Česká společnost uživatelů otevřených systémů EurOpen.CZ
Czech Open System Users' Group
www.europen.cz

50. konference

Sborník příspěvků

Statek Česká lípa, Myslovice
14.–17. 5. 2017

Programový výbor:

Zdeněk Šustr
Jan Kynčl
Jan Okrouhlý
Jiří Bořík
Jiří Sitera

Sborník příspěvků z 50. konference EurOpen.CZ, 14.–17. 5. 2017

Plzeň 2017. První vydání.
Editor: Vladimír Rudolf
Sazba a grafická úprava: Ing. Miloš Brejcha – Vydavatelský servis, Plzeň
e-mail: servis@vydavatelskyservis.cz

Tisk: Typos, tiskařské závody, s. r. o.
Podnikatelská 1160/14, Plzeň

Příspěvky neprošly redakční ani jazykovou úpravou.

ISBN 978-80-86583-32-7

Obsah

IP anycast

Pavel Poláček

E-mail: pavel.polacek@ujep.cz

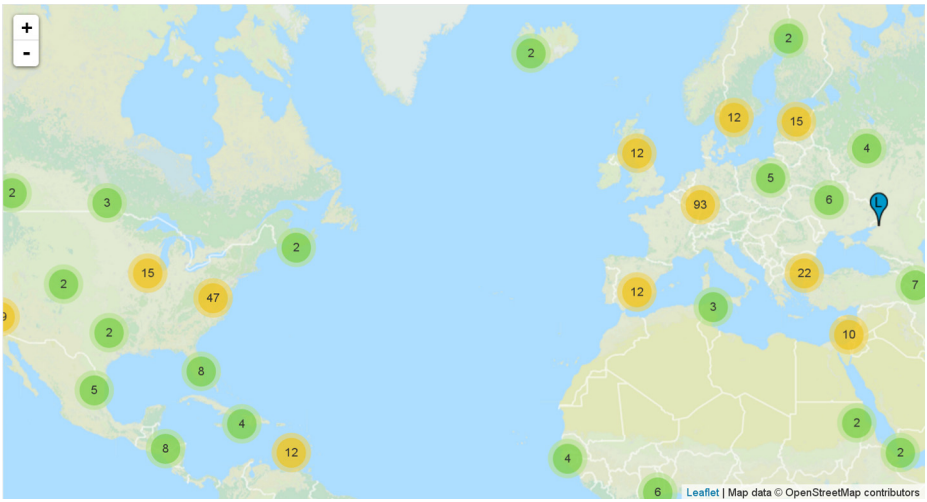
Broadcast, unicast, multicast a anycast. Ze všech druhů šíření paketů je IP anycast používán nejméně. Z toho důvodu může čtenářům vonět exotikou. Přitom samotný princip je prostý. Více uzlů v síti má kromě své vlastní IP unicastové adresy i jednu sdílenou IP adresu anycastovou. Směrovače posílají provoz na nejlepší cílový uzel dle metrik daného směrovacího protokolu. Tím IP anycast zajišťuje nejen vysokou dostupnost, ale je schopen i rozkládat zátěž mezi jednotlivé cílové uzly. Je zřejmé, že IP anycast je ideální pro bezestavové služby přenášené nespojovým protokolem jakým je například UDP.

Využití v praxi

Nejvíce se IP anycast používá pro rozklad zátěže mezi kořenovými DNS servery [1]. Za relativně krátkým statickým seznamem kořenových DNS serverů v konfiguraci rekurzivních resolverů je skryto poměrně velké množství fyzických strojů (viz Obr. 1). Velkou výhodou tohoto řešení je, že přidáním dalších pracovních uzlů škálujeme výkon a zároveň zvyšujeme dostupnost této největší distribuované databáze na světě. Vhodným rozmístěním uzlů blíže ke klientům zajistíme nízkou latenci. Cloudflare používá IP anycast k doručování obsahu ve své CDN [2]. V síti CESNET2 provozujeme službu WAYF/DS v rámci federace identit eduID.cz ve třech městech ČR právě nad IP anycastem (IPv4 i IPv6) [3].

Jednotlivé komponenty tutoriálu

Guláš si uvaříme ve VirtualBoxu. Pět virtuálních strojů s interním síťováním. Dva stroje budou mít úlohu směrovačů a zbývající tři budou poskytovat obsah. Quagga nám z Linuxu vytvoří směrovač, který se bude domlouvat BGP protokolem s ostatními uzly sítě. K tomu nám quagga přidá konfigurační rozhranní bližší síťářům (Cisco like). Na straně anycast uzlů využijeme webový server nginx pro servírování obsahu. Přidáme směrovacího daemona Bird, který bude vyjednávat BGP session s routery. Pak si ještě vytvoříme skript hlídající provozuschopnost služby a tím budeme řídit, zda uzel propagovat do směrovacích tabulek či nikoliv.



Obr. 1: Rozmístění kořenových DNS serverů

Cílový stav cvičení

Routery R1 a R2 spolu mají navázané BGP session a vyměňují si routovací tabulky ve svém autonomním systému (AS). Pracovní uzly W1, W2 a W3 poskytují webovou službu. Zároveň má každý uzel navázanou BGP session s routerem R1 nebo R2. Kontrolní skript zjišťuje, zda webová služba je v provozuschopném stavu. Pokud ano, je routovací daemon instruován k propagaci host routy ke svému BGP sousedovi. Pokud naopak v provozuschopném stavu není, je propagace host routy zakázána. Tím je zajištěno, že provoz jde pouze na "zdravé" uzly. Routery R1 a R2 akceptují od uzlů pouze anycastové host routy. Jelikož používáme službu nad TCP protokolem, tak pracovní uzly propagují host routy s rozdílnou metrikou. BGP protokol celkem pomalu konverguje, proto pro zrychlení konvergence využijeme Bidirectional Forwarding Detection (BFD).

Literatura

[1] http://www.root-servers.org

[2] https://blog.cloudflare.com/a-brief-anycast-primer/

[3] http://www.eduid.cz

Internet věcí do každé školy

Martin Malý

E-mail: maly@maly.cz

České školství má řadu neduhů. Je rigidní, jednou nohou stojí v dobách Marie Terezie, druhou opatrně našlapuje po nejmodernějších technikách, měnící se politické garnitury s ním zacházejí jako s hračkou, neustále přicházejí nové „pokyny“ a „koncepce“, které v konečném důsledku připomínají spíš antikoncepci...

O tom, že je potřeba změny, snad není nutno vést debaty. Absolventi mnohých škol připomínají spíš lexikon, než mladé lidi s kompetencemi pro život ve společnosti začátku 21. století. Naše generace, která chodila do školy převážně v 80. letech, na vlastní kůži zažila, že školní vědomosti v praxi byly skoro nepoužitelné, protože náš svět se během několika let proměnil před očima k nepoznání. Kdo se neučil dál, měl smůlu. Jenže právě to „učení dál“ byl mnohdy problém: škola totiž už tehdy selhávala v tom, aby v žácích vzbudila radost z učení, z poznávání nových věcí, ze zlepšování dovedností.

Nevím jak vaše škola, ale třeba ta naše fungovala jako plnicí linka: tady jsou studenti, my do nich nalijeme předepsané penzum vědomostí, a podle toho, kolik z toho v nich zůstalo, je roztřídíme do kvalitativních kategorií 1 až 5. Stokrát mohli opakovat, že se učíme pro sebe a pro život, ne pro známky, ale výuka tomu moc neodpovídala. Já třeba za sebe říkám, že jsem se naučil spoustu věcí, které jsem opět zapomněl a úspěšně se bez nich obejdu, zato jsem se nenaučil jiné, které bych užil mnohem víc. Třeba schopnost stoupnout si před lidi a mluvit – to jsem se učil až v dospělosti, bolestivě a zdlouhavě. Přitom by to nemělo být tak těžké...

Škola má dané věci, které musí žáky naučit. Většinou jsou formulované jako schopnosti, ale pod nimi jsou ukryté jen a pouze vědomosti: „Žák musí být schopen vyjmenovat alespoň tři meziválečné autory a pohovořit o jejich díle.“ Nezlobte se na mne, jediná schopnost, kterou se takto podaří otestovat, je schopnost „zapamatovat a reprodukovat“. Není tam nic o použití nabytých vědomostí, o schopnosti formulovat myšlenku, ... Jen „nauč se a reprodukuj“. V dobách průmyslové pásové výroby to byla jistě žádaná kompetence: naučit se sekvenci pohybů a tu dělat, k tomu se naučit i nějaké chybové stavy a adekvátní reakce... Jenže svět se změnil, a jestliže my jsme ve škole jen tušili, že třeba hodina týdně s počítačem nám dá mnohem víc praktických kompetencí než hodina větného

rozboru, tak dnešní studenti ten rozpor vidí dnes a denně. Náš svět je plný počítačů, internetu, komunikace – a škola se zmůže většinou jen na zákaz používání mobilních telefonů a na zákaz her ve školní počítačové učebně.

Já bych byl velmi nerad, kdyby to vyznělo, že mám něco proti všeobecnému přehledu. Naopak, já jsem velkým zastáncem toho, co se nazývá „všeobecný přehled"; není větší utrpení, než se bavit s člověkem, který zná „jen to svoje" a není schopen mluvit o čemkoli jiném, o světě, o životě, o knihách, hudbě, o jídle, o cestování, o historii... Je smutné potkat se s absolventem, co se považuje za vysoce kvalifikovaného odborníka v IT, ale nerozumí třeba rčení o arše Noemově. „Cože? To je z Bible? To já neznám, nejsem katolík!" Myslím si zkrátka, že všeobecný přehled je nezbytná součást vyzrálé osobnosti. Bohužel ale nefunguje v encyklopedické podobě, pouze tehdy, když vytváří kontext, propojený systém, ...

A jsme zase u toho: škola by neměla učit jen vědomosti, ale také dovednosti. Třeba schopnost spojit dvě oddělené informace, nalézat analogie, aplikovat znalosti na svět okolo sebe a na problémy... Už dlouho se hovoří o tom, že tyto schopnosti si člověk nejlépe natrénuje tehdy, když propojuje znalost s nějakým vlastním zážitkem. Když zůstaneme u aktuálního příkladu: v zadání maturitních témat bylo např. „vypracovat referát o národním obrození". Nepochybuju ani trošku o tom, že národní obrození hrálo zásadní roli při vzniku našeho státu, ale zkuste se vžít do kůže devatenáctiletého studenta a ruku na srdce – chce se vám psát referát o něčem, co bylo před dvěma sty lety a je to nudné a suché? A na druhou stranu: můžeme si myslet o youtuberech cokoli, třeba že to jsou nezajímaví frackové, ale neoddiskutovatelný fakt je, že mají mnohasettisícové publikum. Každý teenager je zná. Já chápu, že obrození je zásadní učivo, ale co takhle dát téma „referát o českých youtuberech"? A soustředit se ne na to, jestli si student plete Dobrovského s Dobruškou a správně si pamatuje „přehled", co byl v učebnici, ale na to, jestli umí pochopit, co se okolo něho děje teď a tady, vytvořit si o věci vlastní přehled, a ten pak srozumitelně formulovat.

Jenže něco takového připadá mnohým zodpovědným jako znesvěcení, znevážení, zlehčení tak důležitého aktu, jakým je maturita. Nezlobte se, ale v čem je znevážení? V tom, že se místo postav z národního obrození používají živé postavy? V tom, že by měl někdo referovat ne o vznešeném historickém dění, ale o něčem „pokleslém", co ale zná z vlastní zkušenosti?

Škola tak docela fatálně selhává, protože staví do protikladu „věci, co musíte znát" a „věci, co vás zajímají a baví". Mít padesátikorunu za každého učitele, který řekl něco jako „znalosti youtuberů vám nikdy k ničemu nebudou, ale kdo byl Dobner, to budete potřebovat vědět vždycky," tak jsem boháč. Vzpomínám si na svou učitelku, která mě nutila, abych psal krasopisně a posmívala se: „Ty si asi myslíš, že to za tebe bude psát počítač, ale to se mýlíš! K počítači se nejspíš nikdy nedostaneš, ale čitelné a úhledné písmo budeš potřebovat vždycky!" To bylo někdy v roce 1985. O dva roky později jsem začal psát úkoly na stroji,

o deset let později jsem přestal psát rukou úplně. Je těžké po takovém zážitku věřit učitelům, co budu a nebudu v životě potřebovat.

Situace je sice vážná, jak nás o tom přesvědčují nejen srovnávací testy, ale nikoli beznadějná. Potkávám se na cestách po republice s desítkami nadšených učitelů, a nepochybuju, že jich bude řádově víc, kteří učí děti opravdu rozumně a snaží se udržet krok s jejich světem, aby z něj mohli při výuce vycházet. Vidím na nich, že často ministerským konceptům a plánům navzdory dělají pro děti něco opravdu užitečného, zajímavého, a že je podporují v získávání kompetencí, které budou za pět, deset let opravdu potřeba.

Před několika lety se ministerstvo opět probudilo, rozhlédlo se kolem sebe a rozhodlo se, že je potřeba žáky víc připravit pro „informační společnost“ a „průmysl 4.0“. Do toho přišel velmi populární termín „Internet věcí“, anglicky Internet of Things (IoT). Samozřejmě, nejde o nic zásadně nového, stroje, co spolu komunikují, jsou na internetu od jeho počátku, ale v posledních letech se konečně setkalo několik faktorů, které umožnily raketový rozvoj oboru. Klesly ceny cloudů, internetové připojení se stalo ještě dostupnějším, a hlavně klesly ceny elektronických kitů a vývojových zařízení, které jsou dostupné amatérům.

Internet věcí na jedné straně táhnou velké firmy, které v něm vidí obchodní příležitosti. Na druhé straně ho postrkuje obrovská různorodá masa „bastlířů“, kutilů, kteří s pájkou a levnými čipy dělají jednu zajímavou věc vedle druhé. V dobách, kdy vývojový kit s malým jednočipem stál alespoň sto dolarů, se taková masa moc netvořila. Zlomilo se to po roce 2005.

V roce 2005 totiž vzniklo Arduino. Arduino je jednoduchý a levný kit, který obsahuje jednočip od Atmelu a rozhraní pro programování, a jinak nic. Cílem bylo vytvořit platformu pro studenty, kterou dokážou jednoduše použít a jednoduše naprogramovat. Platforma je otevřená (open-source), a i díky tomu vznikly desítky klonů, napodobenin, vylepšených verzí, ... Arduino opravdu proměnilo svět.

