaci M2M.

Knihovna „LMQTT_Client“ poskytuje jeden funkční blok pro CPU S7-1500 a S7-1200. Funkční blok „LMQTT_Client“ integruje funkci klienta MQTT a umožňuje přenášet zprávy MQTT zprostředkovateli (role publisher) a vytvářet odběr (role subscriber). V tomto případě může být komunikace zabezpečena prostřednictvím připojení TLS. Ke zprostředkovateli se můžete dostat statickou IP adresou nebo kvalifikovaným názvem domény. [3] Následující obrázek ukazuje zabezpečené zprávy mqtt s kartou SIMATIC S7-1500. [3]



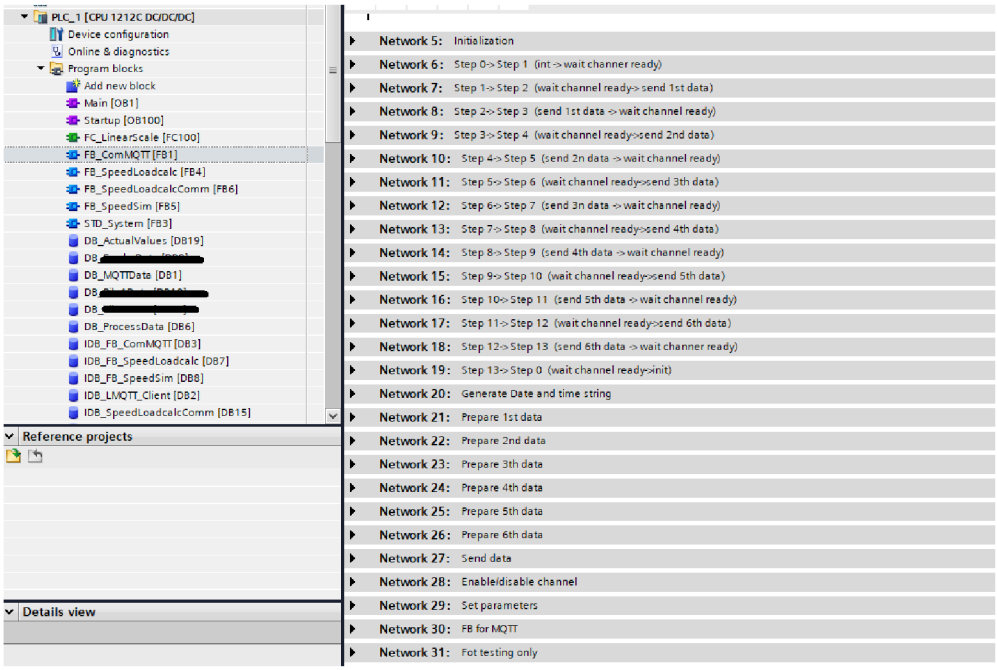
Obrázek 4.3-1 Zabezpečení a cesta zpráv mezi Broker a Client [3]

Knihovna byla použitá ve dvou provozech v různých společností. Implementace se pokaždé liší kvůli odlišnosti dat. Nepodílel jsem se ani na jedné implementaci, jelikož to bylo v rámci jiných projektů společnosti ve, které pracuji a byli na to vyhrazení členové již pracujících týmů.



Obrázek 4.3-2 Klientský blok MQTT komunikace pro S7 sérií PLC

Tento jeden blok není samozřejmě jediný, co se podílí na komunikaci. Při inicializaci se nejdříve nastaví topic v datovém formátu string. Poté následuje zjištění počtu zpráv nutných pro přenesení informace, jelikož je jedna zpráva omezena knihovnou na délku 1500 charakterů. Další důvod dělení je logický, kdy se třeba NOK kusy z různých stanic linky přenášejí v jiné zprávě než informace o lince, jestli nejede a další parametry. Následuje rozparování zprávy společně s kontrolou dostupnosti kanálu a odeslaní zpráv/y.



Obrázek 4.3-3 Náhled na projekt posílající data pomocí LMQT z s7-1200

4.3.1 Popis bloku MQTT

4.3.1.1 Vstupní parametry bloku

Parametr	Datový typ	Popis funkce
enable	BOOL	Kontrolní parametr. True – připojení k MQTT brokeru je navázáno. False – bez připojení.
publishData	LMQTT_typePublishData	Pokud je enable true pošlou se data do broukeru.
subscribeToTopic	LMQTT_typePublishData	Pokud je enable true příjmou se data z MQTT brokeru.
tcpConnParam	LMQTT_typeTcpConnParamData	Konfigurace TCP parametrů pro připojení k MQTT borkeru.
mqttParam	LMQTT_typeParamData	Konfigurace MQTT parametrů pro připojení k brokeru.

Tabulka 4.3-1 Vstupní parametry MQTT bloku [4]

Enable – tento parametr je nastavený na TRUE v datovém bloku komunikace DB_MQTTData.

publishData – nastavený do hodnoty true ve funkčním bloku FB_ComMQTT pokaždé když příznak #SI_Seq.StepNb má hodnotu sudého čísla od 2 po 12. Příznak #SI_Seq.StepNb odpovídá sekvencéru pro přípravu a odeslaní dat.

subscribeToTopic – Tento parametr není využíván.

tcpConnParam – Parametr je nastavený ve funkčním bloku FB_ComMQTT podle specifikace sítě u zákazníka.

mqttParam – Hodnoty parametru jsou nastavovány jedno ve funkčním bloku FB_ComMQTT pouze jednou. Konkrétně parametr "DB_MQTTData".mqttParam.keepAlive určující dobu v sekundách pro

udržení spojení a parametr "DB_MQTTData".mqttParam.clientIdentifier sloužící jako unikátní jménu pro identifikace klienta na straně MQTT brokeru.

4.3.1.2 Výstupní parametry bloku

Parametery	Datový typ	Funkce
tcpEstablished	BOOL	True při navázaní TCP spojení.
mqttEstablished	BOOL	True při navázání MQTT spojení.
done	BOOL	Vrací true pokud byla splněna jedna z následujících úloh: 1) Navázáno připojení. 2) Zpráva úspěšně doručena. 3) Zpráva úspěšně přijata.
busy	BOOL	Vrací true pokud je vykonávaná jedna z následujících úloh: 1) PING je odesilan do MQTT brokeru. 2) Připojovaní. 3) Zpráva se odesílá. 4) Zpráva se přijímá.
Error	BOOL	Vrací true při chybě.
Status	DWORD	Kód chyby.
StatusID	INT	Stav, ve kterém došlo k chybě.
subscriptionMessage	LMQTT_typeSubscriptionData	Přijetí zprávy a samotná zpráva.

Tabulka 4.3-2 Výstupní parametry MQTT bloku [4]

tcpEstablished – Používá se jako podmínka pro odeslaní dat nebo pro otevření nového kanálu. Proměnná je uvnitř bloku LMQTT_Client závislá na několika stavech: Chyba komunikace, ukončení navázaní spojení nebo nečinnost spojení.

mqttEstablished – Proměnná je použita společně s tcpEstablished v logickém OR. Hodnota je měněna ve stejných částech kódu jako tcpEstablished uvnitř LMQTT_Client. Identická funkcionalita doplněna o chyby MQTT komunikace.

done – Je použita ve všech místech kódu, kde dochází k odeslaní dat jako podmínka pro odeslaní. Čeká se na logickou 0.

busy – Proměnná je použita jako jedna z podmínek v logickém součinu pro nastavení kanálu do stavu ready. V bloku LMQTT_Client je nastaven do logické jedničky při jakékoliv probíhající operaci.

Error – Není využit v testovací verzi kódu.

Status – Není využit v testovací verzi kódu.

StatusID – Není využit v testovací verzi kódu.

subscriptionMessage – Není využit v testovací verzi kódu.

4.3.1.3 LMQTT_typeTcpConnParamData

LMQTT_typeTcpConnParamData

	Name	Data type	...	A...	...	...	...	Comment
1	hwIdentifier	HW_ANY	0	☑	☑	☑	☐	HW_ID PROFINETího interface daného CPU
2	connectionID	CONN_OUC	16	☑	☑	☑	☐	ID TCP připojení
3	ipAdressBroker	Array[0..3] of U...		☑	☑	☑	☐	IP Addresa brokeru, přík.: pro addresu "192.168.0.10"
4	ipAdressBroker[0]	USInt	0	☑	☑	☑	☐	"192"
5	ipAdressBroker[1]	USInt	0	☑	☑	☑	☐	"168"
6	ipAdressBroker[2]	USInt	0	☑	☑	☑	☐	"0"
7	ipAdressBroker[3]	USInt	0	☑	☑	☑	☐	"10"
8	localPort	UInt	0	☑	☑	☑	☐	Lokální číslo portu CPU
9	mqttPort	UInt	...	☑	☑	☑	☐	Vzdáleny port MQTT broker
10	activateSecureConn	Bool	fal	☑	☑	☑	☐	Pokud TRUE, komunikace je zabezpečená pomocí TLS protokolu.

Obrázek 4.3-4 Základní parametry datového typu "typeTcpConnParamData" [3]

4.3.1.4 LMQTT_typePublishData

LMQTT_typePublishData

	Name	Data type	...	...	...	...	...	Comment
1	publishMessage	Bool	fal	☑	☑	☑	☐	Nastaven true. Je nastaven do false při sestupné hraně proměnné done.
2	publishTopic	WString[200]	WS	☑	☑	☑	☐	Velikost je možné upravit, ale musí být změněna i MAX_SENDBUFFER and MAX_PUBLISH_MESSAGE v FB
3	publishMessageData	WString[500]	WS	☑	☑	☑	☐	Velikost je možné upravit, ale musí být změněna i MAX_SENDBUFFER and MAX_PUBLISH_MESSAGE v FB
4	publishQoS	Int	0	☑	☑	☑	☐	Definuje QoS level pro MQTT zprávy. 0,1 a 2.
5	publishRetainFlag	Bool	fal	☑	☑	☑	☐	Pokud true, zpráva byla uložena v MQTT brokeru.

Obrázek 4.3-5 Datový typ „LMQTT_typePublishData“ [3]

Komunikaci PLC s brokerem byla sestavena programátory v závodě, kde je testovací nasazení sběru dat. První testování proběhlo s MQTT.fx programem, který byl jako klient pro přijetí zprávy z MQTT brokeru.

4.3.2 Konfigurace zabezpečení TLS/SSL PLC do MQTT broker

Pro navázaní bezpečné šifrované komunikace mezi PLC (MQTT klient) a MQTT brokerem je nutné nejdříve splnit následující body:

- MQTT broker musí podporovat TLS/SSL šifrovanou komunikaci.
- Certifikát pro MQTT brokera je vygenerovaný a platný.
- Čas na CPU v PLC je nastavený na aktuální čas. Certifikát vždy obsahuje platnost na určitou dobu (většinou 365 dní). Pro použití tohoto certifikátu je nutné, aby čas v PLC odpovídal době platnosti. Nové nebo PLC nebo po celkovém resetu má defaultní hodnotu času, která nebude odpovídat skutečnosti. [3]

4.3.2.1 Generování certifikátu

Vytváření certifikátu probíhá na Ubuntu serveru pomocí programu openssl. Jako první se vytvoří privátní klíč a požadavek pro podepsaní certifikátu (CSR):

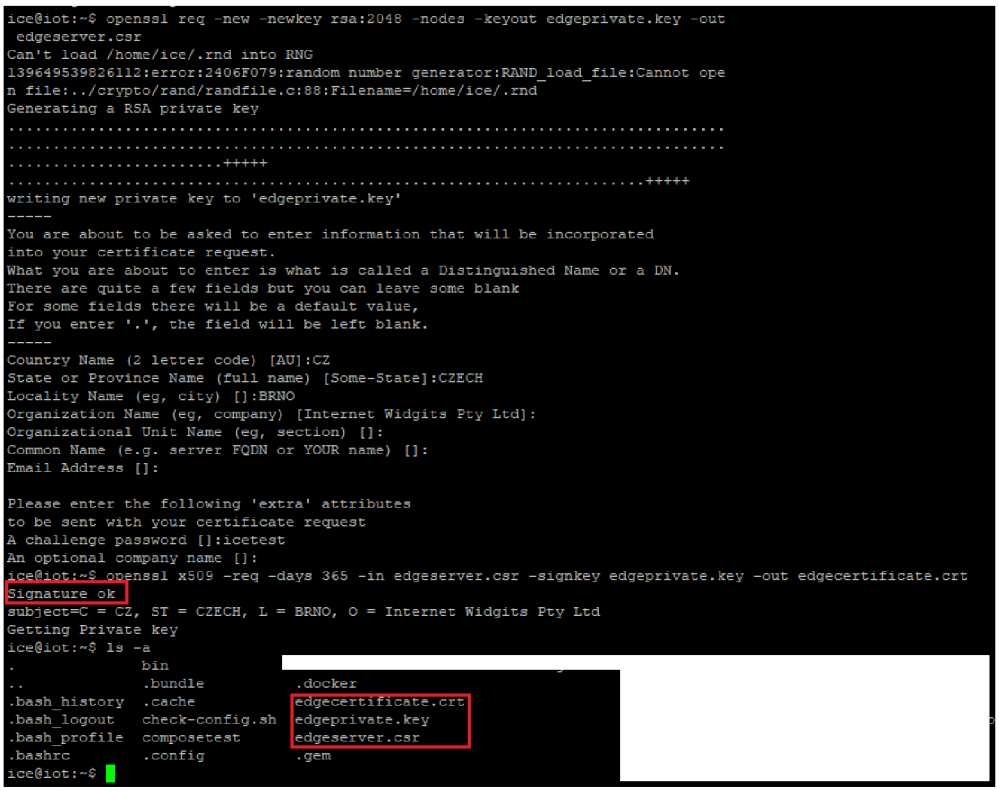
```
openssl req -new -newkey rsa:2048 -nodes -keyout edgeprivate.key -out edgeserver.csr
```

CSR většinou je podepsaný v CA ale v tomto případě je certifikační autorita je edge server na kterém probíhá generování.

K podepsaní a vytvoření certifikátu slouží následující příkaz.

```
openssl x509 -req -days 365 -in edgeserver.csr -signkey edgeprivate.key -out
edgecertificate.crt
```

Ve složce, kde byli spuštěné příkazy jsou teď vytvořené tři soubory.



Obrázek 4.3-6 Vytvoření a podepsaní certifikátu pro MQTT broker.

Počet dní je libovolně volitelný. Po zadání příkazu se program doptá ještě několik základních informací a do domovského adresáře uloží certifikát společně s klíčem. Tyto tři soubory jsou přemístěné do adresářů používaných mosquittem.

Následuje úprava konfiguračního souboru MQTT brokeru. Pro mosquitto, které je použito je tento soubor v adresáři /etc/mosquitto/mosquitto.conf.

```
port 8883 // SSL-TLS port

cafile /etc/ssl/certs/edgeserver.crt // CA certificate

keyfile /etc/ssl/private/edgeprivate.key // Private key

certfile /etc/ssl/certs/ edgecertificate.crt // Server Certificate that is
given by CA
```

Další variantou je možnost sáhnutí certifikátu ze stránek mosquitta přidat jej do CPU.

4.3.2.2 Certifikát podepsaný sám sebou

Je certifikát, který kdokoliv může vystavit nějakým nástrojem k tomu určeným. V mém případě to je na Ubuntu serveru nástroj openssl. Nevýhoda

takového to certifikátu je, že není podepsaný (veřejný klíč) důvěryhodnou certifikační autoritou jako Let´s encripted, google atd… Zařízení, které chtějí používat takto vystavený klíš musí jej mít přiřazeny mezi důvěryhodné zdroje.

4.4 Lokální server

Lokální server slouží jako kolektor dat z vícero linek/plc s využití programu Docker v návaznosti na IoT Edge, který umožňuje spravovat kontejnery (aplikace běžící pomocí virtualizace) na dálku z prostředí Azure. PC tak slouží jako vstupní brána dat do Azure a zároveň jako prostředek pro nasazení nových verzí případných aplikací. Na obrázku 2.1.1 je uveden jako Lokální server.

Do prvního testovacího provozu bylo nasazeno PC od Siemens SIMATIC IPC127E na kterém běží operační systém Ubuntu s Docker runtime a IoT Edge. Nebudu procházet instalaci Ubuntu na počítač. Nijak se neliší od klasické instalace na jakékoliv zařízení. Důležité je pouze použít verze Ubuntu 16.04 nebo 18.04. Tyto verze jsou podporované pro použití s IoT Edge runtime. [5]

Obrázek 4.4-1 Instalace lokálního serveru pro testování

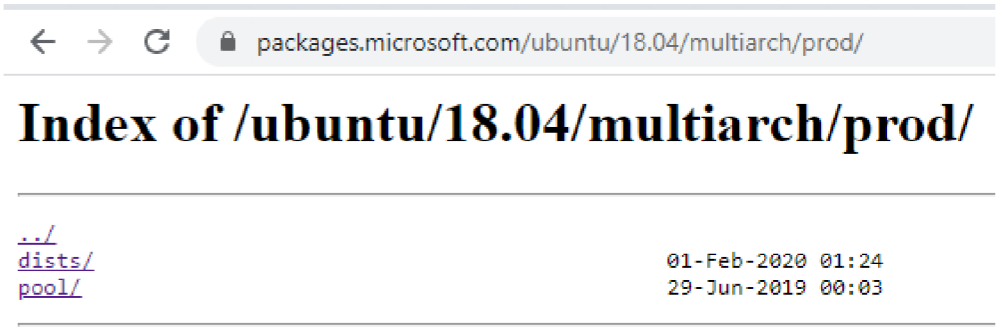
4.4.1 Instalace Azure IoT Edge na lokální server

IoT Edge runtime je několik programů za pomocí, kterých je možné používat PC jako IoT Edge zařízení, které se stará o běh nasazení a zabezpečení všech kontejnerů, které běží na tomto zařízení. Více rozepsané možnosti IoT Edge jsou v kapitole 4.5.5.

Podle použité verze Ubuntu se nalinkuje list programů s vždy poslední verzí instalačních balíčků pro Ubuntu 18.04:

```
curl  https://packages.microsoft.com/config/ubuntu/18.04/multiarch/prod.list
> ./microsoft-prod.list
```

Uvnitř listu je hyper textový odkaz pro systémy založené na debianu s architekturou amd64, arm64 a armhf. V adresářích se potom nacházejí jednotlivé programy, které se instalují níže. [5]



Obrázek 4.4-2 Archív programů pro IoT Edge

Nakopírovaní vygenerovaného listu programů do adresáře /etc/apt/sources.list.d/.

```
sudo cp ./microsoft-prod.list /etc/apt/sources.list.d/
```

Kvůli instalací následujících balíčku od Microsoftu pro Linux je nutné přidat veřejný klíč Microsoft GPG public key do listu důvěryhodných zdrojů. Tento klíč slouží k verifikaci podepsaného certifikátu zaslaného z repositáře při navazování zabezpečeného spojení. V podstatě tím říkáme Linuxu, že certifikáty podepsané tímto klíčem jsou validní a od všeobecně známé certifikační autority. [5]

```
curl https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > microsoft.gpg

sudo cp ./microsoft.gpg /etc/apt/trusted.gpg.d/
```

Po těchto nezbytných krocích se může přejít k instalací prvních dvou balíčků.

```
sudo apt-get install moby-engine | sudo apt-get install moby-cli
```

Moby je framework vytvořený společností Docker pro specializované kontejnery jako je IoT Edge Runtinme. Moby-cli slouží k instalaci a zprávě nových kontejnerů. Nebudu se více zabývat ve své práci tomuto frameworku. Všechny balíčky, které jsou součástí tohoto framworku a jsou použité v řešení jsou popsané níže v kapitole 4.5.

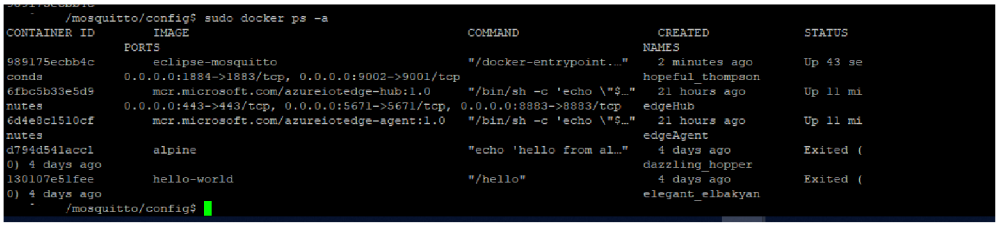
Na závěr se instaluje IoT Edge Security Daemon, který poskytuje a udržuje zabezpečení na IoT Edge zařízení. Tento program je spuštěný při každém startu PC a uvádí do činnosti zbytek IoT Edge runtime. [5]

```
sudo apt-get install iotedge
```

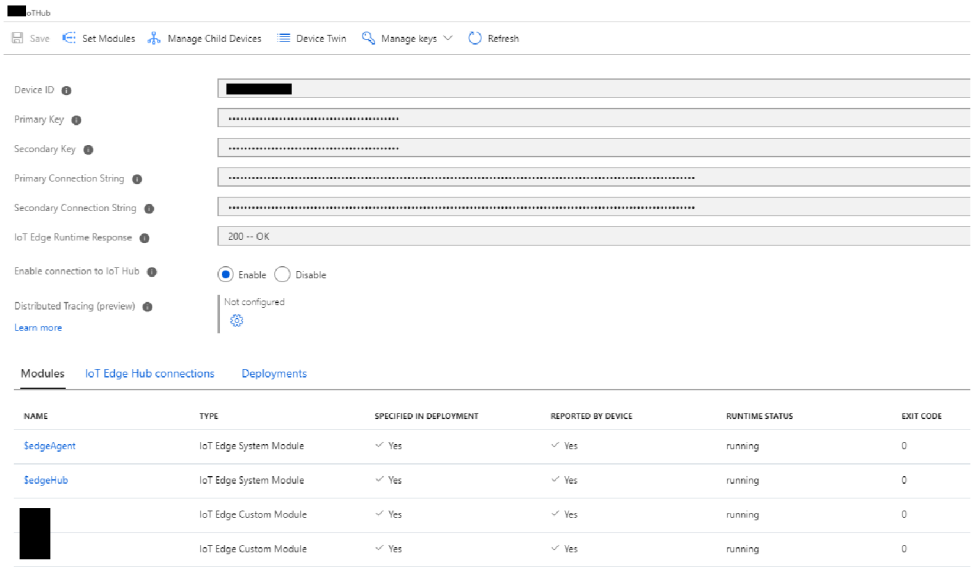
4.4.2 Konfigurace IoT Edge daemon

Konfigurace slouží k nalinkování fyzického zařízení na jeho digitální dvojče[4] uložené v Azure IoT Hub. Konfigurace probíhá pomocí konfiguračního souboru uloženého v adresáři /etc/iotedge/config.yaml. Je možnost manuálního nebo automatického poskytování informací mezi kontejnerem a cloudem. Součástí je nastavení zabezpečení komunikace. [5]

Jakmile je navázáno spojení s IoT Hub v azure je vytvořeno propojení a digitální dvojče nasazení daného IoT Edge zařízení.



Obrázek 4.4-3 Výpis kontejnerů běžících pod docker runtime



Obrázek 4.4-4 Reprezentace IoT Edge zařízení ve webovém prostředí IoT Hub

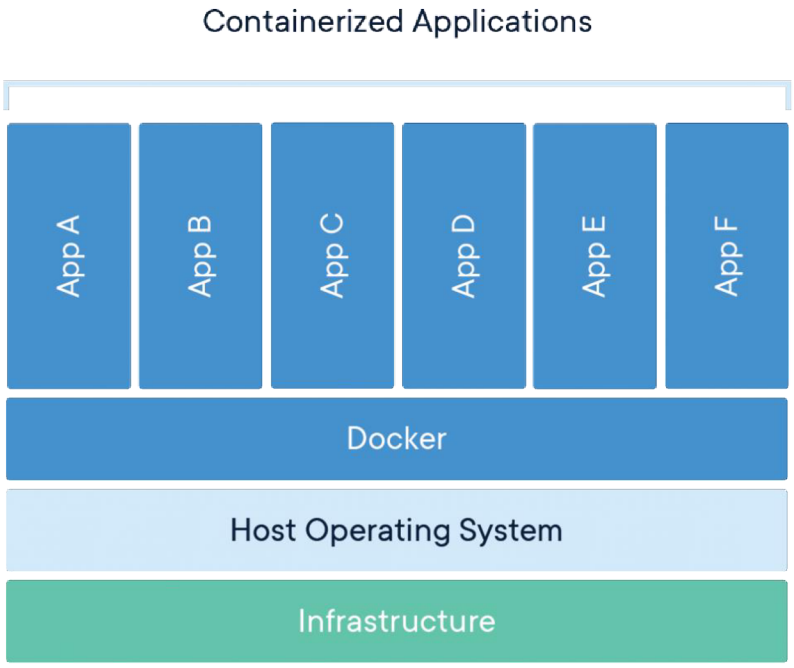
4.5 Docker

Docker Engine je de facto průmyslový modul umožňující běh kontejnerů, který běží na různých operačních systémech Linux (CentOS , Debian , Fedora , Oracle Linux , RHEL , SUSE a Ubuntu) a Windows Server . Docker vytváří jednoduché nástroje a univerzální přístup k balíku, který sdružuje všechny aplikační závislosti uvnitř kontejneru, který je poté spuštěn v Docker Engine. Docker Engine umožňuje kontejnerovým aplikacím běžet kdekoli důsledně na jakékoli infrastruktuře, řeší kontabilitu pro vývojáře a provozní týmy a eliminuje problém funkčnosti na různých platformách. [6]

[4] Digitální dvojče je digitální model reálného zařízení nebo programu.

