Azure a životní cyklus aplikací

Bc. Pavel Balcárek

Diplomová práce
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2016/2017

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Pavel Balcárek**
Osobní číslo:	**A15217**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**kombinovaná**
Téma práce:	**Azure a životní cyklus aplikací**
Téma anglicky:	**Azure and Application Lifecycles**

Zásady pro vypracování:

1. **Nastudujte a v rámci teoretické části práce popište možnosti využití cloudového prostředí Azure při tzv. post-coding fázi vývoje aplikací na platformě .NET.**
2. **Shrňte požadavky na vícevrstvý demonstrační systém v prostředí platformy Azure a na jejich základě sestavte jeho kostru.**
3. **Navrhněte a implementujte postupy pro automatizaci procesu sestavení (build) a nasazení (deploy) aplikace.**
4. **Sestavte a implementujte základní sadu aplikačních testů.**
5. **Navrhněte úpravy v demonstračním systému pro širší využití v cloudovém prostředí.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **BRIGGS, Barry a Eduardo KASSNER. Enterprise Cloud Strategy. Redmond, Washington 98052-6399: Microsoft Press, 2016. ISBN 978-1-5093-0196-6.**
2. **COLLIER, Michael a Robin SHAHAN. Fundamentals of Azure: Second edition. 2.vydání. Redmond, Washington 98052-6399: Microsoft Press, 2016. ISBN 978-1-5093-0296-3.**
3. **RABELER, Carl. Migrating SQL Server Databases to Azure: Microsoft Azure Essential. Redmond, Washington 98052-6399: Microsoft Press, 2016. ISBN 978-1-5093-0292-5.**
4. **RAINEY, Rick. Azure Web Apps for Developers: Microsoft Azure Essentials. Redmond, Washington 98052-6399: Microsoft Press, 2015. ISBN 978-1-5093-0059-4.**
5. **MCKEOWN, Michael. Azure Automation: Microsoft Azure Essentials. Redmond, Washington 98052-6399: Microsoft Press, 2015. ISBN 978-0-7356-9815-4.**
6. **GUTHRIE, Scott, Mark SIMMS, Tom DYKSTRA, Rick ANDERSON a Mike WASSON. Building Cloud Apps with Microsoft Azure: Best practices for DevOps, data storage, high availability, and more. Redmond, Washington 98052-6399: Microsoft Press, 2014. ISBN 978-0-7356-9565-8.**
7. **TULLOCH, Mitch. Introducing Windows Azure: For IT Professionals. Redmond, Washington 98052-6399: Microsoft Press, 2013. ISBN 978-0-7356-8288-7.**

Vedoucí diplomové práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **3. února 2017**

Termín odevzdání diplomové práce: **16. května 2017**

Ve Zlíně dne 3. února 2017

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomové práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně 12.5.2017

.....

podpis autora

ABSTRAKT

Tato diplomová práce se zabývá životním cyklem enterprise systému v cloudovém prostředí Azure od firmy Microsoft s důrazem na post-coding fázi a automatizovaný přístup. V teoretické části jsou popsány fáze vývoje software a je představen koncept cloudových řešení. Dále jsou v ní popsány komponenty Azure využité v této práci. Praktická část obsahuje ukázkový systém upravený pro prostředí Azure a demonstrující jeho možnosti při konfiguraci, buildovacím a deployovacím procesu a následném testování včetně load a integration testů.

Klíčová slova: Azure, vícevrstvý systém, sestavení, nasazení, kontinuální dodávka

ABSTRACT

This Master's thesis deals with an application life cycle of an enterprise system in Azure cloud environment from Microsoft. Emphasis is placed on post-coding phase and an automated approach. The teoretical part describes phases of a software development and a concept of a cloud solution is introduced. It also describes the Azure components used in this work. Practical part contains a demonstration system modified for the Azure environment. On the demonstration system are presented its capabilities in configuration, build-up and deployment processes, and subsequent testing including load and integration testing.

Keywords: Azure, multilayer system, build, deploy, continuous delivery

Velmi rád bych na tomto místě poděkoval vedoucímu své práce, panu Ing. Radku Valovi, Ph.D., za obětavost, připomínky a cenné rady při vedení mé diplomové práce.

OBSAH

ÚVOD

V současné době informačních technologií je stále častěji skloňován pojem „cloud“. Tento termín určuje jakým způsobem využíváme hardware i software - obojí používáme jako službu. S trochou nadsázky lze nákup cloudových služeb přirovnat k objednávce jídla v restauraci. Prohlédneme si menu, vybereme na co máme chuť a objednáme. Pak již stačí jen chvíli počkat a můžeme si pochutnat na pokrmu, který jsme si vybrali. Nemusíme řešit vybavení kuchyně, učit se recept či nakupovat suroviny. S cloudovými službami je to podobné. Vybereme požadovanou cloudovou službu, zkontrolujeme cenu a během chvilky můžeme používat. Není třeba řešit výstavbu infrastruktury, nákup hardware či konfiguraci software. Proces, který by „svépomocí“ mohl trvat měsíce, je možné vměstnat do několika minut. Používání cloudových služeb by tedy mělo být především rychlé a snadné.

Jak je to ale v případě, že si nekupujeme již hotové řešení, ale nakoupené služby budeme chtít použít například pro hostování vlastní webové aplikace či jako databázové úložiště? Je možné naši webovou aplikaci nahrát na web také tak snadno jak šlo službu objednat? Jak je to s aktualizacemi takové webové aplikace? Je možno tento proces automatizovat? Je možné nahranou webovou aplikaci nějakým způsobem monitorovat? Na tyto a některé další otázky se pokouší tato diplomová práce najít odpověď.

Diplomová práce je rozdělena na dvě části - na teoretickou a praktickou. V teoretické části jsou nejprve uvedeny fáze životního cyklu aplikace a poté je proveden úvod do problematiky cloudových řešení. V rámci této problematiky jsou také představeni vybraní cloudový poskytovatelé. Větší pozornost je věnována cloudovému řešení *Azure* od společnosti *Microsoft*, které je použito v praktické části práce.

Praktická část čerpá z poznatků získaných v teoretické části, v rámci kterých nejprve dojde k navržení, implementování a testování vícevrstvého demonstračního systému. Na demonstračním systému jsou dále předvedeny dva způsoby nasazení. V prvním - plně manuálním způsobu jsou popsány kroky, které je nutno provést pro nasazení systému za pomoci webového rozhraní portálu *Azure* a vývojového prostředí *Visual Studio*. Druhý způsob poskytuje návod na automatizaci kroků prováděných v prvním způsobu. Tento postup by měl tvořit jeden z hlavních výstupu práce. Závěr praktické části je věnován rozšířením demonstračního systému pro širší využití v cloudovém prostředí.