Nebyl to první kit takového typu. Existovaly i jiné, jenže se nikdy nedočkaly masovějšího rozšíření – snad s výjimkou BASIC Stamp. Arduino bylo otevřené, jednoduché, levné, dostupné, zároveň pro něj začaly vznikat standardizované periferie (shieldy) a vznikaly spousty návodů, článků, knih... Během několika let se Arduino stalo de facto hobbystickým standardem a vlastní „Arduino-like“ desky nabídli i velcí výrobci, včetně třeba Intelu. Úspěch Arduina spočíval v jeho variabilitě – jak poznamenal jeden novinář: „Arduino je jakékoli zapojení, co vás napadne. Chcete meteostanici? Automatizaci? Ovládání robota nebo 3D tiskárny? Na všechno zní odpověď: Arduino!“

Arduinu pomohla v rozšíření jeho jednoduchost nejen přímo, ale i trochu paradoxně: v diskusních fórech o mikroelektronice, kde do té doby diskutovali jen elektroinženýři z praxe, se začali objevovat začátečníci s Arduinem a kladli začátečnické dotazy. Staří ostřílení inženýři se k nim chovali velmi přezíravě a pohrdavě: oni to tolik let studovali a tak dlouho sbírali zkušenosti v praxi,

a najednou má nějaký matlal dojem, že si může s elektronikou hrát, a ještě chce radu? Díky tomuto vytlačení ze světa „kluků od mikroelektroniky, co spolu mluví“ si Arduino vytvořilo vlastní komunitu uživatelů, nadšenějších a vstřícnějších, kde jim nikdo pohrdavě nepsal, že Arduino je pitomá hračka pro vyhulené teenagery, a tak mohli v klidu udělat třeba knihovny pro téměř každou periferii, co se dá sehnat, a nabídnout je ostatním.

Nakonec i někteří profíci přišli na to, že Arduino je skvělá věc třeba pro rychlé prototypování. Než připravíte desku a naprogramujete firmware, to nějakou dobu zabere. S Arduinem použijete shield, co potřebujete, a za odpoledne postavíte prototyp, co bude fungovat.

Arduino se stalo i vhodnou platformou pro výuku. Je dostatečně jednoduché, abyste mohli jít hned k „hello world“ a nestrávili předtím tři hodiny teorií. Je dostatečně laciné (při nákupu v Číně pro osobní potřebu vás vyjde do deseti dolarů). A v neposlední řadě je rozšiřitelné, není to strop, ale můžete se s ním vyvíjet dál a dál, ke složitějším systémům.

S kolegou Štěpánem Bechynským jsme se rozhodli použít Arduino jako vhodnou platformu pro výuku základů práce s mikroelektronikou. O naše kurzy byl veliký zájem, absolventi byli nadšení, protože do té doby netušili, podle jejich slov, „jak je to jednoduché“, a opravdu se i něco naučili. Na kurzy chodili lidé mladí i starší, muži i ženy, a nám se potvrdilo, že s vhodnou prezentací lze udělat ten nejdůležitější krok, totiž vzbudit zájem.

Na Letní škole Czechitas, což je výukový kurz IT pro dívky 15–18 let, vzbudilo Arduino vždy velký zájem. Stačilo představit, ukázat, vysvětlit jak věci fungují, a najednou jsme koukali, s jakými nápady dívky přicházely. A nejen s nápady – dokázaly je i realizovat! Jejich reakce byly velmi povzbudivé – „je to skvělé, zajímavé, dají se z toho postavit fakt dobré věci, bavilo nás to, čekaly jsme, že to bude těžší. . . “

Tím se dostáváme na začátek: jeden z největších taháků, který přiváděl lidi na naše kurzy, byl právě Internet věcí. My jsme se mu sice nevěnovali přímo, ale dali jsme lidem dostatek znalostí a schopností, aby mohli k vlastním experimentům přistoupit. Viděli jsme, jak velký zájem o tuto oblast mezi lidmi je. Dálkové řízení, sběr dat, hlídání objektů, všechno tohle byly témata, co se pravidelně opakovaly v dotazech posluchačů.

Nemalé procento posluchačů byli učitelé, co jezdili a říkali: „Chci se to naučit, abych mohl pak s dětma zkoušet nějaké zajímavé věci, abych jim to mohl ukázat, protože si myslím, že tohleto je bude bavit!“ Nakonec jsme tedy kurz pozměnili a koncipovali tak, že učíme učitele, a zájem je obrovský.

Internet věcí totiž spojuje ty dvě oblasti, které škola, jak jsem už psal, trochu uměle staví do protikladu: je to něco, co je zábavné, zajímavé, co dokáže studenty nadchnout a podnítit k vlastním experimentům, a zároveň na tom lze učit širokou škálu dovedností, od programování přes práci s hardwarem až po některé vzdálenější obory, jako je ergonomie nebo design. Přitom náklady nejsou

nijak vysoké – jak Arduino, tak o něco vyspělejší Raspberry Pi, stojí jen několik stokorun.

Nadšení učitelé často nahlas počítali a přemýšleli, do jaké škatulky schovají nákup několika Arduin, popřípadě jak řediteli vysvětlí, že by měl uvolnit pět tisícovek na vybavení školního kroužku elektroniky.

Co víc si přát? Jsou tu nástroje, a jsou poměrně levné. Je k dispozici dostatek materiálů. Jsou i nadšení učitelé. Výsledek je zajímavý i pro děti, dá se s tím postavit leccos, co klidně využije i ty proklínané mobily... Připadá vám to příliš fantastické?

Po pravdě řečeno, mně taky. Když se ocitnu na nějaké „velké“ konferenci o „velkém IoT“, kde se střídají řečníci z nadnárodních firem a ministerstev a hovoří o smělých plánech, a v publiku sedí ředitelé notoricky známých „dodavatelů IT a IS pro státní správu“, mám pocit, že jsem v jiném světě a že výsledkem bude zase nějaký předražený nesmysl typu „aquapark v jedné z nejsušších vesnic“ nebo „cyklostezka z návsi doprostřed pole“. Nevěřím, že by se Internet věcí dostal do škol odsud, z nablýskaných akcí s hosteskami a prezenčkou plnou úctyhodných názvů firem a ministerských odborů, kde se ale promlátí tolik prázdné slámy, že by vystačila středně velké farmě na celou zimu... Ne, tady se nic zajímavého nestalo, a asi ani nestane.

Raději jdu z nazdobených sálů dolů, mezi ty, kterých se to týká. Mezi učitele a jejich studenty, kteří by chtěli zkusit dělat něco zajímavého, a intuitivně tuší, že právě Internet věcí bude za pár let nejen běžnou součástí našich životů, ale i tak otevřený obor s hladem po schopných lidech, jako je dnes programování.

Mladí nejsou dnes zásadně jiní než jsme byli my. Také intuitivně tuší, co ve škole má smysl a co je „učení pro učení“, jako jsme to tušili my. Hlavní rozdíl je ten, že mají mnohem snazší přístup k informacím, ten nejsnazší, jaký kdy kdo v dějinách měl. Ke všem. Škola by je měla učit s těmito informacemi pracovat a používat je, ne zakazovat, omezovat, zesměšňovat... Je tím sama proti sobě, protože jen posiluje ve studentech dojem, že je naprosto mimo realitu.

Internet věcí má obrovský potenciál, nejen v průmyslu a technologiích, ale i ve školství. Je to jedna z mála oblastí, kde vidím u studentů opravdový zájem a nadšení něco zkusit, něco udělat. Pojďme místo stesků na to, že „děcka zajímá jen mobil a internet a hry“, jejich zájem využít a podnítit. Ne, nemyslím si, že bychom měli před školama rozhazovat Arduina lopatou, to je nesmysl. Stačí povzbudit, nebránit, pomáhat, ...

Nasazení Internetu věcí do správy majetku

Karel Hřib, Adéla Mikschiková

E-mail: karel_hrib@cz.ibm.com, adela_mikschikova@cz.ibm.com

Abstrakt

An increasingly connected world facilitated by the Internet of Things (IoT) — an integrated network of devices, data, connections, processes and people — is transforming products, services and business models. The volume of data generated by the IoT provides a new means to understand and monitor the performance of these assets.

Sophisticated analytics and modeling technologies can now be applied to volumes of operational data. This includes data generated by manufacturing equipment and machinery, and during actual product usage in the field. These analytics provide organizations with a more detailed and accurate insight into process efficiency and product behavior.

Klíčová slova: Internet of things, predictive maintenance, asset management, quality, innovations, reduce costs

1 Internet věcí a správa majetku

Efektivní správa a údržba majetku je jednou z nejdůležitějších činností každého podniku. Správná údržba dokáže firmám snížit náklady na údržbu až o 25 %. Díky internetu věcí jsme schopni získat cenná data ze všech aktiv a zařízení ve firmě. Tyto data mohou poskytovat přehled o aktuálním stavu majetku. Všechna data z výrobního prostředí jsou tak v reálném čase na dosah ruky. V reálném čase jde také za pomoci Internetu věcí vidět anomálie například na výrobní lince.

Cílem Internetu věcí v průmyslovém prostředí může být například:

- Porozumět stavu zařízení a snížit náklady na údržbu.
- Predikovat možné poruchy zařízení.
- Zlepšit zákaznickou zkušenost.

2 Porozumění zdravotního stavu zařízní a majetku

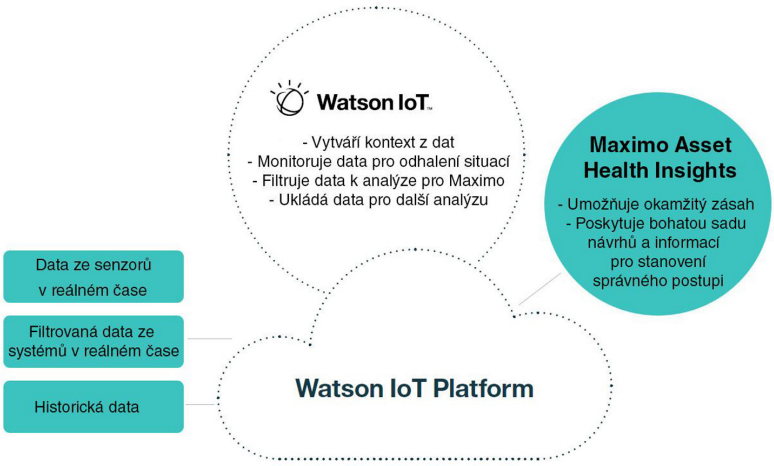
Studie ukazují, že skoro polovina preventivních investic na údržbu mají nulový vliv na provozu schopnost zařízení.

- 30 % z preventivních činností údržby se provádí příliš často.
- 40 % z preventivních nákladů na údržbu jsou vynaložené na opravách, které mají minimální vliv na provozu schopnost zařízení.
- 45 % veškerého úsilý na údržbu je neúčinné.

Novým efektivním způsobem údržby majetku je sledování „zdravotního stavu“ majetku. Propojení historických záznamů a informací s aktuálním stavem zařízení v reálném čase, s daty z externích datových systémů a s dalšími vlivy, jako je například počasí. Díky propojení těchto údajů má podnik možnost okamžitého zásahu a možnost proaktivně rozhodovat o údržbě majetku. Data získané ze senzorů v průběhu času poskytují lepší přehled o zařízení a hlubší informace potřebné k plánování a návrhu údržby.

K vyhodnocování „zdravotního stavu“ majetku podniku slouží **IBM Maximo Asset Health Insight** společně s **Watson IoT Platformou**.

Watson IoT Platforma je cloudová služba, která jednoduše a bezpečně propojuje senzory, čidla a další zřízení do cloudu. Data sesbírané v IoT Platformě jsou analyzována a následně odeslána do IBM Maximo Asset Health Insights, kde je zobrazeno celkové hodnocení a skóre zdravotního stavu aktiv.



Obr. 1: Propojení IBM Maximo Asset Health

3 Prediktivní údržba

Prediktivní údržba řízená obchodními analýzami umožňuje nahlédnout přímo do uživatelských systémů, mobilních zařízení, nebo kamkoliv na pracoviště. Příslušný software analyzuje různé typy dat, včetně dat o užívání, opotřebení a podmíněných charakteristických znacích, z různých zdrojů a aktivně rozpoznává vzorce poruch. Software zasílá tyto poznatky a optimalizovaná doporučená rozhodnutí přímo lidem s rozhodovacími pravomocemi, takže firmy mohou snížit provozní náklady, zlepšit produktivitu majetku a zvýšit efektivitu procesů. **IBM Predictive Maintenance and Quality** využívá prediktivní, preskriptivní a deskriptivní analýzu pro údržbu majetku a kvalitu produktu.

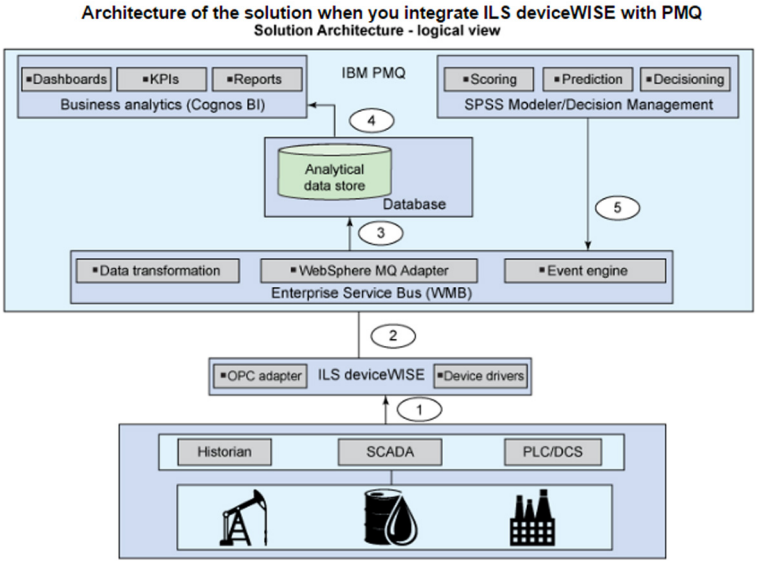
Jedním z příkladů může být plynárenská firma, která má množství točivých strojů, jako jsou turbíny, čerpadla, kompresory, generátory a motory. Každá plynová turbína má rotační kompresor, který generuje potrubní data, například sací a výtlačné tlaky, teploty plynu, okolní teploty a další. Data, která jsou přijímána z nízko-energických senzorů, slouží k analýze a sledování zdravotního stavu turbín. Data jsou zachycena v podobě událostí v reálném čase a poté odeslána do PMQ. V PMQ jsou vytvářeny tzv. značky. V našem případě přístroj zaznamenává 9 značek, jak můžete vidět v tabulce 1. Všechna data jsou poté v PMQ vyhodnocena a jsou navrhnuty opatření, které by měly nastat.