4.5.1 Kontejner

Kontejner je standardní jednotka softwaru, která balí kód a všechny jeho závislosti, takže aplikace běží rychle a spolehlivě z jednoho výpočetního prostředí do druhého. Image kontejneru Docker je lehký, samostatný spustitelný balíček softwaru, který obsahuje vše potřebné pro spuštění aplikace: kód, runtime, systémové nástroje, systémové knihovny a nastavení. Image (obrazy) kontejnerů se stávají kontejnery za běhu a v případě kontejnerů Docker – image se stávají kontejnery, jakmile jsou spuštěny na Docker Engine. Balíčkový software, který je k dispozici pro aplikace založené na systému Linux i Windows, bude vždy stejný, bez ohledu na infrastrukturu. Kontejnery izolují software od jeho prostředí a zajišťují, že pracuje jednotně i přes rozdíly například mezi vývojem a inscenováním. [7]



Obrázek 4.5-1 Struktura virtualizace pomocí Docker Engine [7]

4.5.2 Porovnání Kontejneru s Virtuálním strojem

Kontejnery a virtuální stroje mají podobné výhody izolace a přidělení prostředků, ale fungují odlišně, protože kontejnery jsou virtualizovány operačním systém namísto hardwarem. Kontejnery jsou přenosnější mezi platformami a efektivnější. [7]

4.5.3 Psaní aplikace jako kontejneru

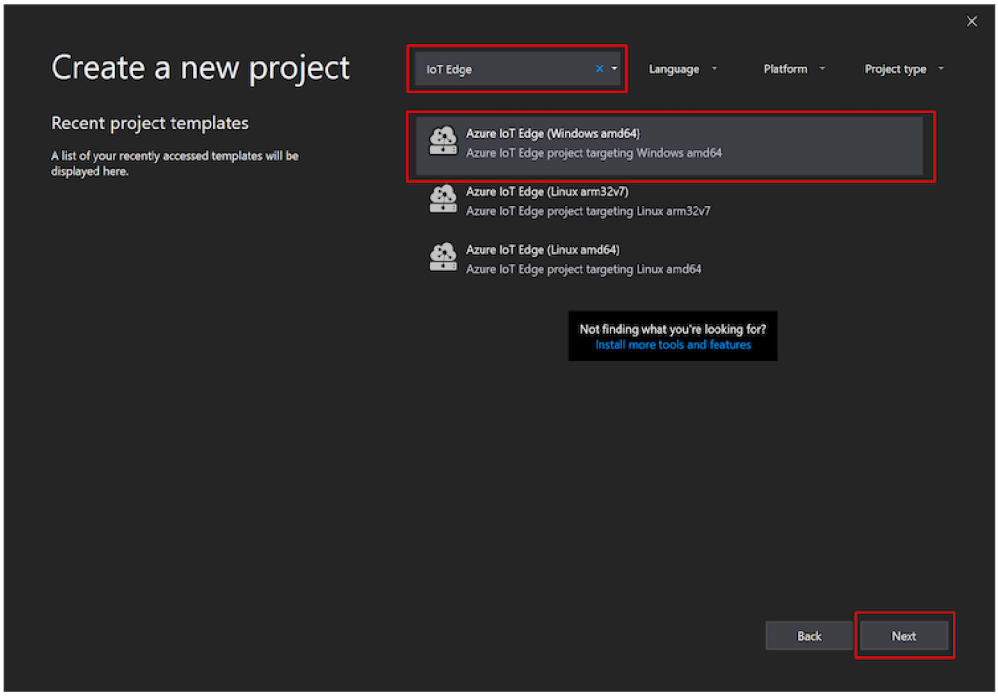
IoT Edge Runtime potažmo Docker (je součástí balíčku Azure IoT Edge Runtime) poskytují vývojovou sadu pro vývoj vlastních aplikací běžících

v prostředí Docker runtime. Microsoft za tímto účelem poskytuje rozšíření zdarma rozšíření pro Visual Studio i Visual Studio Core.

Výhody IoT Edge vývojové sady pro Visual Studio:

- Vytvoření, sestavení a spuštění s možnosti debugu projektů Azure IoT Edge a jejích modulů na lokálním PC.
- Možnost přímo nasadit novou verzi prostřednictvím IoT Hub.
- Podpora jazyku C a C# se všemi benefity Visual Studia ale pro všechny platformy a operační systémy.
- Možnost spravovat IoT Edge zařízení uživatelského rozhraní. [8]

Ve visual studiu při vytváření nového projektu se vybere možnost IoT Edge.



Obrázek 4.5-2 Vytváření nového projektu pro IoT Edge Runtime. [8]

Jako další na výběr je jazyk, v jakém se má aplikace psát, ale hlavně je tu možnost vložit repositář pro modul odkud bude potom dostupný i pro IoT Hub pro nasazení na IoT Edge zařízení. V základním nastavení je tam localhost:5000/<jméno modulu>. Pro Azure Container registry je adresa tvořena <jméno registru>.azurecr.io/<jméno projektu>. Po vygenerování projektu je vygenerovaný JSON soubor deployment.template.json s popisem dané aplikace, která definuje moduly použité v řešení a jejích vzájemné propojení. Ukázková aplikace vytvořená při každém založení projektu obsahuje dva moduly, SimulatedTemperatureSensor (simulovaný teplotní senzor) a IoTEdgeModul1, který zajišťuje komunikaci s IoT Hub na cloudu. Oba moduly jsou napsány pomoci frameworku .NET Core 2.1. Zároveň projekt obsahuje soubor module.json, který popisuje metadata daného modulu (použité knihovny, verze atd.). [8]

4.5.4 MAB (mosquitto azure bridge)

Aplikace zajišťuje spojení mezi IoT Hub na Azure a lokálním broker serverem mosquitto, který běží na stejném lokálním serveru jako samotná aplikace. Aplikace byla vytvořena pomoci MqttNet knihovny pro framework .NET Core 2.1. Její jediný úkol je vyzvednutí zprávy z brokeru a odeslání do IoT Hub.

```
35          static void Main(string[] args)
36          {
37              ConnectToIotHubModule().Wait();
38              MqttConnect().Wait();
39
40              // Wait until the app unloads or is cancelled
41              cts = new CancellationTokenSource();
42              AssemblyLoadContext.Default.Unloading += (ctx) => cts.Cancel();
43              Console.CancelKeyPress += (sender, cpe) => cts.Cancel();
44              WhenCancelled(cts.Token).Wait();
45          }
```

Obrázek 4.5-3 Main MAB

MqttConnect je asynchronní objekt zajišťující spojení s brokerem. Nastavuje připojovací řetězec MQTT brokeru, uživatelské jméno s heslem a ID MQTT klienta. Dále nastavuje pomocí objektů parametry připojení k brokeru a stará se o připojení a odpojení. Jedná se o součást knihovny MQTTNet, která na svých stránkách uvádí příklady publishera i subscribera.

```
var clientOptions = new MqttClientOptions
{
    ChannelOptions = new MqttClientTcpOptions
    {
        Server = s_deviceConnectionString
    },
    ClientId = iotedgeDeviceId + "/" + iotedgeModuleId,
    CleanSession = false
};
```

Obrázek 4.5-4 Přiřazení nastavení komunikace s MQTT brokerem pomocí proměnných prostředí

`mqttClient.ApplicationMessageReceivedHandler` zajišťuje příjem zprávy z brokeru. V těle objektu je volaná metoda pro odeslání zprávy do cloudu. Při vytvoření nového objektu se zprávou je hned odeslaná do cloudu.

```
sendMessageToHub(ioTHubModuleClient, e.ApplicationMessage.Topic, Encoding.UTF8.Ge
tString(e.ApplicationMessage.Payload));
```

```
47          public static async Task MqttConnect()
48          {
49              try
50              {
51                  var clientOptions = new MqttClientOptions···
60
61
62
63                  mqttClient.ApplicationMessageReceivedHandler = new MqttApplicationMessageReceivedHandlerDelegate(e =>···
74
75                  mqttClient.ConnectedHandler = new MqttClientConnectedHandlerDelegate(async e =>···
82
83                  mqttClient.DisconnectedHandler = new MqttClientDisconnectedHandlerDelegate(async e =>···
101
```

Obrázek 4.5-5 MqttConnect

`ioTHubModuleClient` je objekt nesoucí informace o připojení ke cloudu.

Další jsou generované metody zajišťující komunikaci s IoT Hub. Připojení, poslaní zprávy, odpojení, zprávy do IoT Edge zařízení a změna proměnných (desired properties), které jsou navázané na digitální dvojče modulu v IoT Hub a mění se z webového prostředí Azure. Při změně proměnné je modul automaticky restartován a spuštěný s novou hodnotou.

```
151         /// Initializes the ModuleClient and sets up the callback to receive
152         /// messages containing temperature information
153         /// </summary>
            1 reference
154         static async Task ConnectToIotHubModule()
155         {
156             AmqpTransportSettings amqpSetting = new AmqpTransportSettings(TransportType.Amqp_Tcp_Only);
157             ITransportSettings[] settings = { amqpSetting };
158
159             try
160             {
161                 // Open a connection to the Edge runtime
162                 ioTHubModuleClient = await ModuleClient.CreateFromEnvironmentAsync(settings);
163                 ioTHubModuleClient.SetConnectionStatusChangesHandler(async (status, reason) =>
164                 {
165                     if (status != ConnectionStatus.Connected)
166                     {
167                         ioTHubModuleClientIsConnected = false;
168                         if (mqttClient.IsConnected)
169                         {
170                             await mqttClient.DisconnectAsync();
171                         }
172                     }
173                     else
174                     {
175                         ioTHubModuleClientIsConnected = true;
176                     }
177                 });
178
179                 await ioTHubModuleClient.OpenAsync();
180             }
```

Obrázek 4.5-6 ConnectionToIoTHubModule metoda

Metoda zajišťující zaslaní zprávy do IoT Hub z IoT Edge zařízení. Je volána ve chvíli přijetí zprávy z brokeru.

```
234     static async void sendMessageToHub(ModuleClient ioTHubModuleClient, string topinName, string topicPayload)
235     {
236         string dataBuffer;
237         try
238         {
239             dataBuffer = "{\"topic\":\"" + topinName + "\",\"payload\":" + topicPayload + "}";
240             Logger.LogMessage("ioTHubModuleClient sending message: " + dataBuffer, LoggerMessageType.Information);
241             Message eventMessage = new Message(Encoding.UTF8.GetBytes(dataBuffer));
242             await ioTHubModuleClient.SendEventAsync(eventMessage).ConfigureAwait(false);
243         }
244         catch (Exception e)
245         {
246             Logger.LogMessage("ioTHubModuleClient - Error when sending message: " + e.Message, LoggerMessageType.Error);
247         }
248     }
249
```

Obrázek 4.5-7 SendMessageToHub metoda

MAB je tak spojení dvou kódu. MQTT klient zajišťující přijetí zprávy z brokeru je součástí MQTTNet a je použito ukázkové řešení ze stránek této knihovny. Odesílaní zpráv do cloudu a navázaní spojení zajišťuje kód vygenerování při založení projektu pro IoT Edge kontejner. Ode mě zde není moc autorské práce. Vše bylo převzato z příkladů dostupných volně na interneru v dokumentaci pro Azure.

4.5.5 Licence pro docker

Docker open source je distribuován pod licencí Apache 2.0.

4.5.6 IoT Edge

Azure IoT Edge se skládá ze tří komponent:

Moduly IoT Edge jsou kontejnery, ve kterých běží služby Azure, služby třetích stran nebo vlastní kód. Moduly se nasazují do IoT Edge zařízení a lokálně se spouštějí na těchto zařízeních.

IoT Edge Runtime běží na každém zařízení IoT Edge a spravuje moduly nasazené na každé zařízení.

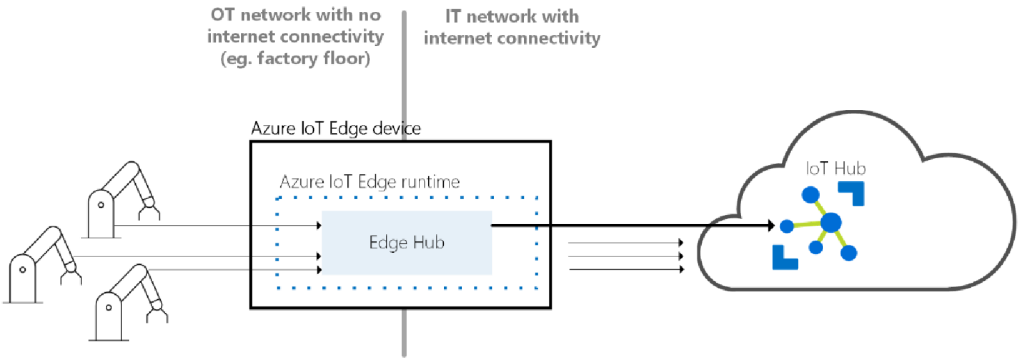
Cloudové rozhraní umožňuje vzdáleně monitorovat a spravovat IoT Edge zařízení. [9]

Více se IoT Edge runtime věnuji v kapitole 5 Cloud.

4.5.6.1 Moduly IoT Edge

Moduly jsou implementovány jako kontejnery kompatibilní s Docker runtime, které spouští aplikace na zařízení s instalovaným IoT Edge Runtime. Docker umožňuje nakonfigurovat vzájemnou komunikaci několika modulů a vytvořit tak kanál zpracování dat. Lze vyvíjet vlastní moduly nebo balit určité služby Azure do modulů, které poskytují přehledy v režimu offline a na IoT Edge zařízení. [9]

IoT EdgeHub je jedním ze dvou modulů, které tvoří modul runtime Azure IoT Edge. Funguje jako místní proxy server pro službu IoT Hub zveřejněním stejné koncové body protokolu jako služby IoT Hub. Taková konzistence znamená, že klienti (ať už zařízení nebo moduly) můžete připojit k modulu runtime IoT Edge, stejně jako do služby IoT Hub. Agenta. IoT EdgeAgent je další modul, který vytvoří modul runtime Azure IoT Edge. Zodpovídá za vytvoření instance moduly, zajištění, že nadále spouštět a hlásí stav modulů zpět do služby IoT Hub. Tato konfigurační data jsou zapsána jako vlastnost vlákna modulu IoT Edgeho agenta. [10]



Obrázek 4.5-8 Schéma komunikace EdgeHub s IoT Hub [10]

Obrázek 4.5-9 Nasazené moduly na Edge zařízeni, ICE Industrial Services a.s.

4.6 MQTT broker Mosquitto

Mosquitto broker je open source projekt vyvíjený konsorciem Eclipse fundation. MQTT broker je server předávají zprávy mezi zařízeními používajícími MQTT protokol pro komunikaci. V rámci této práce spojuje broker PLC s IoT Hub v cloudu Azure. IoT Edge Runtime komunikuje s IoT Hub pomocí MQTT protokolu po portu 1883. Díky tomu není nutné provádět žádné změny ve formátu zprávy. Výhodou mosquitta je jeho dostupnost jako docker image, takže může být spravovaný z prostředí Azure pomocí IoT Hub.

Obraz mosquitta si natáhne sám docker po zadání příkazu:

```
docker pull eclipse-mosquitto
```

Při spuštění mosquitta se zadávají komunikační porty a cesta ke konfiguračním souboru mosquitta.

```
sudo docker run -itd \
--name="mosquitto" \
--restart on-failure \
-p 1883:1883 \
-p 9001:9001 \
-v /home/***/mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf \
-v mosquitto_data:/mosquitto/data \
-v mosquitto_data:/mosquitto/log \
eclipse-mosquitto
```

Přidání mosquitta do IoT Edge Runtime probíhá stejně jako přidání vlastního kontejneru popsaného v kapitole 5 Cloud. Za tímto účelem existuje veřejně dostupná repositář s obrazem mosquitta na serveru docker pod url: docker.io/eclipse-mosquitto.

Kvůli zabezpečení dat je možnost vytvořit uživatele s jménem a hesle, které zabrání vyzvednutí nebo publikování dat neoprávněným klientem. Nejdříve se vytvoří uživatelské jméno a heslo pomocí příkazu:

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd <user_name>
```

Po zadaní příkazu se dvakrát vyplní heslo pro uživatele a následně se přidá do konfiguračního souboru mosquitta v zdrojovém adresáři /etc/mosquitto/mosquitto.conf. Kde se vloží cesta k uživatelskému jménu a heslu a zakážou se anonymní uživatelé.

```
password_file /etc/mosquitto/passwd

allow_anonymous false
```

Jak jsem popsal už víše je možnost přenos dat šifrovat pomocí certifikátu, který si uživatel buď sám vygeneruje a přidá do mosquitta jako důvěryhodný anebo je možnost použít certifikát vygenerovaný pro mosquitto dostupný na webu test.mosquitto.org.

V rámci nulté verze byla použita možnost pouze pro autentifikaci mosquitta. Klienti si tak tento certifikát přidají mezi důvěryhodné zdroje. Zjednodušuje se tak nastavení a vzhledem k často uzavřené lokální sítí ve výrobním podniku je toto řešení podle mého úsudku dostačující. K tomu je samozřejmě ještě nutná znalost topicu pod kterým jsou data publikována z PLC.

4.6.1 Licence

Mosquitto má vlastní licenci Eclipse Public License (EPL) založenou na GNU. Více je popsaná mezi ostatními licencemi v kapitole 9 Licence.

5 Cloud

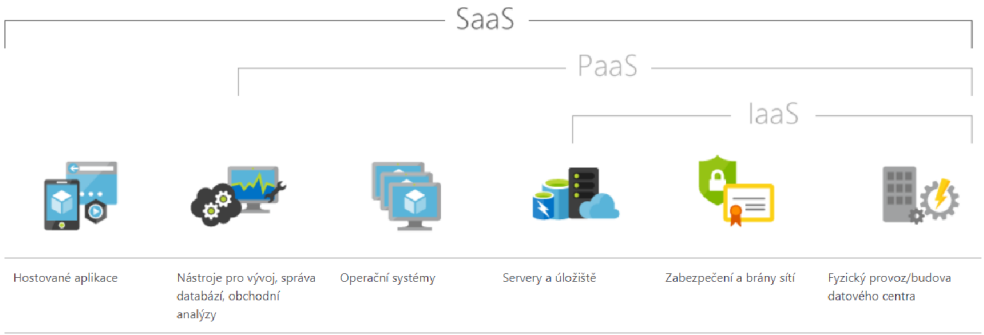
Řešení s cloudem vyplynulo jako hlavní směr, kterým se má aplikace ubírat. Důvodů bylo několik například: možnost přístupu k datům odkudkoliv bez nutnosti řešit zabezpečení dat na uložišti, možnost zálohy dat, a to i geografické mezi datacentry, API pro vizualizační nástroje nebo další zpracovaní dat.

Na českém trhu je několik poskytovatelů cloudových služeb. Vzhledem k zastoupení, podpoře a dokumentaci bylo rozhodnuto pro cloud Azure od Microsoftu.
Srovnání cen mezi poskytovateli nedává moc smysl vzhledem k těžko stanovitelnému odhadu ceny služeb u jednoho cloudu kvůli faktu, že víc cena závisí na jednotlivých datacentrech než na samotném poskytovateli. Další roly hraje i stáří HW v datacentru a typ zálohy dat (zdvojení v jednom serveru, v odlišném serverovém reku nebo geograficky v jiném datacentru).

Azure je neustále se rozšiřující sada služeb cloud computingu[5], které pomáhají organizacím překonávat překážky v podnikání. Poskytuje svobodu při sestavování, správě a nasazování aplikací v rozsáhlé globální síti pomocí nástrojů a architektur.

5.1 Dělení cloudu

Cloud je možné dělit podle služeb a prostředků, které si tam pronajímáme. Základní dělení je SaaS, PaaS a IaaS a serverless. Jednotlivé případy jsou popsaný níže. V této práci se prolínají hlavně SaaS, serverless a PaaS služby cloudu.



Obrázek 5.1-1 Rozdělení cloudu [8]

[5] Cloud computing znamená, že přistupujete k výpočetním službám, jako jsou servery, úložiště, síť a software, přes internet („cloud“) od poskytovatele, jako je Azure. Například místo ukládání osobních dokumentů a fotografií na pevném disku osobního počítače nyní využívá mnoho lidí online uložení: to je cloud computing. [8]

5.1.1 SaaS

Model SaaS (Software jako služba) umožňuje uživatelům připojit se ke cloudovým aplikacím a používat je přes internet. Obvyklými příklady jsou e-mail, kalendáře nebo kancelářské nástroje (jako je Microsoft Office 365).

SaaS poskytuje úplné softwarové řešení, které zakoupíte na základě průběžných plateb od poskytovatele cloudové služby. Pronajímáte si pro organizaci možnost použití aplikace a uživatelé se k ní připojují přes internet obvykle pomocí webového prohlížeče. Veškerá podpůrná infrastruktura, middleware, software a data aplikace jsou umístěné v datovém centru poskytovatele služeb. [11]

5.1.2 PaaS

Platforma jako služba (PaaS) je úplné prostředí pro vývoj a nasazení v cloudu, které vám poskytuje prostředky umožňující dodat cokoli od jednoduchých cloudových aplikací po propracované podnikové aplikace s podporou cloudu. Potřebné prostředky nakupujete od poskytovatele cloudových služeb na základě průběžných plateb a přistupujete k nim přes zabezpečené internetové připojení. [12]

Podobně jako IaaS zahrnuje PaaS infrastrukturu (servery, úložiště a sítě), ale také middleware, vývojářské nástroje, služby business intelligence (BI), systémy správy databází a další nástroje. PaaS je navržený tak, aby podporoval celý životní cyklus webové aplikace: sestavení, testování, nasazení, správu a aktualizace. [12]

5.1.3 IaaS

Infrastruktura jako služba (IaaS) je okamžitá výpočetní infrastruktura, poskytovaná a spravovaná přes internet. Je to jeden ze čtyř typů cloudových služeb, spolu se softwarem jako služba (SaaS), platformou jako služba (PaaS) a bez serverů. [13]

5.1.4 Serverless

Bez serverová architektura umožňuje vývojářům rychleji vyvíjet aplikace, protože díky ní nemusejí spravovat infrastrukturu. Při použití bez serverových aplikací poskytovatel cloudových služeb automaticky zřizuje, škáluje a spravuje infrastrukturu vyžadovanou ke spuštění kódu.

K pochopení definice bez serverové architektury je důležité si uvědomit, že kód stále běží na serverech. Pojem bez serverový vychází ze skutečnosti, že úlohy související se správou a zřizováním infrastruktury nejsou pro vývojáře viditelné. [14]

5.2 IoT Hub

Jedná se o službu hostovanou v cloudu, která funguje jako centrum zpráv pro obousměrnou komunikaci mezi aplikací IoT a zařízeními, která spravuje. Pomocí služby Azure IoT Hub lze vytvářet řešení umožňující spolehlivou a zabezpečenou komunikací mezi velkým počtem zařízeními IoT a back-endem

řešení hostovaným v cloudu. Lze ke službě IoT Hub připojit prakticky jakékoli zařízení. [15]

IoT Hub podporuje komunikaci ve směru zařízení-cloud i cloud-zařízení. Taky podporuje několik způsobů zasílání zpráv, například zasílání telemetrických dat typu zařízení-cloud, nahrávání souborů ze zařízení a metody požadavek-odpověď, kterými můžete ovládat svá zařízení z cloudu. [15]

SDK pro zařízení Azure IoT umožňuje vytvářet aplikace, které se spouští v Edge zařízení a komunikují se službou IoT Hub v cloudu. Mezi podporované platformy patří několik distribucí Linuxu, Windows a operační systémy pracující v reálném čase. Mezi podporované jazyky patří: C, C#, Java, Python, Node.js

IoT Hub a sady SDK pro zařízení podporují připojení zařízení pomocí následujících protokolů: HTTPS (port 443), AMQP (port 5671), MQTT (port 8883).

Při nemožnosti použití knihoven lze zařízení připojovat k IoT Hub nativně pomocí protokolů MQTT v3.1.1, HTTPS 1.1 nebo AMQP 1.0. [15]

IoT Hub rovněž nabízí možnost nastavit přístup do něj pouze z definovaných IP adres. Společně s autentifikací pomocí připojovacích řetězců obsahující klíč pro každé připojené IoT Edgte zařízení je komunikace mezi IoT Hub a IoT Edge zařízením bezpečná a důvěryhodná.

Capability	Basic tier	Free/Standard tier
Device-to-cloud telemetry	Yes	Yes
Per-device identity	Yes	Yes
Message routing, message enrichments, and Event Grid integration	Yes	Yes
HTTP, AMQP, and MQTT protocols	Yes	Yes
Device Provisioning Service	Yes	Yes
Monitoring and diagnostics	Yes	Yes
Cloud-to-device messaging		Yes
Device twins, Module twins, and Device management		Yes
Device streams (preview)		Yes
Azure IoT Edge		Yes
IoT Plug and Play Preview		Yes

Obrázek 5.2-1 IoT Hub Standard vs Basic

IoT Hub má dvě základní úrovně dále děleny podle edice. Důležité pro sběr dat je především možnost nasazení nových modulů a spravování stávajících. K tomuto účelu slouží „Device twins, Module twins“. Tři služby na obrázku výše jsou pro mě v době psaní DP nové, jelikož při prvním setkání s IoT Hub

ještě nebyli nabízené. To neznamená ale že se časem nemůžou ukázat jako důležité pro sběr dat.

5.2.1 Směřování zprav do dalších služeb

Integrovaná funkce směrování zpráv poskytuje flexibilitu umožňující nastavení automatického odesílání zpráv založeného na pravidlech: Směrování zpráv slouží k určení, kam Hub odesílá telemetrii zařízení. Za směrování zpráv do více koncových bodů se neúčtují žádné další poplatky. Další služby si zprávy potom z těchto koncových bodů mohou samy vyzvednout.

5.3 Datový tok mezi lokálním serverem a cloudem

IoT Edge posílá data do IoT Hub ve formě zpráv. Uživatel IoT Hub na azure platí za předplatné za určitý počet zpráv podle verze IoT Hubu. Dalším faktorem je i komunikace zpět do Edge zařízení[6]. Na azure fungují platební skupiny tzv. subscriptions. Každý platební skupina může mít maximálně 50 IoT Hubů a jeden IoT Hub z toho zadarmo. Free verze IoT Hub má povoleno 0,5 kB na jednu zprávu. Placené verze mají kvótu 4 kB na zprávu. Více se tomuto tématu věnuji níže. [16]

Datovým tokům se věnuji v kapitole 8 kde je popsaná více podrobněji cesta dat z lokálního serveru (IoT Edge zařízení) do Cloudu Azure.

5.4 Stream Analytics

Je služba určena k analýze a zpracování vysokého objemu rychlých streamování dat z více zdrojů současně v reálném čase a komplexní modul pro zpracování události. Modely a vztahy se dají identifikovat v informacích extrahovaných z řady vstupních zdrojů, včetně zařízení, senzorů, informačních kanálů sociálních médií a aplikací. Tyto vzory se dají použít ke spuštění akcí a zahájení pracovních postupů, jako je vytváření výstrah, ukládání transformovaných dat pro pozdější použití atd. Stream Analytics je také k dispozici v modulu Azure IoT Edge runtime a podporuje stejný jazyk nebo syntaxi jako Cloud. [17]

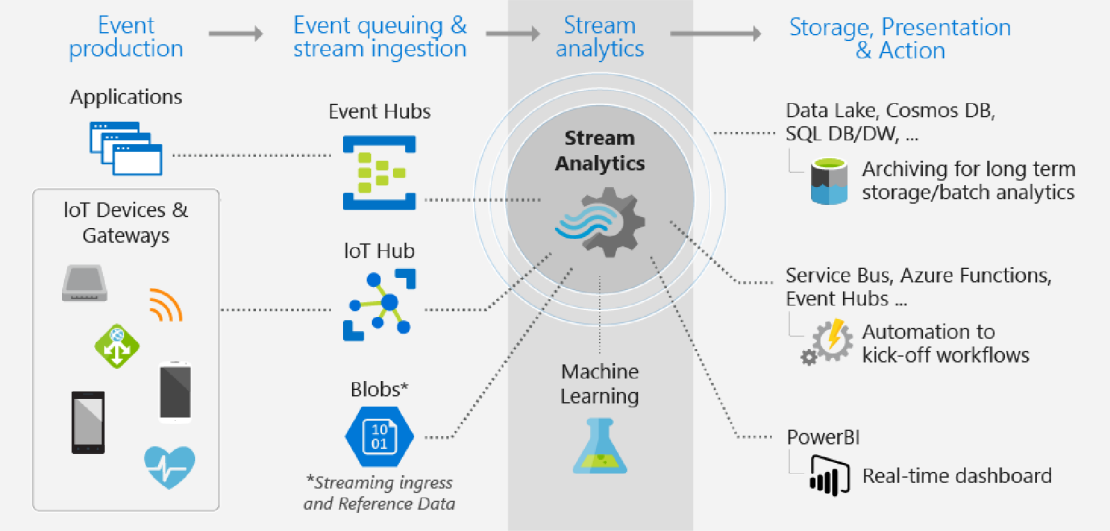
Následující scénáře jsou příklady, kdy můžete použít Azure Stream Analytics:
Analýza streamů telemetrie v reálném čase ze zařízení IoT
Webové protokoly/analýza navštívených webových stránek
Geoprostorová analýza pro správu vozového parku a vozidel bez řidiče
Vzdálené monitorování a prediktivní údržba prostředků s vysokou hodnotou
Analýza dat z prodejen v reálném čase pro účely zjišťování anomálií a řízení zásob [17]

5.4.1 Jak funguje stream analytics

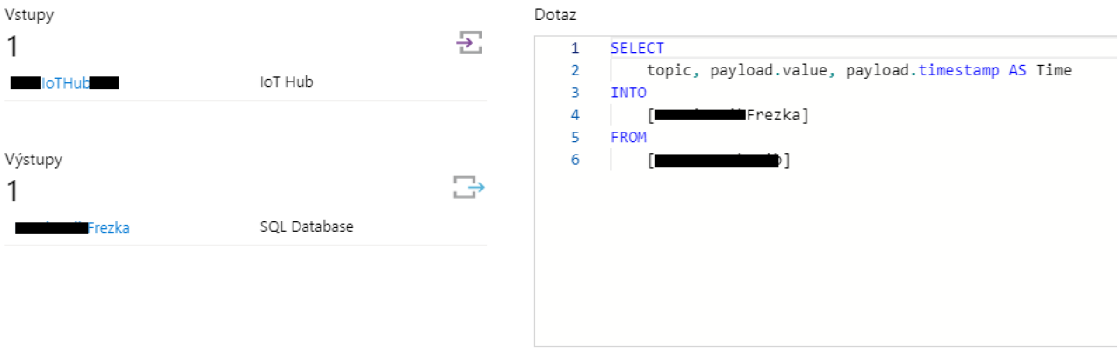
Azure Stream Analytics využívá skoro stejné syntaxe jako SQL databáze. Její úloha se skládá ze vstupu, dotazu a výstupu. SA zpracovává data z Azure

[6] Edge zařízení – označení pro lokální server, který zajišťuje komunikaci s cloudem.

Event Hubs, Azure IoT Hub nebo Azure Blob Storage ve formátu JSON. Dotaz, který je založený na jazyku dotazů SQL, se dá použít k snadnému filtrování, řazení, agregaci a připojení streamování dat v časovém intervalu. Jazyk pro SA lze obohatit pomocí JavaScriptu a C# uživatelsky definovaných funkcí (UDF[7]). Lze tak snadno upravit možnosti řazení událostí a dobu trvání časových oken při předzpracování agregačních operací prostřednictvím jednoduchých jazykových konstrukcí anebo konfigurací. [17]



Obrázek 5.4-1 Práce s daty v SA a vstupy/výstupy úlohy [17]



Obrázek 5.4-2 Přehled služby SA v portálu Azure

V naší aplikaci používáme SA na extrakcí dat z formát JSON a následné uložení do SQL tabulky. Pro aktuální testovací aplikací není třeba samotné služby Stream Analytics ale do budoucna bude u větších a náročnější aplikací využitá více.

5.5 Azure functions

Azure funkce je služba poskytovaná na cloudu Azure umožňující spuštění drobných kousků kódu nazývaných funkce. Odpadá tak nutnost řešení

[7] UDF – (user defined functions) uživatelské funkce.

programového rozhraní. V rámci služby Azure zajišťuje všechny potřebné prostředky a servery v aktuálních verzích softwaru pro běh funkcí v potřebném měřítku. [18]

Funkce jsou spouštění („triggered“) na specifické typy událostí. Například reakce na změnu dat, reakce na zprávu (IoT Hub vysílá zprávy), spuštění podle plánu, nebo jako odpověď na http požadavek. [18]

Interakce s ostatními službami je zjednodušena pomocí vazeb poskytovaných rozhraním. Vazby deklarují přístup k široké části služeb na Azure, a to i služeb třetích stran. [18]

5.5.1 Funkcionality

- **Bez serverové aplikace** („serverless applications“): Umožňují vývoj a nasazení aplikací, které nemají přisazený hardware na Azure. [11]
- **Výběr programovacího jazyka**: Lze použít C#, Javu, JavaScript, Python a PowerShell. [11]
- **Platba za použití**: Azure účtuje peníze pouze za čas, který funkce potřebovali k vykonání. [11]
- **Možnost vlastních knihoven**: Funkce podporují balíčkové správce jako NuGet nebo NPM. [11]
- **Integrovaná bezpečnost**: Automatická protekce http-spouští. [11]
- **Zjednodušení integrace**: Snadno propojitelné s ostatními Azure službami spadajícími do kategorie Služba jako servis (SaaS). [11]
- **Flexibilní vývoj**: Možnost nastavit CI/CD[8] pro vývoj funkce. [11]
- **Open-source**: Zdrojový kód Azure Functions je dostupný na GitHubu a je možnost se zapojit do vývoje a integrovat tak své požadavky. [18]

5.5.2 Možnosti Azure Functions

Funkce se hodí pro zpracování informací nebo záznamů, které sdílí společný zdroj nebo cíl jako je Azure SQL databáze. Integrace systému, které pracují s IoT nebo vytváření jednoduchých API[9] pro mikro služby. [18]

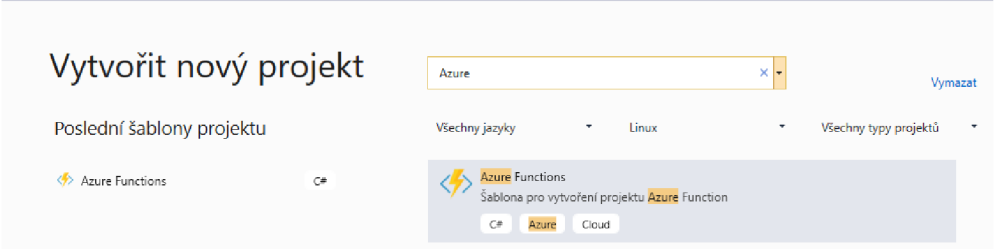
5.5.3 Vývoj aplikace pro Azure Functions

Program je možné psát ve Visual Studiu, Visual studio Code nebo příkazové řádce. Nastavením prostředí Visual Studia není předmětem této práce a je skvěle popsáno v dokumentaci azure docs.

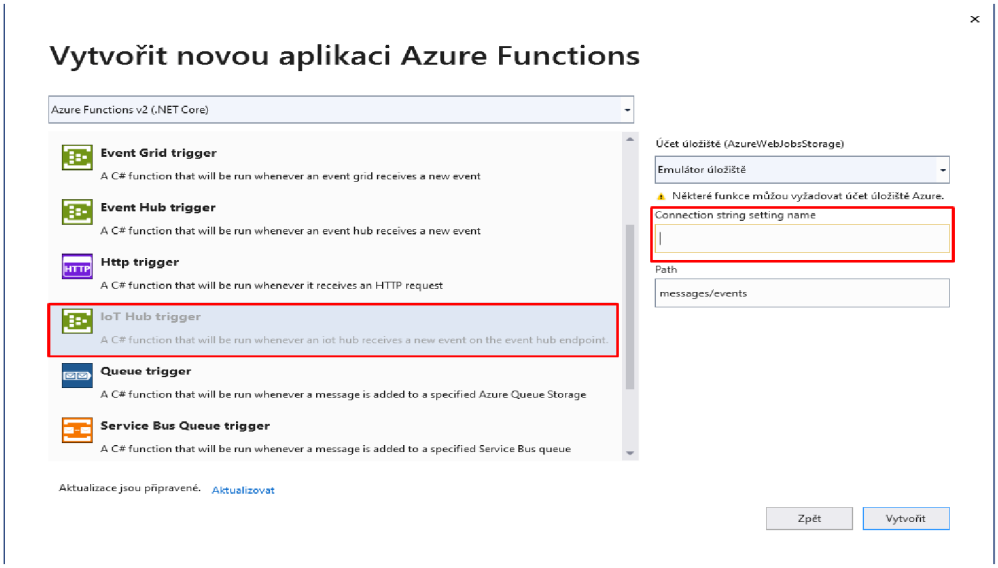
Po správném nainstalování prerekvizit do Visual Studia se funkce vytváří přímo jako projekt. Na začátku se zvolí druh události na, který má funkce reagovat. Poté je zaintegrováno a připraveno vše nutné k implementaci

[8] CI/CD (continuous integration and continuous deplivery) je proces, který umožňuje nepřetržité dodávání aplikací k testování.
[9] API (Application Programming Interface) označuje v informatice rozhraní pro programování aplikací.

vlastního kódu bez nutnosti instalace zkoumání knihoven a API pro propojení s ostatními službami.



Obrázek 5.5-1 Vytváření nového projektu Visual Studio



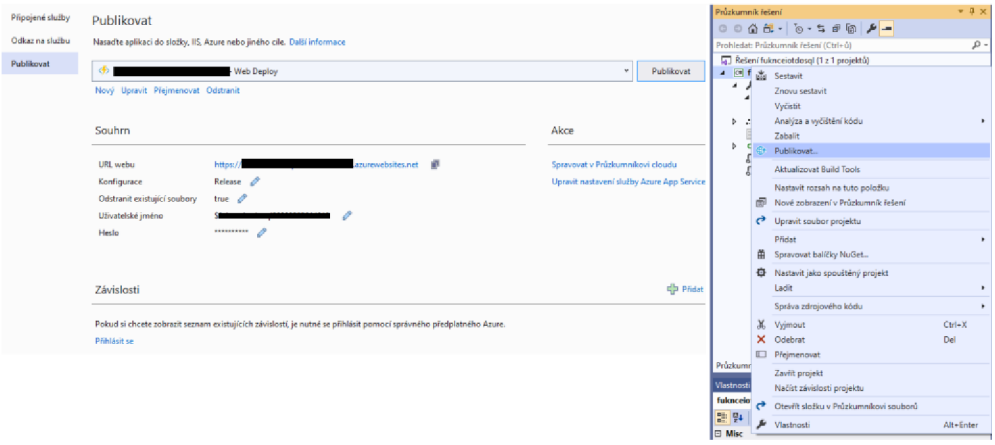
Obrázek 5.5-2 Volba události a možnosti napojit událost na službu

Funkcionalita Azure funkce pro sběr dat zajišťuje komunikaci na Azure mezi IoT Hub a procedurou v Azure SQL databází. Nejprve reaguje na příchozí zprávu do IoT Hub. Následně se pomocí připojovacího řetězce napojí na databází a zašle zprávu ve formátu JSON do databáze, kde je dále zpracována.