I. TEORETICKÁ ČÁST

1 Životní cyklus aplikace

Aplikace prochází za svého života několika fázemi. Tyto fáze jsou souhrnně označovány jako *životní cyklus aplikace*. Tato kapitola tyto fáze popisuje a poskytuje základní přehled o jednotlivých krocích a o jejich očekávaných výstupech.

1.1 Zadání

Počáteční fází životního cyklu je *zadání*. Tato fáze obvykle bývá spuštěna myšlenkou využít k nějakému úkolu počítač. Zadání by mělo být zpracováno v písemné formě.

V této fázi dochází k zapojení zadavatele a analytika. Zadavatel v tomto případě zná podstatu problému a analytik pomocí vhodných otázek vytváří formální zadání. Této fázi je nutno věnovat zvláštní pozornosti a vyvarovat se nepřesnostem, protože zásadnější změny požadavků v pozdějších fázích jsou časově i finančně náročné. [1]

Formální zadání je tvořeno dvěma druhy požadavků:

- *funkční požadavky*,
- *nefunkční požadavky*. [1]

1.1.1 Funkční požadavky

Funkční požadavky jsou zaměřeny na *problémový obsah systému* - tedy velmi zjednodušeně řečeno na to, co by požadovaný systém měl umět řešit. Při sestavování funkčních požadavků jsou zadavateli kladeny otázky. Tyto otázky by měly přinést odpovědi na to *proč* je třeba tento systém vytvořit, *k čemu* má sloužit a *kdo* ho bude používat. Řeší také *vstupy* a *výstupy systému* (jaké informace bude systém uchovávat a jaké poskytovat), *okolí systému* (s kým systém komunikuje) a v neposlední řadě též jaké *funkce* bude plnit. Poslední zmiňované slouží jako podklad pro funkční analýzu. [1, 5]

1.1.2 Nefunkční požadavky

Druhá skupina požadavků, *nefunkční požadavky*, je zaměřena na podmínky a omezení řešení. Pomocí nich je možno specifikovat *priority výsledného systému*, *způsob řešení* či ostatní *vnější požadavky*.

Priority mohou udávat nároky na efektivitu, přenositelnost či rychlost odezvy systému. Způsob řešení umožní předepsat použitý programovací jazyk, formu dokumentace nebo použití standardů. Vnější požadavky zohledňují omezení legislativou, harmonogram vývoje či cenu systému. [1, 5]

1.2 Analýza

Pro úspěšnou implementaci systému, řešící daný problém, je nejprve třeba tento problém pochopit. To je možné díky fázi nazývající se *analýza*, jež se věnuje studiu problému za účelem jeho *poznání*, *popisu* a *modelování*. Na analýze se kromě analytiků podílí i zadavatel a v ideálním případě také budoucí uživatelé.

Výsledek analýzy může mít dvě podoby. Prvním z možných výsledků může být rozhodnutí systém *neimplementovat*. Součástí tohoto výsledku je i odůvodnění odmítnutí. Druhým, pozitivním výsledkem, je *schválení* implementace obsahující specifikace systému.

Specifikace systému je tvořena:

- cílem řešení,
- podrobnou dokumentací požadovaného cílového stavu (pro závěrečné porovnání splnění požadavků),
- důležitými parametry (harmonogram, cena, výkon).

Budoucí systém je podroben třem základním analýzám:

- datová,
- funkční,
- dynamická. [1]

1.2.1 Datová analýza

Tento typ analýzy je důležitý především pro systémy, jejichž podstatou je ukládání dat a jejich vyhledávání. Datová analýza spočívá ve výběru potřebných objektů a jejich atributů. Mezi těmito atributy se vyhledá funkční závislost a navrhne se struktura databáze.

Výsledkem této analýzy je:

- seznam typů entit a jejich atributů,
- úplný grafický tvar v podobě *Entity-Relationship diagramu*,
- datový slovník. [1, 5]

1.2.2 Funkční analýza

Po dokončení datové analýzy je provedena *funkční analýza*. Funkční analýza, jak již název napovídá, slouží k popisu funkcí systému - tedy operací, které lze provádět. Mezi tyto operace řadíme např. ukládání, mazání, vyhledávání či třídění. Podklady pro tuto analýzu jsou opět získány ze zadání. Z těchto podkladů je vytvořen *funkční model*. Tento model vyjadřuje logický sled a podstatu transformací údajů vstupujících do systému a ze systémů vystupujících.

Pro tvorbu funkčních modelů se používají dva přístupy:

- shora dolů (vnější pohled),
- zdola nahoru (vnitřní pohled). [1]

Shora dolů V tomto přístupu je nejprve vytvářena struktura a hierarchie funkcí systému. Tato struktura je následně zjemňována. Model vznikající tímto způsobem je vhodné zpracovat graficky, proto se při jeho tvorbě využívá především *diagramu datových toků* (Data Flow Diagram, DFD) a *diagramu případů užití* (Use Case Diagram, UCD). [1]

DFD grafickou formou reprezentuje jednotlivé funkce systému. Diagram je tvořen ze čtyř základních prvků:

- *proces* (process),
- *tok dat* (data flow),
- *sklad dat* (data store),
- *rozhraní* (interface).

Proces reprezentuje modelovanou funkci, může se jednat např. o aktivitu v systému či transformaci dat. Tok dat slouží k výměně informací mezi procesy. Data jsou ukládána v datovém skladu. Rozhraní zastupuje např. externího uživatele či jinou část systému. [2]

Diagramy případů užití taktéž graficky popisují systém, ale na rozdíl od DFD se na funkce dívají z pohledu uživatele (aktéra). Tento diagram je využíván především při popisu objektově orientovaných systémů. UCD využívá k popisu grafický jazyk *Unified Modeling Language* (UML). UCD je taktéž složeno ze čtyř základních prvků:

- *aktér* (actor),
- *případ užití* (use cases),

- *relace* (associations),
- *hranice systému* (system boundary). [3]

Zdola nahoru Zde jsou popsány jednotlivé popisy procesů na nejnižší úrovni. Jednotlivé algoritmy procesů se zapisují stylem „*IF... THEN...*“, tedy: „*pokud platí něco, proveď následující*“. Funkce na vyšších úrovních se nepopisují, protože jsou složeny z funkcí nižší úrovně.

