Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Datové protokoly v IoT

Pavel Sixta

Bakalářská práce

2016

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Pavel Sixta**
Osobní číslo:	**I13221**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Datové protokoly v IoT**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem bakalářské práce je podrobně popsat datové protokoly MQTT, CoAP, AMQP, Websocket, Node. Popis bude obsahovat formát zpráv, průběh komunikace a příklady použití v praxi. Praktická část se zaměří na vytvoření úlohy, která bude demonstrovat propojení s platformou Azure IoT Hub. Jednotlivé kroky při realizaci praktické části budou zdokumentovány.

Rozsah grafických prací:

Rozsah pracovní zprávy: **40**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

OASIS: MQTT Version 3.1.1 [online]. 2014 [cit. 2016-10-20]. Dostupné z: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

Core IEC Standards [online]. IEC, 2016 [cit. 2016-10-20]. Dostupné z: http://www.iec.ch/smartgrid/standards/

Microsoft Azure. Microsoft [online]. U.S.: Microsoft Corporation, 2016 [cit. 2016-10-24]. Dostupné z: https://azure.microsoft.com/en-us/documentation/services/iot-hub/

Vedoucí bakalářské práce: **Ing. Soňa Neradová, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2016**

Termín odevzdání bakalářské práce: **12. května 2017**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 05. 05. 2017

Pavel Sixta

PODĚKOVÁNÍ

Tímto bych rád poděkoval vedoucí mé bakalářské práce Ing. Soně Neradové Ph.D., za cenné a nápomocné rady, které mi během psaní této práce poskytla. Dále bych chtěl poděkovat svým rodičům, kteří mě po dobu mého studia podporovali.

ANOTACE

Tato práce se zabývá popisem datových protokolů v IoT, konkrétně MQTT, CoAP, AMQP, a WebSocket. Poté je popsáno prostředí Node. V praktické části je popsáno připojení k platformě Azure IoT Hub.

KLÍČOVÁ SLOVA

IoT, MQTT, CoAP, WebSocket, Node, Azure

TITLE

Data protocols in IoT

ANNOTATION

This work deals with description of data protocols in IoT, specifically MQTT, CoAP, AMQP, and WebSocket. Node environment is described afterwards. Connection to platform Azure IoT Hub is described in practical part.

KEYWORDS

IoT, MQTT, CoAP, WebSocket, Node, Azure

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

ACK	Acknowledgement
AMQP	Advanced Message Queuing Protocol
CoAP	Constrained Application Protocol
CON	Confirmable
DDS	Document delivery service
DOFF	Data offset
DPWS	Devices Profile for Web Services
DUP	Duplicate delivery of a PUBLISH Control Packet
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
LSB	Least Significant Bit
M2M	Machine-to-machine
MQTT	MQ Telemetry Transport
MSB	Most Significant Bit
QoS	Quality of Service
RAM	Random-Access Memory
RFC	Request for Comments
ROM	Read-Only Memory
RST	Reset
SASL	Simple Authentication and Security Layer
SCTP	Stream Control Transmission Protocol
SNMP	Simple Network Management Protocol

TCP	Transmission Control Protocol
TKL	Token Length
TLS	Transport Layer Security
TLV	Type-Length-Value
TTL	Time to live
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
XMPP	Extensible Messaging and Presence Protocol

0 ÚVOD

Možnost propojení veškerých hardwarových a softwarových komponentů zařízení do Internetu je cílem Internetu věcí. V rámci informačních technologií musí platforma umět transformovat data, která shromáždí, a odeslat je do cloudu prostřednictvím standardů IoT Datových protokolů používaných v Internetu věcí je značná řada (CoAP, DDS, DPWS, MQTT, SNMP, UPnP, XMPP, ZeroMQ). V první kapitole práce je stručný přehled o Internetu věcí a o tom co vlastně Internet věcí je. Dále je v práci pokračováno popisem datových protokolů MQTT, AMQP, CoAP, WebSocket a prostředí Node. Pro každý protokol je v práci popsán způsob jejich komunikace, formát rámců případně paketů, jenž si posílají, a praktické implementace těchto protokolů.

V poslední části práce je podrobně popsán způsob jak se pomocí Node úspěšně připojit na Microsoft Azure IoT Hub. Následně je popsán postup tvorby několika konzolových aplikací Node v jazyce JavaScript, pomocí kterých vytvoříme zařízení, ze kterého na IoT Hub odešleme několik testovacích zpráv.

1 INTERNET VĚCÍ

Pojem Internet věcí je označení pro propojení fyzických zařízení, mechanických a digitálních strojů, budov, zvířat, lidí a jiných položek opatřených unikátním identifikátorem, elektronikou, softwarem, senzory a schopností přijímat a předávat data po síti.

Internet věcí umožňuje vzdáleně vnímat, kontrolovat a ovládat zařízení přes již existující síťové zázemí, což dovoluje integraci více fyzických prvků do počítačových systémů, to má za následek zvýšenou efektivitu a kvalitu nejen práce, ale i každodenního života.

„Věcí“ v Internetu věcí se může myslet například farmářské zvíře opatřené biočipem, chytrý telefon, zámek, který lze odemknout či zamknout pomocí telefonu, zrcadlo, které z internetu získá data o předpovědi počasí a následně je předloží uživateli. Se senzory, ať již vestavěnými či přidanými, se může jednat o automobil, který upozorní řidiče, když bude tlak v pneumatikách příliš nízký, lednice, jenž snímá druh, množství případně kvalitu potravy, případně celé chytré kuchyně a domy. „Věcí“ může být i člověk s implantátem na srdci sloužící ke kontrole srdečního tepu a následným uložením a odesláním informací o něm doktorovi. K Internetu věcí se skutečně může připojit téměř cokoliv a kdokoliv. Dnes je jediným omezením lidská představivost a vynalézavost.

1.1 Historie

Koncept Internetu věcí nebyl pojmenován do roku 1999, přesto byl ve vývoji dlouho předtím. Na začátku 80. let byl na Carnegie-Mellonově univerzitě v Pittsburghu v Pensylvánii automat na kolu. Programátoři se k automatu mohli připojit přes internet, zkontrolovat jeho stav a zjistit zda by na ně čekal vychlazený nápoj, pokud by se rozhodli pro cestu k automatu. Oficiální první zařízení Internetu věcí stvořil John Romkey v roce 1990. Byl jím toaster, který se dokázal zapnout a vypnout přes internet. Od této doby uplynula již řada let a Internet věcí se stále rozšiřuje a povědomí o něm stále roste.

2 MQTT

V jednotlivých IoT systémech (například chytré energetické síti či domácnostech) by snímače neměly rozhodovat o úkonech, které má kdo vykonat. Snímače by měly pouze snímat a tyto zprávy odesílat dále. Pokud bude mít uživatel zájem o tyto zprávy, přihlásí se k jejich odebírání, a následně na ně bude reagovat. MQTT je protokol sloužící pro předávání těchto zpráv mezi klienty. Pracuje na výše zmíněném principu uveřejnit-odebírat. Zprávy se odesílají do centrálního uložiště, tzv. „*MQTT broker*“, které dále funguje i jako distributor těchto zpráv. Zprávy jsou tříděny podle témat tzv. „*topic*“. Každé zařízení má možnost odebírat jakékoli téma nebo libovolné téma uveřejňovat. Jednotlivá zařízení může odebírat libovolné množství témat, totéž platí i pro zveřejňování.

MQTT přenáší data pomocí TCP, případně může využívat i jiné síťové protokoly, které zajistí spolehlivé doručení bez ztráty dat, doručení dat ve správném pořadí a obousměrný přenos. Přenos zpráv v protokolu MQTT je nezávislý na obsahu. Dále poskytuje tři úrovně QoS.

- „Nejvýše jednou“, kdy se zprávy mohou ztratit a nemusí být doručeny. Tato úroveň je kupříkladu používána, když nezáleží na ztrátě snímaných dat, protože budou brzy nahrazena daty novými.
- „Alespoň jednou“, kdy zprávy musí být doručeny, ale mohou se objevit duplikáty.
- „Přesně jednou“, kdy se zajistí, že zprávy budou doručeny přesně jednou. Tato úroveň, může být použita například v systému plateb, kde by ztráta zprávy nebo její duplikát mohly způsobit nesprávné vyúčtování.

MQTT přidává jen naprosté minimum servisních dat potřebných pro doručení, čímž zajišťuje minimální velikost zpráv. To napomáhá redukovat provoz na síti. Také obsahuje mechanismy pro upozornění, když se zařízení odpojení nestandardním způsobem.

2.1 Formát řídících paketů

Protokol MQTT komunikuje pomocí řídících paketů, které jsou popsány v následující části textu. Tyto řídící pakety mohou obsahovat až tři části. Které jsou vyobrazeny v následující tabulce (Tabulka 1).

Tabulka 1 – Struktura řídícího paketu MQTT

Pevná hlavička, přítomna ve všech řídících paketech MQTT
Proměnná hlavička, přítomna v některých řídících paketech MQTT
Zatížení, přítomen v některých řídících paketech MQTT

Zdroj: (OASIS, 2014)

2.1.1 Pevná hlavička

Na následující tabulce (Tabulka 2) je zobrazen formát pevné hlavičky řídícího paketu, kterou musí obsahovat každý řídící paket MQTT.

Tabulka 2 – Formát pevné hlavičky MQTT

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
byte 1	Typ řídícího paketu MQTT				Příznaky specifické pro každý typ řídícího paketu MQTT			
byte 2…	Zbývající délka							

Zdroj: (OASIS, 2014)

První 4 bity označují typ řídícího paketu. Tyto jsou vypsány v tabulce (Tabulka 3).

Tabulka 3 – Typy řídících paketů MQTT

Název	**Hodnota**	**Směr komunikace**	**Popis**
Rezervováno	0	Zakázáno	Rezervováno
CONNECT	1	Klient-Server	Klient vysílá požadavek pro připojení se k serveru
CONNACK	2	Server-Klient	Potvrzení připojení
PUBLISH	3	Klient-Server nebo Server-Klient	Zveřejni zprávu
PUBACK	4	Klient-Server nebo Server-Klient	Zveřejni potvrzení

Název	Hodnota	Směr komunikace	Popis
PUBREC	5	Klient-Server nebo Server-Klient	Zveřejnění přijato (zaručené doručení část 1)
PUBREL	6	Klient-Server nebo Server-Klient	Zveřejnění uvolněno (zaručené doručení část 2)
PUBCOMP	7	Klient-Server nebo Server-Klient	Zveřejnění dokončeno (zaručené doručení část 3)
SUBSCRIBE	8	Klient-Server	Žádost klienta o odběr
SUBACK	9	Server-Klient	Potvrzení odběru
UNSUBSCRIBE	10	Klient-Server	Žádost o zrušení odběru
UNSUBACK	11	Server-Klient	Potvrzení o zrušení odběru
PINGREQ	12	Klient-Server	Žádost PING
PINGRESP	13	Server-Klient	Odpověď PING
DISCONNECT	14	Klient-Server	Klient se odpojuje
Rezervováno	15	Zakázáno	Rezervováno

Zdroj: (OASIS, 2014)

Zbývající bity prvního bytu, číslované 3-0, označují příznaky jedinečné pro každý typ kontrolního paketu (Tabulka 4). Pokud je příznak označen „Rezervováno“, je rezervovaný pro budoucí použití a musí být nastaven na hodnotu použitou v tabulce. Pokud budou přijaty nesprávné příznaky, příjemce musí přerušit síťové spojení.