```
16  namespace [redacted]
17  {
        Počet odkazů: 0
18      public static class Function1
19      {
20          private static HttpClient client = new HttpClient();
21          //Microsoft.Azure.WebJobs.ConnectionStringNames();
22          [FunctionName("Function1")]
23
            Počet odkazů: 0                                  jméno IoT Hub          Připojovací řetězec
24          public static void Run([IoTHubTrigger("[redacted]", Connection = [redacted], ConsumerGroup = "$Default")]EventData message, ILogger log)
25          {
26              string smsg = Encoding.UTF8.GetString(message.Body.Array);
27              log.LogInformation($"C# IoT Hub trigger function processed a message: {Encoding.UTF8.GetString(message.Body.Array)}");
28
29              string str;
30
31              try
32              {
43                  using (SqlConnection conn = new SqlConnection(str))
44                  {
45                      state = conn.State;
46                      log.LogInformation($"state:{ conn.State }");
47                      if (state.ToString() != "Open")
48                      {
49                          conn.Open();
50                          log.LogInformation("Opened");
51                      }
52                      log.LogInformation($"The database state of connection:{conn.State}");
53                      SqlCommand cmdSQL = new SqlCommand("ImportEvents", conn);
54                      cmdSQL.CommandType = System.Data.CommandType.StoredProcedure;
55                      cmdSQL.Parameters.AddWithValue("Events", Encoding.UTF8.GetString(message.Body.Array));
56
57                       cmdSQL.ExecuteNonQuery();
58                  }
60              }
61              catch (Exception ex)
62              {
63                  log.LogError($"C# Event Hub trigger function exception: {ex.Message}");
64              }
65          }
66      }
```

Obrázek 5.5-3 Azure Funkce kód

Pod dokončení aplikace a lokálním otestováni v emulátoru je možné funkci nasadit do Azure a spustit.

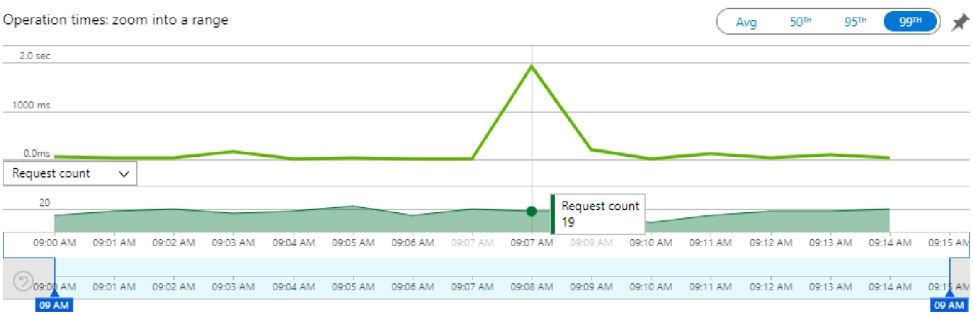


Obrázek 5.5-4 Nasazení funkce v prostředí Visual Studio

Před samotným publikováním a nasazením aplikace je nutné mít nejdříve založený úložný účet („storage account“) nebo SQL databází. Tyto dvě služby slouží jako kontejnery s rozhraním pro komunikaci a interakci s Azurem. Všechny služby na Azure jsou v pozadí jako kontejnery.

Po úspěšném publikování je možné ve webovém rozhraní zkontrolovat běh aplikace a základní telemetrií aplikace. Slouží k tomu klasické metriky Azure. Další možností u Azure Functions je webová konzole, která zobrazuje aktuální logy funkce od spuštění konzole.

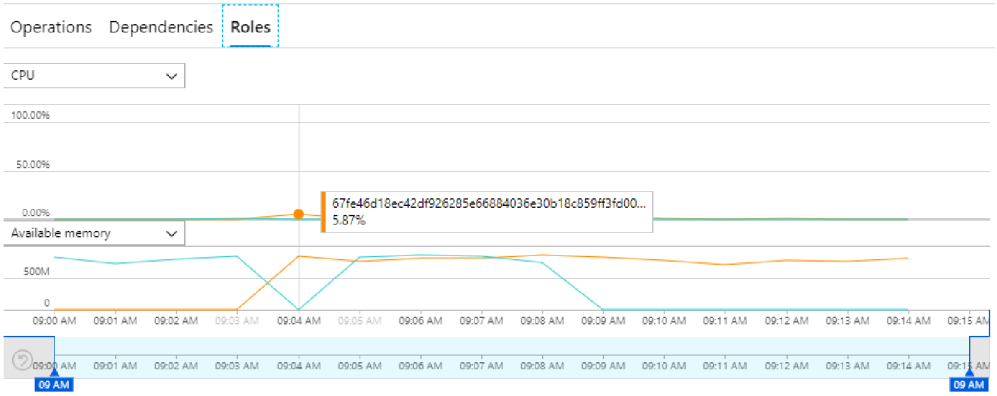
Na obrázku 5.5-8 je vidět výpis z konzole funkce v jedné z dřívějších verzi kdy docházelo ke ztrátě některých dat z důvodu chyb v formátu JSON nebo chybějící hodnotě časového razítka, která nesměla být NULL. Další problémy, na které jsem narazil byly způsobeny špatně napsanou procedurou v databází, která jednou za čas nevyzvedla všechny zprávy nachystané v zásobníku funkce.



Obrázek 5.5-5 Průměrná odezva na dotazy společně s počtem dotazů každou minutu.

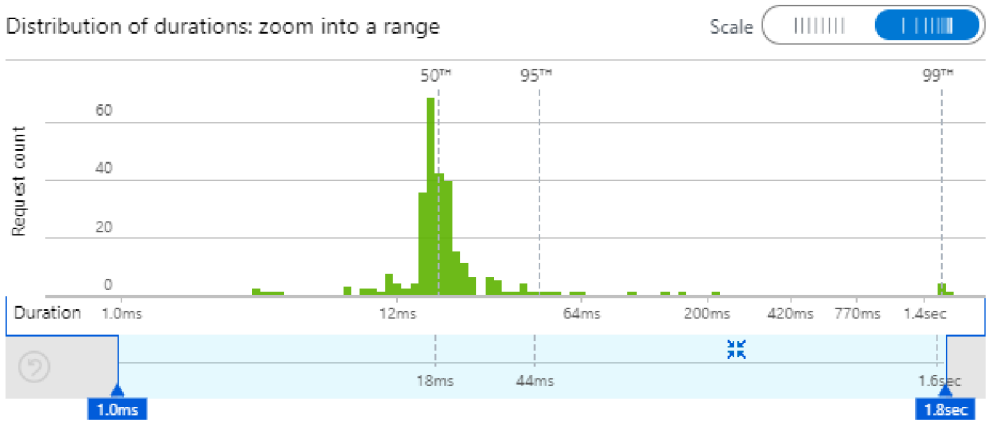
Na výše uvedeném obrázku je vidět využití AF, která obsluhuje dva testovací provozy Zvýšení odezvy AF na cca 2 sekundy se neperiodicky opakuje celý den. Důvodem je pokles zpráv z předchozí minuty, kdy Azure vyhradí pro další

zprávy menší výkon. Při opětovném nárustu zpráv za minutu se to projeví právě zvýšenou odezvou.



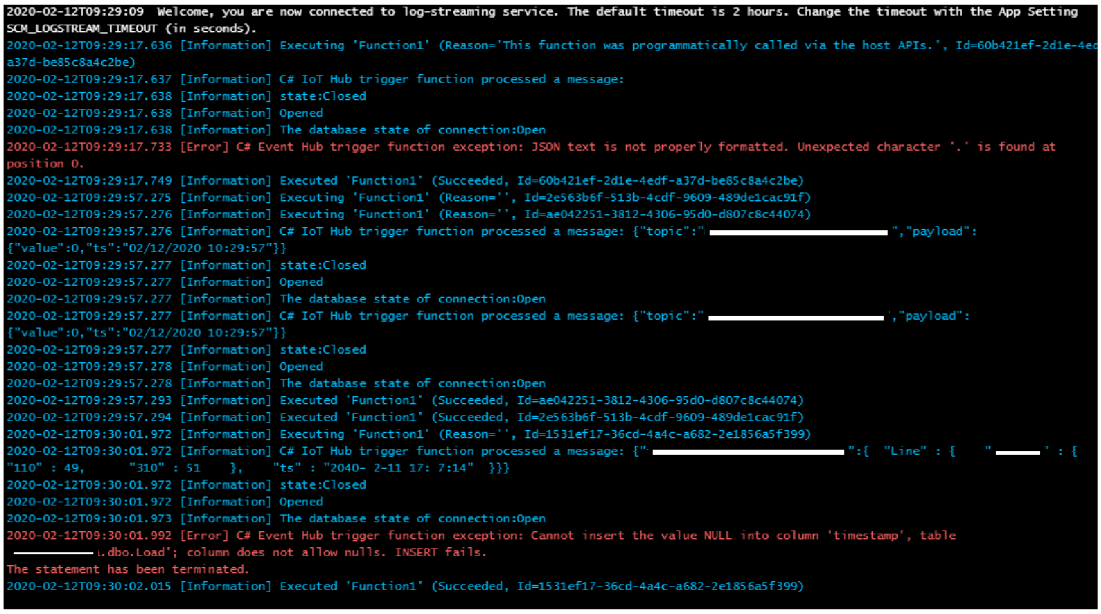
Obrázek 5.5-6 Procenta vytíženosti Azure Funkce podle spouští

Prodlevy v předchozím grafu jsou způsobené nutností rezervovat více výkonu CPU pro zpracování nárazových zpráv z druhého testovacího provozu, kde jsou některé zprávy zasílány periodicky ale další jsou odeslány z PLC při změně stavu. To způsobuje změnu počtu zpráv mezi požadavky. Azure funkce, jak jsem již zmiňoval výše je bez serverovou službu, takže nemá neustále přiřazený výpočetní výkon za, který by bylo nutné platit. Proto větší počet dotazů, než je obvyklé vyvolá malou prodlevu ve zpracování cca do 2 sekund od přijetí požadavku (trigger). Zaznamenal jsem i 20 sekund po výpadku druhého testovacího provozu a opětovného nahození.



Obrázek 5.5-7 Histogram odezvy Azure Functions na spoušť v log. Měřítku v časovém intervalu 15 min.

Na histogramu je patrná nejčastější odezva na požadavek do cca 20 ms a nejpomalejší do 1,4 sekundy. Vzhledem k tomu, že v rámci DP systém přenáší data pouze jednou za minutu sice ze dvou testovacích provozů, ale přesto je tato doba více než postačující vzhledem k obnovovací frekvenci dashboardů po jedné minutě.



Obrázek 5.5-8 Výpis konzole Azure Functions

U chybové hlášky je vidět, že se hlásí jako Event Hub spouštěná funkce i když se jedná o IoT Hub. Důvod je, že IoT Hub vychází z velké míry z Event Hubu.

5.6 Azure functions vs stream analytics z pohledu zpracování zpráv do SQL databáze

Přímé srovnání těchto dvou služeb není vhodné jelikož, se jedná o naprosto rozdílné služby, které se mohou ve specifickém případě jako tento překrývat v použití ale přesto s rozdíly.

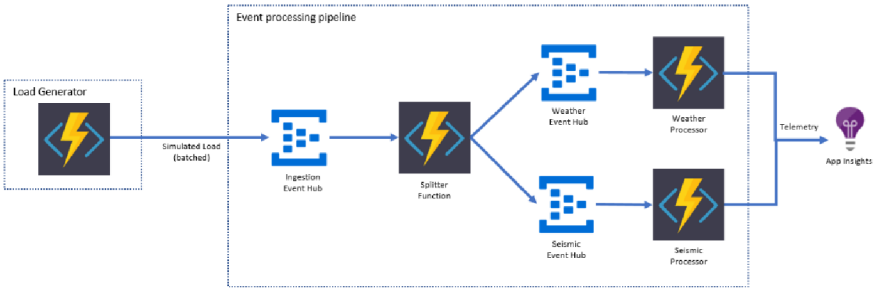
Hlavní rozdíl vidím v modelu ve, kterém tyto služby operují. Stream Analytics je služba, která má neustále alokovaný výpočetní výkon, pokud ji nevypneme. Naproti tomu Azure Functions se spouští pouze na příchozí trigger[10], kdy vykoná nashromážděné úlohy v bufferu. Tato skutečnost se promítá do ceny služeb. Stream Analytics je ale služba stavěna ke zpracování a rozdělování velkých objemů dat za sekundu. Této vlastnosti však nebylo zatím potřeba.

Při testovaní Stream Analytics se vyskytl další rozdíl, a to ve schopnosti využívat procedury v SQL databázi. Na výstupu vstupního bodu dat do SA[11] jsou data serializováná podle zadaného datového typu (JSON, Avro, CSV, Protobuf, XML…) a tento typ se už znova na výstupu SA neobjeví. Dál data z SA proudí po jednotlivých separovaných proměnných. Problém nastává, když na vstupu procedury v SQL očekáváte JSON formát dat. Nejedená se o velký problém, který by se nedal snadno obejít ale společně s vyšší provozní cennou a omezenýma schopnostmi query bylo od SA prozatím opuštěno.

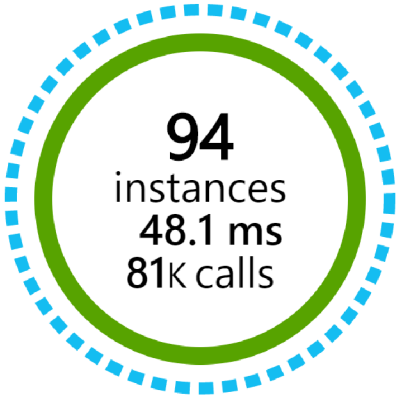
V průběhu testování SA jsem využíval i službu Automation Accounts, která umožňuje podle předem nastaveného času nebo spouště reagovat

[10] Trigger – spoušť služby
[11] SA – Stream Analytics cloudová služba poskytovaná Azure

spuštěním nebo vypnutím dané služby nebo serveru (možné využití této služby je daleko rozsáhlejší). Nedostatkem tohoto řešení byla ztráta dat před spuštěním služby. Spuštění služby mohlo trvat od desítek sekund k jednotkám minut. Naproti tomu Azure Functions tento problém nemá a není nutné využívat další služby k managmentu prostředků využívaných danou službou. V dokumentaci jsou rovněž dostupné návody, jak docílit s Azure Functions až 100 000 zpracovaných událostí (trigger) za sekundu. Toto však nebylo ještě odzkoušeno a nebylo to zatím třeba.



Obrázek 5.6-1 Návrh řetězce služeb umožňující zpracování 100 000 událostí/sek [19]



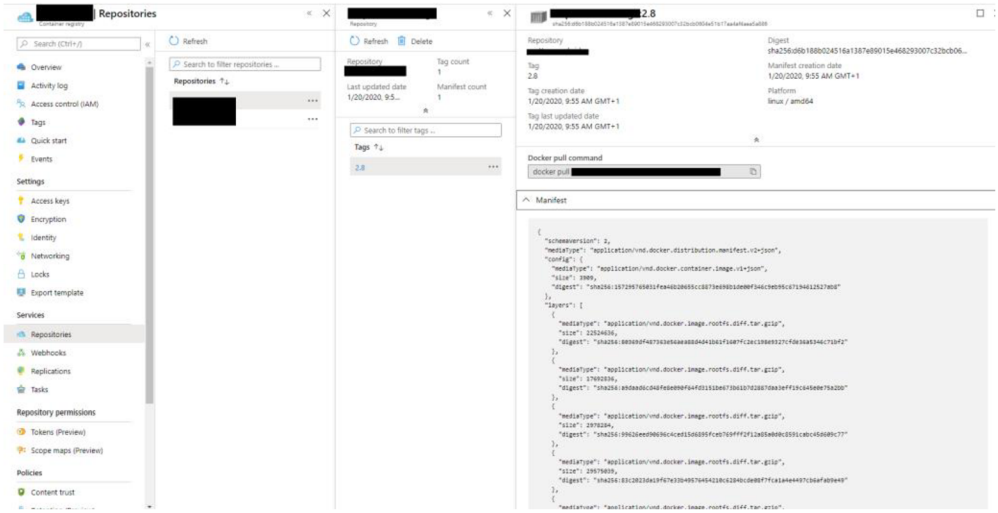
Obrázek 5.6-2 Statistika Azure Funkce za posledních 7 dní

Ze statistik za posledních 7 dní vyplívá, že se ještě ani zdaleka systém neblíží k možnostem Azure Funkcí a Stream Analytics jsou tak naprosto zbytečné. Kvůli nízkému vytížení tak za Azure Funkce nic neplatíme, jelikož bezplatný grand od Azure na AF je na měsíc jeden milion spuštění. Více se tomu věnuji v kapitole 6 Cloud, kde porovnávám ceny služeb.

5.7 Přidání IoT Edge modulu do IoT Edge runtime

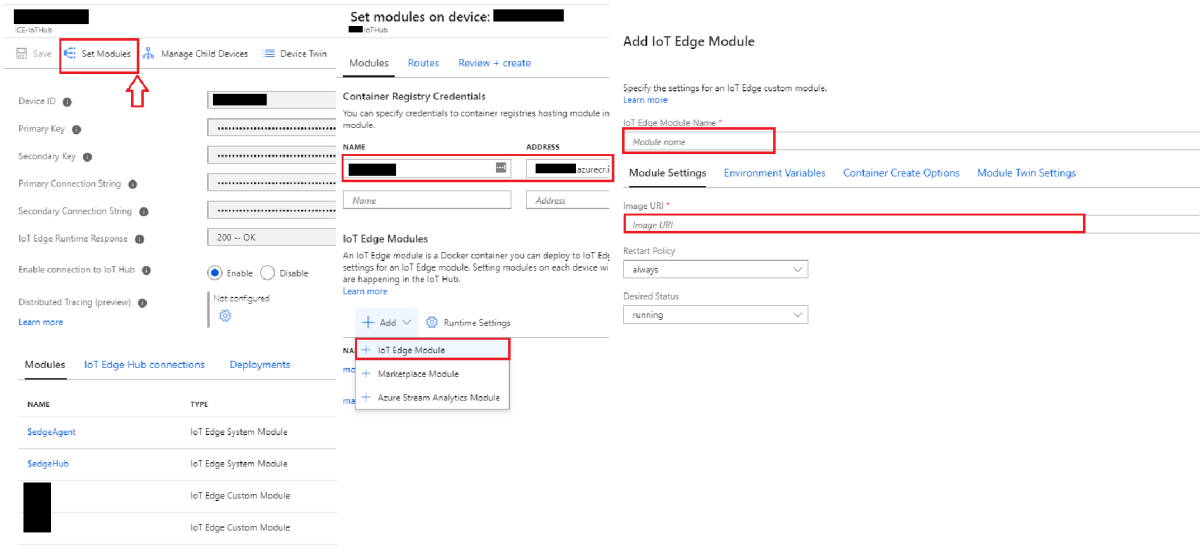
Přidání nového kontejneru do IoT Edge runtime probíhá skrze IoT Hub na portálu azure. Kontejnery se dělí do 3 kategorií. IoT Edge module, Market place module nebo Azure stream analytics modul.

Vlastní kontejner spadá do kategorie IoT Edge Module. Kontejner je uloženy na veřejném nebo privátním repositáři. Existuje řada repositářů jako Container Registry na azure nebo Docker hub. IoT Hub si tento image následně nalinkuje a nahraje jej do IoT Edge zařízení.



Obrázek 5.7-1 Container Registry

IoT Hub přidává kontejnery za pomocí url repositáře, kde se kontejner nachází. Při nasazení se zadává název, základní nastavení, jestli kontejner má hned po nasazení být spuštěný nebo nastavení podmínek restartování kontejneru. Další možnosti jsou zadaní proměnných prostředí, s kterými kontejner pracuje nebo digitální dvojče reprezentující modul v JSON.



Obrázek 5.7-2 Nasazení kontejneru v IoT Hub

5.8 Virtuální server s grafanou

Posledním kouskem systému sběru dat je interpretace a zobrazení samotných dat uložených v databázi. K tomuto účelu slouží upravena verze grafany. Grafana jako taková je více popsaná v kapitole 7 Vizualizace.

Nabízí se zde několik řešení. Grafana může běžet na stejném lokálním serveru jako IoT Edge Runtime, pokud je na to myšleno při realizaci vzhledem k výkonu serveru. Je nutné ale říct, že obě aplikace jsou nenáročné a zvolené průmyslové PC by s nimi nemělo problém. Jediný problém vystává, pokud je server v uzavřené lokální sítí s pouze jedním vystaveným portem pro

komunikaci do IoT Hub nebo dvěma pro zpětnou komunikaci s databází. Problém může nastat v certifikaci domény nebo v přístupu k datům z vnější. Nejedná se o velký problém ale možná zbytečný.

Jednodušší a elegantnější řešení mi přijde využít serveru poskytovaného od Azuru. Proto toto řešení zde více rozvedu.

5.8.1 Výběr serveru

Azure umožňuje celou řadu různých typů VM[12] rozdělených do rodin podle úlohy, kterou tento server má plnit. Je možnost vybrat s několik operačních systému. Samotná velikost serveru se odvíjí od zvoleného způsobu pronajímaní. Je tu možnost si pronajat celé pole s výpočetním výkonem a rozdělit si ho na jednotlivé VM dle potřeby.

Já se zabýval pouze modelem Pay as you go (platba při odchodu), jedná se o platbu na konci měsíce za využitý výkon za jednotlivé servery dle zvolené velikosti.

5.8.1.1 Rodiny VM

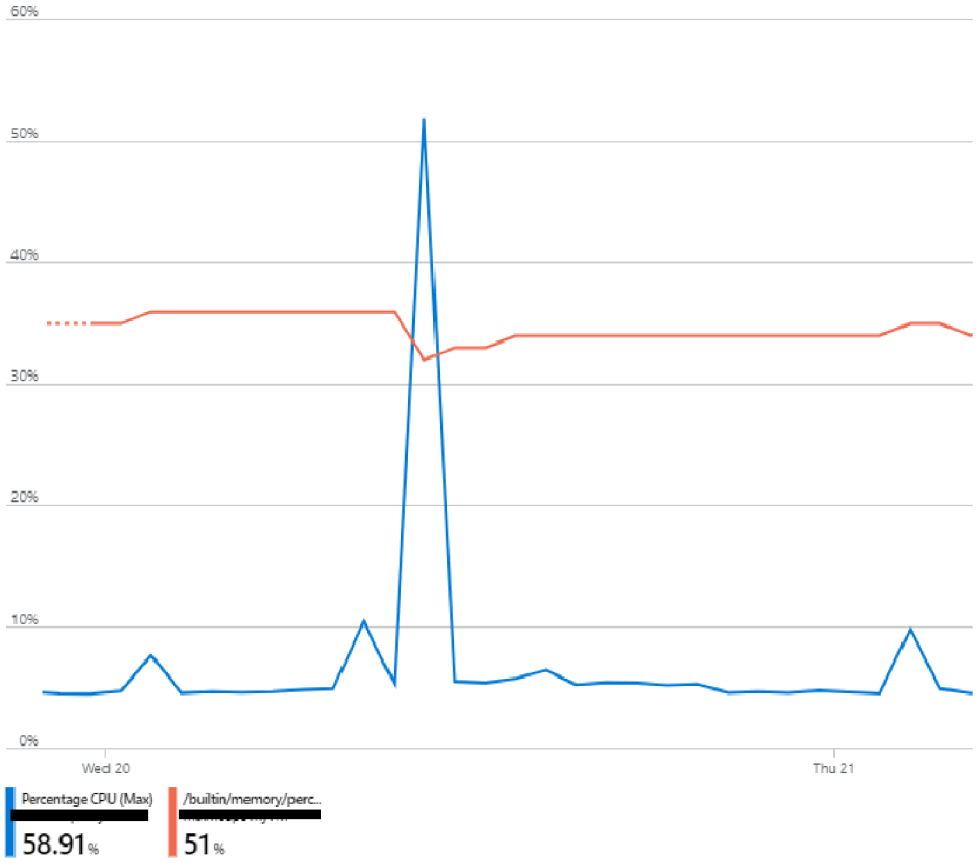
Rodina	Velikosti	Popis
Obecné použití	B, Dsv3, Dv3, Dasv4, Dav4, DSv2, Dv2, Av2, DC, DCv2	Vybalancovaný výkon CPU k paměti. Vhodné pro vývoj a testovaní, malé a střední databáze nebo nízký a střední provoz webových serverů.
Výpočetně optimaliz.	Fsv2	Vysoký poměr mezi výkonem CPU ku RAM. Vhodný pro střední provoz webových serverů, aplikační servery a pro dávkové procesy.
Paměťově optimaliz.	Esv3, Ev3, Easv4, Eav4, Mv2, M, DSv2, Dv2	Vysoký poměr RAM ku CPU. Vhodný pro databázové servery, střední a velké chatch paměti a vnitro paměťové analýzy.
Úložně optimaliz.	Lsv2	Vysoká propustnost disku a IO. Ideální pro velké databáze.
Graficky optimaliz.	NC, NCv2, NCv3, ND, NDv2 (Preview), NV, NVv3, NVv4	Specializované VM pro graficky náročné operace nebo strojové učení.
Vysoko výkonné	HB, HBv2, HC, H	Nejrychlejší a nejvýkonnější CPU aplikace s optimalizovaným vysoce optimalizovaným internetovým rozhraním.

Tabulka 5.8-1 Rony VM [20]

Jednotlivé velikosti je možné dohledat dál v dokumentaci k VM od microsoftu. Pro tento typ úlohy vyhověl nejlépe v poměru cena výkon VM z rodiny pro obecné použití ve velikosti Bm1S. Serveru se dá kdykoliv změnit velikost a výkon Velikost a rodina serveru je spojena z jeho fyzickými atributy jako je počet virtuálních jader procesoru, velikost RAM paměti, počet datových disků, dočasné uložiště a podpora SSD.

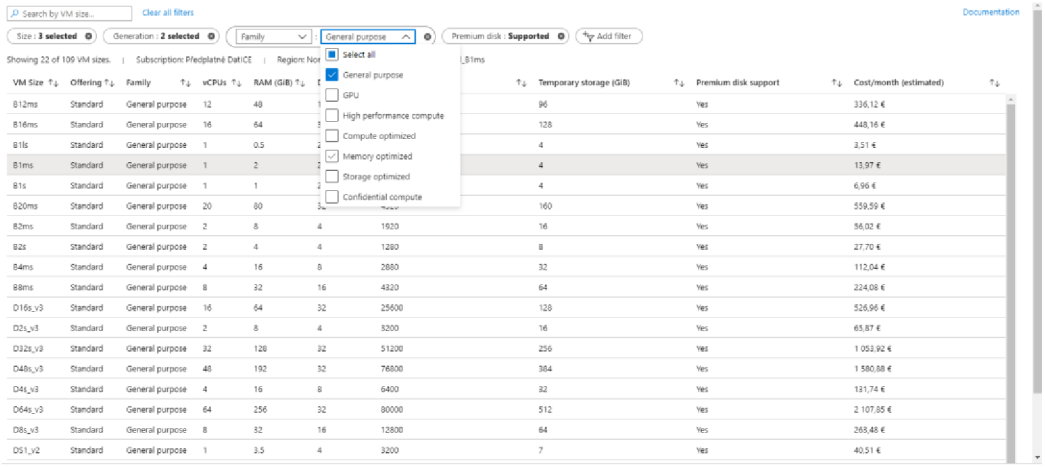
B1ms: 1 vCPU, 2 GiB RAM, 2 Data disky, 4 GiB dočasného uložiště.

[12] VM – Virtual Machine (virtuální stroj)



Obrázek 5.8-1 Využití CPU a RAM serveru za 24 hodin provozu

Při větším počtu přihlášených uživatelů se využití paměti přehoupne přes 50% což mi znemožňuje využít menší velikosti serveru.

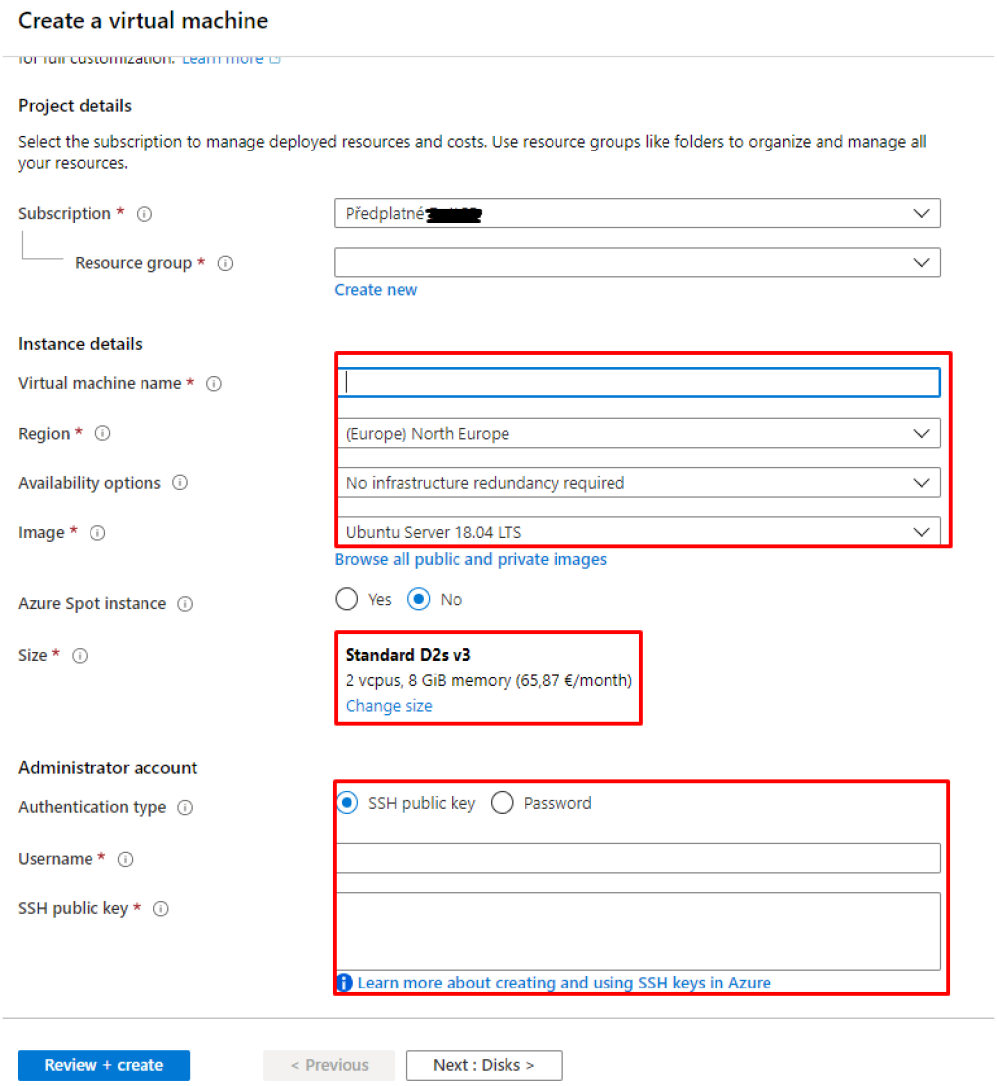


Obrázek 5.8-2 VM z rodiny Obecné použití

5.8.2 Konfigurace serveru

Jako první je nutné zvolit požadovaný operační systém serveru v tomto případě linuxová distribuce Ubuntu Server 18.04 LTS. Tento operační systém je bezplatný a open source.

Další důležité prvky konfigurace nového VM jsou jeho umístnění (odvíjí se cena podle zvolené lokality), velikost a způsob přihlašování k nově vytvořenému serveru. Na výběr je mezi heslem nebo SSH klíčem.

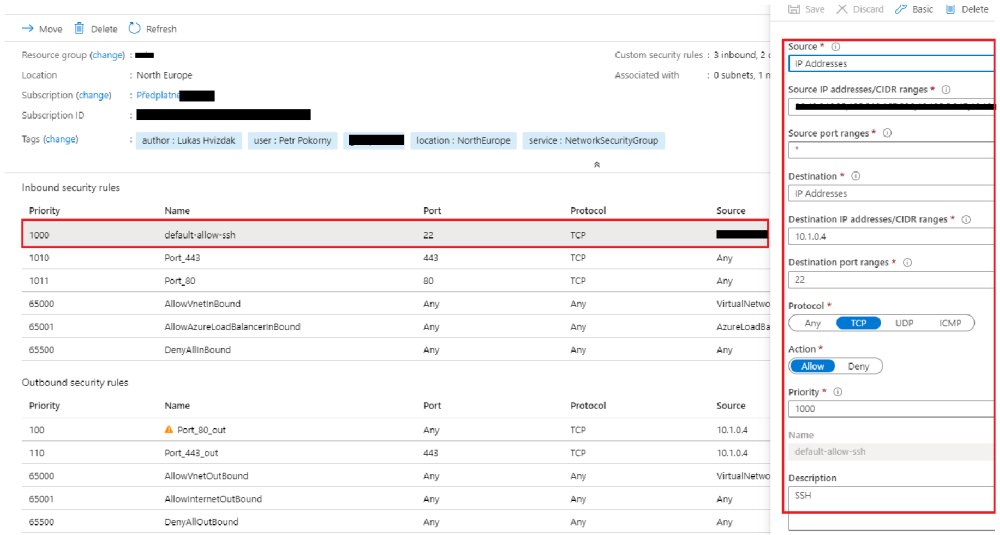


Obrázek 5.8-3 Základní výběr možností při konfiguraci VM

Já osobně preferuji vygenerovat pomocí putty klíč a používat jej pro autorizaci společně s povolenou IP adresou odkud se připojuji pro port 22.

Další položky obsahují základní nastavení virtuální lokální sítě, nastavení zabezpečení firewallu a konfigurace veřejní IP adresy serveru. Nebudu zde procházet jednotlivé nastavení. Ukážu pouze výsledné nastavení firewallu umožňujícího přistup a komunikaci serveru směrem ven i dovnitř. U nastavení veřejné IP je důležité nastavit pevnou IP, jelikož je standartně nastavená na dynamickou.

Firewall zajišťuje služba Network Security Group.



Obrázek 5.8-4 Network security group nastavení ip a portů pro přístup k serveru

Tento firewall není pouze pro jeden VM ale pro celou virtuální síť. Dělí se na dvě částí příchozí a odchozí komunikace.

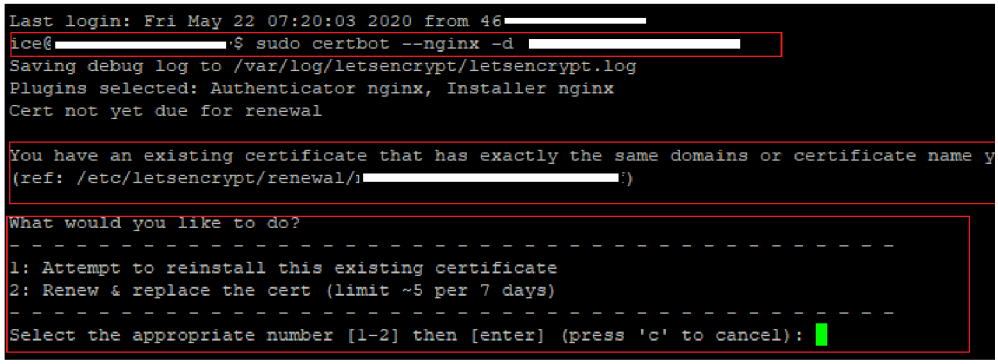
Po připojení k VM přes putty následuje instalace důležitých komponent. Certbot zajišťuje certifikaci domény a serveru, nginx slouží jako reverzní proxy server, a nakonec samotná grafana, která je schovaná v vnitřní virtuální síti uvnitř VM.

5.8.2.1 Certifikace s certbotem

Je nutné mít instalovaný modul pro nginx.