Tab. 1: Data sbíraná ze senzorů

Name	Definition	English unit	Metric unit
CTIM	Compressor inlet temperature	Fahrenheit (°F)	Celsius (°C)
CTD	Compressor discharge temperature	Fahrenheit (°F)	Celsius (°C)
CPR	Compressor pressure ratio	None	None
AFPCS	Compressor inlet pressure transducers 96CS	in H2O	mm H2O
CPD	Compressor discharge press max select	psi	bar
AFQ	Compressor inlet air mass flow	lb/s	kg/s
CompEff_Mean	Mean compressor efficiency	%	%
AFPAP	Barometric pressure transducers 96AP	in	mm
CMHUM	Specific humidity	Ratio	Ratio

Cíle prediktivní údržby a přístupu orientovaného na kvalitu jsou následující:

a) Předvídat údržbu majetku a problémy s kvalitou výrobku.
b) Snížit počet neplánovaných odstávek.
c) Vynakládat méně času na řešení problémů s výrobními stroji a majetkem firmy.
d) Zlepšit produktivitu majetku a kvalitu procesů.
e) Sledovat výkonnost zařízení v reálném čase a předvídat budoucí vývoj.
f) Rozpoznat problémy s kvalitou dříve než běžnými metodami kontroly kvality.



Obr. 2: Architektura prediktivní analytiky

4 Zlepšení vnímání zákazníka

Data nemusí sloužit pouze k určení toho co se děje ve výrobním procesu, ale také po něm. IBM ve spolupráci s Whirpool vytvořila řešení, kdy jsou domácí spotřebiče Whirpool osazeny inteligencí, která pomáhá zlepšovat zákaznický servis. Kognitivní analytika dokáže uživatelům zefektivnit používání zařízení a úsporu vody a energie.

Whirpool může sledovat, za pomoci dat uložených na cloud, jak zákazníci využívají zařízení a prostřednictvím služeb IBM Watson dále rozvíjet a inovovat na základě analýzy dat.

Hlavní snahou je také optimalizace produktu a více možností přizpůsobení produktu specifickým potřebám uživatelů. Kognitivní počítačové systémy tak poskytují podnikům možnost analyzovat většinu dat rychle a efektivně vytvářet pohledy na výrobek.

Další výhodou je zlepšení služeb pro zákazníky. Například když spotřebitel zavolá na zákaznickou podporu, zaměstnanec na ústředně hned vidí, jak bylo se zařízením zacházeno a jaká možná porucha mohla na základě údajů a dat nastat.

Inteligence v domácích spotřebičích přináší i výhody samotným uživatelům, kdy mohou ovládat spotřebiče na dálku pomocí aplikace.

Literatura

[1] Interní materiály, IBM Corporation, 2016.

[2] http://www.gartner.com

[3] Integrate IBM Predictive Maintenance and Quality (PMQ). In *IBM* [online]. [cit. 13. 9. 2016]. Dostupné z: http://www.ibm.com/developerworks/library/ba-pmq-devicewise/index.html

[4] *Whirlpool Corporation, IBM Collaborate on Cognitive Solutions for Connected Appliances* [online], 1 [cit. 13. 9. 2016]. Dostupné z: https://www-03.ibm.com/press/us/en/pressrelease/48762.wss

[5] *Understanding asset health with Maximo* [online], 1 [cit. 13. 9. 2016]. Dostupné z: https://www.ibm.com/blogs/internet-of-things/announcing-maximo-asset-health-insights/

Malý cloud i pro větší organizaci

Michal Švamberg, Jan Krčmář

E-mail: svamberg@civ.zcu.cz, honza801@civ.zcu.cz

Oproti dřívějšku je k dispozici velká škála virtualizačních technik, od jednoduchých na jednorázové pokusy až po komplexní prostředí. Na Západočeské univerzitě jsme od roku 2003 používali virtualizaci založenou na technologii Xen. Podařilo se nám úspokojivě vyřešit síťování, správa diskového prostoru, provoz i instalace. Jako nevýhodu lze spatřovat absenci intuitivního rozhraní pro koncové uživatele.

Měli jsme na výběr upravit současné řešení a přizpůsobit jej uživatelům nebo zkusit něco nového. Jako základ nového řešení se nabízelo virtualizační API libvirt. Backend k API jsme zvolili hypervisor KVM. Frontendů existuje celá řada, my jsme po delším hledání objevili webový projekt WebVirtCloud.

Stav k dubnu 2017 je takový, že máme dvě virtualizační technologie a připravujeme přesun virtuálů z Xen nad KVM. Také se do toho trochu přimíchala aktualizace distribuce Debian, což některé věci usměrnilo jinam, než jsme před více než rokem zamýšleli. O tom proč a jak jsme nové řešení doslova poskládali si můžete přečíst v následujícím textu.

Výchozí stav

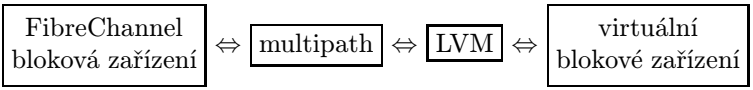
Podrobně popsanou virtualizaci jsme prezentovali na EurOpenu v Chudenicích v roce 2006[1]. Vyzkoušet si Xen naživo byla na jarním tutoriálu EurOpenu na Pradědu v roce 2009[2].

Pro představu výchozích podmínek je třeba uvést výchozí stav virtualizace. Jedná se spíše o virtualizační cluster. Odlehčenou verzi cloudu z toho teprve hodláme vytvořit.

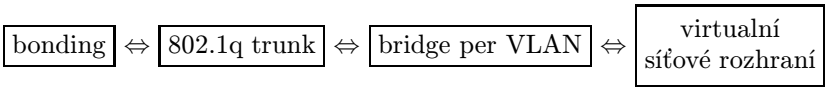
Základem naší virtualizace bylo použití Logic Volume Managera (LVM) pro správu disků nad diskovým polem připojeným přes FibreChannel. Po negativních zkušenostech s clusterovým LVM jsme přešli na ruční řízení, tj. po každé změně se pustil `lvscan` přes všechny uzly clusteru ručně. Pokud toto řešení používáte střídmě, tak je to to nejlepší, co pro stabilitu můžete udělat. Bohužel to

[1] http://www.europen.cz/Proceedings/28/xen.pdf
[2] http://europen.cz/Anot/34/HLAVNI.pdf

brání automatickému zakládaní volumů. Nevýhodou je také neefektivní využití prostoru, každému virtuálu se alokuje celý diskový prostor, nejen ten reálně zabraný. Provisioning diskového prostoru pak musí zvládnout blokové zařízení – tedy diskové pole, což v našem případě sice pole umělo, ale výsledek byl nevalný. Komunikace mezi jednotlivými vrstvami diskového subsystému probíhá takto:



Druhou důležitou infrastrukturní součástí virtuálních strojů je připojení k síti. Jednotlivé podsítě ve formě VLAN jsme měli přivedeny k hypervizorům přivedenou jako trunk 802.1q. Tyto podsítě se pak separovali na bridgích[3] v hypervizorovi. Časem jsme se naučili používat také bonding[4]. Schématicky lze síťovou komunikaci znázornit takto:



Celá konfigurace stroje, včetně přidělení paměti a procesorů, se vešla do deseti řádkového souboru. Fixní konfigurace je sice jednoduchá, ale má významnou nevýhodou v nemožnosti drobných zásahů vlastníkem virtuálního stroje, například nemůže změnit jádro virtuálního stroje bez zásahu správce hypervizoru.

Kromě výše uvedených nevýhod s použitím LVM jsme naráželi na neautomatizované zakládání virtuálních strojů a možnost jejich ovládání vlastníky. Příprava konfiguračních souborů (ikdyž velmi krátkých) vyžadovala oprávnění ke všem virtuálním strojům – nebylo možné to svěřit uživatelům jednotlivých virtuálů.

Výhled

Napřed jsme stanovili, co od nové virtualizace chceme. A když už začínáme znovu, tak toho chceme co nejvíce. Rozhodně chceme **přívětivé uživatelské rozhraní** se základním ovládáním stroje, jako je vytvoření, zrušení, vypnutí, zapnutí a restart. Vše další (konzole, snapshoty, ...) bude jako bonus. **Jednoduchou správu virtualizační infrastruktury**, která nebude vyžadovat složité zaškolení administrátorů. Vše by mělo běžet na **distribuci Debian**, protože s ní máme největší zkušenosti. Maximálně využít existující **centrální konfigurační management** a to jak u hypervizorů tak virtuálních strojů. Potřebujeme **škálovatelné řešení** v možnostech omezeného rozpočtu, lidské zdroje šetřit maximálně, investice do hardware ušetříme. Vybrat jen **používaná řešení**, které

[3] Jeden bridge = jedna podsíť = jedna VLAN.
[4] Redundantní síťové připojení.

mají naději dalšího udržitelného vývoje, tak abychom to nemuseli za chvíli celé překopat. **Zlepšit využití diskového prostoru** a nespoléhat se na služby diskového pole. **Redundantní řešení** tak aby bylo možno provádět postupnou údržbu či reagovat na výpadky.

Z některých částí jsme sice museli lehce slevit, ale jinde jsme přidali. Zhodnotili jsme nejčastěji používané virtualizační řešení. Nejde zcela o objektivní hodnocení, ale vysvětluje naši volbu:

- **OpenNebula, OpenStack** jsou příliš komplexní, vhodné nasadit v opravdu velkých instancích, vyžaduje detailní znalosti o celé technologii,
- **oVirt** je primárně pro RedHat Enterprise Linux, agent není připravený pro Debian,
- **mist.io** vyžaduje účet na `mist.io` a je to spíše pro velké cloudy,
- **Archipel** využívá XMPP pro komunikaci, potřeba XMPP serveru, nechtěli jsme záviset na další technologii,
- **WebVirtCloud** využívá libvirt, je rozumně malý a rozšiřitelný, což vyváží drobná omezení.

Teď je nutné celé řešení začít stavět a ptát se, zda daný dílek skládačky má pro nás přínos.

Diskový systém

Trvali jsme na škálovatelném řešení a největší potíž v takovém případě je s diskovým prostorem. Variantu blokových zařízení nad LVM libvirt ve verzi v Debian 8 nepodporuje. Museli bychom se vzdát snapshotů, ale také provisioningu a možnosti snadného způsobu změny velikost přiděleného prostoru virtuálnímu stroji. Proto jsme se poohlédli po nějakém vhodném clustrovém souborovém systému:

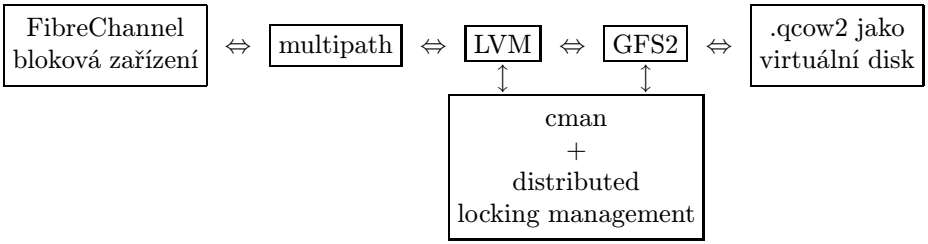
OCFS2 používáme pro Oracle servery, máme s ním dobré zkušenosti, ale občas (2x za rok) se zvláštně zasekne nebo nepohodne. Nikdy jsme nepřišli o data, což je spíše zásluhou žurnálu Oracle databáze. Při pokusu o nasazení se všechny nody clusteru zastavily v kernel panic.

GFS2 máme dobré zkušenosti při nasazení v RHEL clusteru v prostředí CentOS. V Debianu je někdy potřeba ručně pošťouchnout cluster při startu.

Nakonec jsme ponechali i LVM, které se sice zdá zbytečné, ale mělo by pomoci v případě migrací na jiné blokové zařízení. Tentokrát jsme se již odvážili do clusterové verze LVM, bohužel v Debianu je standardně `lvm` zkompilované bez podpory clusterovaných metadat a neumožňuje úplně všechny operace.

Pro potřeby GFS2 (a také kvůli cluster LVM) je nutné nakonfigurovat cluster manager cman. To umožňuje snadnou migraci mezi hypervizory a klonování strojů. Vzhledem k jeho verzi v Debian 8 je pravděpodobně zdrojem občasných potíží. Při aktualizaci na Debian Stretch (měl by vyjít koncem jara 2017) máme plán vyměnit `cman` za `pacemaker`.

Data virtuálního stroje jsou uložena v souboru formátu qcow2. To umožňuje vytvoření snapshotu a úsporu diskového prostoru. Používáme obrazy bez oddílů, takže je snadnější resize, ale musíme mít vlastní grub image, který načte konfiguraci bootloaderu z image virtuálního stroje. Jak vytvořit takový obraz je popsáno dále. Aktuální schéma používání diskového systému lze zakreslit:



Síť

Jako jediná součást zůstala z původního řešení. Pokud neuvažujete o virtuálních sítích, pak není třeba nasazovat technologie SDN. V DNS a DHCP systému jsme navíc vyhradili část IP rozsahu, který jsme předdefinovali spolu s hostname. Tento rozsah se pak používá pro nové stroje, které si mohou uživatelé vytvořit sami. Nemusí tak žádat o registraci v obou systémech a stroj jim funguje do minuty (podle velikosti image) od odeslání formuláře. WebVirtCloud jsme pak rozšířili o možnost vybírat právě z těchto hostname, podle kterého se vyplní i příslušná MAC. Tato „rezervace“ je určena pro stroje, které jsou opravdu jen na testy. Pokud chci virtální stroj s jiným jménem, pak je nutné počkat na propagaci DNS a DHCP registrace, jinak v tom není rozdíl.