Tabulka 4 – Bitové příznaky MQTT

Řídící paket	Příznaky pevné hlavičky	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Rezervováno	0	0	0	0
CONNACK	Rezervováno	0	0	0	0
PUBLISH		DUP[1]	QoS[2]	QoS	RETAIN[3]
PUBACK	Rezervováno	0	0	0	0
PUBREC	Rezervováno	0	0	0	0
PUBREL	Rezervováno	0	0	1	0
PUBCOMP	Rezervováno	0	0	0	0
SUBSCRIBE	Rezervováno	0	0	1	0
SUBACK	Rezervováno	0	0	0	0
UNSUBSCRIBE	Rezervováno	0	0	1	0
UNSUBACK	Rezervováno	0	0	0	0
PINGREQ	Rezervováno	0	0	0	0
PINGRESP	Rezervováno	0	0	0	0
DISCONNECT	Rezervováno	0	0	0	0

Zdroj: (OASIS, 2014)

DUP – Pokud hodnota příznaku nastavena jako 0, jedná se o první instanci pokusu doručení zprávy, ať již klientem či serverem. Pokud je hodnota příznaku 1, že se jedná o další pokus doručení paketu, který byl již odeslán alespoň jednou.

[1] Duplicitní doručení řídícího paketu PUBLISH
[2] QoS paketu PUBLISH
[3] Příznak RETAIN paketu PUBLISH

QoS – Tento příznak určuje úroveň jistoty doručení. Tyto jsou k nalezení v tabulce níže (Tabulka 5).

RETAIN – Příznak RETAIN určuje, zda si server, či klient, příchozí zprávu uloží. V případě, že je hodnota příznaku v paket PUBLISH poslaného klientem serveru 1, server si tuto musí, společně s QoS této zprávy, uložit, aby jí mohl později odeslat klientům, kteří odebírají dané téma. V případě vzniku nového odběru, musí server odeslat poslední takto uloženou zprávu na dané téma. Pokud je hodnota příznaku RETAIN 1 a zároveň je hodnota příznaku QoS 0, musí server zahodit všechny předešle uložené zprávy pro dané téma. Pokud server posílá paket PUBLISH klientu, v případě, že se jedná o nový odběr, musí být příznak RETAIN nastaven na hodnotu 1. Když se o nový odběr nejedná, nastaví příznak na hodnotu 0.

Tabulka 5 – Úrovně QoS MQTT

QoS hodnota	Bit 2	bit 1	Popis
0	0	0	Doručení nejvýše jednou
1	0	1	Doručení alespoň jednou
2	1	0	Doručení přesně jednou
-	1	1	Rezervováno – nesmí se použít

Zdroj: (OASIS, 2014)

V pevné hlavičce v bytu 2 začíná označení „zbývající délky“. Tato nám označuje počet bytů přítomných ve zbytku aktuálního paketu, včetně informací uložených v proměnné hlavičce a zatížení. Nezapočítávají se do ní byty z ní samé. Využívá kódovací schéma, které používá 1 byte k uložení hodnot až do 127. 7 bitů je použito k zakódování informací a 1 bit je použit k označení, zda následují další byty. Každý byte zakóduje 128 hodnot a pokračovací bit, přičemž nejvyšší počet bytů v části zbývající délka jsou 4. To nám umožňuje uložit hodnotu až 268 435 455 (0xFF, 0xFF, 0xFF, 0x7F).

2.1.2 Proměnná hlavička

V některých řídících paketech se vyskytuje proměnná hlavička, jejíž obsah závisí na typu řídícího paketu. Nachází se mezi pevnou hlavičkou a zatížením (Tabulka 1) zpravidla obsahuje 2 byty označující MSB a LSB identifikátory daného řídícího paketu (Tabulka 6) neboli také identifikátor paketu. Obsahují ho pakety PUBLISH (kde je QoS > 0), PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK.

Tabulka 6 – Identifikátor paketu MQTT

Bit	7	6	5	4	3	2	1	0
byte 1	MSB identifikátor paketu							
byte 2	LSB identifikátor paketu							

Zdroj: (OASIS, 2014)

Řídící pakety protokolu MQTT mohou na svém konci obsahovat zatížení (Tabulka 7). Obsah zatížení je různý podle typu řídícího paketu. Zatížení v paketu typu CONNECT, může podle příznaků uchovaných v proměnné hlavičce uchovávat identifikátor klienta, jméno uživatele, heslo, téma, případně Aplikační zprávu. Zatížení v paketu typu PUBLISH obsahuje pouze pole Aplikační zprávu.

Tabulka 7 – Zatížení řídících paketů

Řídící paket	Zatížení
CONNECT	Vyžadováno
CONNACK	Žádné
PUBLISH	Volitelné
PUBACK	Žádné
PUBREC	Žádné
PUBREL	Žádné
PUBCOMP	Žádné
SUBSCRIBE	Vyžadováno
SUBACK	Vyžadováno
UNSUBSCRIBE	Vyžadováno
UNSUBACK	Žádné
PINGREQ	Žádné
PINGRESP	Žádné
DISCONNECT	Žádné

Zdroj: (OASIS, 2014)

2.2 Praktické příklady použití protokolu MQTT

S protokolem MQTT se jistě setkalo mnoho uživatelů, aniž by o tom měli tušení. Důvodem je jeho využití ve Facebook Messenger, který využívá části tohoto protokolu pro online komunikaci. Protokol MQTT najde využití i u cloudových služeb jako Azure IoT, AWS IoT, Openstack, Amazon IoT a další. Společnost EVRYTHING ho využívá pro spojení milionů

„chytrých“ zařízení. Také ho lze nalézt v různých systémech sloužící pro automatizaci domácnosti tzv. chytré domácnosti.

3 COAP

CoAP je specializovaný internetový protokol určený pro převod a výměnu dokumentů. Tuto vlastnost sdílí s HTTP. Na rozdíl od HTTP je však přizpůsoben potřebám omezených zařízení a sítí. Omezených v tomto kontextu znamená ztrátových, s omezených výpočetní kapacitou či úložištěm, případně energií. Uzly v takovýchto sítích mají často 8bit mikrořadiče, které mají k dispozici pouze malé množství ROM a RAM, zatímco omezené sítě mývají často vysokou chybovost přijímaných paketů a typickou propustnost okolo 10 kbit/s. Proto byl tento protokol zamýšlen a byl navrhnut pro M2M aplikace, například automatizaci budov a chytré energetické sítě.

CoAP je protokolem pracující na aplikační vrstvě. Využívá model žádost/odpověď pro interakci koncových aplikačních zařízení. Také podporuje zabudované objevování služeb a zdrojů a jsou v něm zabudovány webové koncepty jako URI a typy internetového média. Byl navržen tak, aby se dal jednoduše přeložit do HTTP pro jednodušší integraci s internetem. Zároveň ale také splňuje specializované požadavky jako je velmi malá režie, jednoduchost a podpora multicast. Všechny tyto vlastnosti jsou velice důležité pro IoT a M2M zařízení, které bývají vybaveny velice omezenými paměťovými a energetickými prostředky oproti tradičním internetovým zařízením.

CoAP nepracuje na TCP nýbrž na UDP, kde spolu zařízení komunikují pomocí datagramů bez navazování spojení. Podporuje unicast i multicast a asynchronní výměnu zpráv. Využívá metod GET, PUT, POST a DELETE v podobném provedení jako HTTP.

3.1 Model zasílání zpráv

Model zasílání zpráv v CoAP je založen na výměně zpráv mezi dvěma koncovými body přes UDP. V CoAP se používá binární hlavička o fixní délce 4 byty. Tato hlavička může být následována binárními možnostmi a zátěží. Takovýto formát je jednotný jak pro žádosti, tak pro odpovědi na tyto žádosti. Každá zpráva obsahuje ID dané právy, pro účely detekce duplikátů a případné zajištění spolehlivosti. Té je dosaženo označením zprávy jako „Potvrditelné“ (CON). Taková zpráva je opakovaně odesílána s přednastavenými přestávkami dokud příjemce nepotvrdí její doručení pomocí „Potvrzovací“ ACK zprávy se stejným ID zprávy jako měla odeslaná zpráva. V případě, že příjemce danou zprávu není schopen zpracovat (tzn., že nedokáže ani odeslat příslušnou chybovou zprávu), odpoví Resetovou zprávou (RST).

3.2 Formát zpráv

CoAP zprávy jsou v jednoduchém binárním formátu (Tabulka 8). Tyto zprávy mají před sebou hlavičku s předem určenou velikostí 4 bytů. Za ní následuje hodnota Token o proměnné délce, 0 až 8 bytů. Poté následuje sekvence nul nebo více CoAP Nastavení v TLV formátu a nakonec i nepovinné zatížení.

Tabulka 8 – Formát zprávy CoAP

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Ver		T		TKL				Kód												ID zprávy											
Token (pokud je, TKL byty)…																															
Nastavení/Možnosti (pokud jsou)…																															
1	1	1	1	1	1	1	1	Zátěž (pokud je)…																							

Zdroj: (IETF, 2014)

Verze (Ver): Bezznaménkový celočíselný datový typ o 2 bitech. Ukazuje číselné označení verze CoAP.

Typ (T): Bezznaménkový celočíselný datový typ o 2 bitech. Ukazuje, zda se jedná o zprávu typu Confirmable „Potvrditelné“ (0), Non-Confirmable „Nepotvrditelné“ (1), Acknowledgement „Potvrzovací“ (2), nebo Reset „Resetovou“ (3).

Délka Token (TKL): Bezznaménkový celočíselný datový typ o 4 bitech. Ukazuje délku hodnoty Token o proměnné délce. Ta nabývá 0 až 8 bytů. Hodnoty 9 až 15 jsou rezervovány a nesmí být odeslány. Pokud bude odeslána zpráva s TKL 9 až 15, musí být takováto zpráva zpracována jako chyba formátu.

Kód: Bezznaménkový celočíselný datový typ o 8 bitech, které jsou rozděleny do *„třídy“* o velikosti 3 bitů (MSB) a *„detailu“* o velikosti zbývajících 5 bitů (LSB). Těchto 8 bitů, je dokumentováno ve formátu *„c.dd“* kde *„c“* je cifra mezi 0 až 7 pro 3 bitové pole třídy a *„dd“* jsou dvě cifry od 00 do 31 pro zbývající 5 bitové pole detailu. Třída označuje, zda se jedná o žádost (0), úspěšnou odpověď (2), chybu na straně klienta (4), nebo chybu na straně serveru (5). Všechny ostatní hodnoty ve třídě jsou rezervovány a nemohou být použity. Výjimkou tohoto pravidla je kód 0.00, který označuje prázdnou zprávu.

ID zprávy: Bezznaménkový celočíselný datový typ o 16 bitech. Používá se k přiřazení ACK a Reset zpráv ke zprávám typu CON a NCON, případně k odhalování duplicit.

Token: Po samotné hlavičce následuje Token o velikosti určené TKL. Slouží k určení vztahu mezi žádostí a odpovědí. Slouží jako lokální identifikátor, mohl by se nazývat „ID žádosti“. Každá zpráva má Token, i v případě, že tento má velikost 0.

Nastavení: Za polem Token následuje buď 0, nebo Nastavení. Tyto specifikují předem definované nastavení CoAP, délku hodnoty Nastavení a hodnotu Nastavení samotného (více v kapitole 3.3.1).