```
sudo certbot --nginx -d foo.bar
```

Tento příkaz spustí certbota, který následně pomocí certifikační autority certifikuje nginx na VM na danou doménu v tomto případě foo.bar.



Obrázek 5.8-5 Odpověď certbotu na příkaz¨

Ve chvíli psaní této práce již byli všechny servery certifikované proto, místo klasického průběhu na to certbot na začátku upozorňuje a dává na výběr mezi obnovením certifikátu před koncem doby nebo přeinstalováním certifikátu.

Certbot je opensource program vytvořený Electronic Frontier Foundation jako část většího balíků programů zajišťující bezpečné šifrované spojení HTTPS se zajištěním úplné anonymity, soukromý a na svobodě projevu. Jako certifikační autoritu používá Let's encrypt.

5.8.2.2 Certifikace SSL/TLS

TLS neslouží k zabezpečení dat v koncovém systému. Pouze se stará o jejích nečitelnost během přenosu po nezabezpečeném internetu, kde by komunikaci mohl kdokoliv číst, pokud by nebyla šifrovaná. [21]

TLS používá kombinaci symetrického a asymetrického šifrovaní. [21]

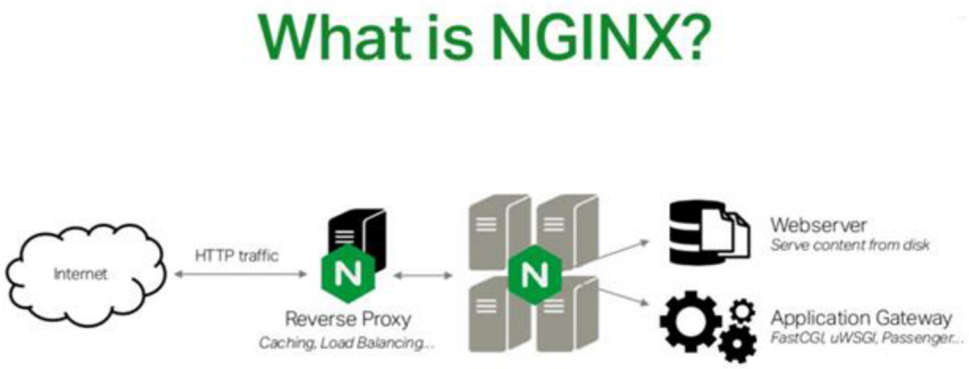
Symetrické šifrování spoléhá na soukromý klíč (private key), který slouží pro zašifrování i dešifrování zpráv na obou stranách přenosu. Nevýhodou tohoto přístupu je nutnost znát klíč na obou stranách. To znamená, že musí dojít k jeho přenosu po sítí. To dává příležitost tento klíč přečíst a použít pro dešifrování zpráv třetí stranou pro, kterou zpráva není určená. Výhodou je naopak nenáročnost na výkon a čas nutný k dešifrování. Standartně se doporučuje délka klíče 256 bitů. [21]

Asymetrické šifrování používá pár veřejný, soukromý klíč. (public, private keys). Veřejný klíč je matematicky odvozený od soukromého ale má pouze omezenou délku což zaručuje, že nepraktické a náročné odvodit soukromý klíč z veřejného. To se využívá k tomu, že veřejný klíč je odeslán druhé straně, aby data pomocí tohoto klíče zašifrovala. Dešifrování potom probíhá za pomocí soukromého klíče. Nevýhoda tohoto typu šifrování je v nutnosti mít dostatečně dlouhý klíč, kdy minimum je 1024 bitů ale doporučuje se 2048 bitů dlouhý klíč. [21]

Z důvodu náročnosti tak TLS využívá kombinaci těchto dvou metod, kdy asymetrické šifrování slouží pouze pro vytvoření předávacího klíče, který pak slouží k šifrovaní zpráv. Předávací klíč je pouze pro jeden přenos a potom je zahozen. Při dalším přenosu se postup opakuje. [21]

CA, Certifikační Autorita je třetí strana, která přidává bezpečnost do přenosu v podobě ověřování platnosti a autentičnosti serverového veřejného klíče. X.509 certifikát k tomuto účelu používá servery třetích stran, které jsou všeobecně uznávané jako důvěryhodné (Let´s encript atd..). [21]

5.8.2.3 Konfigurace nginx



Obrázek 5.8-6 Nginx jako reverse proxy webserver [22]

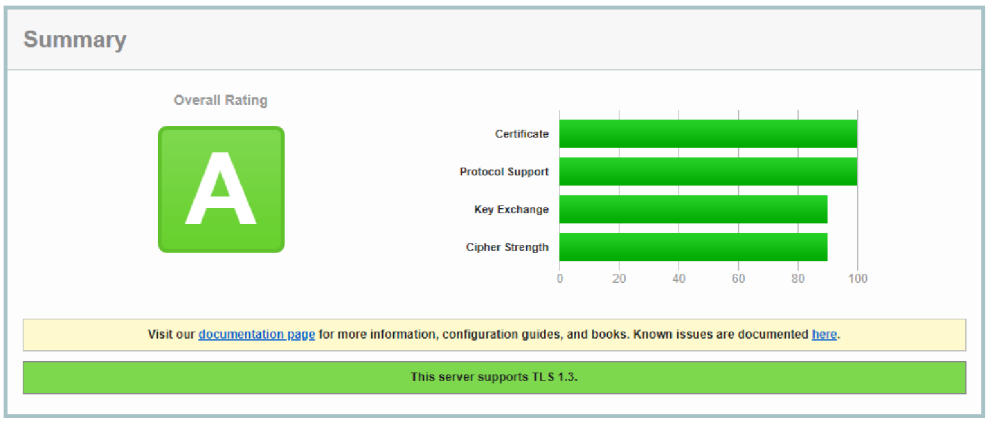
Nginx je open source program pro webové služby. Má řadu funkcí jako je reverzní proxy, ukládání do mezi paměti, vyrovnávání zatížení mezi servery, streamování médií a další. Začal jako webový server navržený pro maximální výkon a stabilitu. Kromě možností HTTP serveru může NGINX fungovat také jako proxy server pro e-mail (IMAP, POP3 a SMTP), reverzní proxy a vyrovnávač zatížení pro servery HTTP, TCP a UDP. [23]

Konfigurační soubor pro nginx se nachází adresáři /etc/nginx/sites-enable/default pro operační systém Ubuntu.