WebVirtCloud

Je menší projekt[5], který využívá libvirt API[6]. Pro komunikaci s hypervizory podporuje všechny možnosti komunikace libvirt (TCP, SSH, socket). Koncept je jednoduchý, snadno rozšiřitelný. Je postaven na webovému frameworku Django. Rozhraní je přehledné a podporuje více uživatelských rolí. Navíc jsme rozšířili

[5] https://github.com/retspen/webvirtcloud
[6] https://libvirt.org/

projekt o uživatelské kvóty. Pro nás bylo důležité také napojení na již existující Single Sign On autentizační infrastrukturu. Framework Django obsahuje rozšíření pro externí autentizaci, takže se nám to podařilo velmi snadno.

Protože WebVirtCloud je postaven čistě na libvirt, pak lze v případě potíží použít přímo nástroje libvirtu pro nouzovou správu. Taktéž lze použít grafický nástroj `virt-manager`. WebVirtCloud si udržuje malou databázi, pokud se však provede nestandardní zásah mimo WebVirtCloud (přímo přes nástroje libvirt), je třeba tuto databázi ručně opravit.

Pro uživatele je velkou výhodou možnost přístupu ke stroji přes webovou konzoli. Dřívější řešení s touto možností nepočítalo a ani nebylo možné takto konzoli snadno zpřístupnit. Technologie zajišťující tuto cestu není odlišná od ostatních (i velkých) řešení. Jedná se o noVNC klient a novncd server.

Valná většina našich úprav, provedené ve WebVirtCloud, byla odeslána vývojářům a přijata.

Virtuální linux

V původním řešení jsme pro vytváření virtálních strojů používali systém FAI, který standardní konfiguraci Debianu upravil pro naše prostředí. Je to ale těžkopádná metoda instalace, která zabere čas na přípravu i samotnou instalaci. WebVirtCloud umožňuje instalaci z templatů, které se připraví standardním způsobem (v Debianu přes `debootstrap`). Disky jsou ve formátu `qcow2`, takže je nejde snadno připojit.[7]

Při prvním spuštění virtálního stroje se spustí skript, který zajistí základní nastavení systému. Jedná se především o resize souborového systému, nastavení hostname a spuštění konfiguračního managementu[8].

Pokud si uživatel vytvoří nový stroj, je třeba mu nějak umožnit se na něj přihlásit. Nechtěli jsme řešit přenos hesla z webového rozhraní. Implicitní heslo jsme nastavili prázdné. To zajišťuje možnost přihlášení z konzole a zároveň zakazuje SSH přístup. První přihlášení pro uživatele musí proběhnout přes webovou konzoli, kde si uživatel sám heslo nastaví. Potom už přihlášení přes SSH bude fungovat.

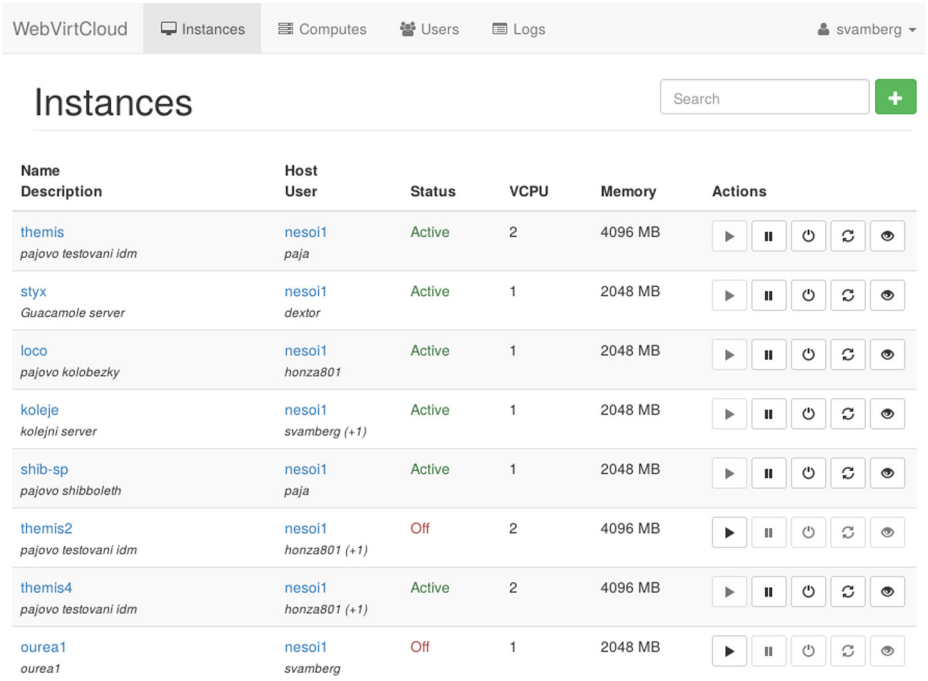
Příklad přípravy nového template

Spustíme nový virtuální stroj `debian9-template` a připojíme si naformátovaný virtuální disk

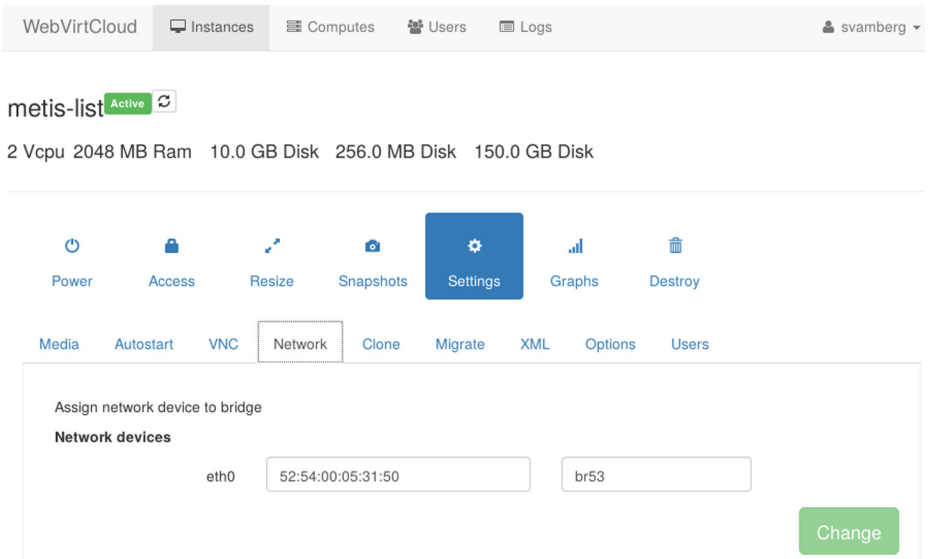
```
virt-rescue -d debian9-template --network
dhclient eth0
```

[7] Lze je připojit nástrojem `qemu-nbd`, nebo instalovat použitím nástroje `virt-rescue`.

[8] Jako konfigurační management používáme CFEngine3.



Obr. 1: Přehledová stránka administrátora s virtuálními stroji



Obr. 2: Správcovské rozhraní pro nastavení sítě virtuálního stroje

```
mkfs.xfs /dev/sda
mount /dev/sda /sysroot
```

Nyní můžeme instalovat základní systém

```
debootstrap stretch /sysroot ftp://ftp.zcu.cz/pub/linux/debian
```

Připravíme si prostředí pro konfiguraci nového virtálního stroje

```
mount -t proc none /sysroot/proc
mount -t sysfs none /sysroot/sys
mount /dev /sysroot/dev --bind
chroot /sysroot bash
```

Nastavíme vhodné repozitáře pro doinstalaci dalšího software

```
cat > /etc/apt/sources.list <EOF
deb http://ftp.zcu.cz/pub/linux/debian stretch main non-free contrib
deb http://ftp.zcu.cz/pub/linux/debian stretch-updates main contrib non-free
deb http://download.zcu.cz/public/software/linux/debian stable main
EOF
apt-get update
```

Doinstalujeme nezbytný software, připravíme zavaděč a smažeme heslo

```
apt-get -o Dpkg::Options::="--force-confdef" -y install grub2 openssh-server
      python perl xfsprogs
dpkg-reconfigure tzdata
update-grub
grub-install /dev/sda
passwd -d root
```

Nastavíme síťové rozhraní

```
cat > /etc/network/interfaces.d/ens3 <EOF
auto ens3
iface ens3 inet dhcp
EOF
```

Na závěr nakopírujeme `guest-init`, jehož spuštění přidáme do `/etc/rc.local`. Tento skript zajistí přizpůsobení stroje, až se spustí. Nyní je možné odpojit připojené svazky a ukončit virtuální rescue stroj. Vzniklý stroj nastavíme ve WebVirtCloud jako template. Template není možné spustit jako virtuální stroj, aniž by se napřed provedlo jeho klonování. Template nebo nový stroj lze vytvořit také z již existujícího virtuálního stroje.

Vytvoření zaváděcího obrazu

Protože nepoužíváme disky rozdělené na oddíly, je problém se zavedením systému. To lze obejít tak, že připravíme samostatný obraz pouze se základním GRUBem, který použijeme jako odrazový můstek pro boot základního systému.

```
grub-mkimage -O i386-pc -o grub.img --prefix="(hd0)/boot/grub" part_msdos ext2
      xfs biosdisk
```

Vytvořený obraz použijeme v nastavení virtuálního stroje a to v .xml souboru virtuálního stroje jako parametr `<kernel>/path/grub.img</kernel>`.

Další operační systémy

Pro Windows 7 nám kolegové od Windows připravili upravený obraz pro ZČU prostředí. Klonování Windows není plně automatizováno. Je třeba zařadit stroj do domény. Poté se lze na stroj připojit standardní ZČU identitou.

Máme připraven i obraz pro CentOS a FreeBSD. Výhradně pro testovací účely a bez konfiguračního managementu.

Poslední možností je prázdný virtuální stroj, ke kterému je možné připojit instalační ISO. Tímto způsobem lze nainstalovat téměř každý běžný operační systém.

Závěr

Nasazením nové virtualizace jsme získali především

- **snadnější správu** oproti původnímu řešení,
- částečnou **automatizaci** při vytváření nových strojů,
- lepší **podporu Windows** a dalších nelinuxových operačních systémů,
- **uživatelské rozhraní** pro přístup a správě virtuálních strojů.

Najdou se samozřejmě i zápory tohoto řešení. Problémy spatřujeme zvláště v částech:

- **cman je hodně zastaralý** a nefunguje na 100 %, plánujeme jej vyměnit za pacemaker,
- **LVM v Debian 8 chybí podpora migrace** v rámci PV, tu ale potřebujeme jen jednou za několik let, kdy se obměňuje diskové pole,
- **libvirt neumí správně zacházet s LVM a qcow2**, doufáme, že se to zlepší.

Posledním úkolem zůstává migrace virtálních strojů z původního řešení. V tomto případě předpokládáme pouze zkopírování dat původních strojů a nainstalování grubu.

Děkujeme Fondu Rozvoje sdružení CESNET, z. s. p. o.[9] za spolufinancování tohoto projektu vedeného pod číslem 571R1/2015[10].

[9] http://fondrozvoje.cesnet.cz/
[10] http://fondrozvoje.cesnet.cz/projekt.aspx?ID=571

O kontejnerech, cloudu, vesmíru a vůbec se zvláštním zřetelem na Kubernetes

Tomáš Kubica

E-mail: Tomas.Kubica@microsoft.com

Myslíte, že je čas podívat se kontejnerům na ozubená kolečka a nasadit vaši první aplikaci v kontejnerovém clusteru? Zkuste to v cloudu, je to nejrychlejší a nejpohodlnější cesta k tomu získat reálné zkušenosti z vývoje, deploymentu a provozu vaší aplikace v kontejnerech. V Azure můžete na kliknutí rozjet Kubernetes, Docker Swarm i Mesosphere DC/OS. Zkuste to.

Proč vůbec potřebujeme měnit způsob jakým nasazujeme a provozujeme aplikace?

Než se pustíme do kontejnerů a jejich orchestrátorů, pojďme se krátce zaměřit na situace, které nás dovedly až k potřebě na něčem takovém zapracovat.

Nekonzistentní prostředí vedou k chybám

Jednou z velmi častých příčin selhání v nasazování aplikací do produkce je nekonzistence vývojových, testovacích a produkčních prostředí. To je obvykle dáno tím, že se tato prostředí spravují z velké části ručně. Díky tomu se drží dlouhou dobu a „opraš ují“, tedy do hotové VM se postupně upgradují různé moduly a knihovny a spouštějí různé testovací sady. Postupně v takové VM vniká dost nepřehledná situace a nikdo vlastně přesně neví, co tam je za konkrétní verze různých součástek nebo zbytků po předchozích verzích, natož, aby se to vědělo přes všechna prostředí od vývojového přes testovací, předprodukční až po produkci.

Má to řešení? Dobrou odpovědí je plně automatizovat a použít strategii phoenix serverů. Ta spočívá v tom, že stav VM není určování klikáním, ale je součástí spustitelné dokumentace, tedy předpisu pro nějaký desired state nástroj jako je Chef, Puppet, Ansible, SaltStack nebo PowerShell DCS. Samotná infrastruktura je řešena jako kód a výsledná kombinace umožní při každé změně zlikvidovat původní infrastrukturu a vytvořit novou (proto název strategie phoenix servery). To může vést až na immutable servery, tedy situaci, kdy do samotné VM není žádný manuální zásah ani možný a po spuštění se nikdy nemění (jakákoli změna nebo instalace nové verze aplikace = nová infrastruktura). Nicméně to má i nepříjemné stránky a o tom je můj další bod.

Automatizace VM a OS je těžkopádná

Situaci umíme velmi dobře řešit, ale celé je to trochu těžkopádné. Obrazy OS jsou dost velké a špatně se s nimi manipuluje. Trvá jim, než nastartují a celkem vzato potřebujeme poměrně dost nástrojů pro rozhýbání toho všeho na úrovni softwarově definované infrastruktury, configuration management systémů a tak podobně.

Řešením by možná bylo něco, co bude menší, pružnější a bude od začátku navrženo pro plnou automatizaci.