K zápisu je možno použít přirozený jazyk, častěji jsou však využívány formálnější nástroje (např. strukturovaný jazyk). [1]

1.2.3 Dynamická analýza

Předchozí typy analýz sloužily k popisu dat a k jejich transformaci. Z nich však nelze nic usoudit o časové návaznosti. Tento nedostatek řeší poslední typ analýzy - *dynamická analýza*. Dynamická analýza modeluje časové návaznosti v závislosti na čase nebo na pořadí funkcí.

K popisu chování se používá *stavový diagram* (State Transition Diagram, STD). STD zaznamenává chování systému vzhledem k působení vnějších událostí nebo na základně změn vnitřních stavů. Stavem se rozumí podmnožina hodnot atributů objektů. Chování objektů je závislé na stavu ve kterém se nachází. [1]

1.3 Návrh implementace

Výsledek analýzy tvoří modely systému popisující data a operace s nimy prováděné. Tyto modely jsou doposud nezávislé na konkrétní implementaci. V této fázi životního cyklu tak dochází k řešení implementačních otázek. Dochází nejen k výběru vhodného hardware (HW) a software (SW), ale jsou řešeny také další otázky týkající se např. organizačních, ekonomických či časových záležitostí.

Návrh implementace se dělí na dvě části:

- *systémový návrh* a
- *vlastní návrh implementace.*

Systémový návrh má za úkol řešit požadavky na výběr HW a SW komponent, prioritizaci požadavků, harmonogram zpracování, rozdělení na funkční celky (moduly), zařazení mezi okolní systémy a návrh použité architektury. Systémový návrh musí brát v úvahu také splnění legislativních náležitostí. V této fázi je také stanovena cena a dochází k uzavření smlouvy.

V druhé části dochází k vlastnímu návrhu implementace, někdy též nazývaným jako *návrh funkčních modulů*. V tomto kroku dochází mimo jiné k upřesnění a optimalizaci algoritmů, kontroluje se rozpoznatelnost stavů získaných z dynamické analýzy či se analyzuje podobnost funkcí. [1]

1.4 Implementace

Teprve v této fázi dochází k vlastní implementaci za pomoci zvoleného programovacího jazyka. Systém je programován po jednotlivých modulech tak, jak byl rozdělen v předchozí fázi.

V této části též dochází k tvorbě dokumentace. Tuto dokumentaci dělíme na *uživatelskou* a *programátorskou* dokumentaci. Uživatelská dokumentace by měla obsahovat specifikace zadání a základní logickou strukturu systému. Měla by též obsahovat požadavky na HW a SW, návod k instalaci, popis jednotlivých funkcí, spuštění a ukončení systému. Programátorská dokumentace naopak popisuje implementační detaily. Součástí dokumentace jsou též výstupy z předchozích fází - specifikace zadání, analýza systému a návrh implementace. [1]

1.5 Testování, validace, verifikace

Tato fáze probíhá zároveň s implementací, což umožňuje včas nalézt chyby nebo případné odchýlení od daných požadavků. V této fázi se provádí tzv. kontrola správnosti. Pod tímto termínem chápána:

- *Validace*. Systém odpovídá požadavkům zákazníka.
- *Verifikace*. Systém správně implementuje specifické funkce.
- *Testování*. Ověření pomocí konečné sady testů, že systém neobsahuje chyby. [1]

Pro provádění testů je vytvářeno *testovací prostředí*, které je naplněno testovacími daty. Testovací prostředí je obdobou produkčního prostředí. Pokud testovaný systém komunikuje s okolními systémy, je třeba v průběhu testování zajistit jejich dostupnost. Pro úspěšné testování je nutné mít připravena vhodná testovací data. Data mohou být vytvořena různými způsoby. Prvním způsobem je kopie z existujícího prostředí. V tomto případě je nutno brát v potaz anonymizaci dat a také, že tato data nemusí obsahovat dostatečnou variabilitu - nemusí být pokryty některé mezní případy. Další možností je vytvoření syntetických dat přímo v testovaném systému. V praxi je nejčastěji používána kombinace těchto přístupů. [4]

Dále jsou uvedeny některé běžné typy testů, které je v této fázi možno provádět.

1.5.1 Jednotkové testy

Jednotkové testy (unit testy) patří k těm nejzákladnějším. Jejich úkolem je testovat základní bloky programu - jednotky, někdy též moduly. Tyto jednotky by měly být co nejmenší. U objektově orientovaných programovacích jazyků se testují metody, případně třídy, u procedurálních testujeme funkce či procedury. Účelem unit testů je prokázat správné chování testovaných jednotek.

Každá jednotka musí být testována nezávisle na ostatních, v jiném případě by mohlo dojít k ovlivnění testu. V reálném prostředí jsou ale jednotky mezi sebou provázané, z toho důvodu se využívají tzv. *mocks* nebo *stubs*. Tyto „zástupné“ objekty simulují skutečné objekty a umožňují tak testy provést.

Princip samotných testů je jednoduchý - testovaným jednotkám jsou poskytovány vstupy a výsledek je porovnáván oproti očekávaným výstupům. Odklonění od těchto hodnot znamená chybu. Vstupy je třeba volit s dostatečnou variabilitou, aby test pokryl co nejvíce případů.

Testování jednotek je výhodné především díky možnosti automatizace a faktu, že zároveň slouží jako *dynamická dokumentace funkčnosti kódu* - při funkční změně jednotky je nutno též pozměnit test. [1, 5]

1.5.2 Integrační testy

V jednotkových testech dochází k testování jednotlivých modulů. Pokud jsou tyto moduly unit testy označeny za správné, je možno postoupit k další fázi - k *integračním testům*. Tyto testy mají za úkol ověřit chování systému jako celku, tedy vzájemné propojení a interakci modulů. Jinak řečeno mají za úkol ověřit, že je systém dostatečně stabilní.

Systémovou integraci lze provádět různými přístupy. Mezi ty nejběžnější patří:

- *Velký třesk.* Všechny moduly jsou integrovány zároveň, což znamená snadnou implementaci, ale zároveň s sebou přináší podstatnou nevýhodu - při výskytu chyby je velmi obtížné najít původce problému, protože se může vyskytovat kdekoliv.
- *Přístup shora-dolů.* Z hlediska hierarchické struktury systému začínáme přidáním nejvýše postaveného modulu a postupným integrováním nižších.
- *Přístup zdola-nahoru.* Nejprve nejnižší moduly, až poté přidávání vyšších.
- *Kombinovaný - sendvičový přístup.* Kombinace shora-dolů a zdola-nahoru ve třech úrovních - na nejvyšší úrovni integrace hlavních modulů přístupem shora-dolů, na spodní úrovni často používané přístupem zdola-nahoru a zbytek je umístěn do prostřední úrovně.