```
{
  listen 80;
  server_name foo.bar;
  return 301 https://foo.bar$request_uri;
}

server {
  listen 443 ssl http2;
  server_name foo.bar;
  root /usr/share/nginx/html;
  index index.html index.htm;
    ssl_certificate /etc/letsencrypt/live/foo.bar/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/foo.bar/privkey.pem;
  ssl_protocols TLSv1.2 TLSv1.3;
  ssl_ciphers HIGH:!aNULL:!MD5;

  location / {
   rewrite /(.*) /$1  break;
   proxy_pass http://localhost:3000/;
   proxy_redirect off;
   proxy_set_header Host $host;
  }

}
```

Nginx požívám v konfiguraci jako reverzní proxy server. Má to mnoho výhod jedná z nich je zabezpečení ale i možnost spravovat více lokálních web serverů (grafan) za jedním reverse proxy serverem a dynamicky jim rozdělovat výkon internetových prostředků. Další výhodou je variabilita použití a konfigurace. Pro zákazníka jsem využil možnost směřovat dotaz

z jedné domény na login stránku grafany pro klasické přihlášení a dotaz z druhé domény využít pro autentifikační proxy, která přihlásila uživatele pod sdíleným účtem v Grafaně s minimálními právy pro zobrazení jednoho dashboardu. Tuto možnost tu sice nebudu rozvádět, ale využívá se to pro zobrazení vizualizací na vzdáleně spravovaných TV panelech.



Obrázek 5.8-7 Výsledná známka od ssllabs serveru

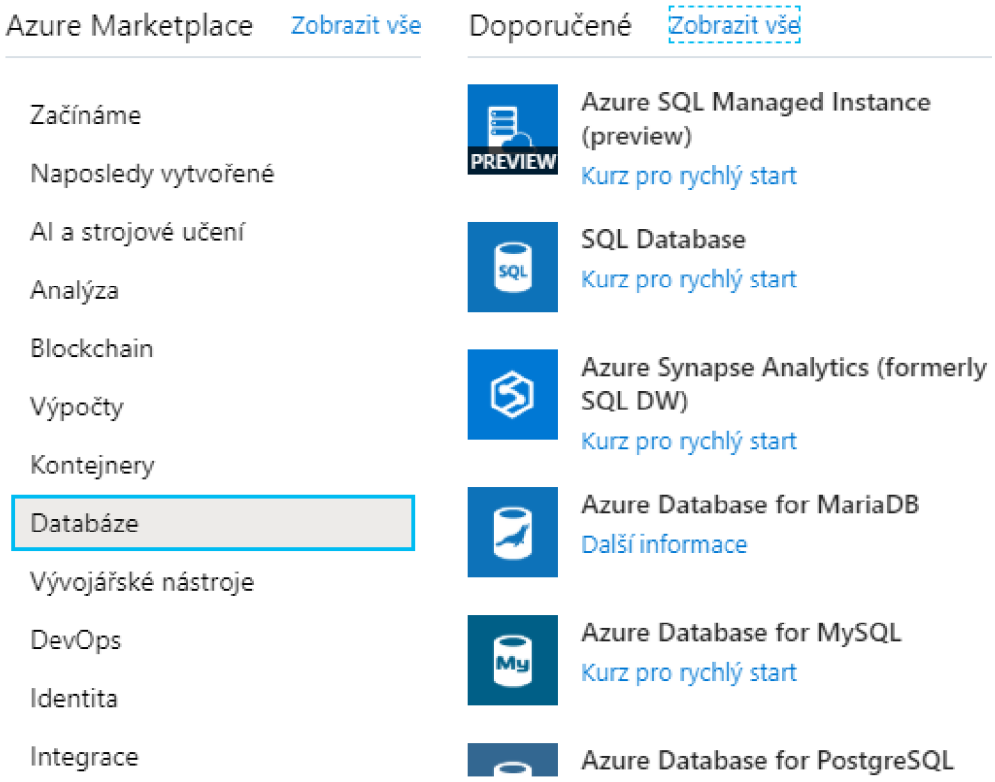
Samotně vystavená grafana na internet při certifikaci pomocí certbotu dosáhla nejlepší známku B jelikož chyběla třeba základní možnost upřesnit jaké verze TLS protokolu webserver podporuje. Dá se jistě dosáhnout lepší známky A+ pomocí výtek a návodu níže na tomto hodnotícím serveru.

5.8.3 Licence Nginx

Nginx využívá BSD licence, která je popsaná v kapitole 9 Licence.

6 Uložení dat a zpracování dat

Azure nabízí celou řadu obou druhů databází. Proto se do budoucna může lišit nasazená databáze podle potřeba konkrétního zákazníka bez toho, aniž by bylo nutné vyvíjet nové dílčí částí řešení sběru dat při integraci nového typu databáze. V tomto směru je Azure velice silný nástroj.



Obrázek 5.8-1 Část výběru služeb spojených s databázemi na portálu Azure

6.1 Základní dělení databázových systémů SQL vs NoSQL

Pokud jde o výběr databáze, největším rozhodnutím je výběr relační (SQL) nebo nerelační (NoSQL) datové struktury. I když obě databáze jsou životaschopnými možnostmi, stále existují určité klíčové rozdíly mezi těmito strukturami, které je nutné zvážit a musí být na paměti při rozhodování. [24]

6.2 SQL Jazyk

SQL databáze definují a manipulují s datovým strukturovaným dotazovacím jazykem (SQL). SQL je jednou z nejvšestrannějších a nejpoužívanějších dostupných možností, což z něj činí bezpečnou volbu zejména pro velké složité dotazy. Ale na druhou stranu to může být omezující. SQL vyžaduje, abyste pro určením struktury svých dat použili předdefinovaná schémata, než s nimi začnete pracovat. Všechna data musí mít stejnou strukturu. To může vyžadovat výraznou přípravu, která znamená, že změna struktury by byla pro celý systém obtížná a narušující. [24]

Databáze NoSQL má dynamické schéma pro nestrukturovaná data. Data jsou ukládána mnoha způsoby. To vede k tomu, že data mohou být orientována

na dokumenty, sloupce, grafy nebo uspořádána jako úložiště KeyValue. Tato flexibilita umožňuje vytváření dokumentů bez pevně definované struktury. Každý dokument může mít také svou vlastní jedinečnou strukturu. Syntaxe se liší od databáze k databázi a můžete přidat pole podle potřeby. [24]

6.2.1 Škálovatelnost

Téměř ve všech situacích jsou databáze SQL vertikálně škálovatelné. To znamená, že můžete zvýšit zatížení jediného serveru zvýšením výpočetního výkonu, jako je RAM, CPU nebo SSD. Na druhé straně jsou databáze NoSQL horizontálně škálovatelné. To znamená, že zvládnete větší provoz sdílením nebo přidáním více serverů do databáze NoSQL. Je to podobné jako přidání více podlaží do stejné budovy versus přidání dalších budov do sousedství. NoSQL se tak nakonec může stát větší a výkonnější, což z těchto databází udělá preferovanou volbu pro velké nebo neustále se měnící sady dat. [24]

6.2.2 Struktura

Databáze SQL jsou založeny na tabulce, na druhé straně databáze NoSQL jsou založeny na principu páru klíč-hodnota. Hodnota může představovat dokumenty, databáze grafů nebo tabulky s proměnlivým počtem sloupců. Díky tomu jsou relační databáze SQL lepší volbou pro aplikace, které vyžadují víceřádkové transakce, jako je účetní systém, nebo pro starší systémy, které byly vytvořeny pro relační strukturu. [24]

SQL	NOSQL
SYSTÉM ŘÍZENÍ DATABÁZÍ (RDBMS)	Nerelační nebo distribuovaný databázový systém.
Tyto databáze mají pevné nebo statické nebo předdefinované schéma	Mají dynamické schéma
Tyto databáze nejsou vhodné pro hierarchické ukládání dat.	Tyto databáze se nejlépe hodí pro hierarchické ukládání dat.
Tyto databáze se nejlépe hodí pro komplexní dotazy	Tyto databáze nejsou tak dobré pro komplexní dotazy
Vertikálně škálovatelné	Horizontálně škálovatelné

Obrázek 6.2-1 Přehled hlavních rozdílů SQL a NoSQL [24]

6.3 SQL databázové systémy

6.3.1 MySQL

MySQL je systém správy databáze. To znamená, že se jedná o část softwaru určeného k definování, manipulaci, získávání a správě dat v databázi. Nejedná se o samotnou databázi, ale pouze o to, jak ukládat a manipulovat s daty v databázi. Je založena na open source s možností široké škály možností integrace. Software MySQL DBMS lze v zásadě volně stáhnout z internetu. Poté ji lze použít tak, jak je, nebo ji upravit tak, aby vyhovovala technologii

základního systému, na kterém je nasazena, aniž by došlo ke ztrátě výkonu. [25]

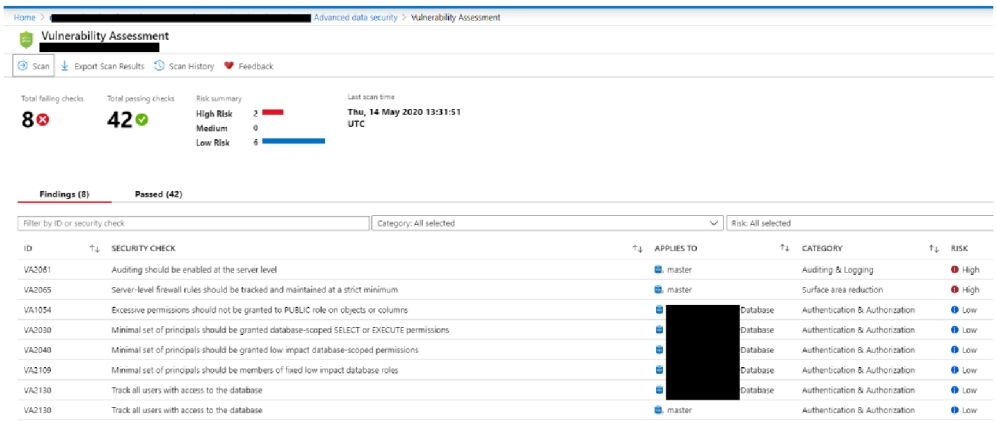
MySQL je proto velmi široce používána. Je kompatibilní a snadno nastavitelná, kompatibilní se všemi hlavními platformami operačních systémů a všemi hlavními programovacími jazyky. Tato flexibilita se však řídí GNU (General Public License). MySQL toho využívá ke kontrole toho, jak se jeho software používá. Pokud je používaná v prostředí, které porušuje tuto licenci, je možnost zakoupit komerční licenci od MySQL. [25]

6.3.2 Azure SQL

Je databáze jako služba od Azure. Nepronajímá se tak přímo server, ale platí se služba, která je škálovatelná. Není nutné konfigurovat hardware, instalovat software nebo konfigurovat rozdělení výkonu mezi databázemi a spoustu dalších věcí. Všechny administrativní úlohy a údržba je zajištěna ze strany cloudu. Na uživatele tak už připadá pouze používání databáze dle vlastního uvážení bez nutnosti v podstatě cokoliv řešit až na navýšení kapacity nebo výpočetního výkonu reprezentovaného DTU[13]. [25]

Další možností reprezentace pronajatého výkonu Azure SQL je pomocí vCores, které reprezentují virtuální procesor a jeho vytíženost. Tento model však nenabízí nižší možnost, než je 1 vCore, který odpovídá zhruba 500 DTU.

Součástí výhod Azure SQL databáze je zabudovaná umělá inteligence vyhodnocující rizika nasazené databáze. Zabezpečení všech připojení k databází pomocí TLS. Inteligentní manažer dotazů, který neustálé kontroluje výkon dotazů a automaticky zlepšuje výkonnost. Automatická změna velikosti databáze při potřebě zvýšeného výkonu. Šifrovaní uložených dat a další věci. [25]



Obrázek 6.3-1 Doporučení UI k zlepšení bezpečnosti databáze

[13] DTU – Database Transaction Units je jednotka pro měření vytíženosti Azure SQL DB. Je v ní zahrnutý fyzický výkon procesu, počet operací s procesorem a diskem a paměť RAM.

6.3.2.1 Porovnání DTU modelu s vCore pronajatého serveru

Toto srovnání je vcelku těžké a nepřesné, jelikož nikdo nevidí do pozadí výpočtu DTU a dokumentace je k tomu pouze kusá a obecná.

Nejdříve je nutné předpokládat, že počet čtení za sekundu a zápisu se projevuje v IOPS[14] přímo bez přepočtů. Potom je nutné předpokládat, že součet těchto dvou operací je celková hodnota IOPS a ještě je nutné si uvědomit, že při DTU nemá nikdo ponětí o skutečném využití RAM paměti. Na základě těchto podmínek je možné společně s použitím kalkulačky DTU od Microsoft srovnat tyto dva modely pronájmu Azure SQL databáze. [26]

Number Cores	IOPS	Memory	DTUs	Service Tier	Comparable Azure VM Size
1 core, 5% utilization	10	???	5	Basic	Standard_A0, barely used
<1 core	150	???	100	Standard S0-S3	Standard_A0, not fully utilized
1 core	up to 4000	???	500	Premium – P4	Standard_DS1_v2
2-3 cores	up to 12000	???	1000	Premium – P6	Standard_DS3_v2
4-5 cores	up to 20000	???	1750	Premium – P11	Standard_DS4_v2
6-13	up to 48000	???	4000	Premium – P15	Standard_DS5_v2

Obrázek 6.3-2 Výsledné srovnání DTU vůči vCores s přiřazenými VM odpovídajícího výkonu [26]

Z obrázku vyplívá, že první řada databáze do 5 DTU je hodně limitována hodí se pro malé, nenáročné a občasné použití, popřípadě pro uchování dat s občasným přístupem k nim. [26]

Druhý stupeň je taky limitující ale už se do něj dají vměstnat jednotlivé databáze z databázového serveru. To znamená, že pro malý počet databází z pravidelným ale nízkým vytížením a přístupem je to levnější alternativa k hostovanému serveru. [26]

Pokud však databází na serveru je více a je nutné mít více procesorový server s větším než 1 TB uložištěm je výhodnější mít server s 2 a více virtuálními jádry. [26]

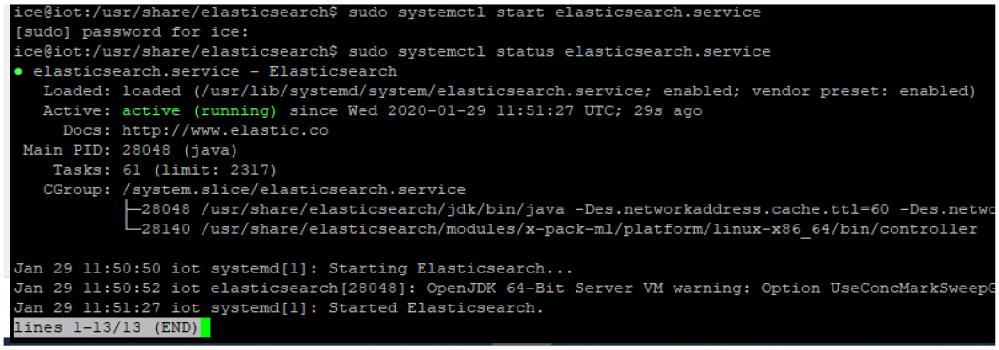
6.4 NoSQL databázové systémy

6.4.1 Elastic Search

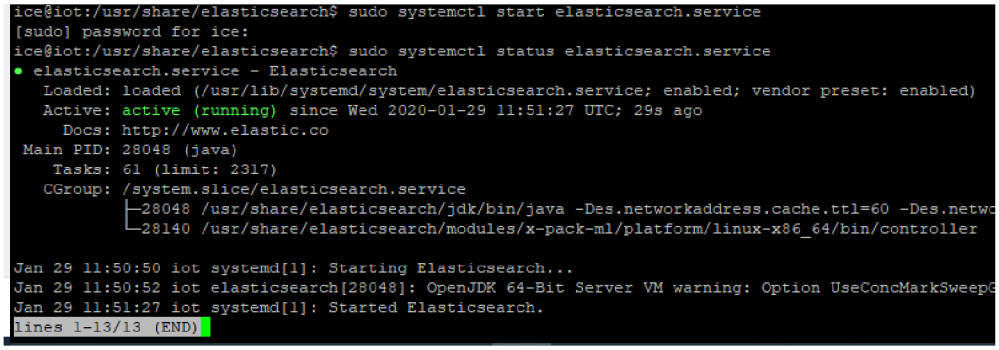
Je databázový nerelační systém s textovým vyhledáváním založený na bázi Apaceh Lucene. Data jsou ukládaná ve formě rejstříků podobně jako u relační databáze. Jediný index může obsahovat data pro uživatele (osobní informace), nebo pro podniky (adresy, pracovníky, telefonní čísla…) atd. Možnost takto rozdělit data do rejstříků (indexů) s různými prvky je klíčovou vlastností této databáze. Elastick search je designovaná především pro horizontální škálování pomocí přidání vícero serverů do clusteru. [27]

[14] IOPS – Input Output Per Second je počet vstupů výstupů do a z CPU vůči disku, RAM a perifériím.

Na prvopočátku projektu jsem zkoušel na lokální Ubuntu serveru Elastic search. S velkými obtížemi se mi podařilo nastavit všechno, aby to byla databáze spustitelná a schopna přijímat data. Bohužel společně s Eslatick search přichází nutnost použití i Logstash, který vytváří logy a strukturuje data pro databází (ELK stack). Největší problém se tak ukázal být, jak poslat data do Lgstash, který nabízí spoustu datových konektorů, ale v podstatě většina je vyvíjená komunitou. V ranní fázi projektu byla možnost data posílat přímo z IoT Hub nebo z Cosmos DB, která byla testována paralelně ES. Datový konektor pro IoT Hub měl výpadky v datech z důvodů na, které jsem nepřišel a ani jedna ze 3 variant pro připojení ke Cosmos DB nefungovala kvůli chybě v dané verzi jruby. Proto od tohoto řešení bylo upuštěno.



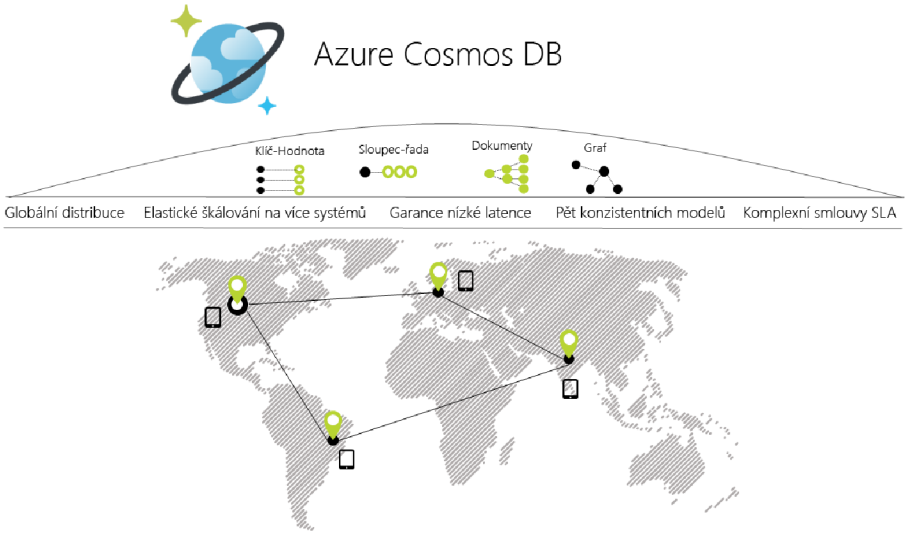
Obrázek 6.4-1 První úspěšné spuštění ElastiSearch na Ubunut serveru



Obrázek 6.4-2 První úspěšné spuštění Logstah na Ubuntu serveru

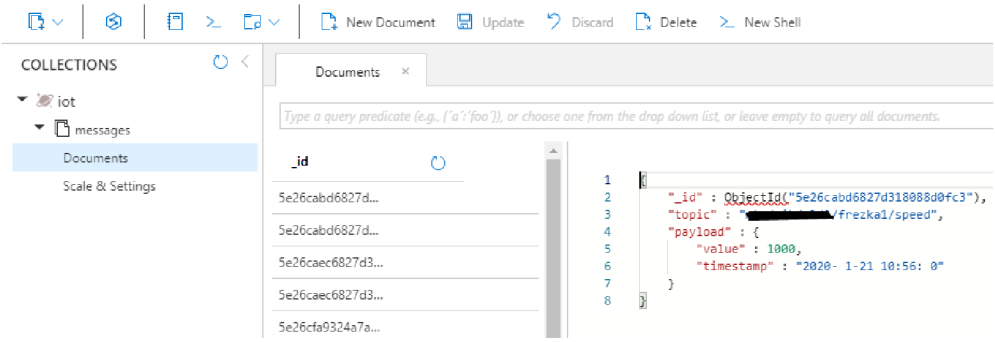
6.4.2 Cosmos DB

Cosmos DB je nerelační databázoví systém od Azure provozovaný jako služba (SaaS). Má oddělenou interní reprezentaci od externě přístupného modelu a API. Dokáže se tak chovat různě, což je fascinující a nejtransformovatelnější databáe v public cloudu. Všechny položky v dokumentech jsou automaticky indexovány. [28]



Obrázek 6.4-3 Přehlde modelů Cosmos DB [28]

Cosmos DB je hodně intuitivní a přehledná databáze ale bohužel v době testování neexistoval konektor pro připojení této databáze do Grafany. Zároveň ve firmě nebyl žádný senior, který by s ní měl zkušenosti, takže od tohoto řešení bylo pro zatím upuštěno.



Obrázek 6.4-4 Webové rozhraní pro náhled dat uložených v Cosmos DB

6.5 Databázový systém pro hot data

V rámci nulté verze jsem testoval několik možností, jak poslat data do Azure a uchovat je. Aktuálně se k tomuto účelu využívá Azure SQL databáze. Nicméně byli testovány i jiné možnosti jako NoSQL řešení. Elastic serach, blob nebo Azure Cosmos DB. Důvodem bylo srovnání ceny a případného přístupu k datům z ostatních služeb poskytovaných Azure nebo i externích. Přesná data o ceně dat jsem neuchovával ale v řádech jednotek euro měsíčně vycházelo řešení s Blob nejlevněji.

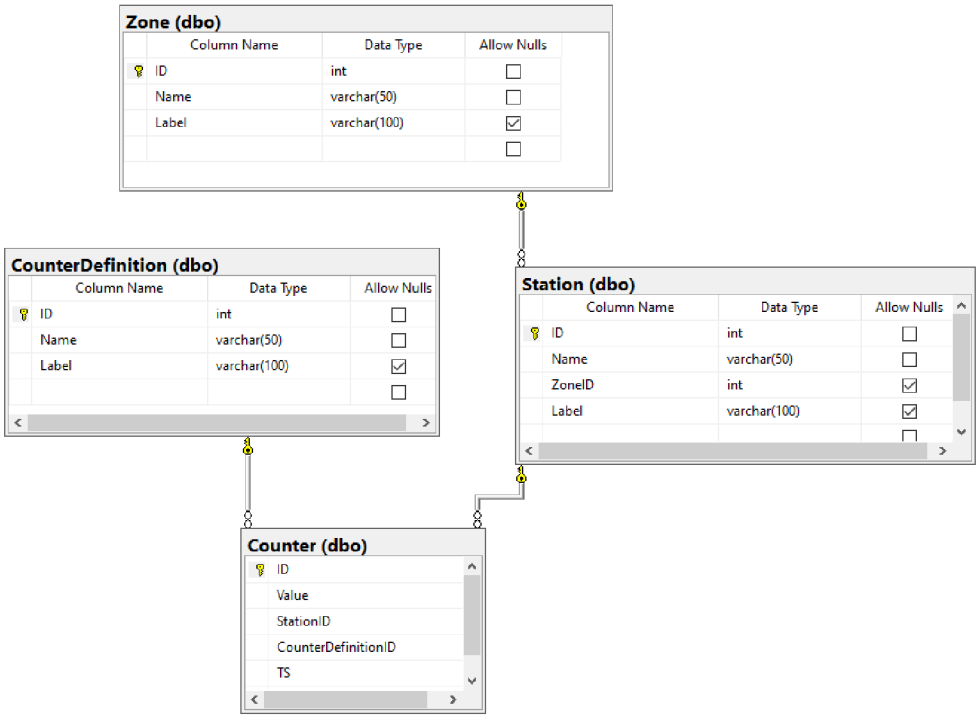
Nevýhodou byl nevyhovující přehled dat při náhledu a delší doba přístupu, jelikož se jednalo o tvz. cold uložiště. Výhodou naopak může být do budoucna formát dat JSON, který je obohacený o informace kdy dorazily data do IoT Hub, kdy odešla z aplikace na Edge zařízení a případné jiné využití dat, které

nabízí Azure a jeho služby. Rovněž je příznivá i cena uložiště pro starší data, která nejsou ihned potřeba zobrazit u zákazníka.

Na základě výše zmíněných vlastností byla pro svoji jednoduchost nasazení a nízkou cenu za malé výpočetně nenáročné databáze zvolena Azure SQL. Alespoň do určité velikosti a počtu databází. Jelikož je tato databáze založena na MS SQL je možné, že se všechny Azure SQL budou časem migrovat pod pronajatý server s MS SQL za účelem ušetření nákladu. To bude mít ale i nevýhody v nutnosti se o server starat a provádět na něm údržbu i kontrolu. Prozatím je však Azure SQL databázoví systém využívaný pro takzvaná hot data, což jsou data, které uživatel potřebuji buď v dané chvíli nebo si je může kdykoliv vyžádat a jsou pro něj důležitá v krátkodobém horizontu pár měsíců. Relační databáze obecně vyhráli nad nerelačními při volbě databázového systému z toho důvodu, že všechny data uložena budou metriky anebo klasifikace prostojů, které ale mohou být předem v databází definované. Není tak nutné vyhledávat v databází pomocí klíčových frází uložených dat.

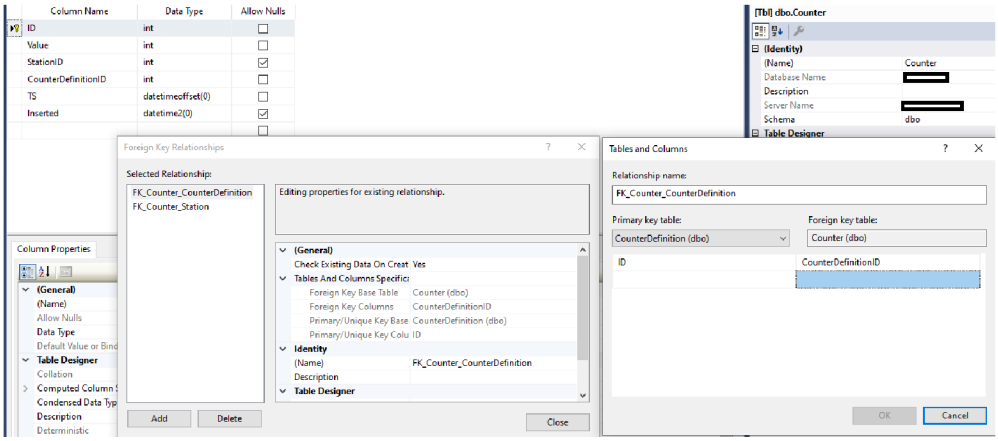
Relační databáze taky jsou lepší i pro složitější dotazy, které jsou postupem častější kvůli požadavkům zákazníku na zobrazení různých analytických metrik, které kombinují několik uložených metrik.

6.5.1 Diagram



Obrázek 6.5-1 Příklad diagramu SQL databáze

Diagram vizualizuje strukturu databáze. Skládá se z jednotlivých databázových tabulek a zobrazuje závislosti mezi těmito tabulkami. Tabulky v sql tak nemusí uchovávat pouze samotná hodnoty metrik, ale také můžou definovat význam těchto metrik (Ok, NOk kusy atd…), které jsou uložené v jiné tabulce.



Obrázek 6.5-2 Vytváření spojitostí mezi tabulkami v programu MSSQL

6.5.1.1 *Stored procedures*

Uložená procedura je předpřipravený sql kód, který je uložený v databází a je možné jej opakovaně vyvolávat a používat.

```
ALTER PROCEDURE [dbo].[ImportEvents]
(
        @Events NVARCHAR(MAX)

)
AS
BEGIN
    DECLARE @Topic varchar(50), @Value int, @Timestamp datetime2(7);
    SELECT @Topic = '$.topic',@Value = '$.payload.value',@Timestamp = '$.payload.timestamp'
    FROM OPENJSON ( @Events )

    IF (@Topic = '             frezka1/speed')
    BEGIN
        INSERT INTO [dbo].[Speed] (topic, value,timestamp)
        VALUES (@Topic, @Value, @Timestamp);

    END
    ELSE
    BEGIN
        INSERT INTO [dbo].[Load] (topic, value,timestamp)
        VALUES (@Topic, @Value, @Timestamp);
    END
END
```

Obrázek 6.5-3 Procedura, která rozděluje zprávy v JSON podle topicu do tabulek

Azure funkce posílá data do procedury skrze @Events. Objekt SqlCommand vyvolá proceduru v databází a následně ji přes metody pošle zprávu.