Je hodně možností pro jednotku deploymentu, ale každá má mouchy

Co je „ta věc“, kterou nasazujeme do produkce? Co je jednotkou nasazení? Je to nějaká instalačka? Balíček v rámci OS nástroje pro package management? Archiv souborů (zip, tarbal)? Tyto klasické metody mohou mít trochu problém, když výchozí stav neodpovídá původním předpokladům (tedy je jednoduché to řešit „do čistého“, ale složitější do prostředí, které není plně pod kontrolou). Jde to, ale kolikrát se vám stalo, že to, co Dev dodal, nebylo Ops schopno rozumně rozchodit bez nutnosti to „poštelovat“? Jinak řečeno tento způsob řízení všech dependencies a podobných souvislostí není vždy stoprocentní.

Dobře, tak co použít hotové VM jako virtuální appliance, která se v Dev připraví a pak cestuje až do produkce? Tady narážíme na už zmíněné potíže s velikostí, těžkopádností a složitější automatizací (např. jak aplikaci sdělíte potřebné parametry jako je connection string do databáze – VM obvykle nemají žádný zabudovaný mechanismus, takže musíte řešení přidat extra, třeba cloud-init nebo napojovat aplikaci jako první krok do Etcd či Consul, což ale znamená aplikaci upravit).

Existují i moderní velmi příjemné postupy, třeba Habitat (by Chef), ale jde to řešit elegantně i předmětem tohoto článku, tedy kontejnery – ale nepředbíhejme.

DevOps bez Dev a DevOps bez Ops

Zažil jsem pár schůzek s lidmi z provozu (Ops) a v podstatě chtěli dělat DevOps, ale tak, aby u toho nemusel být Dev. Současně někdy vývojáři chtějí dělat DevOps bez Ops, hlavně aby o tom v provozu nevěděli, jinak to zablokují. To není udržitelné. Cesta od byznys nápadu přes jeho vývoj až po nasazení do produkce by měla být jedna trubka, jeden vývojovod, jeden proces, kde každý tahá stejným směrem a neustále se vrací častá a včasná zpětná vazba do Dev i k byznysu. O tom je DevOps a stávající Dev a Ops nástroje ke spolupráci možná neinspirují dostatečně.

Mikroslužby potřebují menší „servery“

V tomto článku se nechci věnovat podrobnostem konceptu mikroslužeb, ale rozhodně s tématem souvisí. Řeknu jen, že mikroslužby přináší velké výhody, ale ne zadarmo – vedou nás do světa distribuovaných aplikací, kde nás čekají nové efekty k řešení, odpadání součástek aplikace (místo „žuchnutí“ celého procesu se vším), nedostávání odpovědí na volání nebo latence a chybovost, problém stavu a jeho konzistence (v rámci CAP už nemůžeme ignorovat „P“, takže pokud chceme konzistenci, musíme se začít zabývat volbou lídra a tak podobně). Nicméně ty výhody možná stojí za to (ale ne vždy, byl bych tady poměrně pragmatický) a pak je tu jedna potíž. Mikroslužba nepotřebuje mnoho zdrojů, a tak bychom jich mohli do jednoho „prostoru“ dát hodně, nicméně v případě VM nám u každé mikroslužby bude sedět operační systém, který bude ukusovat CPU, paměť i disk. Potřebovali bychom něco jako deduplikaci paměti, ale to k dispozici úplně není.

Nebyla by nějaká možnost provozovat mikroslužby jako oddělené jednotky, ale nemít u každé jejich vlastní OS stack? Jasně, že byla.

Proč jsou kontejnery tak populární odpověď?

Jak bylo asi zřejmé, kontejnery jsou zajímavou odpovědí na některé z naznačených problémů. Kontejnerová technologie existuje už mnoho let. Jde o vlastnosti na úrovni operačního systému, které dokáží například vytvořit izolovaný strom procesů, síťový namespace (například možnost nevidět skutečnou síťovku) a Linux bridge či OVS, schopnost řídit přidělování CPU a RAM zdrojů nebo izoloce na úrovni souborového systému a vrstvený file system. Nic z toho není úplně nové, ale Docker to dokázal zapřáhnout všechno dohromady a dát nad to krásné a jednoduché API. Docker nevymyslel kontejnerové technologie, ale podnítil jejich masovou použitelnost. Dokonce do takové míry, že Microsoft ve Windows 2016 uvedl svoje kontejnerové technologie, které jsou pochopitelně z hlediska implementace jiné než prostředky používané v Linuxu, ale která nahoru má ono stejné populární Docker API.

Dnes tedy Windows kontejnery můžete spouštět ve Windows, Linux kontejnery v Linux a to všechno řídit z jednotného Docker API a výhledově zařadit obě platformy pod jeden orchestrátor kontejnerů. Navíc se objevují implementace, které vám umožní chovat se ke kontejnerů „kontejnerově“, ale přitom to spouštět v plně hardwarově asistované izolaci čili tak, jak to dělá hypervisor s VM (příkladem jsou Hyper-V kontejnery ve Windows 2016). V tomto kontextu tedy dokonce kontejnery nastavují normu moderního ovládání, kterou můžeme aplikovat i na svět VM. Zdá se tedy, že kontejnerové hnutí opravdu obohatilo IT velmi výrazně.

Kontejner je malý, šikovný a efektivní

Uvnitř kontejneru je jen to, co je specifické pro aplikaci, kterou hostuje. Vše ostatní si bere ze svého hostitele, tedy především OS kernel. Proto je malinkatý, startuje velmi rychle, dobře se s ním pracuje. Přestože stejně jako v případě VM můžete kontejner spustit, ručně tam něco nainstalovat a pak z něj vytvořit image, nástroje vás k tomu neinspirují. Jde to, ale na rozdíl od světa VM je tady normou kontejner „plnit“ zcela automaticky (např. Dockerfile nebo jiné prostředky jako jsou Cloud Foundry či Heroku buildpacky apod.). Kontejnery jsou zrozené pro automatizaci.

Kontejner je pomíjivý, je immutable, neoprašuje se

Původní stav kontejnerů byl takový, že v runtime nic není perzistentní. Nemá smysl si psát něco do lokálního souboru, všechno je stateless. Nemá smysl si poznamenat IP adresu uvnitř kontejneru, nehraje roli. Od té doby se mnoho změnilo (třeba skutečně perzistentní „volume“), ale základní koncept zůstává – kontejner je by design immutable. Předpokládá se, že vytvoříte kompletní image a ten spustíte tolikrát, kolik chcete, ale nepolezete do něj. Pokud kontejner po startu potřebuje získat nějaké informace (connection stringy, přístup do jiných API, service discovery) tak mu to řeknete přes Docker vstříknutím proměnné prostředí nebo se kontejner bude napojovat na nějakou service discovery službu s metadaty (v rámci orchestrátoru nebo vaší instanci Etcd, Consul apod.).

Nemáte tedy různé kontejnery pro dev, test a produkci. Je to stále totéž, stejný vnitřek co do verzí knihoven i aplikace, mění se maximálně jen vstříknuté informace (connection string apod.). Máte jeden kontejner co do jeho obsahu a z něj máte jeho instance běžící v různých prostředích. Co jste otestovali v test prostředí je tedy úplně totéž jako to, co pak poběží v produkci.

Kontejner je jednotkou deploymentu

V předchozím bodu jsem popsal to, co současně dělá z kontejneru jednotku deploymentu. Ta „věc“, která nás dostává z kódu do jeho nasazení může být kontejner. Je to jednotka, do které se automatizovaně dostane aplikace a všechno co pro svůj běh potřebuje (pokud vás teď napadlo jak, vydržte, to je přesně něco, co se dá lépe řešit na úrovni platformy nad orchestrátorem, třeba v OpenShift, Cloud Foundry, Deis Workflow). Tuto jednotku pak testujeme a nasazujeme. Mnoho výrobců dnes nabízí své nástroje či aplikace jako hotové kontejnery minimálně pro možnost si rychle řešení vyzkoušet. Například než řešit, jak nainstalovat, můžete si spustit Azure CLI, SQL Server pro Linux nebo .NET Core pro Linux jako kontejner. Pro Microsoft je to cesta jak vám rychle, bezbolestně a spolehlivě doručit funkční systém třeba na rychlé vyzkoušení.

Proč bez orchestrátoru je kontejner spíš hračka?

Co je skvělé na notebooku nemusí být dobré pro provoz

Na notebooku vývojáře to všechno vypadá nádherně, ale víte, jaký problém spuštění na větší infrastruktuře bude myslím největší ranou pro provoz? Networking. To, jak Docker začal, bylo vytvoření lokální sítě v rámci hostitele a veškerá externí komunikace se řešila jako překlad adres a portů na IP adresu hostitele. To je peklo. Služby běžící na nestandardních portech a s tím spojené problematické discovery (tradiční pojetí je najít si A záznam v DNS a to nestačí, buď musím koukat i na SRV nebo nasadit jiný discovery mechanismus jako je Etcd či Consul). Závislost na lokálním prostředí a „leaky abstractions“ s tím související (tedy žijete si v abstrahovaném světě, kde jednoduše aplikaci přiřadíte port a ono se to možná nepovede v závislosti na tom, na kterém hostiteli běžíte, protože už má ten port obsazený).

Fajn, řeknete si. Tohle je případ pro SDN overlay. Máte pravdu, ale kontejnery obvykle nasazujete do VM (přeci jen v rámci regulace a bezpečnosti chcete mít alespoň tematické clustery odděleny pořádně) a to je svět, ve kterém je také čím dál častější nasazování overlay. To vede na overlay přes overlay. Troubleshooting je noční můra, očekávejte problémy s MTU, penalizaci výkonu.

Zkrátka kontejnery a zejména jejich síťařinu je vhodné řešit nějak koordinovaně. Je to víc než skupina Docker hostitelů. Chceme takhle řešit celou skupinu, tedy cluster – a to je situace pro orchestrátor.

Kdo bude hrát Tetris?

Když mám jeden notebook, spouštím kontejnery na něm. Jednoduché. Když mám celý cluster, musím si vybrat, kde kontejner poběží. Mám to rozhodovat sám? Počet hostitelů i kontejnerů se mi může neustále měnit a já potřebuji aplikovat nějaké složitější politiky. Například říct, že chci kontejner ve třech kopiích a to tak, ať jsou na různých hostitelích. Pokud hostitel nebo kontejner umře, chci to vyřešit (tzn. pokud počet běžících kontejnerů neodpovídá požadovanému, chci to opravit automaticky). Možná budu chtít hostitele obsazovat způsobem, že chci napakovat co nejvíc kontejnerů do co nejmenšího počtu hostitelů. Nebo naopak chci kontejnery rovnoměrně rozprostřít přes všechny hostitele. V některých případech možná potřebuji nějaké složitější politiky – tento kontejner musí běžet někde, kde má pod sebou SSD, tenhle musí běžet na procesoru s vysokou frekvencí, tenhle tam, kde je 10G síťovka.

Orchestrátor obsahuje scheduler. Plánovač, který vyhledá toho správného hostitele pro váš kontejner na základě různých kritérií. Také bude monitorovat

stav hostitelů a kontejnerů a postará se o to, že váš kontejner běží v počtu instancí, které požadujete – pokud ne, sjedná nápravu bez vašeho zásahu.

Potřebuji balancovat provoz a služby se musí mezi sebou najít

Jeden kontejner může nabízet službu jinému. Jak se najdou? A co když hostitel selže a kontejner se znovu vytvoří někde jinde? Od orchestrátoru budeme potřebovat nějaké service discovery ideálně s možností získat metadata o kontejnerech. Typicky nabízí základní DNS službu a pokročilejší discovery spočívající v možnosti sáhnout do nějakého API (často orchestrátor pro sebe využívá Etcd a dá vám přístup k některým metadatům).

Druhá věc je, že z výkonnostních důvodů chci službu reprezentovat třeba třemi instancemi kontejneru. Jak to zařídit, aniž bych musel ručně řešit, kde která instance je a jak na ní budu balancovat provoz? Orchestrátory typicky mají mechanismus, jak to řešit, a to jak pro interní komunikaci (jedna sada kontejnerů k jiné sadě kontejnerů) tak pro externí (klient přistupuje na web). U té externí to bývá složitější, protože už se dostáváme do reálného světa a například v případě Kubernetes je možnost přímo z orchestrátoru ovládat infrastrukturní load balancer v Azure. Nezapomeňte, že orchestrátor možná dokáže přijmout provoz na kterémkoli nodu a balancovat to na správný kontejner, ale uživatelé musíte také balancovat na nody orchestrátoru. Dělat to ručně je nepohodlné a integrace Kubernetes s infrastrukturou tady dává obrovský smysl.

Jak budu nasazovat nové verze aplikace?

Pokud je kontejner immutable jednotkou deploymentu znamená to, že při nasazování nové verze aplikace půjde o nový kontejner. Potřeboval bych nějaký mechanismus, který dokáže mou mikroslužbu upgradovat za provozu. Tedy balancovat provoz na stávající verzi a postupně přidávat kontejnery s novou verzí, ty se starou postupně odebírat, až nakonec zbydou jen kontejnery s novou verzí. Takový automatizovaný postup obvykle orchestrátor nabízí.

Co orchestrátor nemá, a přesto to potřebuji aneb jak vypadá moderní PaaS?

Orchestrátory kontejnerů dávají obrovskou míru flexibility. Neříkají vám, jak je máte používat. To je někdy výhoda (můžete si to upravit pro svoje potřeby), ale mnohdy nevýhoda – třeba se vám hodí nějaký popisnější systém, který vás vede k cíli s využitím best practices a dává vám vyšší hodnotu. Jste tak schopni získat funkční aplikace rychleji a snadněji. Orchestrátor je jako psát v C, PaaS

jako psát v Ruby. To první je určitě velmi flexibilní a výsledky mohou být velmi optimalizované, ale kód je delší, méně lidí mu rozumí a trvá to déle napsat. Ruby vás navádí svým směrem, je jednoduché a intuitivní, rychle se dostanete k cíli, ten ale možná nebude tak dokonale optimalizovaný pro vaše potřeby. Oba přístupy mají velký smysl a záleží na konkrétním projektu co je pro vás důležitější.