Zátěž: Zatížení je volitelné a nemusí být vůbec přítomno. Pokud přítomno je, musí mít prefix Označením zátěže o velikosti 1 byte (0xFF), který ukazuje konec Nastavení a začátek Zatížení. Velikost samotného zatížení je následně vypočítána z velikosti samotné zprávy. Zátěží může být reprezentace výsledku provedené akce či použitých zdrojů případně diagnostická zpráva pokud nastala chyba.

3.2.1 Formát Nastavení/Možnosti

CoAP má spoustu předem definovaných Nastavení, které mohou být využity v instanci Nastavení. Každé pole Nastavení obsahuje Číslo Nastavení označující definovanou Možnost, délku hodnoty Nastavení a samotnou hodnotu (Tabulka 9). Pole ovšem neobsahuje Číslo Nastavení přímo, toto musí být dopočítáno. Je nutné, aby jednotlivé instance přišli ve správném pořadí. Každé Číslo Natavení je vypočítáno jako součet Delty nové instance a Čísla Nastavení instance předešlé. Pro první instanci ve zprávě se předpokládá předešlé Číslo Nastavení nula. Ve zprávě může být uloženo více instancí stejné Možnosti využitím Delty nula.

Tabulka 9 – Formát Nastavení/Možností CoAP

0	1	2	3	4	5	6	7	
Delta Nastavení				Délka Nastavení				1 byte
Delta Nastavení (prodloužené)								0-2 byty
Délka Nastavení (prodloužené)								0-2 byty
Hodnota Nastavení								0 a více bytů

Zdroj: (IETF, 2014)

Hodnota Nastavení může být zapsána několika různými formáty:

- Prázdná sekvence bytů.
- Neprůhledná sekvence bytů.
- Celé nezáporné číslo, které je na síti reprezentováno jako seřazení určitého počtu bytů, podle Délky Nastavení.
- Unicode řetězec zakódovaný pomocí UTF-8 v NET-Unicode formě.

3.3 Nastavení/Možnosti

V žádostech i odpovědích může být použito jedno či více Nastavení. CoAP definuje jeden ustálený seznam, jehož položky budou popsány v následujícím textu:

- Content-Format (Formát-Obsahu).
- ETag (EŠtítek).
- Location-Path (Cesta-Umístění).
- Location-Query (Dotaz-Umístění).
- Max-Age (Maximální-Věk).
- Proxy-Uri.
- Proxy-Scheme (Proxy-Schéma).
- Uri-Host.
- Uri-Path (Uri-Cesta).
- Uri-Port.
- Uri-Query (Uri-Dotaz).
- Accept (Přijmout).
- If-Match (Pokud-Shoda).
- If-None-Match (Pokud-Žádná-Shoda).
- Size1 (Velikost1).

Content-Format Nastavení označuje formát ve kterém je uloženo Zatížení zprávy. Ten je dán jako číselný identifikátor, který je definován v registru „CoAP Content-Formats“. Formáty a jejich ID jsou text/prostý; charset=utf-8 (0), aplikační/link-format (40), aplikační/xml (41), aplikační/octet-stream (42), aplikační/exi (47) a aplikační/json (50). Pokud toto Nastavení není přítomno, není předpokládána žádná výchozí hodnota, to znamená, že formát Zatížení je neurčitý.

ETag, neboli Entity-tag představuje identifikátor zdrojů na lokální úrovni pro rozlišení různých verzí stejného zdroje, například přes delší časový úsek. Generuje ho server a koncový bod, který ho přijímá, s ním musí zacházet jako s neprůhledným, tzn., že nesmí dělat žádné předpoklady ohledně jeho struktury či obsahu.

Location-Path a Location-Query dohromady určují relativní URI, který se skládá buď z cesty absolutní, dotazovacího řetězce, nebo obou. Každé Location-Path Nastavení specifikuje jeden segment absolutní cesty a každé Location-Query Nastavení specifikuje jeden argument parametrizující daný zdroj. Mohou obsahovat jakoukoli sekvenci znaků. Hodnota Location-Path nesmí být znak tečka „**.**“. Tentýž znak se v hodnotě Location-Path nesmí vyskytovat ani dvakrát po sobě, příklad „**..**“.

Max-Age ukazuje maximální možnou dobu, po kterou bude odpověď považována za „čerstvou“, tzn., že může být použita jako odpověď na dotazy, aniž by bylo nutné kontaktovat zdrojový server. Hodnotou je celé číslo označující sekundy v rozmezí 0 až 2^32-1 včetně (zhruba 136.1 let). V případě absence hodnoty v odpovědi se předpokládá hodnota 60 sekund, což je také hodnota výchozí.

Proxy-Uri Nastavení se používá v případě žádosti odeslané dopředné-proxy. Ta odešle z platné mezipaměti a následně vrátí odpověď. Hodnotou Nastavení je absolutní-URI tzn. absolutní forma URI bez identifikátoru fragmentu. Při použití Proxy-Scheme je absolutní-URI nahrazena novou CoAP URI, která je vytvořena z odeslaných Uri-* Nastavení.

Uri-Host, Uri-Port, Uri-Path, a Uri-Query Nastavení jsou používány ke specifikování cílových zdrojů žádosti CoAP serveru. Uri-Host specifikuje zařízení v síti s požadovaným zdrojem, Uri-Port určuje číslo portu na transportní vrstvě daného zdroje, každé Uri-Path Nastavení označuje jeden segment absolutní cesty ke zdroji, a každé Uri-Query Nastavení označuje jeden argument parametrizující daný zdroj. Výchozí hodnotou pro Uri-Host je IP literál reprezentující cílovou IP adresu žádosti. Stejně tak je výchozí hodnotou Uri-Port cílový UDP port. Explicitní Uri-Host a Uri-Port Nastavení se většinou používají, pouze pokud je na koncovém zařízení vícero virtuálních serverů. Uri-Path a Uri-Query mohou obsahovat libovolnou sekvenci znaků, ale Uri-Path nesmí obsahovat „**.**“, nebo „**..**“.

Accept Nastavení může být použito k indikování, který Content-Format je pro klienta akceptovatelný. Nastavení je dáno jako numerické označení identifikátoru Content-Format. Pokud není udána žádná hodnota, klient nebude vykazovat žádnou preferenci, tudíž přijme libovolný Content-Format.

If-Match Nastavení může být použito k zaslání žádosti, která bude podmíněna momentální hodnotou nebo existencí ETagu pro jednu nebo více reprezentací cílového zdroje. Hodnotou je buď samotný ETag, nebo prázdný řetězec. Pokud je hodnotou prázdný řetězec, podmínce vyhoví jakákoli existující reprezentace, tj. jakýkoli ETag, včetně prázdných. If-Match Nastavení se může vícekrát opakovat. Pokud se bude jakékoli shodovat, podmínka bude považována za splněnou.

If-None-Match Nastavení může být použito k zaslání žádosti, která bude podmíněna neexistencí ETagu na cílovém zdroji. Pokud cílový zdroj existuje, podmínka je považována za nesplněnou. Není vhodné kombinovat Nastavení If-Match a If-None-Match v jedné žádosti, protože obě podmínky nemohou být nikdy splněny najednou.

Size1 Nastavení uchovává informace o velikosti reprezentace zdroje v žádosti. Hodnotou je celé číslo představující počet bytů. Používá se k indikování maximální velikosti žádosti, kterou je server schopen a ochoten zpracovat.

3.4 Využití v praxi

CoAP je specializovaný protokol navržený pro použití na omezených sítích a uzlech v Internetu věcí. Byl navržen pro M2M aplikace jako chytré energetické sítě a automatizace budov, a vyvíjen jako Internetový Standard RFC 7252, který má za cíl vydržet desetiletí.

4 AMQP

AMQP je binární protokol aplikační vrstvy navržený pro efektivní podporu široké škály aplikací pro zasílání zpráv a komunikačních schémat. Mezi jeho hlavní vlastnosti patří orientace a směrování zpráv, jejich řazení, zajištění spolehlivosti a bezpečnosti. AMQP zabezpečuje kontrolovaný tok komunikace s různými stupni garance doručení (QoS) „nejvýše jednou“, „alespoň jednou“, a „přesně jednou“. Rovněž autentizace popřípadě šifrování založeném na SASL a/nebo TLS.

4.1 Typový systém

Typový systém AMQP definuje množinu běžně používaných primitivních datových typů pro dosažení interoperabilní reprezentaci dat. Hodnoty AMQP mohou být okomentovány dodatečnými sémantickými informacemi, i těmi, které nejsou běžně spojované s dotyčným primitivním datovým typem. To dovoluje spojit AMQP hodnotu s vnějším datovým typem, který není přítomen v primitivních datových typech AMQP. Například, URL je běžně reprezentován jako řetězec, nicméně ne všechny řetězce jsou platné URL. Typový systém AMQP dovoluje definovat kód, který může být použit k okomentování řetězce, pokud má být tento chápán jako reprezentace URL. Stejně tak, lze okomentovat například datový typ map, jenž obsahuje páry klíč-hodnota. Kupříkladu „jméno“, „adresa“ by mohly být okomentovány jako reprezentace datového typu „zákazník“.

4.1.1 Primitivní datové typy

Typový systém AMQP definuje standardní kolekci primitivních datových typů pro reprezentaci běžných skalárních hodnot i běžných kolekcí. Skalární datové typy zahrnují boolean, celá čísla, čísla s plovoucí čárkou, časová razítka, UUID, znaky, řetězce, binární data, a symboly. Kolekce obsahují řady, listy, a mapy.

4.1.2 Popisné datové typy

Primitivní datové typy AMQP mohou být použity pro přímou reprezentaci dat používaných ve většině populárních programovacích jazycích. V praxi má ovšem i ta nejjednodušší aplikace vlastní kolekci vlastních datových typů. V aplikacích pro výměnu a odesílání zpráv musí být tyto speciální datové typy zformovat pro přenos. AMQP toto dovoluje pomocí okomentování datového typu tzv. popisovačem. Popisovač sdruží vlastní datový typ aplikace a datový typ AMQP. Toto sdružení má za úkol označovat, že daný AMQP datový typ je ve skutečnosti reprezentací vlastního datového typu. Výsledná kombinace AMQP datového typu a popisovače se nazývá po popisný datový typ. Popisný datový typ obsahuje dva odlišné

druhy typových informací. Identifikuje AMQP datový typ i vlastní datový typ, včetně jejich vztahu. Aplikace se znalostí vlastního datového typu, může popisný datový typ dekódovat a zpracovat jako daný vlastní datový typ. Aplikace, jenž nemá znalost tohoto typu, ho může dekódovat a zpracovat jako datový typ AMQP.

4.1.3 Složené datové typy

AMQP definuje několik různých složených datových typů, které jsou používány pro strukturovaná data jako například těla rámců. Složený datový typ definuje složenou hodnotu, kde každá jednotlivá hodnota identifikována pomocí pojmenovaného pole. Každá definice složeného datového typu obsahuje řazenou řadu polí, kde každé pole má konkrétní jméno, typ, a mnohočetnost. Definice složených datových typů rovněž obsahují jeden či více popisovačů, symbolických a/nebo numerických, pro identifikaci jejich definované reprezentace.

4.1.4 Omezené datové typy

Omezený datový typ je nový datový typ definovaný AMQP, odvozený od existujícího datového typu, kde povolená hodnota nového datového typu je podmnožinou hodnot již existujícího datového typu. Omezené datové typy jsou běžně používané v programování, nejčastěji jako typy „výčtové“. Omezené datové typy AMQP mohou reprezentovat otevřenější omezení jako například URL, o kterém můžeme přemýšlet, jakož to o omezení nad datovým typem řetězec. Definice omezeného datového typu může tudíž limitovat dovolené hodnoty na předem definovanou fixní nebo otevřenější množinu hodnot. V dříve jmenovaném případě je každá povolená hodnota nazývána volbou a všechny takto povolené volby jsou vypsány v definici daného omezeného datového typu. V později jmenovaném případě je toto omezení specifikováno jako text v definici datového typu. Existující datový typ, z něhož je omezený datový typ odvozen se nazývá zdroj.