```
SqlCommand cmdSQL = new SqlCommand("ImportEvents", conn);
cmdSQL.CommandType = System.Data.CommandType.StoredProcedure;
cmdSQL.Parameters.AddWithValue("Events",Encoding.UTF8.GetString(message.Body.Arra
y));
```

Open JSON je funkce MS SQL a Azure SQL databáze, která dokáže překonvertovat JSON zprávu na pole prvků.

Pomocí klíčového slova DECLARE je se vytvoří dočasné proměnné, do kterých se následně přiřadí hodnoty pomocí SELECT. Proměnné odpovídají jednotlivým sloupcům tabulek. Pomocí INSERT jsou tyto hodnoty přesunuty do řádků jednotlivých tabulek.

6.6 Databázový systém pro cold data

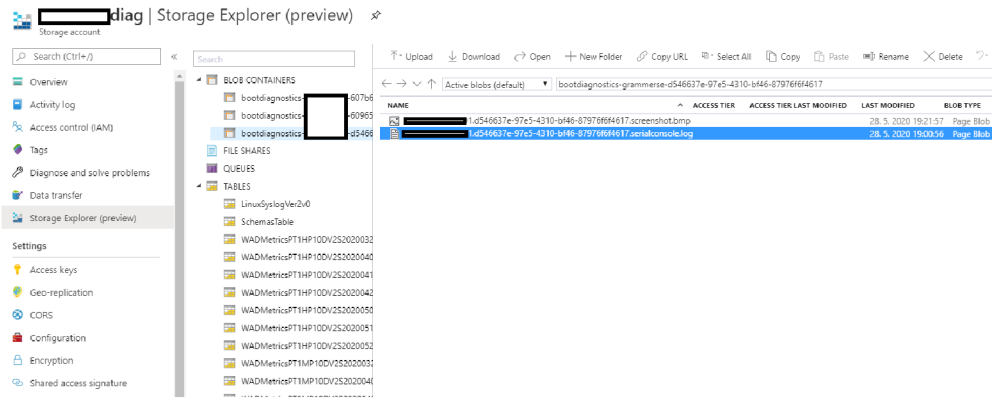
Azure Blob je uložiště optimalizované pro velké množství nestrukturovaných dat. [29]

Blob je určený pro:

- Poskytování dokumentů nebo obrazů přímo pro prohlížeč
- Ukládaní souborů sdílený přístup.
- Streamování audia a videa.
- Ukládaní záloh a jejích znovuobnovení nebo archivace dat.
- Ukládaní dat pro analýzu od lokálních služeb nebo Azure hostovaných služeb. [29]

Blob byl zkoušený ještě před Cosmos DB. Dostat do něj data je jednoduché a zároveň cena na den provozu byla zanedbatelná. Měl jedinou vadu na kráse a to, že nebylo možné číst zprávy uložené v blobu od IoT Hub jelikož je nedokázalo webové rozhraní Azure přeložit z JSON formátu v sekci náhledu pro blob. V té době bylo nejdůležitější zjistit co chodí ve zprávách do cloudu a zda jsou zprávy úplné a v jaký čas přišli oproti časovému razítku odeslání. Proto bylo od blobu upuštěno a přešlo se na Cosmos DB, kde se dalo na zprávy podívat přímo ve webovém rozhraní azure.

Od té doby nebyl Blob využit, ale je hned několik způsobu, jak do něj archivovat data z Azure SQL. Společně s tím je nejelevnější uložiště na cloudu Azure. Proto bude v budoucnu určitě využit pro uchování starších dat.



Obrázek 6.6-1 Náhled na webové rozhraní Blob uložiště, kam se ukládá diagnostika VM

6.7 Cenová politika na Azure

Azure nabízí tři základní varianty pořízení a správy služeb, softwaru a hardwaru. Veškeré platby v Azure se týkají takzvaného subscription (přeplatné), které má hodně kategorií podle velikosti služeb a velikosti částky, kterou Azure platíte za této služby. S problematikou druhů předplatných se nebudu zabývat.

Podkapitola se věnuje hlavně Pay as you go (PAYG) ale zmiňuji i Enterprise Agreement (EA) nebo Cloud Solution Provider (CSP) Dále rozeberu ceny jednotlivých služeb použitých v řešení.

6.7.1 Enterprise Agreement (EA)

Jedná se o variantu pro společnosti, co chtějí provozovat velké množství služeb a dat na cloudu. Umožňuje smluvně zahrnout celé portofilo Microsoft služeb a stává se tak partnerem Microsoftu. Tato smlouva umožňuje nabití kreditu a následné čerpání na jakékoliv služby a případné doplacení po přečerpaní. Také umožňuje rozdělit kredit mezi libovolný počet vytvořených subscription, které následně můžete přidělovat týmům lidí nebo libovolně spravovat s využitím například Azure Active Directory. Výhodou této varianty je i partnerský program DPOR, který využívají další společnosti za účelem poskytování podpory při využívání Azure.

6.7.2 Cloud Solution Provider (CSP)

Umožňuje přenechat veškerou starost o Azure a služby partnerek společnosti. To znamená, že společnost využívající tento model nemá do Azure ani přístup ale kupuje si služby a řešení třetí strany, která je přizpůsobí pro potřeby svého zákazníka a využije k tomu služeb Azure.

6.7.3 Pay as you go (PAYG)

Model vhodný pro malé společnosti nebo začátečníky s Azure, kteří by nevyužili plný potenciál cloudu anebo je pro ně zbytečné hned uzavírat smlouvu typu EA. Tento model využíváme ve společnosti, kde pracuji. Pro vývoj, testovaní, zkoumání je tento model podle mě nejlepší. Platí se pouze za využité služby, a to podle času a výkonu, který tyto služby spotřebovaly. Ke každému předplatnému je potřeba zadat kreditní kartu pro ověření identity a jako platební metodu. Tento model plateb je únosný pouze do určitého bodu pro velké částky za celé infrastruktury služeb jako se rýsují do budoucna pro tento systém sběru dat je lepší přejít pod EA.

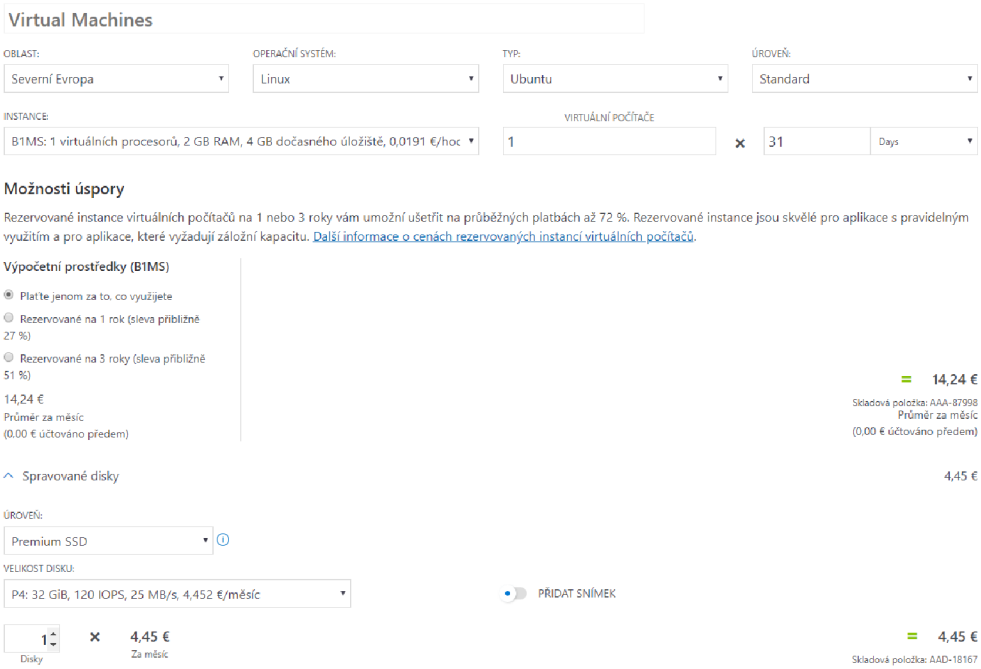
6.7.3.1 Rozdíly mezi datacentry

Azure má specifiky platební model v tom, že se liší podle lokality, která je zvolena pro hostovaní dané služby nebo infrastruktury. Cena se může změnit na vybraném uložišti při vytváření nové služby doslova ze dne na den například kvůli nasazení nového hardwaru nebo rozšíření portfolia funkcí dané služby, jelikož všechny služby na Azure jsou upravovány, vylepšovány a měněny časem. Jako příklad uvádím virtuální servery, Azure SQL nebo IoT Hub na třech různých uložištích. Tyto tři služby jsou asi nejpodstatněji pro sběr dat na straně cloudu. Nekombinuji to už s redundancí na datovém centru nebo mezi vícero datovými centry (geologická redundance). Všechny tři služby je lepší mít v jedné lokalitě, jelikož si Azure účtuje za přesun dat mezi službami umístněnými v různých lokalitách.

6.7.3.2 Virtuální servery s Ubuntu

Uvažuji velikost serveru B1MS při jednom premiovem ssd disku pro operační systém o velikost 32 GB a standartní úrovně s platbou po měsíci. Délka měsíce je 31 dnů.

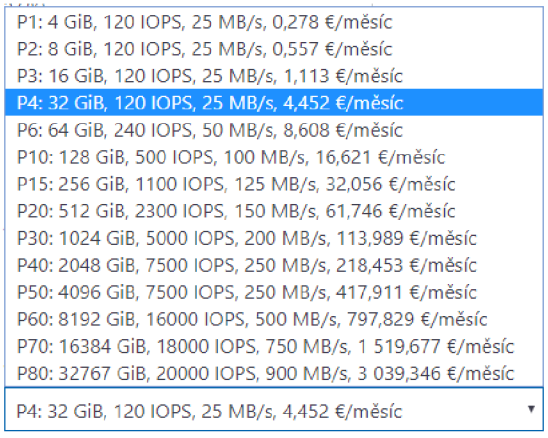
- Německo středozápad: 19,96 eur/měsíčně.
- Západní Evropa: 19,96 eur/měsíčně.
- Severní Evropa: 18,69 eur/měsíčně.



Obrázek 6.7-1 Kalkulačka Azure cloudu

Je vidět, že je možnost velké úspory, pokud je server pronajatý na celý rok dopředu při EA.

Samotné disky mají svoje ceny uvedeny taky pokud se mají v rámci balíčku služby lišit.

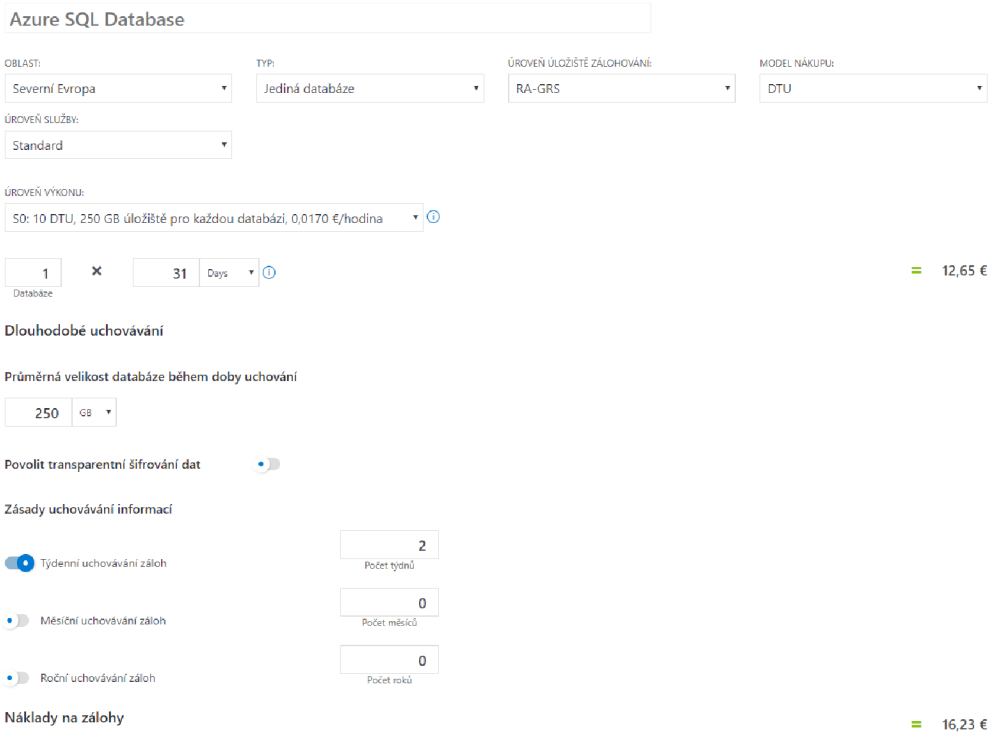


Obrázek 6.7-2 Obrázek 6.7 2 Ceny premium SSD disků v Severní Evropě

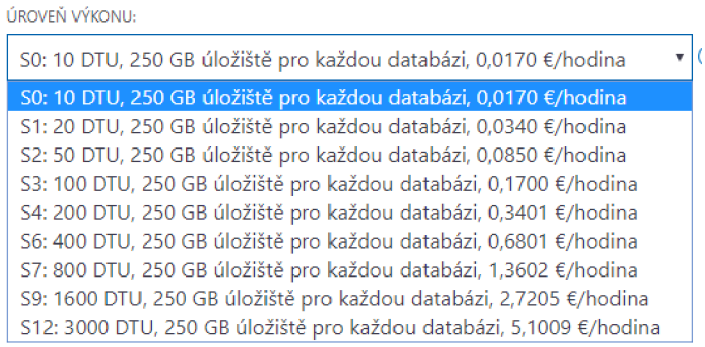
6.7.3.3 Azure SQL

Stejně jako u virtuálních serverů budu počítat jednotný model v tomto případě zvolený v jednotkách DTU pro S0 standard s 10 DTU pořád k dispozici a maximální velikosti uložiště 250 GB. Pro větší uložiště je nutné zvolit jiný vyšší počet DTU nebo mode vCore. Dále je uvažována cena vzhledem k zálohování databáze na týdenní bázi z délkou zálohy dva týdny.

- Německo středozápad: 33,38 eru/měsíčně.
- Západní Evropa: 30,21 eur/měsíčně.
- Severní Evropa: 28,88 eur/měsíčně.



Severní Evropa se liší proti západní v ceně za výpočetní výkon a Německo má dražší jak uložiště, tak výpočetní výkon. Oproti virtuálním serverům ale není rozdíl v pronájmu na měsíc nebo rok ale cena může naskočit s příplatkovou podporou nebo jiným modelem zálohování.

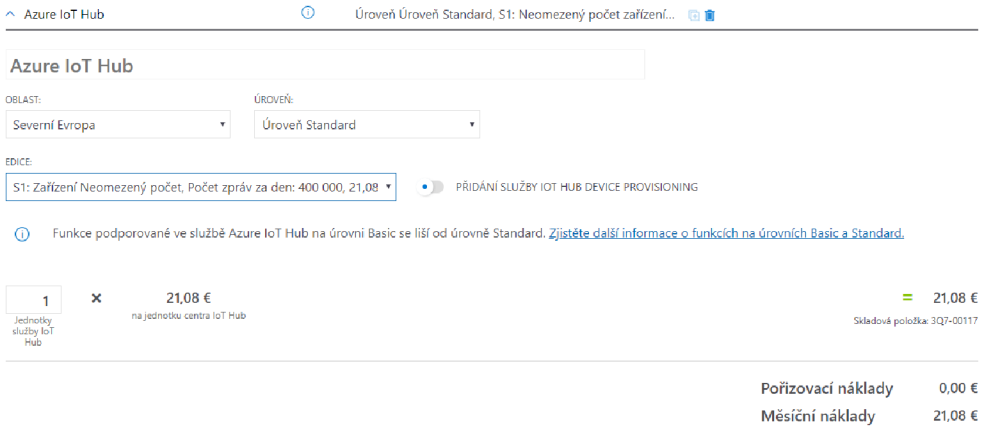


Obrázek 6.7-3 Úrovně výkonu databáze v modelu DTU pro Severní Evropu

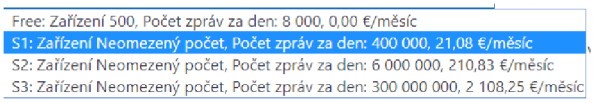
6.7.3.4 IoT Hub

Cenová politika IoT Hub je asi nejjednoduší. Pro standartní úroveň jsou zde čtyři verze. Porovnání je pro úroveň S1, která umožňuje připojit neomezený počtu IoT Edge zařízení, 400 000 zpráv za den.

- Německo středozápad: 27,41 eur/měsíčně.
- Západní Evropa: 21,08 eur/měsíčně.
- Severní Evropa: 21,08 eur/měsíčně.



Obrázek 6.7-4 Cenová kalkulace IoT Hub S1 pro Severní Evropu na měsíc



Obrázek 6.7-5 Standartní úrovně IoT Hub pro Severní Evropu

Všechny výše uvedené ceny jsou platné k 28.05.2020 a mohou se lišit v budoucnu.

6.7.3.5 Stream analytics vs Azure Functions

Srovnání uvádím z důvodu, že obě služby vykonávali stejnou funkci v tomto konkrétním případě (zápis zpráv do Azure SQL). Je nutné brát ale v potaz rozdílné fungování a model služeb. Azure Funkce nemají přiřazený stály výpočetní výkon, ale při nízkém množství dat jako v tomto případě je AF naprosto dostačující s velkými rezervami.

Za čtyři dny provozu stála služba SA 4,20 euro. Azure Functions nestála za pět měsíců provozu ani jedno euro, jelikož systém nevyčerpal základní bezplatný grand poskytovaný cloudem na každý měsíc.

MĚŘENÍ	CENA	BEZPLATNÝ GRANT (ZA MĚSÍC)
Doba spuštění*	€0,000014/GB-s	400 000 GB-s
Celkový počet spuštění*	€0,169 za milion spuštění	1 milionů spuštění

Obrázek 6.7-6 Bezplatný grand [30]

7 Vizualizace dat

Stejně důležité jako získat a uchovat data je důležité je i správně interpretovat. V tomto směru proběhlo několik zkoušek s oběma níže zmíněními řešeními vizualizace.

I když jsou obě řešení nabízené na Azure jsou mezi nimi rozdíly, které jsou pak reflektovány i při samotném provozu na portálu Azure. Klíčové vlastnosti a rozdíly jsou uvedeny níže.

7.1 Grafana

Analytická platforma pro všechny metriky. Grafana umožňuje dotazovat, vizualizovat, varovat a porozumět datům bez ohledu na to, kde jsou uloženy. Lze vytvářet vlastní nebo použít cizí dashboardy s možnosti sdílet se svým týmem.
Vytváří se v ní účty s přístupovými právy, které umožňují oddělit vývojáře dashboardů nebo analytika vytvářejícího dotazy od uživatele jehož zajímá aktuální stav produkce.

Obrázek 7.1-1 Příklad Dashboardu z webu grafany

Grafana je navržená pro vizualizaci metrik a nepodporuje fulltextové vyhledávání. Podporuje však celou řadu grafických analyzátorů dotazů jako například: graphite, influxdb, Prometheus, elasticsearch, AWS CloudWatch.

Je opravdu intuitivní na první použití s velkým množstvím návodů. Během první hodiny se mi podařilo ji spustit z PC a připojit se na cloudové uložiště a zobrazit jednoduchý graf dat.

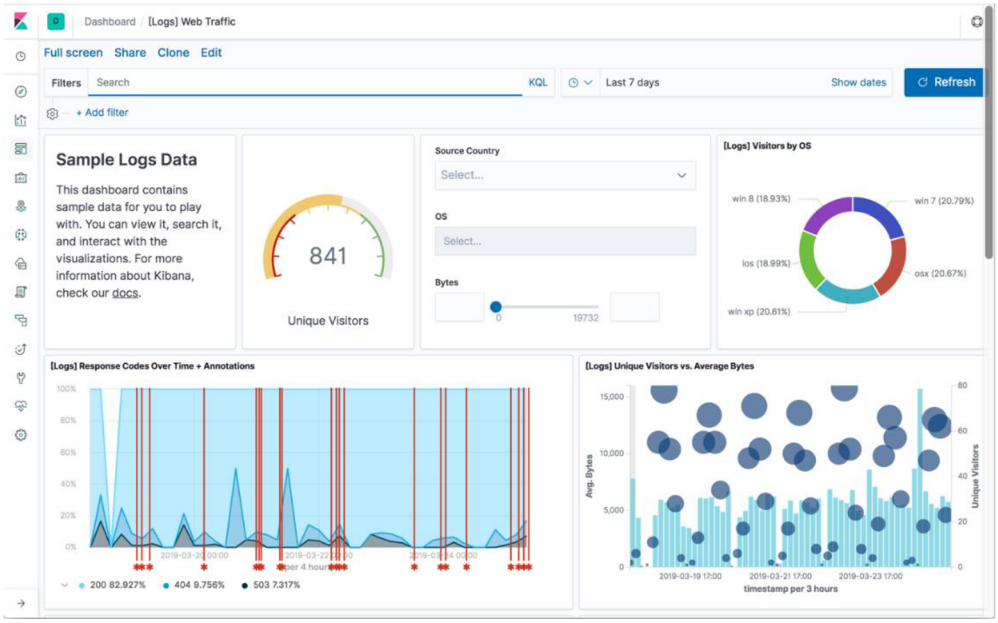
7.1.1 Licence

Grapaha je zcela open source projekt založeny na licenci Apache 2.0. Která je popsaná v kapitole 9 Licence.

7.2 Kibana

Stejně jako Grafana je Kibana analytický nástroj umožňující zobrazení dat a vytváření dashboard. Rovněž umožňuje kontrolovat přístup jednotlivých uživatelů a je možné vytvářet role. Z mé krátké zkušenosti je komplexnější a méně přehledný nástroj z back end pohledu.

Umožňuje fulltextové vyhledávání v datech díky nativní podpoře ElasticSearch. Rovněž ElasticSearch je jediný analyzátor dotazů, který Kibana podporuje.



Obrázek 7.2-1 Příklad Dashboardu v prostředí Kibana

7.2.1 Licence

Kibana využívá naprosto stejnou licenci jako Grafana, tedy Apach 2.0, která je popsaná v kapitole 9 Licence.

7.3 Grafana vs Kibana

7.3.1 Protokoly vs. Metriky¨

Klíčový rozdíl mezi těmito dvěma vizualizačními nástroji vyplývá z jejich účelu. Grafana je navržen pro analýzu a vizualizaci metrik, jako je systémové CPU, paměť, využití disku a I / O. Grafana neumožňuje fulltextové dotazování dat. Kibana, na druhé straně, běží na vrcholu Elasticsearch a používá se primárně pro analýzu zpráv protokolu.

Pokud budujete monitorovací systém, oba dokážou tuto práci dobře, i když stále existují určité rozdíly, které budou popsány níže. Pokud jde o protokoly, které sledujete, v kterémkoli z případů použití, které protokoly podporují –

řešení problémů, forenzní analýza, vývoj, zabezpečení, je Kibana jedinou možností. [31]

7.3.2 Nastavení, instalace a konfigurace

Kibana i Grafana se velmi snadno instalují a konfigurují. Podporují instalaci na Linuxu, Macu, Windows, Dockeru nebo umožňují sestavení ze zdrojových souborů. Kibana podporuje širší škálu možností instalace pro každý operační systém, ale celkově – zde není žádný velký rozdíl. Protože se Kibana používá na vrcholu Elasticsearch, je nutné spojení s vaší instancí Elasticsearch.

Grafana je konfigurována pomocí souboru INI, který je relativně snadnější zvládnout ve srovnání s konfiguračními soubory YAML citlivými na syntaxi společnosti Kibana. Grafana také umožňuje přepsat možnosti konfigurace pomocí proměnných prostředí. [31]

7.3.3 Zdroje dat a integrace

Grafana byl navržen tak, aby pracoval jako uživatelské rozhraní pro analýzu metrik. Jako takový může pracovat s více časovými řadami datových úložišť, včetně vestavěných integrací s Graphite, Prometheus, InfluxDB, MySQL, PostgreSQL a Elasticsearch a dalších zdrojů dat pomocí pluginů. Grafana má pro každý zdroj dat specifický editor dotazů, který je přizpůsoben funkcím a schopnostem, které jsou v tomto zdroji dat obsaženy. Kibana je naopak navržena tak, aby pracovala pouze s Elasticsearch, a proto nepodporuje žádný jiný typ zdroje dat. [31]

7.3.4 Upozornění

Klíčovým rozdílem mezi Kibanou a Grafanou jsou upozornění. Od verze 4.x se Grafana dodává s vestavěným upozorňovacím modulem, který umožňuje uživatelům připojovat podmíněná pravidla k panelům řídicích panelů, jejichž výsledkem jsou upozornění na koncový bod oznámení podle vašeho výběru (např. E-mail, Slack, PagerDuty, vlastní webhooks). Kibana nepřichází s varovnými schopnostmi připravenými k okamžitému použití. Chcete-li přidat upozornění pro uživatele Kibana, mohou se uživatelé buď rozhodnout pro hostovaný zásobník ELK, jako je Logz.io, implementovat ElastAlert nebo použít X-Pack. [31]

7.4 Výběr

Z důvodu nemožnosti připojit Kibanu k jiný databázovým systémům, nebo respektive s velkými obtížemi byla vybrána Grafana jako hlavní nástroj pro vizualizaci metrik ze zařízení. Power BI má pěkné prostředí, ale nenabízí možnost změny celého zdrojového kódu (kdyby to bylo třeba) a zároveň je součástí vlastního ekosystému, kdy je nutné platit za licenci pro vytváření

dashboardů pro další organizace nebo pro vývoj vlastních pluginů[15]. Společně s tím se pojí i nutnost vlastnit office 365.

7.4.1 Vývojářské prostředí pro úpravu grafany

Grafana je vyvíjená komunitou a je možné její zdrojové kódy stáhnout z github repositáře. Je koncipovaná pro vývoj na platformách Linuxu nebo macOS. Její backEnd je napsaný v jazyce GO a frontEnd je z převážné částí tvořený typescriptem s využitím teď už pouze frameworku React s dalšími knihovnami a open source programy pro webové rozhraní.

Z těchto důvodů změny v Grafaně probíhají na vzdáleném Ubuntu serveru. Jakýkoliv program je možné na platformách Linuxu vyvíjet v podstatě v textovém editoru. Osobně jsem pro přehlednost a pohodlí zvolil Visual Studio Code. Umožňuje připojení k serveru pomocí ssh na dálku. Při připojení na daný adresář projektu si automaticky stáhne všechny nutné rozšíření pro přeložení a porozumění kódu v dané složce.

7.4.2 První build developer verze Grafany

Grafana se skládá ze dvou částí frontend[16] a backend[17], které tvoří celek, se kterým interaguje uživatel ve webovém rozhraní. Každá část se dá kompilovat nezávisle na sobě. Grafanu je možné stáhnout z jejího repositáře na github kde je i struční návod na kompliace obou částí. Důležité je mít všechny nezbytné komponenty, která využívají obě částí (Go, Node.js, yarn, Git). Go je programovací jazyk, v kterém je napsaný backend Grafany. Důležité je mít aktuální verzi Go. Node.js umožňuje běh webových aplikací napsaných v javascriptu a html. Grafana si z Node.js využívá pouze balíčkový manažer yarn, který je postavený na nativní manžeru npm standartně přítomný v Node.js. Je vyžadována verze Node.js větší než 12 ale menší než 13, což odpovídá poslední stabilní verzi 12.17 LTS ale aktuální verze je 14.3. Tuto podmínku jde obejít přepsáním package.json ale může to vést k chybě při kompilaci frontendu. Po instalaci Go je nutné natavit domovský adresář kam chce umístit projekty v tomto případě Grafanu. Slouží k tomu následující příkaz. Já jsem zadal dovoský adresář na Ubuntu. Je to důležité jinak se nedá grafana stáhnout a kompilovat pomocí Go.