Svět není stateless

Kontejnery mají svůj původ v systémech masivně distribuovaných postavených na eventuální konzistenci a stateless principech. Váš objednávkový systém ale není distribuovaný, chcete striktní konzistenci (transakční zpracování) a v principu je stavový. Ano, state lze externalizovat (třeba do Azure DocumentDB nebo Azure Redis), striktní transakce implementovat s využitím externí ACID databáze (třeba Azure SQL Database) a zbytek postavit jako distribuované stateless služby. Jde to, ale možná je to hodně práce, která nepřinese zas tak velkou užitnou hodnotu. Ti velcí to udělat musí, aby mohli škálovat, ale pro vaší stabilní bázi uživatelů to třeba nepotřebujete a proč pak dělat takové cvičení?

Jsou různé systémy, které se snaží se stavovými aplikacemi vypořádat. Už i některé orchestrátory něco málo v tomto dělají (Kubernetes). Dobrý příkladem platformy jako služba (PaaS), která řeší stav velmi elegantně, je Microsoft Service Fabric. Upřímně řečeno je zatím ideální pro C# a Windows svět, ale pracuje se intenzivně na jeho generalizaci pro Linux a jiné jazyky. Uvádím spíš pro zajímavost, že tento rámec nabízí programátorům přímo jednoduchý způsob, jak si mohou držet state v paměti (tradičními prostředky z pohledu programátora) a platforma to na pozadí pro ně řeší jako distribuovaný systém. Tedy tento state replikuje mezi členy a sama řeší problém konzistence, tedy volbu lídra a tak dále. To je příklad velmi elegantní pomoci s řešením problému stavovosti, který jinak musíte rozlousknout sami (třeba externalizací, což ale znamená větší latenci).

Kontejner je jako binárka, ale já chci automatický kompilátor a packager

Až dosud jsme předpokládali, že kontejner s aplikací tak nějak je. Orchestrátor si ho převezme a začne kouzlit. Kontejner je ale něco jako binárka, je to výsledek nějaké kompilace, kompletace, instalace knihoven apod. Jak se ze zdrojového kódu stane kontejner? To můžete vyřešit různými způsoby, ale právě platformy (PaaS) nabízí integrované řešení typicky ve formě nějakých build packů. Jak v případě (Pivotal) Cloud Foundry, (Red Hat) OpenShift tak (Microsoft) Deis Workflow dáváte platformě zdrojový kód a manifest obsahující potřebné informace o dependencies. Build pack pak zajistí všechno potřebné pro vznik kontejneru. Všechny tři platformy jsou open source a zejména OpenShift a Deis staví

nad Kubernetes orchestrátorem (Cloud Foundry typicky využívá svůj vlastní orchestrátor).

Existuje svět tam venku (API, DBaaS, BigData as a Service, ...)

Kontejner není kladivo na všechno. Velmi často budete chtít využít nějakou službu mimo samotný cluster – databázi jako služba, cache jako služba, kognitivní API a tak podobně. V takovém případě chcete například pro účely testování vytvořit v rámci spouštění kontejneru nějakou novou databázi a login do ní zprostředkovat kontejneru jako environmentální proměnnou. Provedete deployment a současně s tím vám nějaký service broker zajistí login do DB, token pro vaše API volání na předpověď počasí nebo cokoli takového. To samotné orchestrátory typicky neřeší, ale přitom je to žádoucí zejména pro Continuous Integration/Continuous Delivery scénáře. Například Cloud Foudry používá service broker API a Deis umožňuje stejné API napojit na Kubernetes.

Směřuje zmenšování jednotky nasazení přes kontejner až k funkcím jako služba (serverless)?

Možná vaše mikroslužba funguje tak, že reaguje nějakou akcí na konkrétní událost. Například když se objeví nový obrázek v objektové storage, provede jeho zmenšení do náhledu, který tam také uloží. Co kdyby tento kód neběžel trvale v nějakém zdroji typu VM nebo kontejner, ale byl spuštěn až v okamžiku, kdy je to potřeba? Nějaká platforma by se dozvěděla o příchozím triggeru a zajistila provisioning zdrojů pro spuštění vaší reakce. Bez serveru, VM nebo kontejneru, o kterém byste věděli a museli se o něj starat. To jsou mikroslužby dotažené do úplného konce a dobrým příkladem takového řešení může být Azure Functions. Stejně jako vždy si umím představit, že vaše aplikace nemusí nutně využívat jen jednu jedinou technologii. Pro frontend možná upřednostníte web jako službu (například Azure App Services), pro byznys logiku kontejnery v Kubernetes třeba v rámci Azure Container Service, pro perzistentní data třeba zvolíte NoSQL jako je Azure DocumentDB, využijete některé kognitivní služby jako je Cortana suite a na některé události budete reagovat s Azure Functions.

Rozhodnuto, chci orchestrátor. Jaký si mám vybrat?

Chcete orchestrátor? Výborně, na výběr jich je několik. Představme si tři nejpoužívanější.

Mesosphere DC/OS (Mesos + Marathon)

První možností je rámec Mesos, který vznikl ještě před Dockerem a jedná se o nesmírně robustní dvou-úrovňový scheduler, který třeba Twitter nasadil v neuvěřitelném množství nodů. Jeho specialitou je schopnost provozovat naprosté odlišné činnosti v jednom řešení. Nejčastěji jde o náročně Big Data workloady dohromady s byznys aplikacemi a kontejnery. V rámci Mesos pak provozujete tzv. frameworky, tedy třídy aplikací. DC/OS, postavený na Mesos, je vlastně operační systém pro celé datové centrum, který velmi precizně přiděluje zdroje aplikacím. Jeden z rámců je Marathon a to je právě orchestrátor pro kontejnery. Na tom jsou vidět ony dvě úrovně. Swarm, Kubernetes nebo Marathon jsou orchestrátory kontejnerů a Mesos je orchestrátor orchestrátorů (ostatně skutečně je technicky možné běžet Kubernetes uvnitř Mesos).

Složitosti stranou. Zvolte DC/OS pokud potřebujete něco, co má delší historii a je skutečně prověřeno životem. Také bych se zaměřil na DC/OS, pokud kromě kontejnerů chcete jedním šmahem vyřešit i Hadoop nebo Spark nad stejnými zdroji. Pokud jdete jen po kontejnerech, zvažte i další varianty.

Docker Swarm

Docker stojí za vším nadšením kolem kontejnerů, a tak není divu, že i v oblasti orchestrátorů chce hrát významnou roli. Po funkční stránce možná zatím nedosahuje schopností konkurentů (i když to se rychle mění), ale má jednu krásnou vlastnost. Pro jeho ovládání používá prakticky stejné API jako Docker na jediném stroji. Pokud už jste se naučili používat Docker na notebooku, pak prakticky stejné příkazy můžete použít pro celý cluster. To se může ukázat jako velká výhoda pro nenáročný přechod do produkčního nasazení.

Kubernetes

Pokud chcete kontejnery, je Kubernetes pravděpodobně nejdál. Některé jeho designové vlastnosti nemusí sednout každému (například koncept podů, tedy to, že jednotkou nasazení není kontejner, ale jeden a více kontejnerů v podu), ale myslím, že je dnes pro kontejnery nejlepší volbou. V čem jsou jeho výhody?

Kubernetes byl první kdo začal razit koncept přímé adresovatelnosti kontejnerů (tedy v jeho případě spíše podů), tedy IP per kontejner. K tomu se ostatní orchestrátory dopracovaly teprve nedávno, když všichni museli uznat, že to dává smysl. Síťově to můžete řešit různými způsoby a díky standardu CNI si mezi nimi lze vybírat. Možná skončíte u overlay přes overlay, ale v on-premise třeba bude lepší využít Calico a v případě Azure určitě CNI plugin právě pro toto prostředí, které zajistí nativní integraci s SDN implementací v Azure.

Druhá pro mě zásadní vlastnost také souvisí se sítí a tou je řešení problému příchozího provozu. Orchestrátory obvykle nabízí balancování a service discovery uvnitř svého světa, ale v něm uživatelé nejsou. Často pak orchestrátor nechá na vás, jak dostanete provoz uživatelů do clusteru a ještě vás potrápí s detaily implementace uvnitř clusteru (to je obvykle řešeno jako reverse proxy). Kubernetes nabízí schopnost orchestrátoru promluvit k infrastruktuře a dohodnout se. To je přesně i příklad provozu v Azure, kdy Kubernetes dokáže přímo mluvit k Azure Load Balanceru a pro vaši službu si vyžádat novou vnější adresu a zajistit balancování. Taková integrace je pro praxi naprosto zásadní.

Můj třetí důvod pro Kubernetes jako hlavní volbu je jeho pokračující snaha přestat předstírat, že všechno na světě je stateless. Pomalu začíná přidávat podporu pro stavové, řekněme tradičnější, aplikace. Loni bych vám tvrdil, že provozovat MySQL v kontejneru v produkci je nesmysl, ale dnes už bych se možná pomalu odvážil říct, že snad ok (nicméně – stále bych bez ohledu na to dal přednosta nějaké DB as a Service službě, kdy pro vás v cloudu databázi někdo vytvoří, zajistí HA a backup a tak podobně). Neřekl bych, že je hotovo, ale rozhodně jde Kubernetes správným směrem.

Předposlední argument pro tento orchestrátor vychází z toho, že byl první, kdo velmi dobře zpracoval desired state principy a především narolování nových verzí kontejnerů (tedy aplikací).

Je tu ještě jedna výhoda – nad Kubernetes se začínají stavět platformy. Projekt OpenShift (tažený firmou Red Hat) se před nějakou dobou transformoval do platformy nad Kubernetes. Deis je nativní řešení pro platformní služby nad Kubernetes a pro jeho výborné vlastnosti a zaměření Azure tímto směrem se Microsoft v dubnu 2017 rozhodl pro akvizici firmy aktivní v tomto open source projektu a chce akcelerovat jeho rozvoj (samozřejmě open source způsobem) a integrovat do Azure řešení. Slyšel jsem i o snahách rozjet Cloud Foundry nad Kubernetes místo proprietárního orchestrátoru Diego.

Proč má smysl provozovat kontejnery v Azure?

Proč a jak provozovat kontejnery v cloudu na příkladu Azure? První je to, že sestavit cluster bezpečně, spolehlivě a opakovatelně vyžaduje dost znalostí. Jeden ze zakladatelů Kubernetes, který se projektu věnoval ještě v Googlu před otevřením kódu komunitě, je nyní Microsoft zaměstnanec a se svým týmem automatizoval celý engine pro provisioning Kubernetes clusteru na kliknutí se všemi best practices (služba je součástí Azure Container Service, která podporuje kromě Kubernetes také DC/OS a Swarm, takže si můžete vybrat, kterou cestou se vydáte). Možná nevíte, jaký orchestrátor použít a jestli vám to bude dávat smysl. Celý cluster pro vás připravíme během několika minut – stačí říct jaký orchestrátor

a kolik a jak velkých nodů chcete. Teprve v ten okamžik začnete platit a jakmile cluster zrušíte, neplatíte ani korunu.

Druhý důvod pro kontejnery v cloudu je jejich nedostatečná izolace z pohledu různých regulací a certifikací. Enterprise nebude chtít provozovat aplikaci, která řeší objednávání obědů pro zaměstnance ve stejném clusteru, ve kterém jsou citlivé údaje o pacientech, protože izolace kontejnerů není zdaleka tak silná, jako v případě hypervisorů (kontejner sdílí kernel a na něm záleží, jestli je to opravdu bezpečné, zatímco v případě hypervisoru izolaci pomáhají například i speciální instrukce CPU, takže oddělení je realizováno i na úrovni železa). Pravděpodobně tedy budete potřebovat clusterů hned několik pro různé třídy aplikací nebo prostředí (test, dev, prod). Vytvářet a rušit cluster tak budete daleko častěji, než jednou za rok a plná automatizace ve službě jako je Azure Container Service tady dává velký smysl. Navíc je tu nákladová stránka věci.

Cluster má nějaké zdroje k dispozici. Může být velký nebo malý. Dokážete dopředu odhadnou kolik přesně zdrojů bude každý cluster potřebovat a zajistit, že vaše spotřeba bude konstantní v čase (tedy nikdy nebudete mít zbytečně moc ani nepříjemně málo). To se v praxi těžko dělá, takže schopnost na kliknutí přidat nebo odebrat nody clusteru (se všemi bezpečnostními záležitostmi typu nakopírování klíčů apod.) se ukazuje jako dost důležitá. Navíc v cloudu platíte jen za to, co opravdu spotřebujete. Klidně můžete některé clustery na noc vypnout a ráno je automaticky vytvořit znova (třeba v dev prostředí). Nebo můžete clusteru na noc nebo víkend ubrat některé nody a ušetřit. Nebo naopak vaše aplikace zaznamenala úspěch, blíží se Vánoce, marketingová kampaň nebo zátěžový test a vy potřebujete 10× víc výkonu? V Azure stačí kliknout a půjčit si víc... třeba jen na pár hodin.

Velmi zásadní je, aby byl Kubernetes integrován s podvozkem zejména s ohledem na síťovou kontektivitu, load balancing od uživatelů nebo perzistenci datového úložiště. Řada zákazníků není v on-premise připravena na takovou míru automatizace a softwarové definovanosti, aby se Kubernetes mohl pěkně napojit. Místo budování lokální IaaS jen pro to, abyste mohli nad tím efektivně provozovat kontejnery, můžete jít do Azure, kde už je to pro vás všechno vymyšlené, odladěné a podporované. Jak síťařina, tak balancing, perzistence a k dispozici máte i privátní repositář pro vaše image integrovaný s identitou v Active Directory (tedy skutečně jde o Enterprise řešení se vším všudy, žádné stahování neověřených imagů z internetu, vše máte plně pod kontrolou s integrací na Enterprise řízení identity).

Běhat kontejnery rovnou nad železem je pro internetové giganty, kteří v zásadě provozují jedinou aplikaci, ale v brutální škále. Věřím, že v Enterprise prostředí potřebujeme něco jiného – schopnost vytvářet, bořit a škálovat vícero clusterů s různou úrovní citlivost dat nebo prostředí a využít infrastrukturních konceptů IaaS pro bezpečný a flexibilní běh našich aplikací. To je důvod, proč jsem přesvědčen, že Kubernetes patří do cloudu a zejména do Azure.

Co přidat ke Kubernetes a získat tak moderní platformu jako služba pro vaše aplikace?