4.2 Komunikace

Síť AMQP se skládá z uzlů spojených spoji. Uzly jsou pojmenované entity, které jsou zodpovědné za bezpečné uložení a/nebo doručení zpráv. Zprávy mohou z uzlů pocházet, končit na nich, či jimi být směrovány. Spoj je jednosměrná cesta (spojení) mezi dvěma uzly. Spoj se k uzlu připojuje v mezníku. Existují dva druhy mezníků, zdroje a cíle. Mezník je odpovědný za sledování stavu toku dat příchozích a odchozích dat. Zdroje sledují odchozí zprávy, zatímco cíle sledují zprávy příchozí. Zodpovědnost za zprávu si jednotlivé uzly postupně předávají podle toho, jak daná zpráva cestuje po síti přes dané uzly. Převod

zodpovědnosti mezi zdrojem a cílem obstarává protokol linkové vrstvy. Uzly se nacházejí v kontejnerech, například distributorských a klientských aplikacích. Každý kontejner může obsahovat více uzlů. Mezi AMQP uzly spadají producenti, konzumenti, a řady. Producenti a konzumenti jsou elementy v aplikaci, které generují a zpracovávají zprávy. Řady jsou entitami, jenž uchovávají a odesílají zprávy.

Aby bylo možno úspěšně komunikovat mezi dvěma kontejnery, musí být nejprve navázáno spojení. Spojení AMQP se skládá z full-duplex, spolehlivě seřazené sekvence rámců. Spojení si dohodnou maximální velikost rámců, což dovoluje jednoduchou defragmentaci toku bytů do kompletních rámcových těl. Je požadováno, aby v případě doručení rámce n, byly doručeny všechny rámce, které tomuto rámci n předcházely. Předpokládá se, že spojení jsou pomíjivá a mohou selhat z různých důvodů, což vyústí ve ztrátu neznámého počtu rámců, nicméně tyto jsou stále předmětem výše zmíněného pravidla spolehlivého řazení. Podobnou garanci zařizují i TCP a SCTP pro toky bytů.

AMQP spojení je rozděleno do dohodnutého počtu nezávislých jednosměrných kanálů. Každý rámec je označen číslem kanálu, které označuje jeho rodičovský kanál. Sekvence rámců každého kanálu je složena do jediné sekvence rámců pro celé spojení. Relace AMQP vytváří vztah mezi dvěma jednosměrnými kanály, pro vytvoření dvousměrné, sekvenční konverzace mezi dvěma kontejnery. Relace zajišťují kontrolu toku dat, vycházející z počtu odeslaných rámců přenosu. Díky tomu, že rámce mají pro dané spojení omezenou velikost, je možné kontrolovat tok podle počtu odeslaných bytů. To může být použito pro k optimalizaci výkonu. Jediné spojení může mít více na sobě nezávislých aktivních relací, až po dohodnutý limit kanálu.

4.3 Formát rámce

Rámce obsahují tři části. Hlavičku rámce s pevně danou šířkou, prodlouženou hlavičku s proměnnou šířkou, a tělo rámce s proměnnou šířkou (Tabulka 10 a Tabulka 11). Hlavička má velikost 8 bytů a vyskytuje se před každým rámcem. Obsahuje povinné informace nutné k analýze daného rámce včetně velikosti a informace o typu rámce. Prodloužená hlavička nemá pevně stanovenou velikost a stojí před tělem rámce. Jedná se o rozšiřující bod, který byl definován z důvodu budoucího rozvoje. Jak se s touto oblastí zachází, závisí čistě na typu rámce. Tělo rámce je sekvence bytů s proměnnou šířkou. Jejich formát závisí na typu rámce.

Tabulka 10 – Obecný formát rámce

	+0	+1	+2	+3		
0	Velikost				Hlavička rámce (8 bytů)	
4	DOFF	Typ	<Specifické-pro-typ>			
8	<Specifické-pro-typ>				Prodloužená hlavička (DOFF * 4 - 8 bytů)	
4*DOFF	<Specifické-pro-typ>				Tělo rámce (Velikost – DOFF * 4 bytů)	

Zdroj: (OASIS, 2012)

První čtyři byty na pozicích 0 -3, obecného rámce (Tabulka 10), obsahují velikost rámce, jako bezznaménkový celočíselný 32bit datový typ, který musí obsahovat celkovou velikost hlavičky, prodloužené hlavičky i těla rámce. Byte 4 v hlavičce rámce obsahuje datový offset (posun). Udává tak pozici těla uvnitř rámce. Jeho hodnota je udána jako bezznaménkový celočíselný 8bit datový typ. Byte 5 udává typ rámce, tzn. formát a účel. Následující byty rámce, mohou být chápány různě podle typu. Kód 0x00 znamená AMQP.

Tabulka 11 – AMQP rámec 0x00

	+0	+1	+2	+3	
0	Velikost				Hlavička rámce (8 bytů)
4	DOFF	Typ	Kanál		
8	<Ignorováno>				Prodloužená hlavička (DOFF * 4 - 8 bytů)
4*DOFF	Úkol				Tělo rámce (Velikost – DOFF * 4 bytů)
	Zatížení				

Zdroj: (OASIS, 2012)

Existuje několik úkolů, které rámce typu AMQP provádějí.

- Open (otevři) – dohodni parametry připojení.
- Begin (začni) – otevři na kanálu relaci.
- Attach (připoj) – připoj do relace spoj.
- Flow (tok) – aktualizuj stav spoje.
- Transfer (přenes) – přenes zprávu.
- Disposition (uspořádání) – informuj vzdálené zařízení o změně stavu doručení.
- Detach (odpoj) – odpoj spoj od relace.
- End (ukonči) – ukonči relaci.
- Close (zavři) – dej znamení o ukončení spojení.

4.4 Formát zprávy

O zprávě můžeme uvažovat ze dvou pohledů. Odesílatel o ní může přemýšlet jako o neměnném balíčku (zatížení), který předává systému či infrastruktuře pro předávání zpráv. Příjemce zprávy o ní nepřemýšlí pouze jako o tomto neměnném balíčku. Příjemce totiž dostane i různé dodatky, vysvětlivky, a komentáře, které k dané zprávě dodal právě systém pro její doručení. Proto rozlišujeme dva pojmy. Holá zpráva označuje zprávu odeslanou odesílatelem bez jakýchkoli dodatků. Komentovaná zpráva je zpráva, kterou dostal příjemce i s informacemi přidanými infrastrukturou pro její doručení. Komentovaná zpráva se skládá z holé zprávy a sekcí určených pro komentáře v záhlaví i zápatí holé zprávy. Komentářů jsou dva druhy, první cestuje se zprávou navždy, druhý, který se zprávou cestuje pouze do příštího uzlu.

Tabulka 12 – Formát zprávy AMQP

			Holá zpráva			
Hlavička	Komentáře doručení	Komentáře zprávy	Vlastnosti	Vlastnosti aplikace	Data aplikace	Zápatí
Komentovaná zpráva						

Zdroj: (OASIS, 2012)

Hlavička obsahuje standardní informace o přenosu zprávy skrz síť AMQP (priorita, TTL, atd.). Pokud není žádná, příjemce předpokládá výchozí hodnoty. Komentáře doručení jsou používané pro uložení nestandardních informací o přenosu od odesílatele příjemci. Pokud se nevyskytuje, předpokládá se, že je přítomna jako prázdná sekce. Sekce komentáře zprávy je

používána pro uložení vlastností zprávy zaměřených na samotnou infrastrukturu, které by měli být šířeny při každém kroku doručení. Pokud není přítomna, předpokládá se prázdná sekce. Vlastnosti představují neměnné vlastnosti zprávy, které musí zůstat stejné při každém odeslání. Vlastnosti aplikace je použita pro strukturování aplikačních dat. Prostředníci mohou použít data uvnitř této struktury pro účely filtrován a směrování. Data aplikace obsahují binární data. Zápatí obsahují informace o zprávě či jejím doručení, které mohou být spočítány nebo ohodnoceny až po doručení celé holé zprávy.

4.5 Implementace AMQP

Mezi momentální implementace AMQP 1.0 patří Apache Qpid a Apache ActiveMQ, což jsou open-source projekty pod záštitou Apache Software Foundation. Další implementací je RabbitMQ, rovněž open-source projekt sponzorovaný společností Pivotal Software. Všechny tyto implementace jsou systémy pro přijímání a publikaci zpráv.

5 WEBSOCKET

WebSocket je komunikační protokol jenž umožňuje současnou obousměrnou komunikaci mezi klientem a vzdáleným hostem, přes TCP spojení. Z historického pohledu, musely webové aplikace, vyžadující obousměrné připojení, využít http, pro odeslání dotazu na server ohledně aktuálních informací, zatímco jím jsou stejným kanálem, proti směru proudících dat, posílány oznámení serveru. Z toho vychází několik problémů.

- Server je nucen používat několik různých TCP spojení pro každého klienta. Jedno pro posílání informací klientovi a jedno nové pro každou novou příchozí zprávu.
- Protokol má vysokou režii. Každá zpráva, mířící od klienta k serveru, má vlastní http hlavičku.
- Skript na straně klienta je nucen udržovat si „mapu“ příchozích spojení pomocí odchozích spojení, kvůli sledování odpovědí.

Jednodušší řešení by bylo udržovat jedno TCP spojení pro komunikaci v obou směrech. WebSocket dělá právě to. Byl navržen pro implementaci ve webových prohlížečích a serverech nicméně ho může využít jakákoli klientská či server aplikace.

5.1 Komunikace

WebSocket komunikace se skládá ze dvou částí, tzv. „handshake“ (potřesení rukou) a přenosu dat. Jakmile si klient a server úspěšně „potřesou rukou“, může začít přenos dat. Data se přenášejí pomocí zpráv, které se skládají z jednoho či více rámců.

Handshake byl zamýšlen pro kompatibilitu s HTTP softwarem a prostředníky na straně serveru, aby se serverem komunikovat HTTP klienti i WebSocket klienti přes jediný port. Proto je WebSocket handshake HTTP Upgrade žádost.

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Odpověď serveru.

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Kromě běžné HTTP Upgrade hlavičky, také klient odešle `Sec-WebSocket-Key`, jenž obsahuje náhodné byty zakódované pomocí base64 kódování. Server odpoví pomocí hash klíče v `Sec-WebSocket-Accept`. Další pole hlavičky jsou používány pro určování nastavení WebSocket protokolu. Typickým příkladem je selektor pod protokolu `Sec-WebSocket-Protocol`, či seznam rozšíření, která klient podporuje `Sec-WebSocket-Extensions`. `Sec-WebSocket-Protocol` může být použit pro indikaci, které pod protokoly jsou pro klienta přijatelné, server si následně jeden vybere (či žádný) a svou odpověď odešle ve svém handshake zpět.

5.2 Formát rámce

WebSocket přenáší data jako sekvenci rámců. Z bezpečnostních důvodů musí klient přiřadit každému rámci masku. Ten pak odešle serveru. Pokud server přijme rámec bez masky, musí okamžitě ukončit spojení. Naopak server nesmí přiřazovat masku rámcům, jenž posílá klientovi. Pokud klient detekuje příchozí rámec s maskou, musí okamžitě ukončit spojení.