Integrační testy se dělí na:

- *Intrasystémové.* Testují se integrace modulů v rámci jednoho systému.
- *Intersystémové.* Testuje se integrace dvou a více systémů. [1, 5]

1.5.3 Systémové testování

Systémové testování má za úkol ověřit, zda systém splňuje požadavky definované v zadání, a měl by tedy být zadavatelem akceptován. Tato fáze je poslední fází, než systém dostane zadavatel k dispozici (pro akceptační testování, viz kapitola 1.5.4), je tedy nutno odhalit co možná nejvíce zbývajících chyb.

Toto testování není omezeno pouze na funkční testy, ale jsou zařazeny testy k ověření mimofunkčních požadavků. Dále jsou uvedeny příklady testů, které je možno v této fázi provádět.

- *Funkční testy.* Splnění funkčních požadavků.
- *Bezpečnostní testy.* Integrita a důvěrnost dat.
- *Testy robusnosti.* Práce systému s neočekávanými vstupy, zotavení se ze selhání.
- *Testy použitelnosti.* Ohodnocení komfortnosti a srozumitelnosti systému z pohledu uživatele.
- *Výkonnostní testy.* Sledování výkonu systému během různých situací. Do této kategorie patří např. zátěžové testy simulující dlouhodobou zátěž, testy hraniční zátěže testující systém během extrémního vytížení systému s cílem zjistit reálné omezení systému či testy škálovatelnosti pro ověření, zda systém bude možno škálovat.
- *A/B testy.* Provádění za účelem zjištění, která z možných variant (verzí) systému má větší přínos. [5]

1.5.4 Akceptační testování

Akceptační testování se zásadně liší od předchozích - je prováděno samotným zadavatelem, resp. uživatelem. Cílem akceptačního testování je ověření, zda systém splňuje tzv. *akceptační kritéria*. Pod pojmem akceptační kritéria rozumíme měřitelné a ověřitelné podmínky pro přijetí produktu. Akceptační kritéria se nemusí týkat jen systému samotného, ale může jít též o zpracování dokumentace či o soulad se zákony či normami.

Během této fáze nejde o hledání chyb, ale o ověření „použitelnosti“ systému. Uživatelé testují systém tak, že provádí běžné aktivity, které se budou odehrávat v reálném

provozu. Problémem zde může být např. situace, kdy by systém měl v určitou chvíli umožnit založení záznamu (a systém funkci založení záznamu korektně implementuje a za jiných okolností je umožněna), avšak založení není umožněno. [5]

1.5.5 Automatizované testy

V ideálním případě by mělo k testování docházet při každé změně, ať již kódu samotného či jiné komponenty, která může ovlivnit chování systému. Dochází tak k opakující se činnosti, kterou je vhodné automatizovat. Cílem *automatizovaného testování* je časová úspora a minimalizace lidských chyb. Doménou automatizovaného testování jsou především jednotkové a integrační testy.

Pro úspěšnou automatizaci jednotkových a integračních testů by měly testy splňovat následující kritéria:

- *Dostatečný a stručný zároveň.* Test by měl pokrýt co nejvíce testovanou funkci, ale zároveň neobsahovat zbytečné části.
- *Jednoduchá kontrola výsledku.* Případná manuální kontrola výsledku by měla být co nejsnazší.
- *Opakovatelnost.* Test by mělo být možno spouštět dle potřeby opakovaně bez nutnosti manuálního zásahu.
- *Stabilita.* Při stejných podmínkách by měl vracet stejné výsledky.
- *Srozumitelnost a trasovatelnost.* Test samotný by měl být srozumitelný - mělo by být bez dlouhého studia snadno pochopitelné co daný test dělá a co testuje.
- *Časová efektivnost.* Test by měl běžet přiměřeně dlouhou dobu.
- *Udržovatelnost.* Test by mělo být možné snadno upravit při změně testovaného systému.

U jednotkových testů by mělo být navíc splněno:

- *Nezávislost.* Testy spuštěné současně by se neměly z pohledu přítomnosti chyb vzájemně ovlivňovat. [6]

1.6 Předání do provozu (nasazení)

Po úspěšné akceptaci zákazníkem je systém připraven na *předání do provozu - nasazení.* Instalace systému je pouze jedna z činností, které je nutno v tomto kroku splnit. V této fázi se také předává dokumentace systému a zaškolují se uživatelé. Pokud se přechází

z jiného systému a bylo tak předem dohodnuto, dochází též ke konverzi starých dat do nového systému. [1]

1.7 Provoz a údržba

Provoz systému a jeho údržba je poslední část životního cyklu systému, zároveň se ale jedná o nejdelší fázi. Systém je nasazen, uživateli běžně používán a na výrobci by mělo být provádění softwarové údržby. Pod tímto termínem se rozumí:

- *Odstraňování nalezených chyb.*
- *Adaptivní údržba.* Přizpůsobení změnám v SW či HW prostředí.
- *Zdokonalování.* Implementace nových funkcí, reakce na nové či změněné požadavky.
- *Preventivní údržba.* Zlepšení udržovatelnosti systému. [1]

2 Softwarová architektura

Softwarová architektura je jednou z dílčích architektur sloužící k prohloubení návrhu budoucího systému. Softwarová architektura popisuje strukturu a vztahy programových modulů dané aplikace.

Softwarová architektura určuje:

- *Způsob provozu.* Jedno či více uživatelská.
- *Počet, uspořádání a funkcionalitu modulu.*
- *Vstupní a výstupní data funkcí, včetně algoritmu na transformaci těchto dat.*
- *Vzájemnou vazbu modulů a interface modulu na jiné aplikace.*
- *Vývojové a provozní prostředí modulu.* [7]

2.1 Monolitické a vícevrstvé architektury

Běžné systémy obsahují tři základní skupiny funkcí:

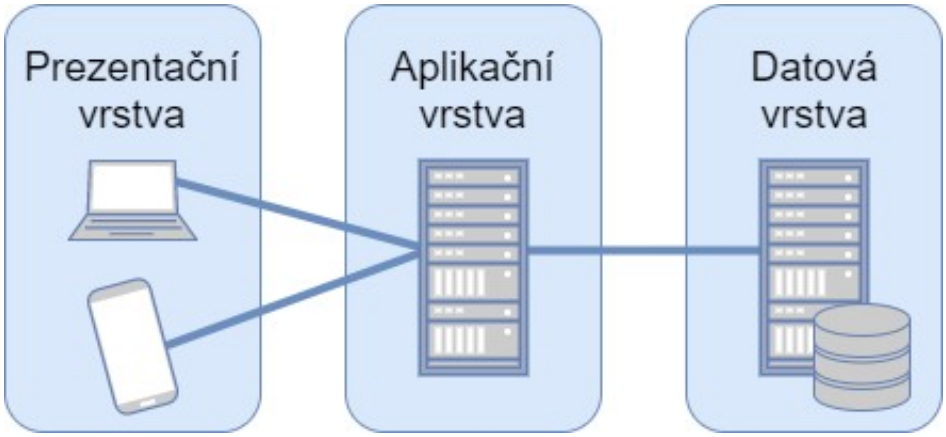
- *Datové funkce.*
- *Business logiku aplikace.*
- *Komunikační funkce.* Zobrazování výsledků, komunikace s uživatelem.