```
export GOPATH= /home/<Název adrsáře>
```

Stáhnutí Grafany a umístnění do zvoleného adresáře je příkazem.

---

[15] Plugin – modul pro nadřazený systém, nemůže být spuštěn sám o sobě. Přidává funkcionality do celkového řešení.

[16] FrontEnd – je část programu, která složí pro vytvoření zobrazení a uživatelského rozhrání tzv. prezenční vrstva do které patří HTML kód a TypeScript.

[17] BackEnd – je část programu, která slouží pro manipulaci s daty, propojuje program s hardwarem na, kterém běží. V tomto případě je to kód napsaný v jazyce GO.

```
go get github.com/grafana/grafana
```

Po stáhnutí je nutné přejít do adresáře Grafany kde se nachází soubor Make společně se všemi zdrojovými soubory:

/home/<Název adrsáře>/go/src/github.com/grafana/grafana/

Následující postup kompilace obou částí už je jednoduchý a lze jej najít i na githubu Grafany.

7.4.2.1 FrontEnd

Kompilace frontend je jednoduchá stačí dva příkazy. První je příkaz pro stažení všech knihoven na, kterých je projekt závislí. Po vykonání jsou umístnění v adresáři Grafany ve složce node_modules.

```
yarn install --pure-lockfile
```

Druhý příkaz je může být s přívlastkem build, start nebo dev. Každý má trochu jiný průběh kompilace ale k prvnímu spuštění a ověření funkčnosti grafany to stačí.

```
yarn start
```

7.4.2.2 BackEnd

Kompilace backendu proběhne spuštěním jediného příkazu a pokud nejsou prováděné úpravy.

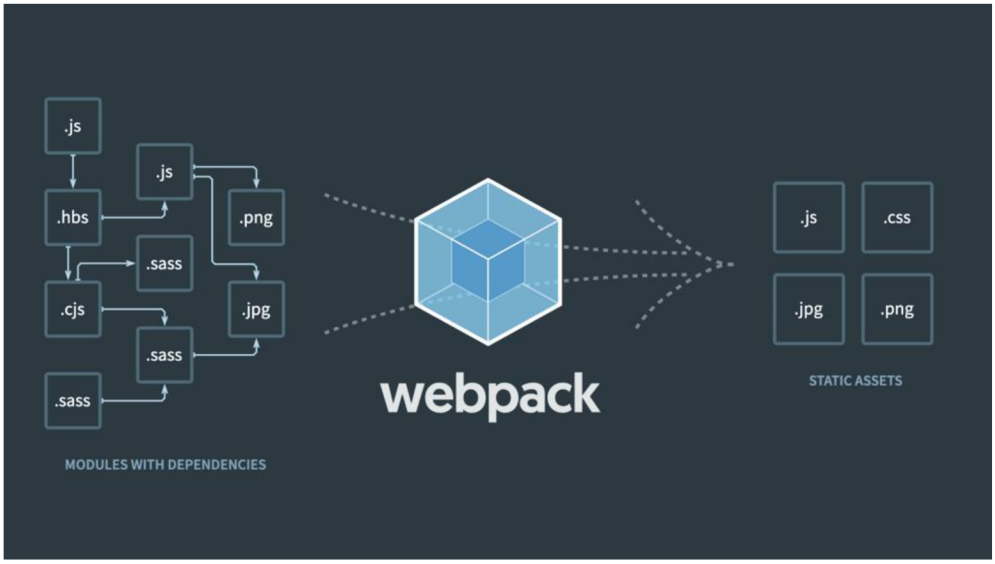
```
make run
```

7.4.3 Branding Grafany

Jedná se o změnu vizuálu, aby reprezentovala celkové řešení dodané zákazníkovi i když Grafana tvoří pouze dílčí část celého sběru dat je nejdůležitější z pohledu uživatele. Logo a pozadí přihlašovací obrazovky jsou práce grafika. Mým úkolem bylo tyto dvě částí implementovat do Grafany změnit barvené schéma s použitím barev společnosti kde pracuji společně s dalšími úpravami, které byly navrhnuté kolegy pro větší přehlednost a vizuální čistotu vizualizace. Úpravy proběhli pro verzi Grafany 6.6.0 a na aktuálně nově vypuštěné verzi 7.0. je nutné je provézt znova jelikož došlo ke změně zdrojových souborů.

Grafana používá kombinaci typescript souborů společně sass soubory a html soubory. Tyto dílčí částí jsou propojeny mezi sebou soubory s názvem index. K propojení a vytvoření celku, který je vidět ve webovém rozhraní slouží nástroj Webpack, který je součástí zdrojového kódu Grafany. Webpack je nástroj pro zpracování souborů a usnadnění práci vývojářů. Jde o kombinaci

balíčkovače (module bundler jako je Browserify) se spouštěčem úloh (task runner jako jsou Gulp nebo Grunt).



Obrázek 7.4-1 Webpack

Osobně jsem si při práci Webpack nijak nešahal. Nechal jsem mu jeho výchozí konfigurací, jelikož jsem nepřidával nové soubory do kompilace pouze měnil nebo ubíral stávající komponenty. Výstupem webpacku jsou soubory formátu javascript, které potom spouští webový prohlížeč. Jsou umístněný v složce publick v domovském adresáři Grafany.

7.4.3.1 TypeScript

TypeScript je open-source programovací jazyk vytvořený a spravovaný firmou Microsoft. Jedná se o nadstavbu nad jazykem JavaScript, která jej rozšiřuje o statické typování a další atributy, které známe z objektově orientovaného programování. Samotný kód psaný v TypeScriptu se kompiluje do JavaScriptu. [32]

7.4.3.2 Sass

Sass je kompilovaný jazyk, který rozšiřuje syntaxi CSS o proměnné, cykly, podmínky, mixiny, funkce aj. Šetří čas, množství napsaného kódu, je přehlednější a snadněji se udržuje. Samotný SCSS soubor nelze spustit, jelikož prohlížeč nezná jeho syntaxi a nemůže tedy ostylovat elementy. [33]

7.4.3.3 Změna vizuálu

Vizuál je součástí celé řady souborů napsaných v typescritpu, sass nebo html. Záleží na konkrétní části Grafany. Po kompilaci jsou všechny soubory podílející se na vizuálu v složce public, kterou tak stačí po úpravě vždy nahradit v nově nainstalované Grafaně stejné verze.

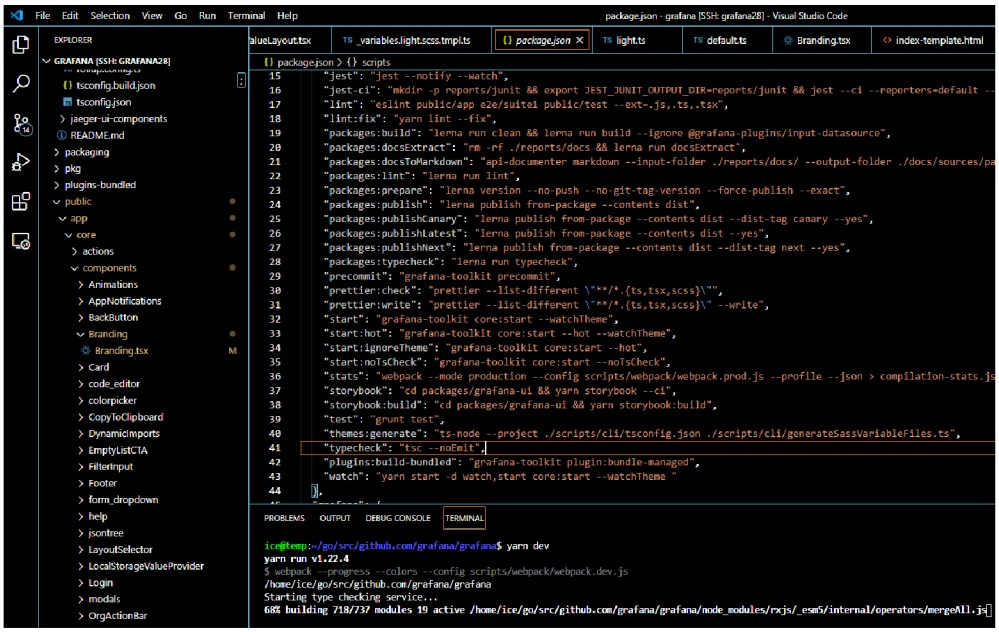
Obecně se dá říct, že sdílená barevná paleta je obsažena v sass souborech _variables.dark.generated.css a _variables.light.generated.css. Tyto soubory jsou však generovaný pomocí skriptu a pro jejich úpravu je nutné nejdříve upravit zdrojové soubory v typescriptu.

Nové proměnné, které obsahují nové barvy pro barevnou paletu se přidávají do souborů light.ts, dark.ts a default.ts v adresáři grafana/packages/grafana-ui/src/theme/. Samotné styly jsou potom generovány ze souboru _variables.light.scss.tmpl.ts a _variables.dark.scss.tmpl. Po požadované úpravě je nutné nejdříve vygenerovat nové css styly pomocí příkazu:

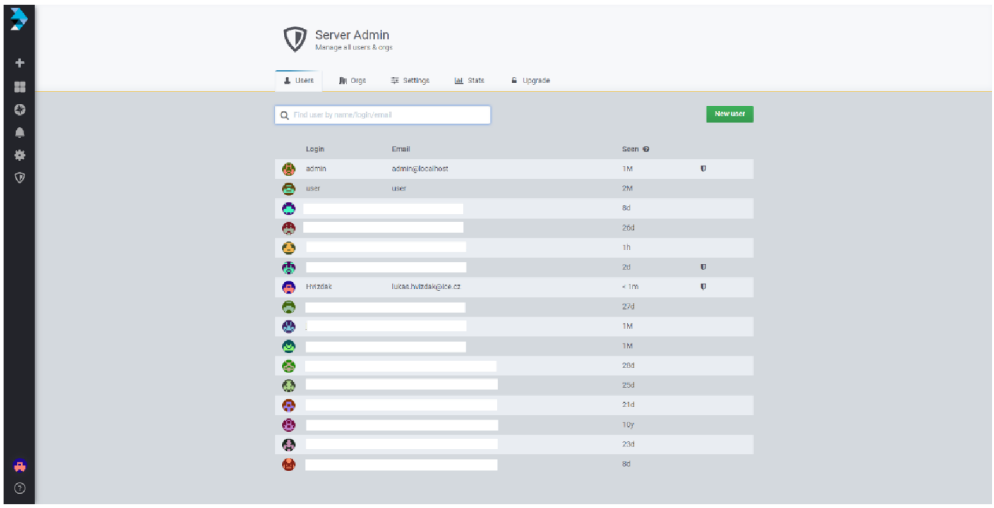
```
yarn themes:generate
```

Potom se spustí klasický příkaz pro kompilaci frontedn stejně jako při první instalaci (yarn dev).

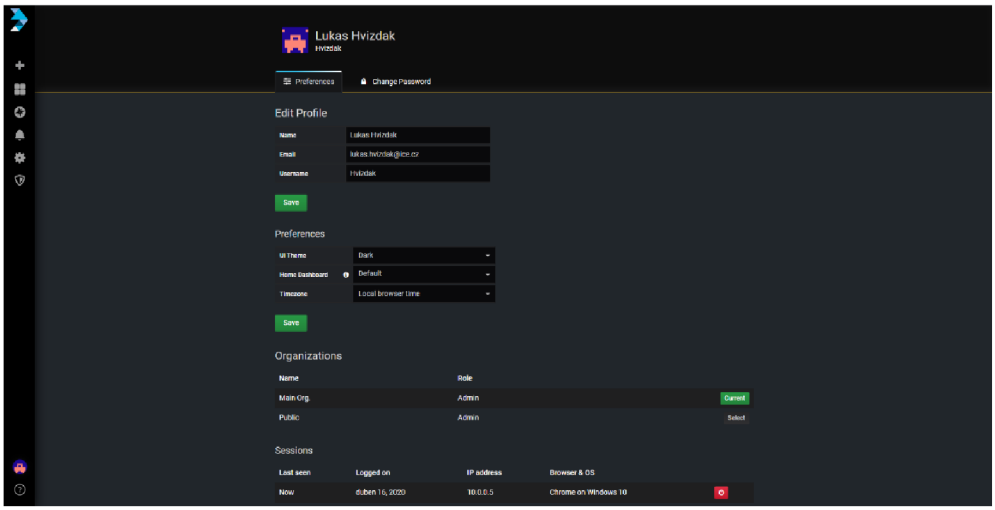
Změna jednotlivých komponent vizualizace se potom provádí v souborech, které definují tyto prvky. Například vlastnosti a funkcionalita záložek (tabs) je v souboru tabs.tsx v adresáři packages/grafana-ui/src/components. Pozadí a ikona pro menu jsou v souboru Branding.tsx. Jedná se o jednoduché úpravy pouze je nutné si příslušné soubory najít.



Obrázek 7.4-2 Pohled na jednotlivé příkazy a vývojové prostředí VS Code v adresáři Grafany



Obrázek 7.4-3 Druhá verze vizuálu světlého schématu



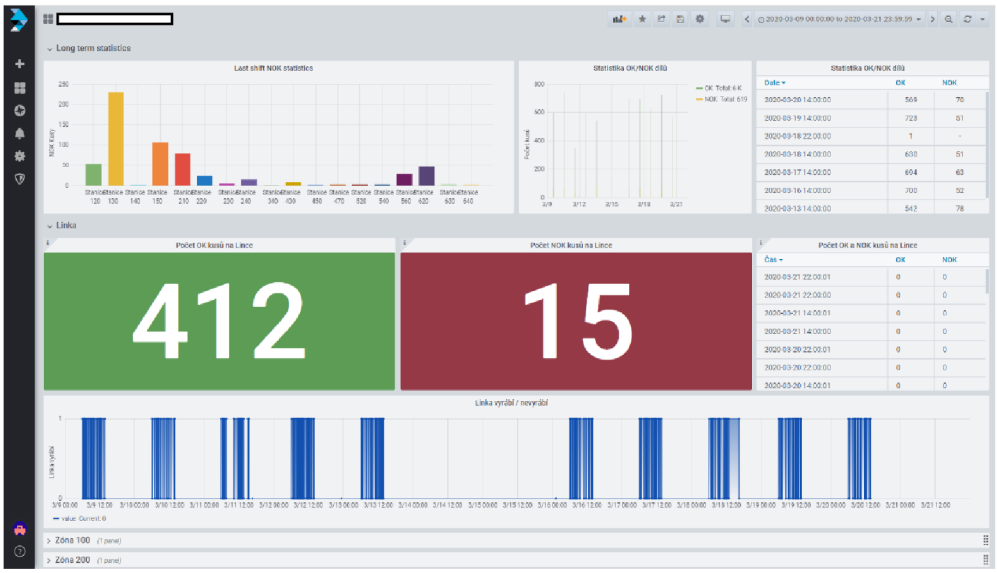
Obrázek 7.4-4 Druhá verze změny vizuálu tmavé schéma

7.4.4 Možnosti vývoje pluginů

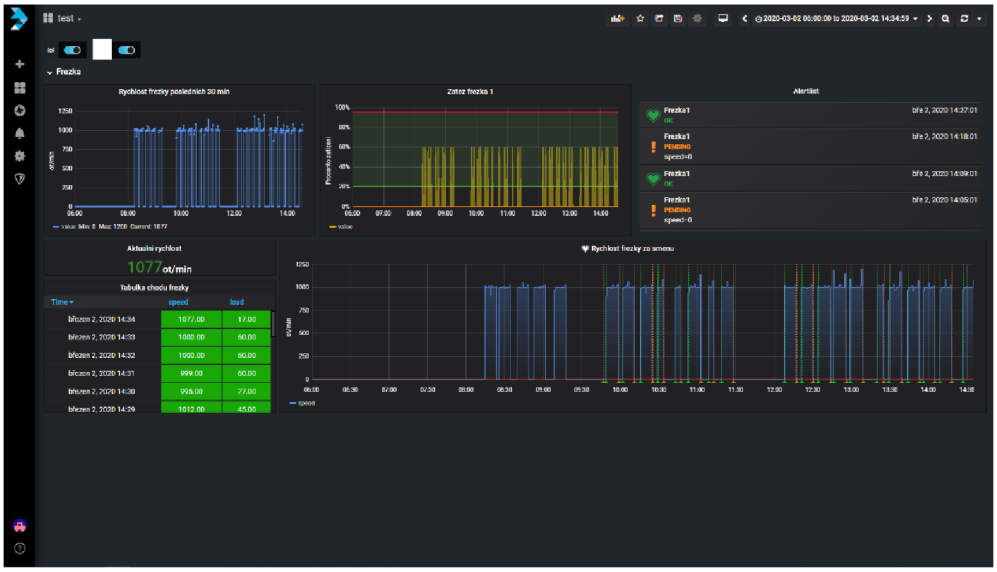
O vytvoření nového pluginů se stará grafana-toolkit, který je součastí souborů, které se stahují z githubu. Návod je dostupný na stránkách grafana.com nebo v samotném adresáři je soubor README.md, který odpovídá návodu na stránkách. Je možnost vytvářet data source pluginy pro připojení nových databází nebo vizualizační pluginy panel plugins. Po založení nového pluginu přichází ta těší část a samotná funcionalita nového pluginu. Nově je možné vytvářet pouze pluginy, které využívají framework React. Sám jsem ještě neměl čas se tomuto věnovat naplno.

7.5 Výsledné dashboardy z testovacích provozů

Obrázky vizualizací níže jsou pro dva testovací provozy. U obou se lišili požadavky na to, co je třeba zobrazit, ale oba jsou tvořeny nativními funkcemi Grafany 6.6.0. Přesto se počítá s vytvářením nových pluginů, které budou umožňovat lepší intepretaci některých dat.



Obrázek 7.5-1 Dashboard pro první testovací provoz zachycen 25.05.2020 světlé schéma



Obrázek 7.5-2 Dashboard druhého testovacího provozu zachycen 02.03.2020 tmavé schéma

8 Datové toky

8.1 Reprezentace dat v JSON

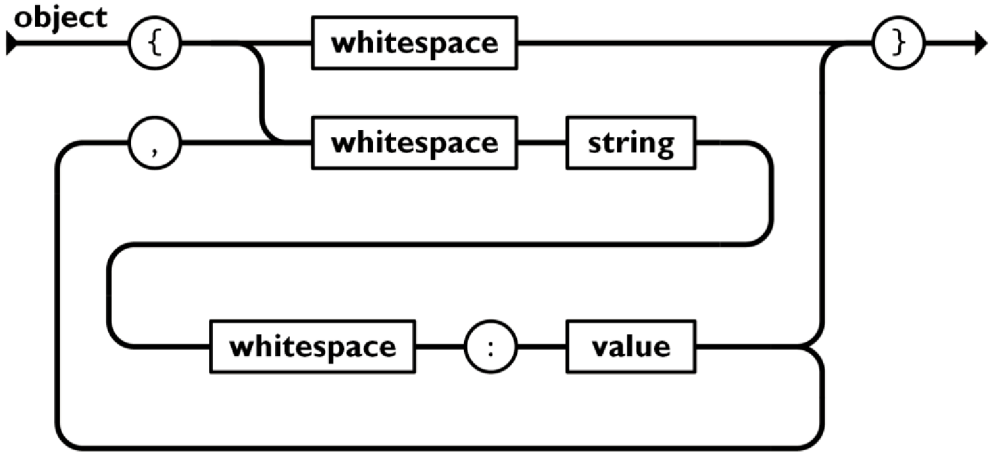
JSON (JavaScript Object Notation) je odlehčený formát pro výměnu dat. Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojově. Je založen na podmnožině Programovacího jazyka JavaScript, Standard ECMA-262 3rd Edition - December 1999. JSON je textový, na jazyce zcela nezávislý formát, využívající však konvence dobře známé programátorům jazyků rodiny C (C, C++, C#, Java, JavaScript, Perl, Python a dalších). Díky tomu je JSON pro výměnu dat opravdu ideálním jazykem. [34]

JSON se skládá ze dvou struktur:

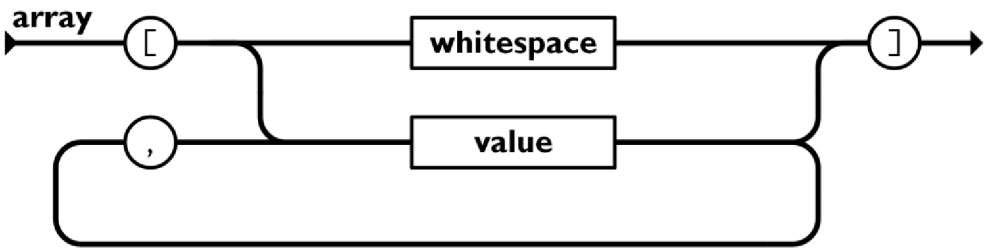
- Soubor párů název/hodnota. Rozličných jazycích bývá realizován jako objekt, záznam (record), struktura (struct), slovník (dictionary), hash tabulka nebo asociativní pole.
- Seřazený seznam hodnot. Ten je ve většině jazyků realizován jako pole, vektor, seznam (list) nebo posloupnost (sequence). [34]

Datové struktury jsou reprezentovány v podstatě stejně napříč všemi moderními jazyky, a proto je logické, aby na nich byl založený i jazykově nezávislí formát jako je JSON. [34]

Objekt je neuspořádaná množina párů název/hodnota. Objekt je uvozen znakem {levá složená závorka a zakončen znakem } pravá složená závorka. Každý název je následován znakem :dvojtečka a páry název/hodnota jsou pak odděleny znakem ,čárka. [34]



Obrázek 8.1-1 Základní konstrukce objektu v JSON [34]



Obrázek 8.1-2 Pole v JSON [34]

Příklad JSON zprávy:

```
myObj = {
  "name":"John",
  "age":30,
  "cars": [
    { "name":"Ford", "models":[ "Fiesta", "Focus", "Mustang" ] },
    { "name":"BMW", "models":[ "320", "X3", "X5" ] },
    { "name":"Fiat", "models":[ "500", "Panda" ] }
  ]
 } [43]
```

8.2 Datové buffery dle vrstev

Jak jsem psal v kapitole 4 MQTT protokol je výhodný pro pomalejší komunikaci ale s jistotou doručení zprávy. Fronty obsahující zprávy jsou tak po cestě do cloudu i v cloudu, aby se zajistilo doručení informace při přerušení spojení mezi jednotlivými komponenty.

8.2.1.1 V PLC

Nebudu uvádět přesnou funkcionalitu a kód pro PLC, jelikož jsem se na této části vůbec nepodílel ale využívá se informace o potvrzení doručení zprávy do MQTT brokeru. Pokud data nebyla odeslaná, tak zůstávají ve frontě tvořenou polem stringů. Při navázání spojení je toto pole po několika zprávách najednou vyprázdněno.

8.2.1.2 MQTT broker

Mosquitto umožňuje uschovávat nevyzvednuté zprávy ve formě fronty a zapisovat tuto frontu na disk. Přesto toto řešení není stoprocentní a může dojít ke ztrátě dat při chashi mosquitta.

V konfiguraci mosquitta jsou nutné následné nastavení:

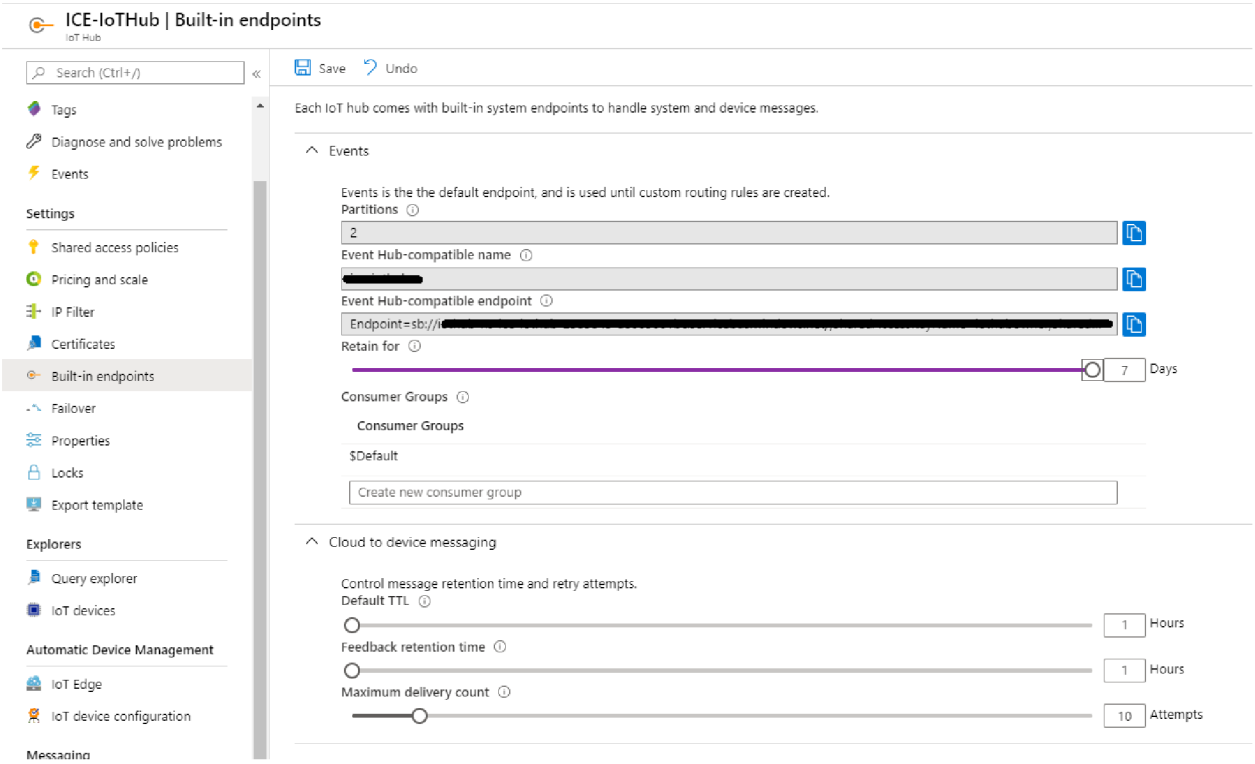
- persistence v hodnotě true. Toto nastavení brokeru říká ať zapíše stav komunikací a frontu zpráv na disk.
- max_queued_messsages <číslo>. Nastavení říká brokru kolik zpráv má udržet v paměti.

- autosave_interval <číslo> v sekudnách. Vyjadřuje, jak často má být zapsaná databáze z RAM na disk, kde běží broker.

Protože broker Mosquitto zapisuje databázi v paměti pouze na disk každých X (kde X je konfigurovatelný) sekund, můžete ztratit data, pokud dojde ke jeho crashi. Taky je tady riziko pokud klient selhává při zpracování zprávy, nikoli při jejím přijetí, můžete zprávu ztratit.

8.2.1.3 IoT Hub

Služba IoT Hub nabízí možnost uchovávat data, než budou vyzvednuta další službou nebo klientskou aplikací. K IoT Hub je možné se připojit pomocí koncového bodu (build in end point) a jeho připojovacího řetězce. IoT Hub využívá HTTPS REST API pro své koncové body.



Obrázek 8.2-1 IoT Hub buffer.

Na obrázku 8.2-1 je vidět nastavení doby po kterou IoT Hub uchovává data. Nastavení je od 1 dne po 7 dnů s krokem 1 den. Nezávisí přitom na velikosti dat.

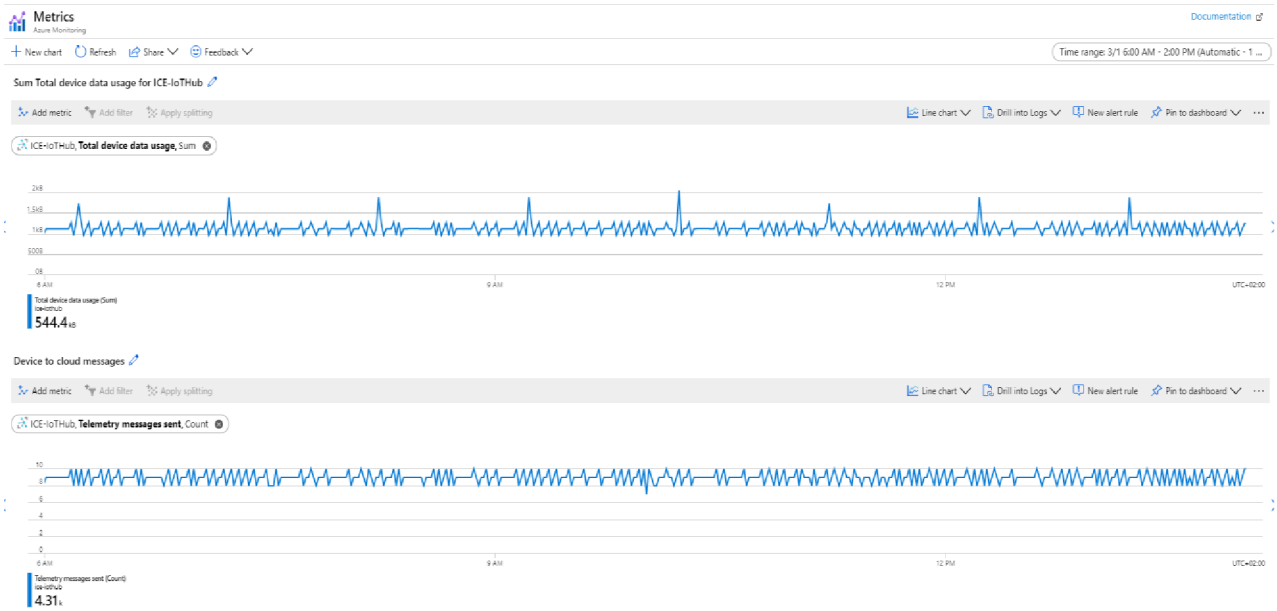
8.3 Rychlost přenosu dat do cloudu

Jak jsem zmiňoval v kapitole 5 IoT Hub se liší podle verze, kterou si platíte. Od toho se odvíjí i maximální počet zpráv za den, kde jedna zpráva se skládá z tzv. chunků, kde jeden má velikost 4 kB. Maximální velikost zprávy tak může činit 256 kB ve směru z Edge zařízení do cloudu nebo zpět. To však nejsou jediná omezení podle verze IoT Hubu. Dále se liší podle i tzv. „operačních škrticích klapek“ (Operation throttles). Tyto omezení jsou aplikované v minutovém intervalu komunikace. [16]

Operace	Ver. Free, B1, S1	B2, S2	B3, S3
Zpráva ze Edge do cloudu	Do 100 odeslaných op[18]/sek nebo 12 op/sek/jednotka[19].	120 odeslaných op/sek/jednotka	6000 odeslaných op/sek/jednotka
Z cloudu do zařízení	1,67 odeslaných op/sek/jednotka	1,67 odeslaných op/sek/jednotka	83,33 odeslaných op/sek/jednotka
Max. počet současně připojených streamujících zařízení.	50	50	50
Zařízení, které iniciují stream v jeden čas	5 nových streamu/sek	5 nových streamu/sek	5 nových streamu/sek
Max. transfer dat za den na jedno zařízení	300 MB	300 MB	300 MB

Tabulka 8.3-1 Výtah některých omezení IoT Hub podle verze předplatného. [8]

Z tabulky 8.3-1 vyplívá, že množství přenesených dat není omezeno ani tak množstvím bytů ale množstvím operací a celkouvým datovým tokem za den. Pokud by se podařilo vytvořit v lokálním Edge zařízení agregovanou zprávu o velikosti 256 KB (64 operací), tak je možné ji za sekundu přenést do cloudu pomocí verzí B2, S2 vyšších. To se nemusí zdát jako mnoho ale tato služba je vytvořena především pro řenost metrik ze zařízení ve formátu JSON.

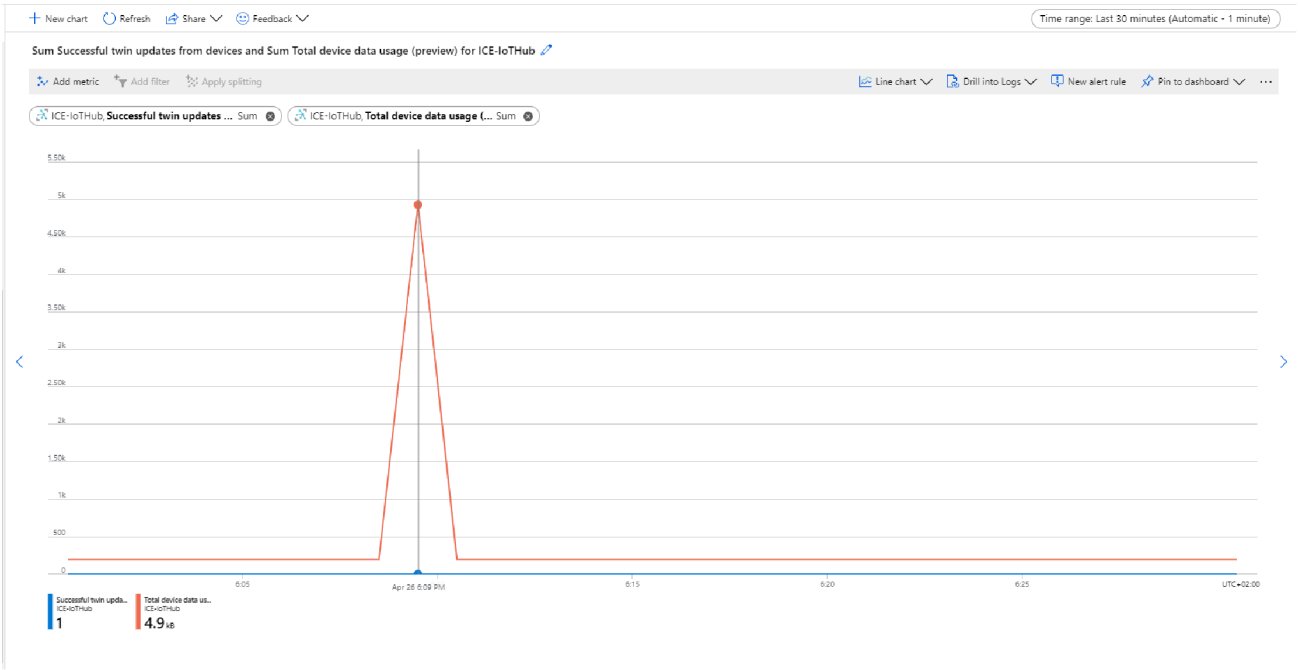


Obrázek 8.3-1 Počet zpráv do cloudu a jejích velikost za 8 h 1.3.2020

Na obrázku je vidět graf celkové velikosti zpráv za 8 h a jejích počet. Data jsou agregované po minutě. Jednotlivé zprávy za minutu měli velikost od 1 do 1,9 KB. Počet zpráv za jednu minutu byl od 8 do 10. Tato v množství a velikosti zpráv byla způsobená tím, že byl jeden IoT Hub sdílený pro dvě testovací

[18] Op – operace, taky jeden chunk v jednom směru
[19] Jednotka – jedna služba IoT Hub

nasazení ve dvou různých fabrikách. V jednom případě chodily z Edge zařízení pouze dvě hodnoty (rychlost frézky a její zatížení). V druhém případě to byli komplexnější data o počtu OK a NOK kusů celé linky dále členěno na NOK kusy z jednotlivých stanic na lince a informace, jestli linka jako celek je v provozu. Data byla odesílaná z PLC jednou za minutu nebo při změně stavu. Z toho důvodu mají grafy na obrázku výše pilovitý charakter.



Obrázek 8.3-2 Celkový počet dat zaslaných do zařízení z cloudu 30 minut s krokem 1 minuta 27.4.2020

Twin operace referuje k obnoven obrazu a proměnných u MAB kontejneru v zařízení IoT Edge. Tento modul se stará o vyzvednutí zpráv z MQTT broukeru a odeslaní do IoT Hubu. Mezi jeho proměnné patří například topic z kterého má vyzvedávat zprávy. K updatu dochází periodicky podle nastavení nebo na změnu.

9 Licence

Níže uvedený texty jsou pouze základní popisy licence. Nejedná se o licence samotné, které jsou mnohem obsáhlejší. Texty tak dává základní a nejdůležitější informace vyplívající z licencí.

9.1 Apache 2.0

Licence Apache (anglicky Apache License) je v informatice název svobodné softwarová licence, jejímž autorem je Apache Software Foundation (ASF). Licence požaduje po uživateli zachování autorství (copyright) a tzv. disclaimer, tedy zřeknutí se odpovědnosti. Jako ostatní licence tohoto druhu, i Apache umožňuje uživateli svobodné užívání softwaru k různým účelům; distribuci, upravování, následné redistribuci upravené verze softwaru apod. To vše je možné, aniž by došlo k porušení licenčních práv. [35]

Poslední vydanou verzí je verze s pořadových číslem 2.0, která vyšla v lednu 2004 a je možné ji stáhnout z webové stránky Apache. Verze je kompatibilní s GPL v3 a je možno ji propojit s kódem, který má jinou licenci. Dále je kompatibilní s Debian Free Software Guidlines (DFSG). Licence je potvrzena jako svobodný software (free software) a otevřený software (open source). Po uživateli není požadován copyleft[20]*.* [35]

9.2 MIT

Licence MIT je svobodná licence vzniknuvší na Massachusettském technologickém institutu (MIT). Software uvolněný pod touto licencí je možné použít jak v proprietárním software (s podmínkou, že text licence MIT musí být dodáván spolu s daným software), tak i s GPL licencovaným software (díky tomu, že GPL explicitně povoluje kombinaci s licencí MIT). [36]

9.3 BSD licence

BSD[21] *licence je licence pro svobodný software, mezi kterými je jednou z nejsvobodnějších. Umožňuje volné šíření licencovaného obsahu, přičemž vyžaduje pouze uvedení autora a informace o licenci, spolu s upozorněním na zřeknutí se odpovědnosti za dílo.* [37]

9.4 GNU licence

GNU General Public License, GNU GPL (česky „obecná veřejná licence GNU“) je licence pro svobodný software, původně napsaná Richardem Stallmanem pro projekt GNU. GPL je nejpopulárnějším a dobře známým příkladem silně copyleftové licence, která vyžaduje, aby byla odvozená díla dostupná pod toutéž licencí. V rámci této filozofie je řečeno, že poskytuje uživatelům

[20] Při vytvoření odvozeného díla z díla, jež je dostupné jen pod copyleft licencí, musí být toto odvozené dílo nabízeno pod stejnou (copyleft) licencí jako dílo původní.

[21] Zkratka BSD označuje „Berkeley Software Distribution“ – obchodní organizaci při University of California, Berkeley, která tuto licenci vyvinula a používala pro práce nad operačním systémem BSD. [37]

počítačového programu práva svobodného softwaru a používá copyleft k zajištění, aby byly tyto svobody ochráněny, i když je dílo změněno nebo k něčemu přidáno. Toto je rozdíl oproti permisivním licencím svobodného softwaru, jejímž typickým případem jsou BSD licence.

GNU Lesser General Public License (LGPL) je upravená, permisivnější verze GPL, původně zamýšlená pro některé knihovny. Existuje také GNU Free Documentation License, která byla původně určena pro dokumentaci k softwaru GNU, která ale byla později použita i jinde, například v projektu Wikipedie. [38]

9.5 EPL

Veřejná licence Eclipse (EPL) je bezplatná a open source softwarová licence, která se používá zejména pro Eclipse IDE a další projekty nadace Eclipse Foundation. Nahrazuje společnou veřejnou licenci (CPL) a ruší některé podmínky týkající se soudních sporů týkajících se patentů.

Veřejná licence Eclipse je navržena jako bezplatná softwarová licence pro podnikání a má slabší ustanovení o kopírování než licence, jako je GNU General Public License (GPL). [7] Příjemce programů s licencí EPL může používat, upravovat, kopírovat a distribuovat dílo a upravené verze, v některých případech je povinen vydat své vlastní změny.

EPL je uvedena jako bezplatná softwarová licence od Free Software Foundation (FSF) a schválena Open Source Initiative (OSI). [39]

9.6 Ubuntu

Ubuntu je distribuován jako svobodný software (free software nebo software libre). Uživateli je dovoleno použití takovéhoto softwaru pro jakékoliv účely včetně jeho změny a distribuce v pozměněné verze. Svobodný software v sobě zakomponováva licence MIT, GNU, Apache, BSD atd.

Veškerý sw zahrnutý v Ubuntu musí umožňovat:

- Distribuci, tzn. právo prodávat anebo dále šířit samotný software.
- Nesmí nárokovat placení za použití softwaru jakýmkoliv způsobem.
- Musí povolovat uplatnění těchto práv společně s poskytnutým softwarem.
- Ubuntu nedistribuuje sw, který by měl speciální nebo odlišné licenční podmínky než samotný operační systém. [40]

10Závěr

Vytvoření systému sběru dat postaveného na modulárních platformách jako je Docker nebo Azure cloud se osvědčilo. Přináší to možnost dynamicky měnit nebo dotvářet funkcionality podle přání zákazníka nebo reagovat na nové trendy a technologie na trhu. Příkladem může být nově vzniklá podpora OPC UA na IoT Hubu a směřování trhu s IoT, ale i PLC k tomuto komunikačnímu standardu právě pro odesílaní dat do cloudu. Používání dílčích řešení třetích stran má i své nevýhody, kdy se nová verze programu může zásadně lišit od předchozí nebo se vývoj zastaví a je nutné hledat alternativu. Volba cloudu byla provedena na začátku projektu na základě doporučení jiné společnosti. Nedokážu tak posoudit, jestli je Azure nejlepší volba na trhu, ale z mé zkušenosti jsem se službami, které nabízí, spokojený. Portál Azure poskytuje ke všem službám unifikovanou platformu bez rozdílu, jestli se jedná o aplikaci vytvořenou Microsoftem nebo aplikaci třetí strany. Některé služby se mohou zdát drahé, ale na druhou stranu je to na každém uživateli cloudu, jaké služby použije a za jakým účelem. V průběhu práce jsem se naučil, že vždy existuje několik alternativ, jak dosáhnout kýženého výsledku. Hledání informací a návodů na internetu mě zavedlo například k návodům pro spojení MQTT brokeru s AWS (Amazon Web Services) cloudem, a i když se jejich služby jmenují jinak, je na první pohled zřejmé, že jejich funkcionalita je velmi podobná.

Je nutné zmínit, že na systému sběru dat s využitím služeb a programů třetích stran nebylo třeba mnoho programování. Práce měla spíš charakter konfigurace a učení se propojovat existující řešení. Proto služby a programy uváděné v této práci nejsou jedinou možností k dosažení stejné funkcionality. Troufám si ale říct, že-řešení prezentované v této práci je efektivní z pohledu cena výkon s velkou konfigurovatelností.

MQTT protokol byl použit pro jeho jednoduchost, možnost implementace do PLC a fakt, že je postavený na TCP/IP protokolu. Umožňuje tak vytvořit bezpečné šifrované spojení pomocí TLS protokolu na lokální síti zákazníka. MQTT není vhodný pro přenos dat, u kterých je vyžadovaná nízká latence, ale naopak je postavený pro pomalejší komunikaci s jistotou doručení dat. Proto je vhodný pro sběr dat za účelem statistik výrobních linek a provozů. V průběhu práce bylo třeba vytvořit propojení MQTT brokeru s cloudem. Práce byla hodně ulehčena tím, že Microsoft zdarma nabízí návody a předpřipravené moduly pro Visual Studio. V podstatě tak stačilo založit projekt pro připojení ke cloudu a využít vhodnou knihovnu pro komunikaci s MQTT broker. Za tímto účelem byla použitá knihovna MQTTnet, která má i vlastní wiki s příkladem kódu pro MQTT klienta. Celý program tak je hlavně spojením těchto dvou řešení.

Program do PLC jsem naopak vůbec neřešil, jelikož testovaní bylo provedeno rovnou implementací do provozu dvou projektů společnosti, kde pracuji naši programátoři, kteří knihovnu poskytovanou Siemensem implementovali do svého kódu. Rovněž vyřešili zásobník zpráv v PLC, pokud nedojde k potvrzení doručení do MQTT brokeru.

Po vyzkoušení různých databázových systému jsem se nakonec rozhodl pro relační typ databáze vzhledem k datům, která jsou sbíraná. Všechny metriky jsou předem známé a neobsahují dlouhé textové zprávy. Všechna data jsou striktně časové řady. Azure nabízí všechny nejpoužívanější SQL databáze, ale pro aktuálně nízkou vytíženost a nespočet výhod byla vybrána Azure SQL jako služba. Azure SQL je založená na MSSQL a do budoucna je možnost pronajmout server s MSSQL a snadno přejít na tuto relační databází.

Od počátku se rozhodovalo mezi Kibanou a Grafanou jako vizualizačním nástrojem. Obě platformy jsou vyvíjené jako open source projekty a je možné je volně upravovat a dál distribuovat. Po straně vizuální byla Kibana dle mého subjektivního názoru hezčí. Rozhodnutí pro Grafanu tak bylo čistě programátorské, kdy Grafana umožňuje napojení na větší množství databází a je určena pro zobrazování metrik. Naproti tomu Kibana funguje v rámci ELK stack a je určena striktně na propojení s Elastick Search. Zároveň je vyvíjená spíš pro vizualizaci logů. S novou verzí Grafany 7.0. se však tyto rozdíly postupně smazávají a jenom mě to utvrzuje, že použití Grafany bylo správné rozhodnutí.

Zabezpečený přístupu k vizualizaci je zajištěn pomocí https protokolu a zároveň všechna data proudící z Azure SQL jsou automaticky šifrované TLS protokolem. Data z lokálního serveru jsou rovněž automaticky šifrované pomocí TLS. Bezpečnost lze zvýšit případnou blokací neznámých IP adres. Za tímto účelem se osvědčilo použití reverzní proxy před samotným web serverem Grafany.

Původně měla být práce o menším počtu služeb s hlubším popisem. Vzhledem k vývoji na trhu z posledního měsíce a půl ale bylo rozhodnuto, aby se v pozměněné verzi začal sběr dat nabízet jako produkt. Z toho důvodu nemohu uvádět podrobnosti u některých dílčích řešení použitých v této práci. Ze stejného důvodu práce nemá přílohy. Než pozměňovat kusy kódu nebo vkládat jen části a anonymizovat screenshoty, je podle mě lepší žádné přílohy nesdílet.

Do budoucna je na projektu ještě hodně práce a vývoje. Nabízí se široké množství směrů, kam se ubírat, a věci, co se dají přidat. Určitě se budou ještě přeskupovat některé služby a na serveru kde běží Grafana bude využíván Docker s vícero instancemi Grafany pro více domén za účelem ušetření nákladů. Budou se vyvíjet další pluginy do Grafany a prozkoumávat možnosti využití umělé inteligence k prediktivní údržbě atd.

11 Reference

[1] MICROSYS, spol. s r.o., „Komunikace přes rozhraní OPC UA,“ MICROSYS, spol. s r.o.. [Online]. [Přístup získán 23 5 2020].

[2] A. Vojáček, „IoT MQTT prakticky v automatizaci - 1.díl - úvod,“ automatizace.hw.cz, 21 1 2017. [Online]. Available: https://automatizace.hw.cz/iot-mqtt-prakticky-v-automatizaci-1dil-uvod.html. [Přístup získán 22 5 2020].

[3] Siemens AG 2009-2020, "FB "LMQTT_Client" for SIMATIC S7-CPU," 12 5 2019. [Online]. Available: https://support.industry.siemens.com/cs/document/109748872/fb-lmqtt_client-for-simatic-s7-cpu?dti=0&lc=en-WW.

[4] Siemens, "MQTT client for SIMATIC S7-1500 nad S7-1200," 12 2019. [Online]. Available: https://support.industry.siemens.com/cs/document/109748872/fb-lmqtt_client-for-simatic-s7-cpu?dti=0&lc=en-WW. [Accessed 20 4 2020].

[5] v. S. v.-s. m. kgremban, „docs.microsoft.com,“ microsoft, 21 2 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/how-to-install-iot-edge-linux. [Přístup získán 17 5 2020].

[6] Docker Inc, "Docker," 2019. [Online]. Available: https://www.docker.com/products/container-runtime. [Accessed 2019].

[7] Docker Inc, „ Docker Inc,“ Docker Inc, 2019. [Online]. Available: https://www.docker.com/resources/what-container. [Přístup získán 2020].

[8] olprod, „Use Visual Studio 2019 to develop and debug modules for Azure IoT Edge,“ microsoft, 27 3 2020. [Online]. Available: https://docs.microsoft.com/cs-cz/azure/iot-edge/how-to-visual-studio-develop-module. [Přístup získán 30 3 2020].

[9] olpro, „microsoft azure, iot edge,“ 28 10 2019. [Online]. Available: https://docs.microsoft.com/cs-cz/azure/iot-edge/about-iot-edge.

[10] olprod, „MicrosoftDocs/azure-docs.cs-cz,“ 27 Listopad 2019. [Online]. Available: https://github.com/MicrosoftDocs/azure-docs.cs-cz/blob/master/articles/iot-edge/iot-edge-runtime.md.

[11] „Co je SaaS?,“ microsoft, [Online]. Available: https://azure.microsoft.com/cs-cz/overview/what-is-saas/.

[12] „Co je PaaS?,“ microsoft, [Online]. Available: https://azure.microsoft.com/cs-cz/overview/what-is-paas/.

[13] „Co je IaaS?,“ microsoft, [Online]. Available: https://azure.microsoft.com/cs-cz/overview/what-is-iaas/.

[14] „Architektura bez serveru,“ microsoft, [Online]. Available: https://azure.microsoft.com/cs-cz/overview/serverless-computing/.

[15] olprod, „MicrosoftDocs/azure-docs.cs-cz,“ 13 Srpen 2019. [Online]. Available: https://github.com/MicrosoftDocs/azure-docs.cs-cz/blob/master/articles/iot-hub/about-iot-hub.md.

[16] R. S. R. S. K. G. Dominik Betts, "MicrosoftDocs / azure docs," 10 3 2020. [Online]. Available: https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/iot-hub/iot-hub-devguide-quotas-throttling.md. [Accessed 26 4 2020].

[17] olprod, „Azure Stream Analytics,“ 21 Červen 2019. [Online]. Available: https://docs.microsoft.com/cs-cz/azure/stream-analytics/stream-analytics-introduction.

[18] c. t. s. c. d. k. c. ggailey777, „azure-docs,“ github, 16 1 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview. [Přístup získán 18 5 2020].

[19] Paul Batum, „azure,“ microsoft, 2017 9 19. [Online]. Available: https://azure.microsoft.com/cs-cz/blog/processing-100-000-events-per-second-on-azure-functions/. [Přístup získán 2019 11 18].

[20] s. l. d. +. cynthn, „Sizes for Windows virtual machines in Azure,“ 3 2 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes. [Přístup získán 20 5 2020].

[21] internetsociety, „TLS Basics,“ internetsociety, [Online]. Available: https://www.internetsociety.org/deploy360/tls/basics/. [Přístup získán 27 05 2020].

[22] [Online]. Available: https://www.clariontech.com/blog/nginx-php-fpm-setup-for-high-traffic-web-sites.

[23] nginx, „NGINX,“ nginx, [Online]. Available: https://www.nginx.com/resources/glossary/nginx/.

[24] Ayusharma0698, „GeeksforGeeks, SQL NoSQL,“ [Online]. Available: https://www.geeksforgeeks.org/difference-between-sql-and-nosql/.

[25] J. H, „Azure SQL vs MySQL: What are the Differences?,“ DreamFactory, 7 4 2020. [Online]. Available: https://blog.dreamfactory.com/azure-sql-vs-mysql-what-are-the-differences/. [Přístup získán 20 5 2020].

[26] A. Mallon, „What the heck is a DTU?,“ SQL Performance.com, 30 03 2017. [Online]. Available: https://sqlperformance.com/2017/03/azure/what-the-heck-is-a-dtu. [Přístup získán 28 2 2020].

[27] D. Berman, „InfluxDB vs. Elasticsearch for Time Series Analysis,“ DZone, 30 10 2017. [Online]. Available: https://dzone.com/articles/influxdb-vs-elasticsearch-for-time-series-analysis. [Přístup získán 1 5 2020].

[28] i. olprod, „Welcome to Azure Cosmos DB,“ microsoft, 23 10 2019. [Online]. Available: https://docs.microsoft.com/cs-cz/azure/cosmos-db/introduction. [Přístup získán 10 1 2020].

[29] olprod, „Introduction to Azure Blob storage,“ microsoft, 18 3 2020. [Online]. Available: https://docs.microsoft.com/cs-cz/azure/storage/blobs/storage-blobs-introduction. [Přístup získán 30 4 2020].

[30] azure, „Ceny funkcí Azure,“ microsoft, [Online]. Available: https://azure.microsoft.com/cs-cz/pricing/details/functions/.

[31] A. Yigal, „logz.io,“ 6 Června 2018. [Online]. Available: https://logz.io/blog/grafana-vs-kibana/.

[32] Kesslpet, „TypeScript,“ wikipedie, 14 12 2014. [Online]. Available: https://cs.wikipedia.org/wiki/TypeScript. [Přístup získán 28 5 2020].

[33] H. Bittner, „Lekce 2 - Úvod do CSS preprocesoru Sass,“ ITnetwork.cz, [Online]. Available: https://www.itnetwork.cz/html-css/webove-portfolio/tutorial-moderni-webove-portfolio-sass.

[34] json.org, „Úvod do JSON,“ json.org, [Online]. Available: https://www.json.org/json-cz.html. [Přístup získán 30 5 2022].

[35] Tchoř, „Licence Apache,“ wikipedia, 9 4 2020. [Online]. Available: https://cs.wikipedia.org/wiki/Licence_Apache. [Přístup získán 17 5 2020].

[36] Harold, „Licence MIT,“ Wikipedia, 26 6 2016. [Online]. Available: https://cs.wikipedia.org/wiki/Licence_MIT. [Přístup získán 17 5 2020].

[37] TXiKiBoT, „BSD licence,“ Wikipedia, 13 9 2009. [Online]. Available: https://cs.wikipedia.org/wiki/BSD_licence. [Přístup získán 17 5 2020].

[38] Lalina, „GNU General Public License,“ wikipedia, 2020. [Online]. Available: https://cs.wikipedia.org/wiki/GNU_General_Public_License.

[39] C. A. Russell, "Eclipse Public License," wikipedia, 29 1 2020. [Online]. Available: https://en.wikipedia.org/wiki/Eclipse_Public_License. [Accessed 24 5 2020].

[40] „ubuntu.com/licensing,“ ubuntu, [Online]. Available: https://ubuntu.com/licensing.

[41] „azure.microsoft.com,“ [Online]. Available: https://azure.microsoft.com/cs-cz/overview/.

[42] R. Gardette, „NGINX Reverse Proxy,“ github, 18 6 2018. [Online]. Available: https://dev.to/remyg/nginx-reverse-proxy-54d7. [Přístup získán 22 5 2020].

[43] [Online]. Available: https://www.w3schools.com/js/js_json_arrays.asp.