Základní rámcový protokol definuje typ rámce s opcode, délkou zatížení a specifickými umístěními pro rozšiřující data a aplikační data, která společně definují data zatížení (Tabulka 13).

Tabulka 13 – Formát rámce WebSocket

<table>
<tr><td>0</td><td colspan="7"></td><td>1</td><td colspan="7"></td><td>2</td><td colspan="7"></td><td>3</td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>0</td><td>1</td></tr>
<tr><td>FIN</td><td>RSV1</td><td>RSV2</td><td>RSV3</td><td colspan="4">opcode</td><td>MASK</td><td colspan="7">Délka zatížení</td><td colspan="22">Prodloužená délka zatížení (pokud je délka zatížení 126 bytů)</td></tr>
<tr><td colspan="32">Pokračování prodloužené délka zatížení (pokud je délka zatížení 127 bytů)</td></tr>
<tr><td colspan="24"></td><td colspan="8">Klíč masky, pokud je MASK 1</td></tr>
<tr><td colspan="24">Klíč masky (pokračování)</td><td colspan="8">Data zatížení</td></tr>
<tr><td colspan="32">Pokračování dat zatížení</td></tr>
<tr><td colspan="32">Pokračování dat zatížení</td></tr>
</table>

Zdroj: (IETF, 2011)

Následuje popis jednotlivých polí nacházejících se v rámcích WebSocket.

FIN (1 bit): Indikuje, zda se jedná o finální rámec zprávy. První rámec zprávy může být zároveň posledním rámcem.

RSV1, RSV2, RSV3 (každý 1 bit): Musí být 0, pokud nebylo při tvoření spojení dohodnuto rozšíření, které by definovalo význam nenulových hodnot. Pokud bude přijata nenulová hodnota, aniž by dohodnuté rozšíření definovalo její význam, musí být spojení ukončeno.

opcode (4 bity): Definuje interpretaci dat zatížení. Pokud je přijat neznámý opcode, spojení musí být ukončeno. Jsou definovány následující hodnoty.

- %x0 – rámec pokračování,
- %x1 – textový rámec,
- %x2 – binární rámec,
- %3-7 – rezervovány pro další nekontrolní rámce,
- %x8 – rámec pro ukončení spojení,
- %x9 – rámec ping,
- %xA – rámec pong,
- %xB-F – rezervovány pro další kontrolní rámce.

Maska (1 bit): Určuje, zda mají data zatížení přiřazenou masku či nikoliv. Pokud je hodnota 1, je v poli klíč masky přítomen klíč masky, který je použit odmaskování dat zatížení. Všechny rámce odeslány klientem serveru mají tento bit nastavený na hodnotu 1.

Délka zatížení (7 bitů, 7+16 bitů, nebo 7+64 bitů): Představuje délku dat zatížení v bytech. Pokud je hodnota 0-125, pak má velikost 7 bitů. Pokud je hodnota 126, k původním 7 bitům se přičtou následující 2 byty. Pokud je hodnota 127, pak následujících 8 bytů, představuje délku zatížení (nejdůležitější bit musí být 0). Délka zatížení představuje délku rozšiřujících dat sečtenou s délkou aplikačních dat. V případě, že se délka rozšiřujících dat rovná nule, představuje délku aplikačních dat.

Klíč masky (0 nebo 4 byty): Všechny rámce odeslané klientem serveru jsou maskované 32bit hodnotou obsaženou v rámci. Toto pole je přítomno, pokud je hodnota bitu masky 1 a není přítomno, pokud je hodnota bitu masky 0.

Data zatížení (x + y bytů): Data zatížení jsou definována jako rozšiřující data a aplikační data.

Rozšiřující data (x bytů): Rozšiřující data jsou 0, pokud nebylo dohodnuto rozšíření. Jakékoli rozšíření musí specifikovat délku tohoto pole, nebo jak se tato délka dá spočítat, a jak se toto

rozšíření musí dohodnout při výměně handshake. Pokud jsou rozšiřující data přítomna, jsou součástí celkového zatížení.

Aplikační data (y bytů): Aplikační data zabírající zbytek rámce, po jakýchkoli rozšiřujících datech. Jejich délka je ekvivalentní délce zatížení mínus délka rozšiřujících dat.

5.3 Implementace WebSocket

Protokol WebSocket je implementován ve webových prohlížečích, serverech, dále také běhových prostředích a knihovnách. Mezi příklady implementace protokolu WebSocket patří Google Chrome 16, Mozilla Firefox 11, Lightstreamer, POCO C++ Libraries, Resin, Opera 12.10, Internet Explorer 10, a další.

6 NODE

Node.js je open-source softwarové prostředí postavené na V8 JavaScript engine, určené pro provádění JavaScript kódu na straně serveru. Byl (je) vyvíjen v programovacím jazyce C a C++. Typicky byl JavaScript zabudovaný přímo v html stránce a prováděl se na straně klienta pomocí JavaScript enginu z webového prohlížeče klienta. Pomocí Node.js můžeme na straně serveru spustit daný skript pro vytvoření dynamického obsahu stránky ještě předtím, než danou stránku odešleme uživateli, tomu dodáme již hotovou stránku s dynamickým obsahem. Node.js má událostmi řízenou architekturu a podporuje asynchronní I/O[4]. Účelem takovéhoto návrhu implementace Node.js je optimalizace průchodnosti webových aplikací, které obsahují vícero operací vstupu a výstupu.