Rozložení těchto funkcí do samostatných aplikací určuje, zda se jedná o *monolitickou*, *dvouvrstvou*, *třívrstvou* či obecně *vícevrstvou architekturu.* U monolitické architektury platí, že všechny tři uvedené skupiny funkcí jsou implementovány v jedné aplikaci. Dvouvrstvá rozděluje skupiny funkcí do dvou samostatných aplikací. [7]

Třívrstvá architektura má pro každou skupinu funkcí samostatnou aplikaci. Tyto skupiny odpovídají jednotlivým vrstvám architektury. Vrstvy architektury tedy jsou (viz Obr. 2.1):

- *Datová vrstva.* Někdy též databázová, tvoří nejnižší vrstvu a jejím úkolem je provádění základních datových operací jako je ukládání, výběr či agregace. Dále se stará o integritu a audit dat.
- *Aplikační vrstva.* Tato vrstva bývá někdy též označena jako funkční, logická, či jako aplikační server. Tato prostřední vrstva obstarává výpočty a operace prováděné mezi vstupem a výstupem.

- *Prezentační vrstva.* Nejvyšší vrstva mající na starost vstup požadavků a následnou prezentaci výsledků. Aplikace v této vrstvě bývají závislé na platformě, může tedy těchto aplikací existovat vícero (aplikace pro desktop a aplikace pro mobilní zařízení). [8]



Obr. 2.1 Třívrstvá architektura [8]

Vrstev může být i vícero, např. pokud započítáme komunikační vrstvu mezi aplikační a prezenční vrstvou. Takové systémy obecně nazýváme jako vícevrstvé.

3 Verzovací systémy

Verzovací systémy, z anglického version control systems (VCS), jsou systémy určené především vývojářům. Tyto systémy mají často v sobě integrovánu *správu zdrojových textů* (Source Code Management, SCM). Termín „verzovací systémy“ pak není úplně přesný, pro naše potřeby však dostačující.

VCS jsou zaměřené na dokumentaci, sdílení a slučování změn, tedy na úkoly časté ve fázi implementace software. Obecné využití verzovacích systémů se však neomezuje pouze na práci s programovým kódem či na vývojáře samotné - často je můžeme nalézt např. v pokročilých textových editorech. [9]

Jak již bylo zmíněno, jsou zaměřeny na často se opakující úkoly (dokumentaci, sdílení a slučování změn). Z tohoto důvodu tak u těchto úkolů roste riziko vzniku chyby ze strany vývojáře. Obzvláště při práci v týmu, kde jednotliví vývojáři pracují paralelně nad společným zdrojovým kódem, je potřeba sdílet a slučovat změny kritická. VCS s těmito úkoly napomáhá, čímž redukuje nejen možnost vzniku chyby, ale též snižuje čas potřebný k těmto úkolům. Vývojáři se tak mohou více soustředit na jejich hlavní činnost - vývoj. [9]

V následujících kapitolách jsou blíže popsány základní principy verzování a je provedeno jejich rozdělení.

3.1 Základní principy

Základním stavebním prvkem *verzovacích systémů* je *repozitář*. Repozitář obsahuje *verzované soubory*. Verzování souborů má na starosti *správce verzí*, jednotlivé verzovací systémy pak poskytují nástroje pro získání těchto verzovaných souborů. Kromě repozitáře existuje také *lokální kopie* - dle verzovacího systému se jedná buď o lokální kopii repozitáře nebo konkrétní verze souborů z repozitáře. Uživatel tedy nepracuje přímo nad repozitářem, ale vždy nad kopiemi verzovaných souborů, které modifikuje. [10]

Nástroje verzovacích systémů poskytují základní operace:

- *Checkout*. Vytvoření lokální kopie z repozitáře.
- *Commit*. Nahrání změn z lokální kopie do repozitáře.
- *Update*. Aktualizace lokální verze z repozitáře.

Při operacích *commit* a *update* může dojít ke konfliktům. Konflikt vzniká současnou modifikací stejného místa (stejného řádku kódu) více vývojáři. Správce verzí nezabraňuje těmto konfliktům, ale nabízí nástroje pro jejich řešení. [10]

3.2 Rozdělení verzovacích systémů

Verzovací systémy se dělí dle místa uložení repozitáře a přístupu k repozitáři. Verzovací systémy se dělí na:

- *Lokální verzovací systémy.* Repozitář existuje pouze na lokálním počítači a pouze na tomto počítači je pracováno s lokálními kopiemi. Jsou tedy vhodné, pokud na projektu pracuje pouze jeden vývojář. Mezi tyto verzovací systémy patří například *Revision Control System* (RCS).

- *Centralizované systémy správy verzí.* Tento typ řeší nedostatek předchozího - repozitář je umístěn na centrálním serveru a umožňuje práci více lidí v týmu. Repozitář (stejně jako u předchozího typu) existuje pouze v jedné kopii. Při selhání centrálního serveru, kde je repozitář umístěn, může dojít ke ztrátě dat. Příkladem těchto verzovacích systémů je například *Concurrent Versions System* (CSV), *Apache Subversion* (SVN) nebo *Team Foundation Version Control* (TFVC).

- *Distribuované systémy správy verzí.* V tomto případě se vytváří společně s lokální kopií souborů také kopie repozitáře. Při selhání stačí použít repozitář z jiného umístění. Tuto kategorii zastupuje například *Git* nebo *Mercurial*.

4 DevOps

DevOps je anglickou zkratkou pro *Development & Operations*, která zjednodušeně řečeno vyjadřuje úzkou spolupráci vývoje a provozu. V této kapitole je nejprve přiblížena historie vzniku DevOps a poté termín samotný.