Na závěr bych se rád vrátil k tématu PaaS vs. orchestrátor. Jsem zastáncem toho, že ideální je použít nástroj, který vede k cíli tak, že můžu co nejdřív získat nějaké výhody, být efektivnější a rychlejší, a ne budovat nějakou platformu, která jednou třeba i k něčemu byznysu bude. Možná je k dispozici nějaká managed služba, která bude rychlejší a jednodušší, než se pokusit nacpat třeba databázi či identity systém do kontejneru. Možná můj webový frontend není mikroslužba a bude rychlejší a efektivnější použít cloudovou PaaS a její prostředky pro zajištění dostupnosti a efektivity. Zkrátka – použijte to, co dává smysl. Kubernetes, platformní služby nebo klidně i obyčejné VM. Vraťme se ale ke kontejnerům.

Rozhodnutí padlo, chcete kontejnery. Je lepší použít orchestrátor nebo nad ním nasadit nějaké PaaS řešení? Jste připraveni získat velkou flexibilitu, ale sami se poprat s některými výzvami jako je deployment aplikace ze zdrojáků, připojení na externí služby typu databáze nebo reverzní proxy, logování a integrace s CI/CD platformou? Pak za mne Kubernetes. Možná ale dává větší smysl nehrát si s technikáliemi a raději získat schopnost co nejdřív provozovat reálné aplikace. Ať vám něco poradí s tím, jak aplikace nasazovat, ať vám něco automatizuje překlopení aplikace ze zdrojáků do funkčního kontejneru, ať vám něco založí novou databázi a připojí vás na ni. V takovém případě, pokud chcete stále Kubernetes jako základní mechanismus řízení kontejnerů, bych šel do Red Hat OpenShift nebo nasadil nástroje z dílny Deis (Helm, Workflow, ...). Třeba to pro vaše prostředí vymyslíte lépe, než oni, ale třeba po roce práce zjistíte, že jste vyvinuli zase jen svůj klon OpenShift nebo Deis a zůstali na to sami.

Kontejnery v cloudu? To mi dává velký smysl. Microsoft do kontejnerů velmi investuje jak ve světě Linux, tak Windows. V Azure můžete provozovat tři open source orchestrátory zcela nativně a máte podporu pro PaaS kontejnerová řešení jako je Cloud Foundry, OpenShift nebo Deis. Klasiká webově orientovaná PaaS Azure App Services dnes podporuje deployment Linux kontejnerů a v Linux kontejneru snadno získáte třeba SQL pro Linux nebo .NET Core. Investujte čas vývoje a provozu aplikací v kontejnerech a získejte co nejdřív zkušenosti. Nejrychlejší a nejpohodlnější cesta k tomu vede přes Azure, vyzkoušejte to.

E-mailové reputační systémy

Ondřej Caletka

E-mail: Ondrej.Caletka@cesnet.cz

Elektronická pošta se často stává terčem nejrůznějších kampaní, šířících spam nebo phishing. Navíc obecně umožňuje komukoli vydávat se za kohokoli. V tomto příspěvku shrneme techniky, které se používají k prokazování autenticity zpráv a potažmo také možnosti, jak neautentické zprávy detekovat a filtrovat.

1 Sender Policy Framework

Principem SPF (původně zkratka znamenala *Sender Permitted From*) je dát možnost vlastníkovi DNS domény deklarovat, které servery jsou oprávněny odesílat e-mailové zprávy jménem domény. Příjemce pošty pak může validovat, zda poštu od dané domény dostává z autorizovaného serveru a podle toho zohlednit další nakládání se zprávou.

Majitel domény `example.com` může například specifikovat politiku, že poštu z adres `@example.com` mohou předávat pouze servery z adresního rozsahu 192.0.2.0/24 takovýmto DNS záznamem:

```
example.com   IN  TXT  "v=spf1 ip4:192.0.2.0/24 -all"
```

1.1 Pass, neutral, fail, softfail

Obsah `TXT` záznamu obsahuje vždy povinné záhlaví `v=spf1` následované mezerou oddělenými slovy, která definují povolené, nebo naopak zakázané rozsahy adres. Každé slovo může být uvozeno kvalifikátorem, jejichž význam shrnuje tabulka 1.

Tab. 1: Význam kvalifikátorů SPF

znak	název	význam
`+`	pass	daný rozsah adres vyhovuje politice (výchozí)
`?`	neutral	pro daný rozsah adres politika neexistuje
`-`	fail	daný rozsah adres nevyhovuje politice
∼	softfail	daný rozsah adres spíše nevyhovuje politice

Jako rozsah adres pak mohou být kromě výše uvedeného příkladu se slovy `all` pro všechny adresy a `ip4` pro daný rozsah adres použita také klíčová slova `a` a `mx`, která doplní IP adresu získanou DNS dotazem na příslušný typ doménového záznamu. Nejlepší současná praxe ale doporučuje se takovýmto zřetězením vyhnout za účelem snížení počtu DNS dotazů, které musí validátor vykonat.

Pravidla se při validaci vyhodnocují v pořadí, v jakém jsou zapsána, po nalezení první shody se další nezkoumají. Neexistující politika vrací stav *neutral*, je tedy ekvivalentní existenci záznamu:

```
example.com   IN  TXT  "v=spf1 ?all"
```

Příjemce pošty, který SPF záznam validuje, tak činí ideálně ještě v průběhu SMTP komunikace. Nejprve by měl pomocí SPF validovat jméno hostitele, kterým se představí klient v SMTP příkazu `HELO`/`EHLO`, následně validuje adresu domény, uvedenou v příkazu `MAIL FROM`. Pokud na základě vyhodnocení SPF rozhodnuto o nepřijetí zprávy, měla by být odmítnuta již během SMTP komunikace.

1.2 TXT nebo SPF?

SPF je jedním z protokolů, jehož vývoj probíhal ve spěchu a tak trochu mimo půdu IETF. Jedním z důsledků kvapného zavádění je způsob, jakým protokol SPF přikládá význam DNS záznamům typu `TXT`. Ty byly původně určeny pro nestrukturované textové informace. SPF těmto informacím přikládá speciální význam a dopouští se tak klasické chyby známé ve světě databází jako *metadata v datech*.

Na půdě IETF proto kdysi vznikla snaha situaci napravit zavedením nového typu DNS záznamu `SPF`, který bude mít stejný význam jako záznam typu `TXT`, bude však určen jen pro potřebu SPF. Tato snaha ovšem dopadla fiaskem; než se podpora pro nový typ DNS záznamu dostala do DNS serverů a jejich uživatelských rozhraní, bylo SPF již velmi rozvinuto. Validátory tak musely kontrolovat dva různé DNS záznamy a řešit případně rozpory. Přítrž tomu učinil nejnovější standard RFC 7208, který záznam typu `SPF` rezervuje pro budoucí revize standardu, zatímco pro aktuální verzi 1 nařizuje – v souladu se současnou praxí – použití záznamu typu `TXT`.

1.3 Problém s přeposíláním pošty

Mnohem zásadnější je ale koncepční problém s přeposíláním pošty na straně příjemce. Mějme příklad se třemi entitami:

- 𝔸 publikuje SPF politiku, obsahující tvrdé selhání pro nepovolené adresy

- $\mathbb{B}$ nemá s SPF nic společného, jen provozuje e-mailové schránky a nabízí svým uživatelům přesměrování pošty na jinou adresu

- $\mathbb{C}$ validuje SPF politiku a poštu, u které SPF selhává, tvrdě odmítá

Problém nastane ve chvíli, kdy uživatel schránky u $\mathbb{B}$ nastaví přesměrování všech nebo vybraných zpráv od $\mathbb{A}$ na adresu u $\mathbb{C}$. K $\mathbb{C}$ se totiž v okamžiku přeposílání připojí server entity $\mathbb{B}$ a bude se snažit doručit zprávu, která bude v SMTP komunikaci `MAIL FROM` označena jako přicházející od $\mathbb{A}$. To $\mathbb{C}$ vyhodnotí jako porušení politiky a zprávu odmítne.

Zpráva se tedy nedoručí a není to ničí vina. Každá z entit však může nějakým způsobem proces ovlivnit s cílem snížit riziko takového nedoručení. Entita $\mathbb{A}$ má možnost použít kvalifikátor *softfail* namísto *fail* (tedy zakončit definici politiky `~all` namísto `-all`). Dává tím najevo, že si sice nepřeje, aby jiné adresy odesílaly poštu jejím jménem, ale nepřeje si tvrdé odmítnutí takových zpráv. Stejně tak entita $\mathbb{C}$ může nastavit svůj systém tak, aby i zprávy, jejichž SPF kontrola selže, podrobila pouze důkladnější kontrole, ale neodmítala. Standard definující SPF totiž záměrně nechává volnost v tom, co se má se zprávou na základě výsledku kontroly stát.

I entita $\mathbb{B}$ může přeposílání upravit tak, aby k porušování SPF politiky nedocházelo. Zdaleka nejjednodušší, nikoli však správné řešení, je přeposílání zpráv s prázdnou obálkovou adresou odesílatele. Negativním efektem takového řešení je, že odesílatel původní zprávy $\mathbb{A}$ se nedozví o případných problémech s doručením zprávy od $\mathbb{B}$ k $\mathbb{C}$. Další, spíše teoretickou možností, kterou připouští SPF standard, je odmítnutí zprávy, jejíž přeposlání by porušilo SPF politiku, návratovým kódem `551`, `User not local` spolu s uvedením adresy, na kterou má být zpráva odeslána přímo původním odesílatelem.

1.4 Přepisování adresy odesílatele podle SRS

Poslední možností, jak může entita $\mathbb{B}$ zabránit porušování SPF politiky, je systematické přepisování obálkové adresy odesílatele během přeposílání tak, aby případné informace o průběhu doručování k $\mathbb{C}$ bylo možné předat zpět odesílateli. Nelze to však udělat jednoduchým přepisem; takové řešení by ze serveru $\mathbb{B}$ efektivně vytvářelo *open relay*, který by mohl být zneužit k rozesílání spamu. Metoda zvaná *Sender Rewriting Scheme* přepisované adresy proti zneužití zabezpečuje buď kryptograficky, nebo použitím databáze.

V prvním případě je pro odesílatele `user@A.org`, posílajícího zprávu na server `B.org` tímto serverem při přeposílání vytvořena následující obálková adresa odesílatele:

```
SRS0=HHH=TT=A.org=user@B.org
```

Písmena `TT` jsou nahrazena časovou značkou, kdy byla přepsaná adresa použita, písmena `HHH` pak jsou nahrazena částí hashe, na jehož vstupu je adresa odesílatele, časový kód a náhodné tajemství, známé pouze serverům, které generují a validují přepsané adresy.

Protože zpráva ze serveru entity $\mathbb{B}$ přichází s obálkovou adresou entity $\mathbb{B}$, nepředstavuje její doručení žádný problém v SPF. Pokud dojde k problému s doručením přeposlané zprávy, je informace o problému poslána na výše uvedenou speciální adresu. Server $\mathbb{B}$ ze speciální adresy extrahuje původní adresu `user@A.org`, ověří, zda souhlasí hash a zda od časové značky neuplynula příliš dlouhá doba, a pokud vše souhlasí, zprávu předá původnímu odesílateli.

Takovýto přepis má jistá úskalí, související především s maximální povolenou délkou uživatelské části e-mailové adresy, kterou RFC 5321 stanovuje na 64 znaků. Zejména pak v případě, kdy k přeposílání a SRS přepisu dojde víc než jedenkrát. Z toho důvodu je pro takové případy definován zjednodušený formát:

```
SRS1=KKK=B.org==HHH=TT=A.org=user@B2.org
```

Písmena `KKK` představují nový hash, který vytvoří server `B2.org`. Při třetím a dalším přeposlání se již formát přepsané adresy nemění – případné zprávy jsou přeposlány přímo serveru, který provedl první přepis a od něj původnímu odesílateli.

Jinou možností přepisu adres je použití databáze. Každá adresa odesílatele se uloží do databáze pod náhodně vygenerovaným klíčem. Adresa se pak přepíše do tvaru:

```
SRS0=<klíč>@B.org
```

Při doručení zprávy na takovou adresu se skutečná identita odesílatele zjistí z databáze. Položky se z databáze se po určité době vyčistí, aby nebylo možné jednou naučenou adresu používat trvale. Výhodou použití databáze je hlavně omezení problémů s délkou uživatelské části e-mailové adresy. Nevýhodou naopak nutnost databázi synchronizovat a sdílet mezi všemi poštovními servery organizace.

Je zřejmé, že SRS je vcelku komplexní služba, která navíc není nativně podporována většinou poštovních serverů. O jejím nasazení se však vyplatí uvažovat zejména u nejrůznějších školních či univerzitních schránek, kde je míra přeposílání pošty velká.

2 DomainKeys Identified Mail

Systém DKIM slouží zabezpečení obsahu e-mailových zpráv před změnou prostřednictvím elektronického podpisu. Od zavedených systémů jako jsou PGP

a S/MIME se liší především tím, že podepisování může provádět kterýkoli článek e-mailové infrastruktury, takže není nutné vyžadovat používání této technologie po uživatelích. Samotný podpis pak putuje se zprávou jako přídavná hlavička, například takováto:

```
DKIM-Signature: v=1; a=rsa-sha256; d=example.com; bh=3yZ...h2=;
 c=relaxed/relaxed; s=dep1; h=Date:From:To:Subject:From;
 b=Wo/qoyff...M0yBh5UqQ=
```

Hodnota `v=1` určuje verzi, `a=` použitý algoritmus podpisu, `d=` je identifikátorem domény, která podepisuje, `bh=` je hash těla zprávy a `b=` vlastní hodnota elektronického podpisu. Binární hodnoty jsou uloženy v kódování Base64.

K ověření podpisu se nepoužívá ani hierarchie veřejných klíčů, ani síť důvěry; veřejný klíč potřebný k ověření podpisu se jednoduše umístí do DNS v podobě `TXT` záznamu, opět v Base64 kódování. Ačkoli se zde také používá obecný typ záznamu `TXT`, kolizi s jinými účely se brání předepsaným prefixem `_domainkey`, za který je připojena doména, která má být zdrojem podpisu. Aby bylo možné klíče hladce měnit a/nebo používat různé klíče v různých částech organizace, je každý DNS záznam s klíčem uvozen tzv. *selektorem* (volba `s=`), což může být naprosto libovolný řetězec, splňující požadavky na platné DNS jméno.