Jakožto asynchronní JavaScript prostředí řízené událostmi, je Node.js navržen pro snadnou rozšiřitelnost síťových aplikací. Následující kus kódu je typickým začátečnickým „hello world“ příkladem, ve kterém může být řízeno několik spojení. Při každém spojení je zavolána callback funkce, ovšem pokud není co dělat, nic se dít nebude.

```
const http = require('http');
const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Pro takovouto práci se dnes běžně používají vlákna operačního systému, nicméně jejich použití v sítích je relativně složité a neefektivní. Uživatelé Node.js se navíc nemusí bát, že by se jim proces dostal stavu uváznutí, tzv. stavu deadlock, z prostého důvodu, že skoro žádná funkce v Node.js neprovádí vstup/výstup přímo, takže procesy nejsou nikdy blokovány.

Node.js je hojně využíván korporacemi jako Microsoft, Cisco Systems, Netflix, Paypal, LinkedIn, Yahoo!, IBM, Walmart, a další.

[4] Asynchronní I/O je forma zpracování vstupu/výstupu, jenž dovoluje jiným procesům pokračovat, než bude zpracována celá transakce.

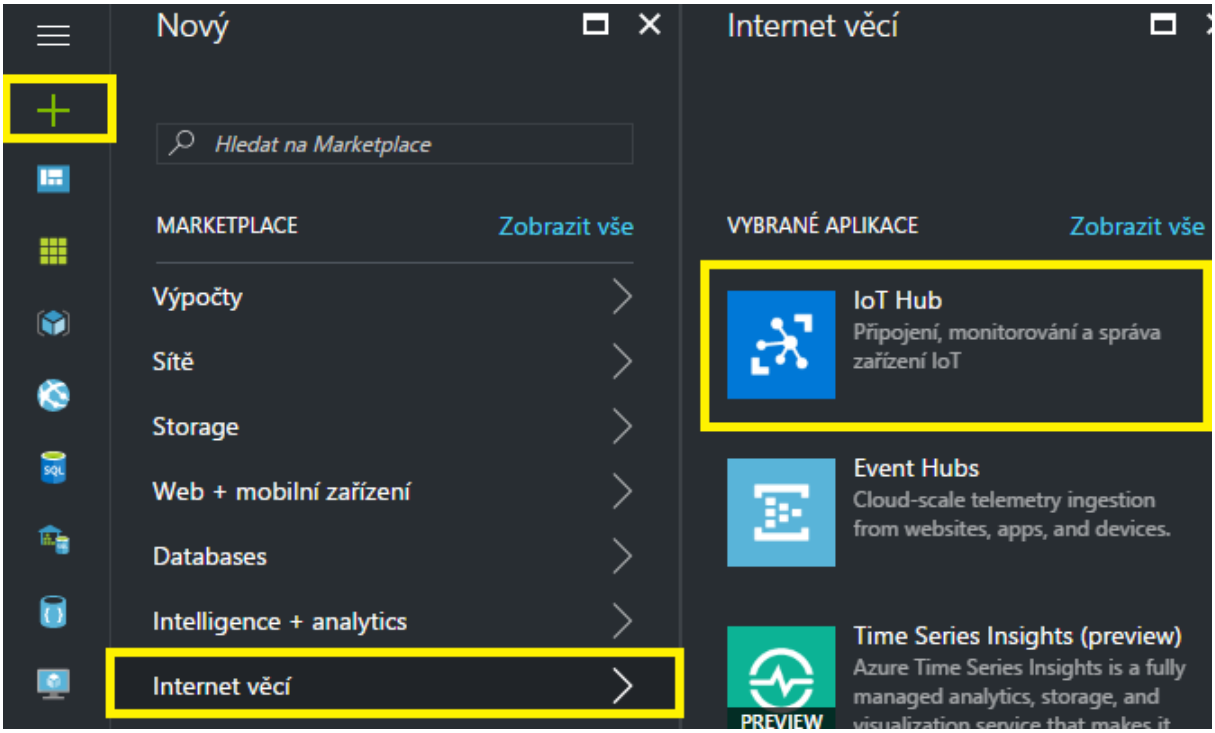
7 PŘIPOJENÍ K AZURE IOT HUB

Azure IoT hub je služba která umožňuje spolehlivou a zabezpečenou obousměrnou komunikaci mezi miliony zařízení Internetu věcí. Vlastnosti Azure IoT hub:

- Nabízí spolehlivou zařízení-cloud a cloud-zařízení komunikaci.
- Dává uživateli možnost zabezpečené komunikace na principu kontroly přístupu a bezpečnostních pověření specifických pro každé zařízení.
- Zahrnuje knihovny pro nejpopulárnější jazyky a platformy, které mohou zařízení využít.

7.1 Vytvoření IoT hub

Pro možnost připojení se, je třeba mít se kam připojit. Prvním krokem je přihlášení, případně registrace na portálu Azure[5]. Po úspěšném přihlášení k platformě Microsoft Azure lze z levého kontextového menu vybrat položku Nový, poté Internet Věcí, a nakonec IoT Hub (Obrázek 1).

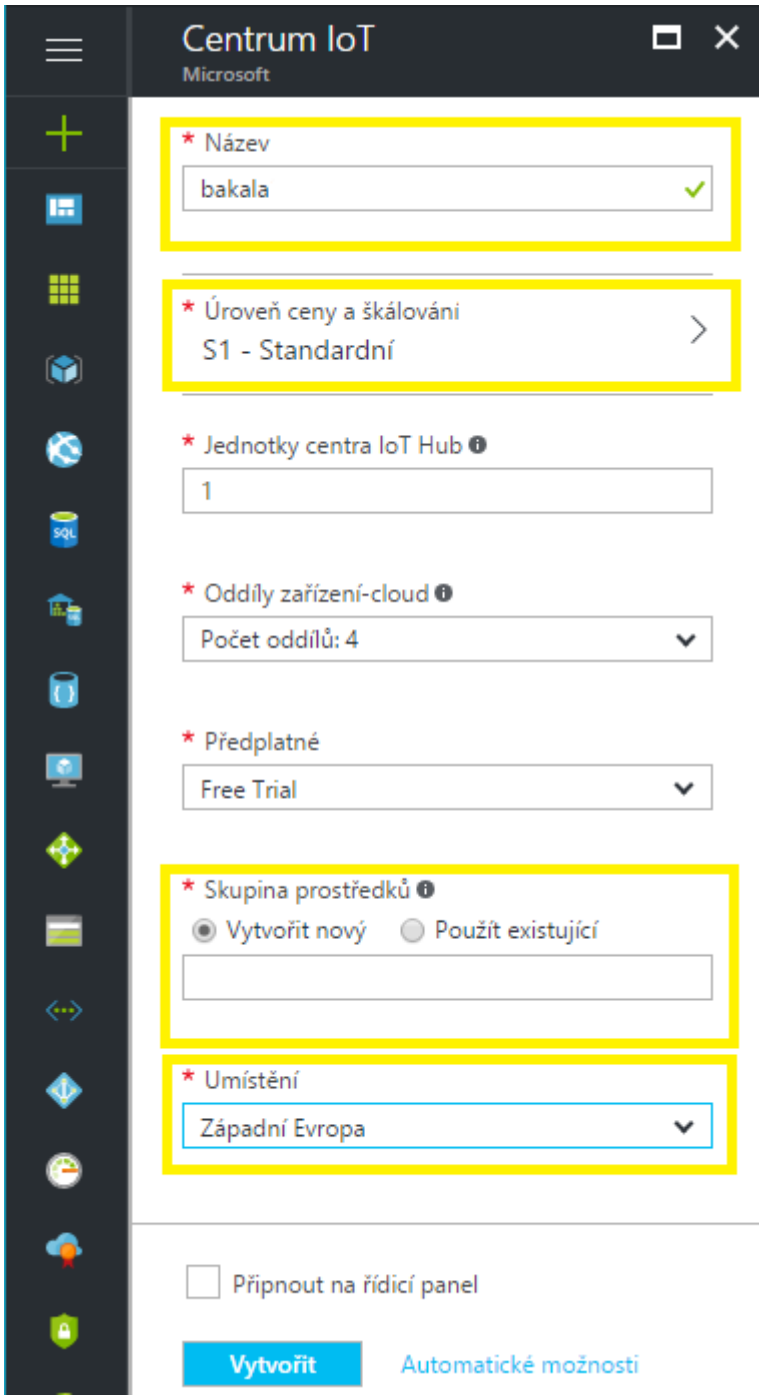


Obrázek 1 – IoT hub vytvoření

Zdroj: vlastní

[5] https://azure.microsoft.com/cs-cz/

Následně je nutné nastavit několik základních parametrů IoT hubu (Obrázek 2).



Obrázek 2 – IoT hub konfigurace

Zdroj: vlastní

Název: Název IoT hubu. V případě, že jméno je validní, objeví se na pravém konci pole zelené zaškrtnutí.

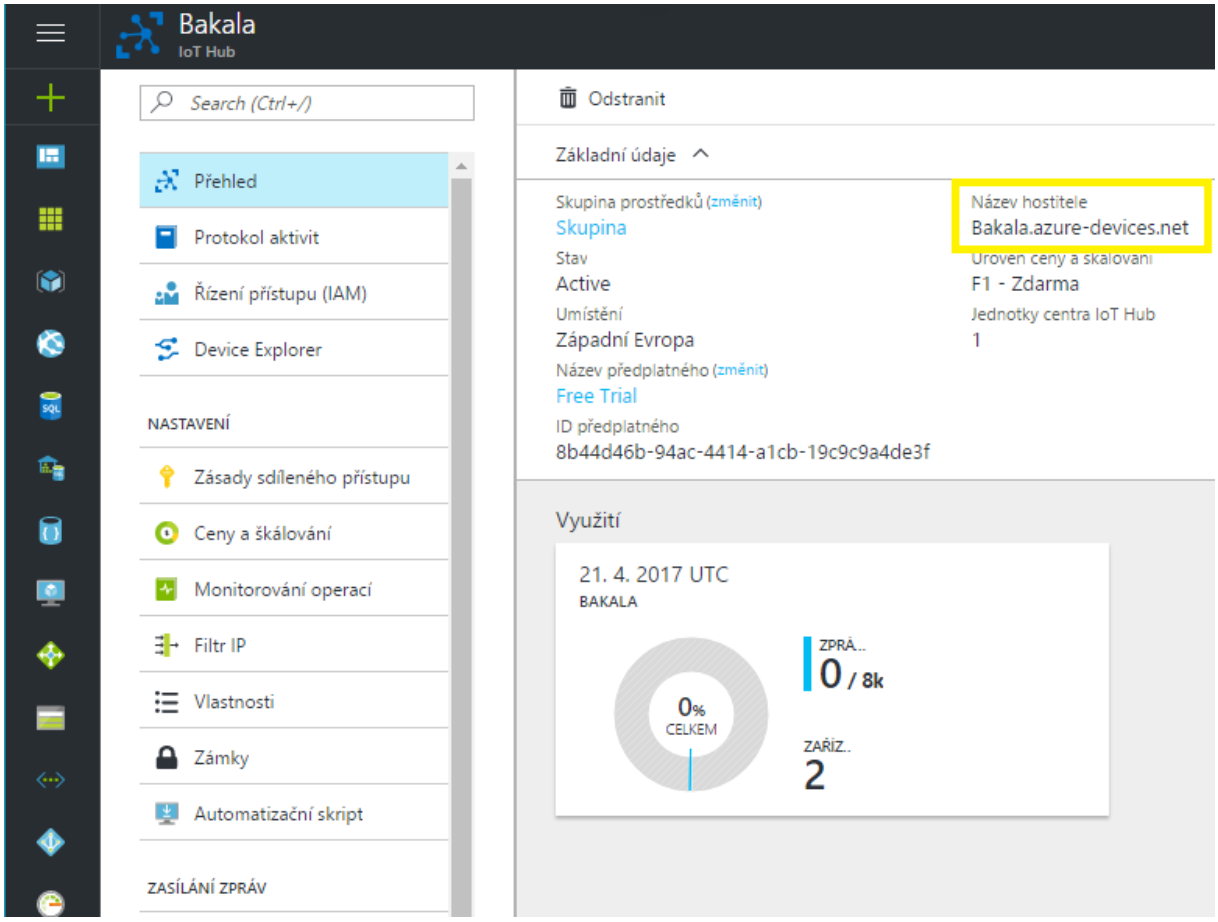
Úroveň ceny a škálování: Určuje cenu služby. S rostoucí cenou roste počet zpráv za den, jednotek a zařízení, které můžeme mít připojené. Ve svém IoT hubu používám úroveň

F1 – zdarma, která je určena pro testovací a vzdělávací účely. Dovoluje připojit jednu jednotku a přijmout 8 tisíc zpráv za den.

Skupina prostředků: Kolekce prostředků, jenž sdílejí životní cyklus, oprávnění a zásady.

Umístění: Lokace, ve které bude IoT hub hostován.

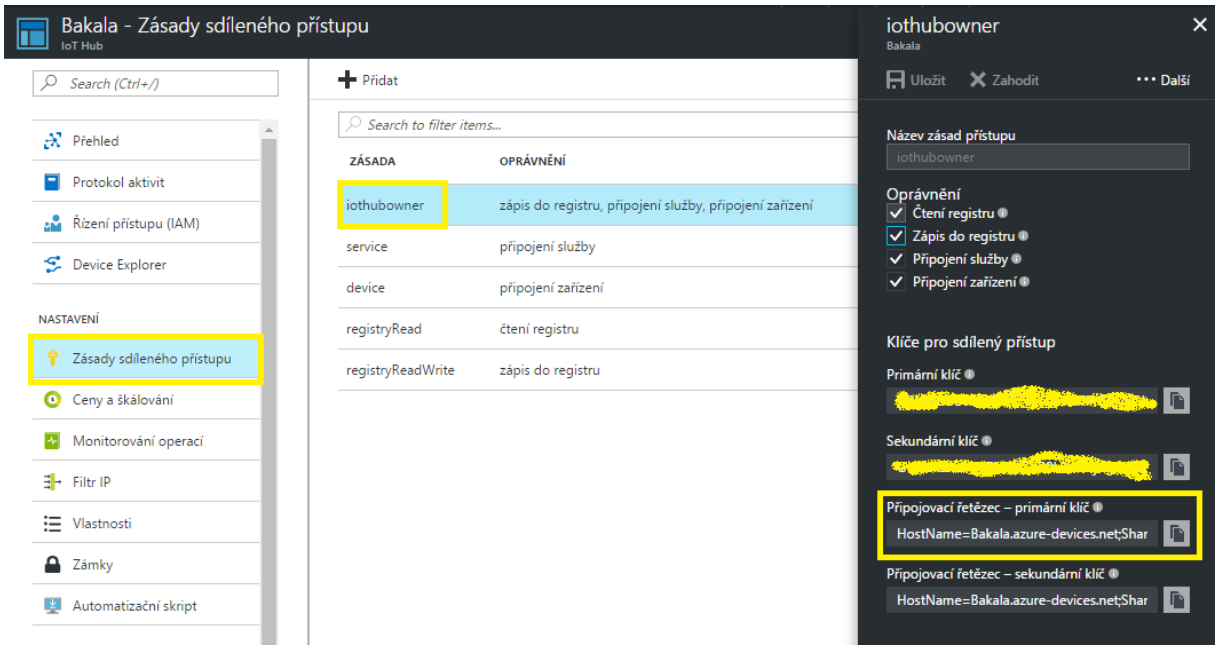
Po vybrání konfigurace a kliknutí na tlačítko Vytvořit, je třeba vyčkat, než Azure vytvoří daný IoT hub. Může to trvat i několik minut. Po vytvoření IoT hubu je třeba jej otevřít a zaznamenat si název hostitele (Obrázek 3).



Obrázek 3 – IoT hub

Zdroj: vlastní

Dále je třeba otevřít zásady sdíleného přístupu, kliknout na iothubowner a dále zaznamenat připojovací řetězec (Obrázek 4). Po těchto krocích byl úspěšně vytvořen IoT hub a je k dispozici název hostitele IoT hubu a připojovací řetězec, které budou potřeba níže.



Obrázek 4 – Zásady sdíleného přístupu

Zdroj: vlastní

7.2 Vytvoření identity zařízení

Je třeba vytvořit zařízení, které bude na IoT hub odesílat zprávy. Vytvoříme si Node.js konzolovou aplikaci, která nám vytvoří identitu zařízení v registru našeho IoT hubu. Když tuto aplikaci spustíme, vygeneruje nám unikátní ID zařízení, které naše zařízení může použít pro svou identifikaci, při komunikaci s IoT hub.

Postup:

1. Vytvoříme novou prázdnou složku s názvem „*createdeviceidentity*“. Ve složce createdeviceidentity, vytvoříme package.json soubor pomocí následujícího příkazu v příkazovém řádku. Přijmeme všechny výchozí hodnoty.

   ```
   npm init
   ```

2. Pomocí příkazového řádku si v naší createdeviceidentity složce, nainstalujeme azure-iothub Service SDK balíček. Použijeme k tomu následující příkaz.

   ```
   npm install azure-iothub -save
   ```

3. Pomocí textového editoru (např. PSPad editor) ve složce *createdeviceidentity* vytvoříme *„CreateDeviceIdentity.js“*.

4. Na začátek *CreateDeviceIdentity.js* souboru přidáme následující require výraz.

```
'use strict';
var iothub = require('azure-iothub');
```

5. Do souboru *CreateDeviceIdentity* přidáme následující kód. Část v uvozovkách nahradíme připojovacím řetězcem, který jsme zjistili v minulé sekci.