4.1 Historie

Všechno má svůj postupný vývoj a DevOps není výjimkou. Historie DevOps sahá do počátku devadesátých let, kdy se poprvé objevilo *denní sestavení aplikace* (Daily Build). Daily Build obsahoval zkompilovaný výstup otestovaný základními testy. Na konci devadesátých let se začíná prosazovat další fáze - *kontinuální (průběžná) integrace* (Continuous Integration, CI). Ta mimo jiné obsahuje též integrační testy. Díky verzovacím systémům je možno CI provádět po každé změně. Přelom století přináší *kontinuální dodávky* (Continuous Delivery, CD). CD, někdy též nazývaná jako průběžné doručování, kromě otestované aplikace samotné obsahuje též všechno související s dodávkou. Jde tedy o proces sestavování (build), testování, konfigurace a nasazení (deploy) z buildu do produkčního prostředí. [11, 12]

4.2 Definice

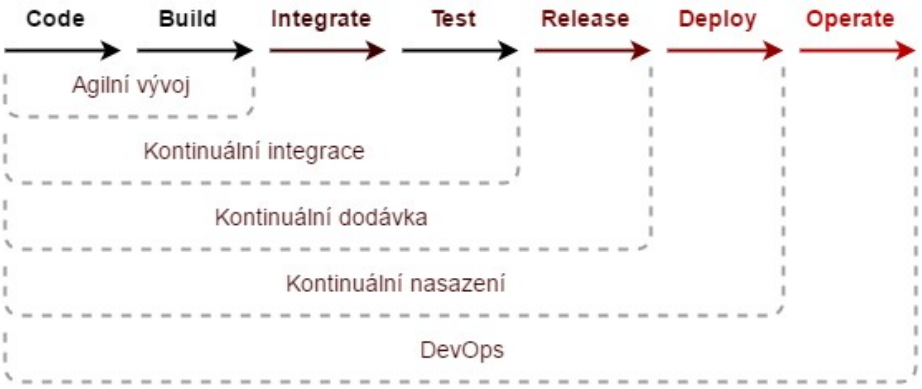
DevOps se definuje jako sada postupů určená k minimalizaci času mezi provedenou změnou a promítnutí této změny do požadovaného prostředí (produkčního) za dodržení vysoké kvality. Z toho vyplývá následující:

- *Vysoká kvalita změn.* Jedna z možností jsou předprodukční automatizované testy. Další možností je použít A/B testování nebo detailněji sledovat chování po aplikování nasazené změny.
- *Spolehlivý proces nasazení.* Proces nasazení musí být snadno opakovatelný a pokud možno bezchybný.
- *Dva důležité časové milníky.* Prvním je nahrání změny a druhým je nasazením do produkčního prostředí.
- *Zaměření na cíl.* Nespecifikujeme nástroje nebo konkrétní postupy, ale cíl.
- *Nejen testování a nasazení.* Definice neomezuje rozsah DevOps jen na testování a nasazení. [13]

4.3 Použití

Předpokladem pro využití DevOps je nejen propracovaná automatizace. Je nutno mít též zvládnutou formální stránku věci - mít definované potřebné postupy či procesy. Součástí těchto procesů je i určení za jakých podmínek je možno provést nasazení do produkčního prostředí. Možnost provádět nasazení po každé změně ještě neznamená, že je to i žádoucí. Nasazení do produkčního prostředí vyžaduje nejen zapojení IT, ale i například byznysu a ostatních procesů. [11]

DevOps navazuje na schopnost kontinuální dodávky (viz Obr. 4.1). Nasazením do produkčního prostředí vývoj nekončí. DevOps pokračuje kontinuálním monitoringem a zapojení Operations. Výstup je použit jako vstup pro další vývoj. [12]



Obr. 4.1 DevOps [11]

5 Cloud computing

V současné době informačních technologií rostou požadavky na zpracování a následné uchovávání dat. Společně s narůstajícím objemem dat jsou kladeny také stále větší požadavky na bezpečnost těchto dat, ale také například na mobilitu či na plnění legislativních a regulačních norem. Dříve firmy k těmto účelům budovaly vlastní infrastrukturu, dnes se stále více přiklání k využívání cloudových řešení nebo-li *cloud computingu*. [11]

V této kapitole je nejprve představen pojem *cloud*, resp. *cloud computing* a základní termíny s ním související. Poté je provedeno krátké rozdělení cloud computingu. Na konci kapitoly je k dispozici krátký přehled současných nejvýznamějších poskytovatelů.

5.1 Základní pojmy a charakteristika

Cloud je o způsobu používání softwaru nebo hardwaru - k obojímu je možno přistupovat formou služeb dostupných prostřednictvím internetu. Z pohledu zákazníka tedy vše probíhá pouze jako nákup služby, ať již jde například o software, infrastrukturu, diskové či databázové úložiště. Zákazník se nemusí starat o žádné své servery a věci s tím spojené - o ty se stará poskytovatel, který může požadovanou službu složit z více dílčích služeb, případně využít služeb dalších poskytovatelů (subposkytovatelů). [14]

Dle [15] je možno cloud computing definovat jako model, který umožňuje síťový přístup dle potřeby ke sdílené skupině konfigurovatelných zdrojů, které jde rychle realizovat s vynaložením minimálního úsilí nebo minimální interakcí poskytovatele služby. Cloud computing je též možno charakterizovat jako:

- *Samoobslužný dle potřeb.* Zákazník musí mít možnost nezávislého uspořádání zdrojů dle potřeby.
- *S širokopásmovým přístupem.* Přístup ke zdrojům prostřednictvím internetu či privátní sítě.
- *Využívající sdružených zdrojů.* Zdroje poskytovatele jsou sdruženy do velkých celků a poskytovány více uživatelům (multi-tenant, viz 5.3). Zdroje jsou dynamicky alokovány a dealokovány dle potřeb uživatelů. Uživatel přímo nekontroluje fyzické zdroje, ani nezná jejich přesné umístění.
- *Vysoce pružný.* Zdroje mohou být odstupňovány - škálovány (v některých případech i automaticky, například dle zatížení), což může působit dojmem „neomezených“ zdrojů.