2.1 Nejde o náhradu SPF

Ačkoli se zdá, že účel, ke kterému DKIM existuje, je podobný účelu SPF, není tomu tak. Připomeňme, že SPF řeší pouze autorizaci e-mailového serveru, či obálkové adresy odesílatele, která je předávána v průběhu SMTP komunikace. SPF nijak nezkoumá ani hlavičky, ani vlastní obsah e-mailové zprávy. Proti šíření nejrůznějších podvržených zpráv, sloužících nejčastěji phishingu, které mají v pořádku obálkovou adresu odesílatele, žádným způsobem nepomáhá.

DKIM naopak autorizuje **obsah zprávy**, zcela bez ohledu na to, jakým způsobem byla zpráva doručena. Samotný standard DKIM také na rozdíl od SPF nepředepisuje vůbec žádnou politiku, která by stanovovala, co se má stát se zprávami, jejichž elektronický podpis nevyhoví. Dokonce jeden z cílů DKIMu vyžaduje, aby zprávy s nevalidním DKIM podpisem byly hodnoceny stejně jako zprávy bez jakéhokoli podpisu. To umožňuje DKIM podepisování nasadit beze strachu ze zhoršení spolehlivosti e-mailového systému.

2.2 Kanonizace a přepodepisování hlaviček

Při konfiguraci podepisovacího softwaru jsou k dispozici volby, které by neměly uniknout administrátorově pozornosti. Tou první je kanonizace, tedy jakási předúprava hlaviček a těla zprávy před provedením vlastního elektronického pod-

pisu. Výchozí hodnota `c=simple/simple`, žádnou úpravu neprovádí a tak k poškození elektronického podpisu dojde i sémanticky nevýznamnou změnou hlaviček (úprava velikosti písmen názvů hlaviček) či obsahu (přidání či odebrání bílých mezer). Ačkoli asi panuje všeobecná shoda na tom, že systémy předávající poštu by takové úpravy dělat neměly, není od věci zabezpečit DKIM tak, aby i takto upravená zpráva byla validovatelná. K tomu slouží nastavení kanonizace na `c=relaxed/relaxed`. První hodnota se vztahuje k hlavičkám, druhá k tělu zprávy.

Další zajímavou volbou je vynucené podepsání některých hlaviček, byť ve zprávě nejsou přítomny. Vzhledem k tomu, že hlavičky se během cesty zprávy sítí přidávají, nepodepisuje DKIM všechny, ale pouze ty, které byly vyjmenovány ve volbě `h=` v samotném DKIM podpisu. Typicky bude obsahovat něco jako:

```
h=Date:From:To:Subject:From;
```

Vyskytuje-li se hlavička vícekrát, musí ji i metadata podpisu obsahovat vícekrát; při validaci se přibírají hlavičky postupně odspodu. Hlavičky, které nejsou v seznamu uvedeny, případně další výskyty uvedených hlaviček, nejsou do podpisu zahrnuty. Pokud v nastavení podepisovacího nástroje zapneme *přepodepisování* hlavičky `From`, bude do podpisu zahrnuta i neexistující druhá hlavička `From`. Díky tomu případný útočník nemůže do zprávy dodatečně vložit druhou hlavičku From, která by nebyla DKIMem kryta a mohla tak příjemce zprávy uvést v omyl, kdo je skutečným autorem zprávy. Stejný princip je možné použít například také pro hlavičku `Reply-To`, čímž lze útočníkovi zabránit nedetekovatelně odklonit odpovědi na zprávu na jinou adresu.

2.3 Politika podepisování ADSP

Samotný DKIM nijak nestanovuje, co se má stát se zprávami, které podepsané nejsou, ani to, jakou váhu má DKIM podpis od jiné domény než té, která je uvedena v adrese odesílatele. Aby mohl DKIM opravdu pomoci proti šíření podvržených zpráv, definuje RFC 5617 politiku *Author Domain Signing Practices*. Jedná se o další DNS záznam typu `TXT` na pevně dané subdoméně `_adsp._domainkey`, který může definovat jednu ze tří politik. Ty shrnuje tabulka 2.

Tab. 2: Přehled ADSP politik

název	**význam**
`dkim=unknown`	politika neexistuje (výchozí)
`dkim=all`	všechny zprávy mají být podepsány autorovou doménou
`dkim=discardable`	zprávy, jejichž validace selže, mají být zahozeny

Nejpřísnější politika `discardable` je určena výhradně pro domény, které odesílají strojem generované e-maily, u kterých je důležitý boj proti podvrženým zprávám, například automatická sdělení bankovních institucí, případně pro domény, které vůbec nejsou určeny pro e-mail. Tato volba **není určena** pro domény, ve kterých mají schránky uživatelé. To proto, že existuje několik málo legitimních služeb, případně služeb *na hraně legitimity*, které přísnou podmínku podepsání nesplní, ale je žádoucí, aby byly přesto doručeny. Takovou službou může být posílání e-pohlednic či článků e-mailem, nebo mnohem častěji e-mailové konference.

2.4 Problém ADSP s e-mailovými konferencemi

Elektronický podpis, který DKIM používá, chrání zprávu a několik jejích hlaviček před modifikací a je tedy neslučitelný s většinou systémů, které zprávy modifikují. Pokud tedy máte ve své síti antivir, který s oblibou do všech zpráv připíše *„Odchozí zpráva neobsahuje viry, zkontrolováno nejlepším antivirem na světě“*, případně vás právní oddělení donutilo ke každému e-mailu připsat *právní doložku* nutící nahodilé příjemce zprávu okamžitě smazat a všechno, co v ní bylo napsáno, zapomenout, je nutné takové systémy buď nepoužívat, nebo zařadit ještě před vytvoření DKIM podpisu.

Zařízení, která legitimně zprávu modifikují, jsou i e-mailové konference. U nich je žádoucí, aby zpráva přeposlaná od konference převzala identitu toho, kdo ji do konference odeslal. Zároveň ale konference zprávu často modifikuje způsobem, který není slučitelný s DKIM podpisem:

1. odstraňuje nevhodné přílohy
2. přidává patičku s informacemi o konferenci
3. přidává název konference do předmětu zprávy
4. přidává hlavičku `Reply-To` na adresu konference

Problém je poměrně obsáhlý a nemá jediné vždy správné řešení. Věnuje se mu celé RFC 6377, publikované též jako BCP 167. Jedním z navržených postupů, jak se má konference chovat k podepsaným zprávám, které modifikuje, je následující:

1. validovat všechny DKIM podpisy
2. výsledek validace uložit do hlavičky `Authentication-Results` při současném odstranení všech předchozích hlaviček tohoto typu
3. odstranit ze zprávy všechny hlavičky s DKIM podpisy
4. podepsat zprávu vlastním klíčem, do podepsaných hlaviček přitom zahrnout i hlavičku `Authentication-Results`

Konference, která zprávy modifikuje, by také měla odmítat příspěvky a pokusy o přihlášení od adres v doménách, jejichž ADSP politika je stanovena na `discardable`. Takové zprávy by totiž po modifikaci konferencí byly nejspíše odmítnuty, což by konference mohla nesprávně vyhodnotit jako nefunkční schránku a respondenta odhlásit.

3 Domain-based Message Authentication, Reporting, and Conformance

Standard zvaný jako DMARC byl publikován v RFC 7489 v březnu 2015. Představuje jakési zobecnění ADSP politiky tak, aby kromě DKIM pokrývala i SPF a to s jediným cílem: zaručit, že adresa v hlavičce `From` je autentická. Zkoumá se tedy, zda dopis přišel z IP adresy, kterou doména odesílatele označuje v SPF záznamu jako povolenou a zda souhlasí podpis DMARC z odesílatelovy, případně přidružené domény. Chování DMARC validátorů by také mělo být konzistentní – specifikace nabízí výrazně méně prostoru pro lokální nastavení. Z toho důvodu by uplatnění DMARC politiky mělo mít přednost před uplatňováním SPF či ADSP politik.

Standard přidává také možnost reportování, které umožňuje držiteli odesílající domény získat automatizovaně informace o porušení politiky. Administrátor tak může odhalit například zaměstnance, kteří v rozporu s firemní SPF politikou posílají poštu přes cizí SMTP server.

3.1 DMARC záznam

Politika DMARC je specifikována opět v DNS záznamu typu `TXT` na pevně dané subdoméně `_dmarc`. Standard předepisuje, že záznam musí být nejvýše jeden. Příklad minimalistického záznamu, který obsahuje obě povinné volby, ale nemá žádný význam, vypadá takto:

```
_dmarc.example.com. IN TXT "v=DMARC1; p=none"
```

Volba `v=` určuje verzi politiky. Volba `p=` určuje vlastní politiku. Kromě hodnoty `none` může nabývat také hodnot `quarantine`, pro umístnění neověřitelných zpráv do karantény a `reject` pro úplné odmítnutí zpráv, které neprošly SPF a DKIM autentizací. Subdomény mohou používat samostatnou politiku definovanou buď v nezávislém DNS záznamu, nebo ve volbě `sp=`.

Číselná volba `pct=` může obsahovat procento zpráv, které DMARC validátorem analyzováno – tím je možné v případě velkých provozovatelů politiku postupně zavádět. Volby `adkim=` a `aspf=` nastavují, jak přísně bude porovnávána doména použitá v DKIM, resp. SPF validaci s doménou v hlavičce `From`

e-mailové zprávy. Ve výchozím nastavení `r` je porovnávání uvolněné, vyhoví tedy i DKIM podpis/SPF záznam subdomény. Volbou `s` vynutíme striktní režim, kdy se doménové jméno musí shodovat přesně.

3.2 Hlášení o problémech

Pro nastavení reportování o problémech s validací slouží volby `rua=` a `ruf=`. První z nich slouží k odesílání agregovaných reportů, které typicky jednou denně shrnují počet úspěšně a neúspěšně ověřených zpráv, druhá slouží pro okamžité reportování každé neúspěšně doručované zprávy. Jediným standardizovaným prostředkem přenosu je prozatím e-mail. Hlášení chodí ve formátu XML, který je komprimován gzipem a přiložen k e-mailové zprávě. Držitel záznamu může při specifikaci omezit maximální velikost zpráv, třeba na 10 MB:

```
rua=mailto:dmarc-rua@example.com!10m
```

Aby se zabránilo nevyžádanému zaplavování cizích e-mailových schránek DMARC reporty, není úplně snadné posílat reporty na e-mailovou adresu v jiné doméně. V takovém případě odesílatel reportu ověří, zda dal adresát dané doméně souhlas. Pokud bude chtít například držitel domény `example.com` posílat hlášení na adresu v doméně `@example.org`, musí k tomu majitel domény `example.org` dát souhlas vystavením následujícího DNS záznamu:

```
example.com._report._dmarc.example.org. IN TXT "v=DMARC1"
```

3.3 Problém s e-mailovými konferencemi

DMARC využívá k ověření autenticity odesílatele jako SPF, tak i DKIM, přičemž k úspěšnému ověření stačí splnit alespoň jednu z podmínek. Tím se eliminuje problém s přeposíláním pošty, neboť taková pošta sice poruší SPF, ale nepoškodí DKIM podpis. Problém s e-mailovými konferencemi však zůstává. Taková konference sice neporušuje SPF pravidla, nicméně posílá zprávy z typicky zcela jiné domény, takže pozitivní výsledek SPF autentizace je pro DMARC nepoužitelný.

V zásadě jediné funkční řešení spočívá v úpravě nastavení konference tak, aby posílala e-maily ze své adresy a adresu skutečného respondenta vkládala do hlavičky `Reply-To`. Takové chování sice eliminuje porušení DMARC politik, ale zase činí problém při validaci elektronicky podepsané zprávy ať už pomocí S/MIME nebo PGP.

4 Shrnutí

S trochou nadsázky to s odstupem času vypadá, že standard SPF řeší zhruba stejné množství problémů, jako sám vytváří. Je to nejspíš důsledek jeho překot-

ného vývoje stylem *nejdřív nasadit – pak standardizovat.* Velcí e-mailoví hráči ho však vzali za svůj, a tak zbytku správců nezbývá než se přizpůsobit. Máte-li v plánu SPF zavést pro svoji doménu, začněte nejdříve průzkumem, kudy uživatelé e-mailových schránek odesílají svou poštu a případně nápravou špatného stavu. Stále totiž existuje nemalé procento lidí, kteří odesílají poštu pochybnými cestami začínajícími obvykle u SMTP serveru internetového poskytovatele. Situace se ale pomalu zlepšuje, i velcí freemailoví poskytovatelé nabízejí autentizované předávání pošty k tomu určenou službou Submission (TCP/587). Vzhledem k problému s přeposíláním rozhodně nelze doporučit zavádět tvrdé `-all` a doufat, že problém bude vyřešen na jiné straně spojení. Volba *softfail* se jeví jako nanejvýš vhodná.

Standard DKIM byl primárně vyvinut jako vylepšení stávajícího systému s důrazem na možnost postupného nasazení bez rizika poškození funkčnosti stávající e-mailové infrastruktury. Postupně zprávy podepsané DKIMem začaly být zvýhodňovány velkými hráči na poli elektronické pošty – Google například v DKIMem podepsaných e-mailech automaticky zobrazuje obrázky – což zvýšilo zájem o technologii, zejména mezi rozesílateli hromadné obchodní korespondence.

Politika ADSP, jejímž cílem je použít DKIM k vynucení jistého standardu v doručování zpráv, již tak úspěšná nebyla, zejména s ohledem na e-mailové konference. Dalšími problémy bylo ne zcela přesně specifikované očekávané chování příjemce zpráv a nemožnost detekovat, že k porušení politiky došlo.

Standard DMARC se snaží vyřešit neduhy jak SPF tak ADSP a přijít s řešením, které umožní nakládat konzistentně s e-maily, které falšují adresu odesílatele. Takové chování je však bohužel běžné u e-mailových konferencí, které je potřeba nastavit tak, aby nepřeposílaly zprávy s původní adresou uživatele v hlavičce `From`. Díky možnosti hlášení je také možné odhalit legitimní poštu, která by byla zahozena, ještě před skutečným uplatněním politiky.