```
var connectionString = '{iothub connection string}';
var registry =
iothub.Registry.fromConnectionString(connectionString);
```

6. Přidáme následující kód, abychom v registru našeho IoT hubu vytvořili definici zařízení. Tento kód vytvoří zařízení v případě, že ho v registru IoT hubu nenajde. Pokud ho najde, vrátí jeho klíč.

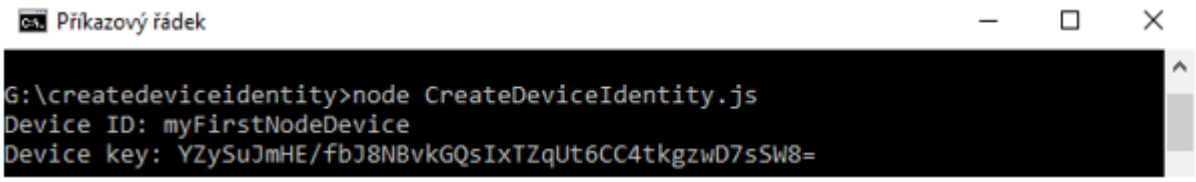
```
var device = new iothub.Device(null);
device.deviceId = 'myFirstNodeDevice';
registry.create(device, function(err, deviceInfo, res) {
  if (err) {
    registry.get(device.deviceId, printDeviceInfo);
  }
  if (deviceInfo) {
    printDeviceInfo(err, deviceInfo, res)
  }
});

function printDeviceInfo(err, deviceInfo, res) {
  if (deviceInfo) {
    console.log('Device ID: ' + deviceInfo.deviceId);
    console.log('Device key: ' +
deviceInfo.authentication.symmetricKey.primaryKey);
   }
 }
```

7. Uložíme a zavřeme *CreateDeviceIdentity.js* soubor.

8. Pro spuštění aplikace, spusťte v příkazovém řádku ve složce *createdeviceidentity* následující příkaz.

```
node CreateDeviceIdentity.js
```

Výsledkem by mělo být zhruba toto (Obrázek 5). Náš *Device ID* a *Device key* zaznamenáme pro pozdější použití.



Obrázek 5 – CreateDeviceIdentity

Zdroj: vlastní

7.3 Přijímání zařízení-cloud zpráv

V této sekci vytvoříme Node.js konzolovou aplikaci, jenž bude číst zařízení-cloud zprávy z IoT hubu. K tomu nám postačí koncový bod Events, který pro čtení zařízení-cloud zpráv vždy používá protokol AMQP.

Postup:

1. Vytvoříme prázdnou novou složku „*readdevicetocloudmessages*“. V této složce vytvoříme *package.json* soubor pomocí následujícího příkazu v příkazové řádce. Přijmeme všechny výchozí hodnoty.

   ```
   npm init
   ```

2. V příkazové řádce, ve složce *readdevicetocloudmessages*, nainstalujeme *azure-event-hubs* balíček.

   ```
   npm install azure-event-hubs –save
   ```

3. Pomocí textového editoru (např. PSPad editor) ve složce *readdevicetocloudmessages* vytvoříme *„ReadDeviceToCloudMessages.js“*.

4. Na začátek *ReadDeviceToCloudMessages.js* souboru přidáme následující require výraz.

   ```
   'use strict';
   var EventHubClient = require('azure-event-hubs').Client;
   ```

5. Přidáme deklaraci proměnné a nahradíme obsah závorek připojovacím řetězcem.

   ```
   var connectionString = '{iothub connection string}';
   ```

6. Přidáme dvě další funkce, které nám vytisknou výstup na konzoli.

```
var printError = function (err) {
  console.log(err.message);
};

var printMessage = function (message) {
  console.log('Message received: ');
  console.log(JSON.stringify(message.body));
  console.log('');
};
```

7. Na závěr přidáme následující kód pro vytvoření EventHubClient, otevření spojení s naším IoT hubem, a vytvoření přijímače pro každý samostatný oddíl.

```
var client = EventHubClient.fromConnectionString(connectionString);
client.open()
    .then(client.getPartitionIds.bind(client))
    .then(function (partitionIds) {
        return partitionIds.map(function (partitionId) {
            return client.createReceiver('$Default', partitionId, {
'startAfterTime' : Date.now()}).then(function(receiver) {
                console.log('Created partition receiver: ' +
partitionId)
                receiver.on('errorReceived', printError);
                receiver.on('message', printMessage);
            });
        });
    })
    .catch(printError);
```

8. Uložíme a zavřeme soubor ReadDeviceToCloudMessages.js.

7.4 Vytvoření simulované aplikace zařízení

V této sekci vytvoříme Node.js konzolovou aplikaci, která má za úkol simulovat zařízení, které bude posílat zařízení-cloud zprávy na náš IoT hub.

1. Vytvoříme prázdnou novou složku *„simulateddevice“*. V této složce vytvoříme *package.json* soubor pomocí následujícího příkazu v příkazové řádce. Přijmeme všechny výchozí hodnoty.

```
npm init
```

2. V příkazové řádce, ve složce *simulateddevice*, nainstalujeme *azure-iot-device* Device SDK balíček a *azure-iot-device-mqtt* balíček.

```
npm install azure-iot-device azure-iot-device-mqtt –save
```

3. Pomocí textového editoru (např. PSPad editor) ve složce *simulateddevice* vytvoříme *„SimulatedDevice.js“*.

4. Na začátek *SimulatedDevice.js* souboru přidáme následující require výrazy.

```
'use strict';

var clientFromConnectionString = require('azure-iot-device-
mqtt').clientFromConnectionString;
var Message = require('azure-iot-device').Message;
```

5. Dále přidáme proměnnou *connectionString* a použijeme ji k vytvoření instance klienta. Nahradíme *{youriothostname}* názvem našeho IoT hubu. Také nahradíme *{yourdevicekey}* klíčem zařízení, které jsme získaly v sekci Vytvoření identity zařízení.

```
var connectionString =
'HostName={youriothostname};DeviceId=myFirstNodeDevice;SharedAccess
Key={yourdevicekey}';

var client = clientFromConnectionString(connectionString);
```

6. Přidáme další funkci pro vypsání výstupu aplikace.

```
function printResultFor(op) {
  return function printResult(err, res) {
    if (err) console.log(op + ' error: ' + err.toString());
    if (res) console.log(op + ' status: ' + res.constructor.name);
  };
}
```

7. Vytvoříme callback a použijeme setInterval funkci k zasílání zprávy našemu IoT hubu každou vteřinu.

```
var connectCallback = function (err) {
  if (err) {
    console.log('Could not connect: ' + err);
  } else {
    console.log('Client connected');

    // Create a message and send it to the IoT Hub every second
    setInterval(function(){
        var temperature = 20 + (Math.random() * 15);
        var humidity = 60 + (Math.random() * 20);
        var data = JSON.stringify({ deviceId: 'myFirstNodeDevice',
temperature: temperature, humidity: humidity });
        var message = new Message(data);
        message.properties.add('temperatureAlert', (temperature >
30) ? 'true' : 'false');
        console.log("Sending message: " + message.getData());
        client.sendEvent(message, printResultFor('send'));
    }, 1000);
  }
};
```

8. Na konec přidáme kód pro otevření spojení k našemu IoT hubu a začneme odesílat zprávy.

```
client.open(connectCallback);
```

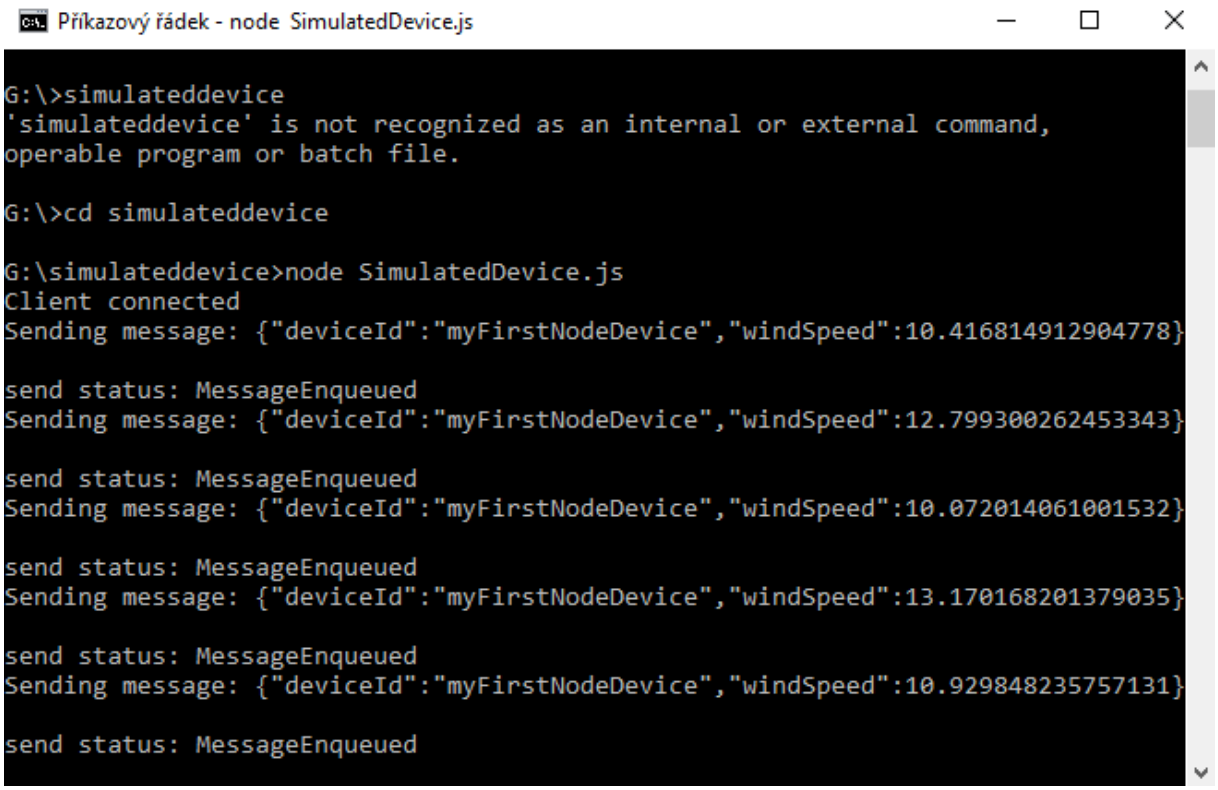
9. Uložíme a zavřeme soubor *SimulatedDevice.js*.

7.5 Spuštění aplikací

Nyní jsme připraveni spustit naše aplikace.

1. V příkazovém řádku ve složce *simulateddevice* spustíme následující příkaz, abychom začali odesílat telemetrii dat našemu IoT hubu.