- *Měřitelný.* Přidělené zdroje mohou být monitorovány a řízeny, což poskytuje transparentní přehled (pro poskytovatele i zákazníka) o skutečném využití služeb. [15]

S pojmem cloud computing úzce souvisí termín *virtualizace*. Cloud computing je na virtualizaci postavený. Virtualizace je systém rozdělování počítačových zdrojů do vícenásobného realizačního prostředí. Základem jsou technologie, které tvoří abstraktní vrstvu mezi hardwarem a softwarem. Virtualizaci je také možno chápat jako abstrakci fyzické komponenty do logických objektů. Tyto logické objekty umožňují větší flexibilitu využití. Díky virtualizaci je tak například možno provozovat vícero virtuálních operačních systémů na jednom fyzickém stroji. [16, 17]

5.2 Rozdělení cloudových prostředí

Cloud computing lze rozdělit na tři základní úrovně služeb:

- *Software as a Service (SaaS).* Poskytování určitých aplikací běžících v cloudové infrastruktuře. Aplikace mohou být přístupné na různých zařízeních, obvykle formou tenkého klienta (např. webový prohlížeč). Zákazníkovi je přístupné pouze specifické nastavení dané aplikace. Mezi tento typ služeb patří například elektronická pošta či nejrůznější informační systémy.

- *Platform as a Service (PaaS).* Poskytování platformy pro provozování aplikací. Aplikace jsou určeny přímo pro danou platformu. Zákazníkovi je umožněno konfigurovat nastavení pro hostování aplikací. Příkladem těchto služeb může být například poskytování webhostingu.

- *Infrastructure as a Service (IaaS).* Jedná se o nejobsáhlejší službu zahrnující poskytování fyzické nebo virtuální infrastruktury. Zde je možno provozovat libovolný software včetně operačních systémů. Zákazník spravuje operační systém včetně nasazených aplikací. V omezené míře také může spravovat vybrané síťové komponenty (např. firewall).

Uvedené úrovně služby se tedy liší především jejich rozsahem a možnostmi konfigurace. Žádná úroveň však neumožňuje správu infrastruktury na které je provozován cloud samotný. [14, 15, 18]

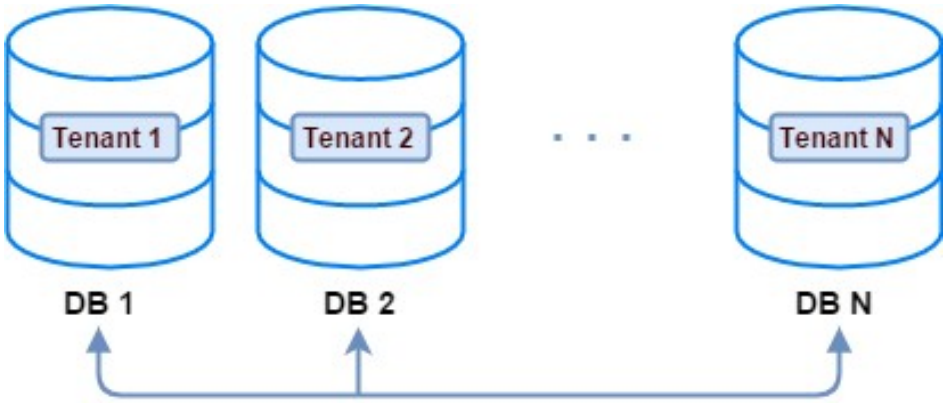
Cloud je možno dále dělit dle způsobu realizace či nasazení na:

- *Privátní cloud.* Infrastruktura je určena exkluzivně pro jednoho zákazníka, často je i v jeho vlastnictví.

- *Komunitní cloud.* Infrastruktura vzniká sloučením zdrojů jednotlivých členů komunity, jež obvykle sdílí nějaký společný zájem. Infrastruktura je pak zpětně poskytována členům této komunity.
- *Veřejný cloud.* Infrastruktura je vlastněna poskytovatelem. Posktovatelem může být soukromá firma, akademická sféra, vládní organizace či kombinací vícero vlastníků. Tyto prostředky jsou poskytovány široké veřejnosti.
- *Hybridní cloud.* Kombinace předchozích. [14, 15, 18]

5.3 Multi-tenancy

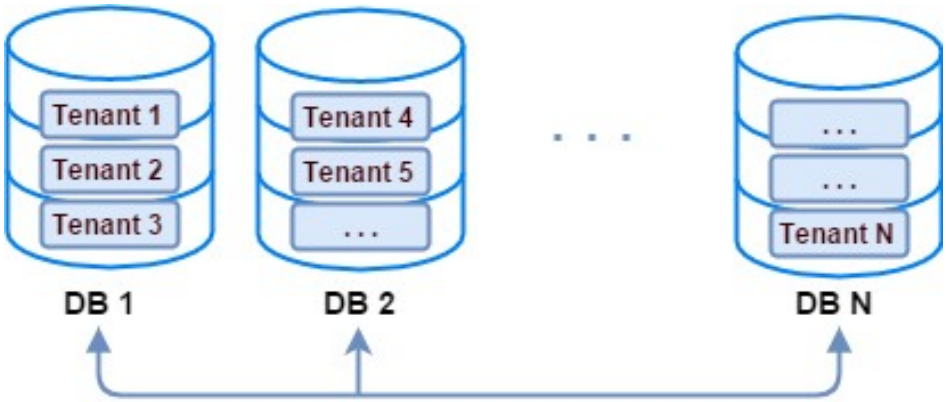
Jak již bylo řečeno, jednou z charakteristik cloud computingu je využití sdílených zdrojů. Poskytovatel tedy může za účelem zvýšení efektivnosti a snížení nákladů *konsolidovat zákazníky* - nájemce (přesneji jim přiřazené zdroje) na jeden počítač. Tato konsolidace je nazývána *multi-tenancy*. Opakem multi-tenancy, tedy využití zdrojů bez jejich sdílení, je nazýváno *single-tenancy*. Multi-tenancy může být aplikována na různých úrovních, např. na úrovni aplikace či databázové vrstvy. Příklad single-tenancy, resp. multi-tenancy u databázové vrstvy je uveden na Obr. 5.1, resp. na Obr. 5.2.