```
node SimulatedDevice.js
```

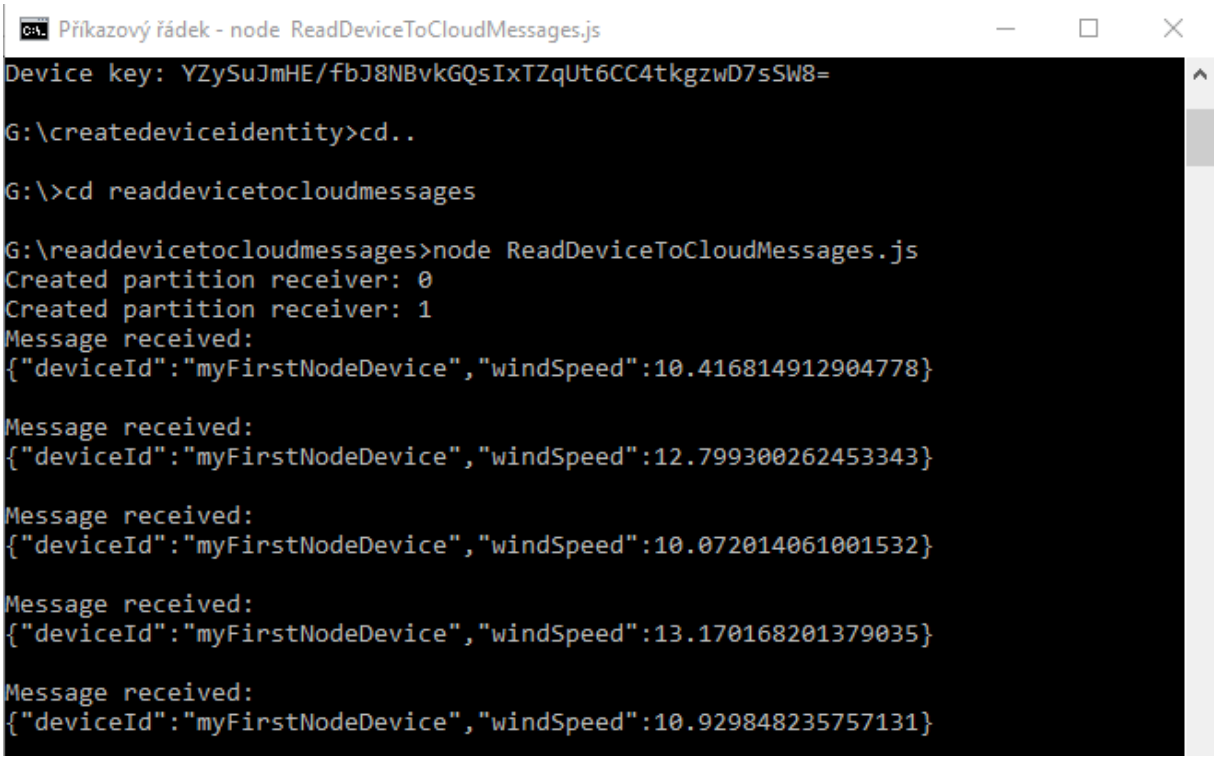


Obrázek 6 – SimulatedDevice

Zdroj: vlastní

2. V příkazovém řádku ve složce *readdevicetocloudmessages* spustíme následující příkaz, pro započetí monitorování našeho IoT hubu.

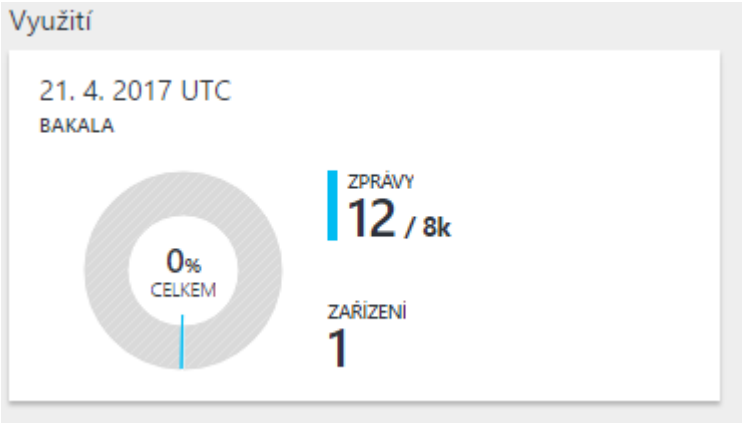
```
node ReadDeviceToCloudMessages.js
```



Obrázek 7 – ReadDeviceToCloudMessages

Zdroj: vlastní

3. Na závěr si portálu Azure zkontrolujeme počet odeslaných zpráv, které IoT hub zpracoval.



Obrázek 8 – IoT využití

Zdroj: vlastní

8 ZÁVĚR

Úkolem této bakalářské práce bylo podrobně popsat datové protokoly, jenž jsou používány v Internetu věcí. Na začátku bakalářské práce bylo uvedeno samotné téma Internetu věcí. Dále byly popsány datové protokoly, konkrétně to byly protokoly MQTT, CoAP, AMQP, a WebSocket. Tyto protokoly se zabývají přenosem dat a jeho řízením. V každé kapitole byl nastíněn způsob komunikace těchto protokolů, vyobrazen formát jejich řídících paketů či rámců a na závěr byly uvedeny příklady jejich využití a implementací v praxi. Poté bylo popsáno softwarové prostředí Node, sloužící pro provádění JavaScript kódu na straně serveru.

V praktické části bylo popsáno a předvedeno jak si vytvořit a propojit se s vlastním systémem Internetu věcí, jak na něj ze simulovaného zařízení odeslat zprávu a následně si ověřit její přijetí. K tomuto účelu byla vybrána platforma Azure IoT hub, na které byla úloha demonstrována pomocí skriptovacího jazyka JavaScript a softwarové prostředí Node.js, nicméně by místo těchto bylo možné využít programovacích jazyků C#, Java, či Python.

Při psaní této práce jsem získal přehled o způsobech využití Internetu věcí a rozšířil jsem si znalosti o datových protokolech, jenž se v Internetu věcí využívají. Znalosti jsem si rozšířil zejména v oblasti způsobu komunikace jednotlivých protokolů.

9 POUŽITÁ LITERATURA

Core IEC Standards [online]. IEC, 2016 [cit. 2017-04-21]. Dostupné z: http://www.iec.ch/smartgrid/standards/

Hivemq.com [online]. [cit. 2017-04-21]. MQTT 101 – How to Get Started with the lightweight IoT Protocol. Dostupné z: http://www.hivemq.com/blog/how-to-get-started-with-mqtt

IETF: RFC 6455 [online]. 2011 [cit. 2017-04-21]. Dostupné z: https://tools.ietf.org/html/rfc6455

IETF: RFC 7257 [online]. 2014 [cit. 2017-04-21]. Dostupné z: https://tools.ietf.org/html/rfc7252

Internet of Things (IoT) [online]. IoT Agenda [cit. 2017-04-21]. Dostupné z: http://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

JAFFEY, Toby. *Eclipse.org* [online]. 2. 2014 [cit. 2017-04-21]. MQTT and CoAP, IoT Protocols. Dostupné z: https://eclipse.org/community/eclipse_newsletter/2014/february/article2.php

MALÝ, Martin. *Root.cz* [online]. 29. 6. 2016 [cit. 2017-04-21]. Protokol MQTT: komunikační standard pro IoT. Dostupné z: https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/ ISSN 1212-8309

MALÝ, Martin. *Zdroják.cz* [online]. 14. 12. 2009 [cit. 2017-04-21]. Web Sockets. Dostupné z: https://www.zdrojak.cz/clanky/web-sockets/ ISSN 1803-5620

Microsoft Azure. Microsoft [online]. U.S.: Microsoft Corporation, 2016 [cit. 2017-04-21]. Dostupné z: azure.microsoft.com/en-us/documentation/services/iot-hub

Nodejs.org [online]. [cit. 2017-04-21]. About Node.js. Dostupné z: https://nodejs.org/en/about/

OASIS: MQTT 3.1.1 [online]. 2014 [cit. 2017-04-21]. Dostupné z: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

OASIS: OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0 [online]. 2012 [cit. 2017-04-21]. Dostupné z: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html#toc

Postcapes.com [online]. 1. 2. 2016 [cit. 2017-04-21]. Internet of Things (IoT) History. Dostupné z: https://www.postscapes.com/internet-of-things-history/

PRESS, Gil. *Forbes.com* [online]. 18. 6. 2014 [cit. 2017-04-21]. A Very Short History Of The Internet Of Things. Dostupné z: https://www.forbes.com/sites/gilpress/2014/06/18/a-very-short-history-of-the-internet-of-things/2/#474c4d755530

Trac.ietf.org [online]. [cit. 2017-04-21]. FAQ. Dostupné z: https://trac.ietf.org/trac/hybi/wiki/FAQ

Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-04-21]. Node.js. Dostupné z: https://en.wikipedia.org/wiki/Node.js

10 PŘÍLOHY

Příloha A – *Zdrojový kód souboru CreateDeviceIdentity.js*

```
// JavaScript Document
'use strict';

 var iothub = require('azure-iothub');

 var connectionString = 'HostName=Bakala.azure-
devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=HPvkdOBEVFSkEGA
21pGXEKDXRGRjv2DlndYhUKyHOSE=';

 var registry = iothub.Registry.fromConnectionString(connectionString);

 var device = new iothub.Device(null);
 device.deviceId = 'myFirstNodeDevice';
 registry.create(device, function(err, deviceInfo, res) {
   if (err) {
     registry.get(device.deviceId, printDeviceInfo);
   }
   if (deviceInfo) {
     printDeviceInfo(err, deviceInfo, res)
   }
 });

 function printDeviceInfo(err, deviceInfo, res) {
   if (deviceInfo) {
     console.log('Device ID: ' + deviceInfo.deviceId);
     console.log('Device key: ' +
deviceInfo.authentication.symmetricKey.primaryKey);
   }
 }
```

Příloha B – *Zdrojový kód souboru ReadDeviceToCloudMessages.js*

```
// JavaScript Document
'use strict';

 var EventHubClient = require('azure-event-hubs').Client;

 var connectionString = 'HostName=Bakala.azure-
devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=HPvkdOBEVFSkEGA
21pGXEKDXRGRjv2DlndYhUKyHOSE=';

 var printError = function (err) {
   console.log(err.message);
 };

 var printMessage = function (message) {
   console.log('Message received: ');
   console.log(JSON.stringify(message.body));
   console.log('');
 };

 var client = EventHubClient.fromConnectionString(connectionString);
 client.open()
     .then(client.getPartitionIds.bind(client))
     .then(function (partitionIds) {
         return partitionIds.map(function (partitionId) {
             return client.createReceiver('$Default', partitionId, {
'startAfterTime' : Date.now()}).then(function(receiver) {
                 console.log('Created partition receiver: ' + partitionId)
                 receiver.on('errorReceived', printError);
                 receiver.on('message', printMessage);
             });
         });
     })
     .catch(printError);
```

Příloha C – *Zdrojový kód souboru SimulatedDevice.js*

```
// JavaScript Document
'use strict';

 var clientFromConnectionString = require('azure-iot-device-
mqtt').clientFromConnectionString;
 var Message = require('azure-iot-device').Message;

 var connectionString = 'HostName=Bakala.azure-
devices.net;DeviceId=myFirstNodeDevice;SharedAccessKey=YZySuJmHE/fbJ8NBvkGQ
sIxTZqUt6CC4tkgzwD7sSW8=';

 var client = clientFromConnectionString(connectionString);

 function printResultFor(op) {
   return function printResult(err, res) {
     if (err) console.log(op + ' error: ' + err.toString());
     if (res) console.log(op + ' status: ' + res.constructor.name);
   };
 }

 var connectCallback = function (err) {
   if (err) {
     console.log('Could not connect: ' + err);
   } else {
     console.log('Client connected');

     // Create a message and send it to the IoT Hub every second
     setInterval(function(){
         var windSpeed = 10 + (Math.random() * 4);
         var data = JSON.stringify({ deviceId: 'myFirstNodeDevice',
windSpeed: windSpeed });
         var message = new Message(data);
         console.log("Sending message: " + message.getData());
         client.sendEvent(message, printResultFor('send'));
     }, 1000);
   }
 };

 client.open(connectCallback);
```