Obr. 5.1 Single-tenancy u databázové vrstvy [19]

Multi-tenancy je možno realizovat následujícími způsoby (souhrnný pohled viz Obr. 5.3):

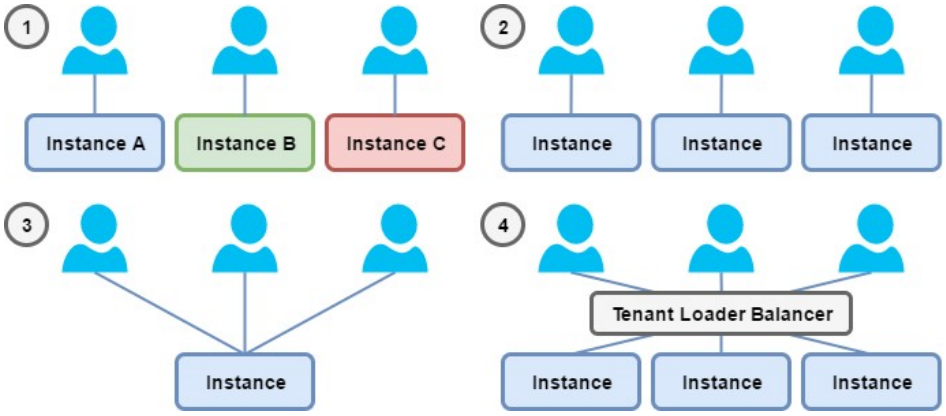
1. *Každý zákazník má vlastní vyhrazenou instanci.* Tento přístup umožňuje největší možnost přizpůsobení jednotlivým potřebám zákazníka (např. volba velikosti úložiště, velikost RAM či rychlost CPU).
2. *Každý zákazník má vlastní vyhrazenou instanci, instance jsou ale stejné.* Tento způsob je stejný jako v prvním případě, ale přizpůsobení je možno pouze konfigurováním aplikace, nikoliv parametrům instance. Pokud aplikace odesílá e-mailové zprávy, je např. možno konfigurovat jazyk nebo patičky zpráv.



Obr. 5.2 Multi-tenancy u databázové vrstvy [19]

3. *Všichni zákazníci sdílí jednu instanci.* Existuje pouze jedna instance a aplikace se přizpůsobuje vždy podle autora požadavku. Budeme-li se držet příkladu s odesíláním e-mailové zprávy, součástí požadavku na odeslání e-mailové zprávy bude i autor požadavku. Před odesláním zprávy se zjistí patička a jazyk pro autora požadavku a zpráva se odešle.

4. *Sdílené instance s balancerem.* Obdobné jako v 3, jen instancí je víc a požadavek je nejprve směřován na balancer, který rozhodne o tom, která instance bude požadavek zpracovávat (např. podle vytížení jednotlivých instancí). [20]

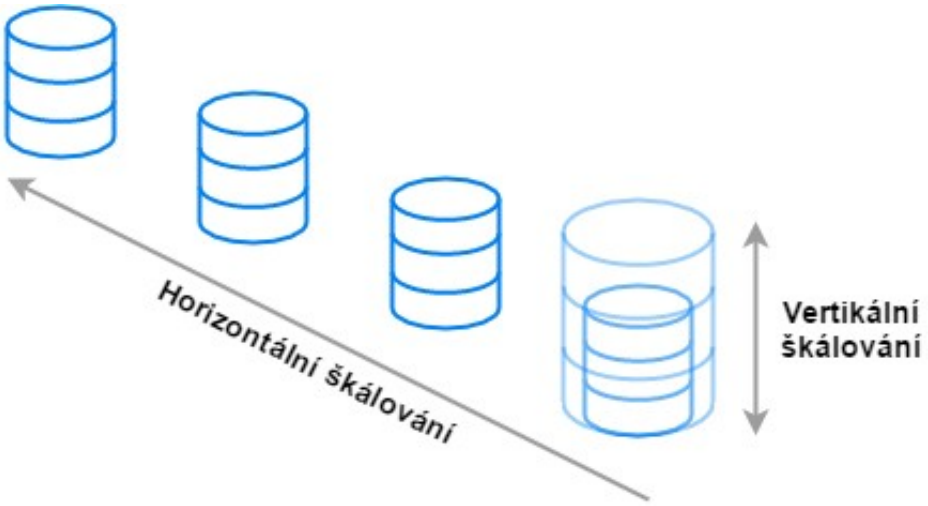


Obr. 5.3 Možnosti řešení multi-tenancy [20]

5.4 Škálování zdrojů

V oblasti cloud computingu je často využíván pojem *škálovatelnost*, resp. *škálovatelnost zdrojů*. Škálovatelnost znamená schopnost přizpůsobit se měnícím se požadavkům, zejména na objemy zpracovaných dat či zátěž systému. Požadavky mohou být vzrůstající i klesající. Existují dva druhy škálování (příklad škálování databáze viz Obr. 5.4):

- *Vertikální škálování (vertical scaling).* Při rostoucích požadavcích je zvyšován výkon příslušného zdroje pomocí výkonnějšího HW (scale up), při klesajích je výkon zdroje snižován (scale down).
- *Horizontální škálování (horizontal scaling).* Tento druh škálování reaguje na rostoucí požadavky přidáváním (scale out) a odebíráním (scale in) počtu instancí zdrojů, na kterých jsou požadavky zpracovány. [19, 21]



Obr. 5.4 Škálovatelnost databáze [19]

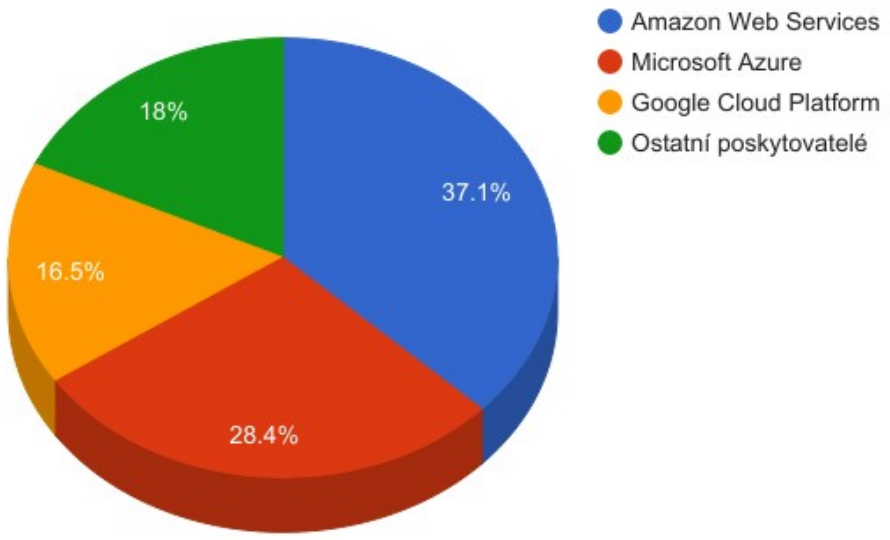
5.5 Přehled vybraných poskytovatelů

V této kapitole jsou ve zkratce představeni vybraní poskytovatelé cloudových služeb. Výběr poskytovatele je závislý na tržním podílu, jako zdroj byly použity data z [22] (viz Obr. 5.5). Jednodlivé podíly je však nutno brát s rezervou, protože porovnatelné údaje poskytovatelé obvykle nezveřejňují. Pro účely této kapitoly jsou vybrány produkty od společností Amazon, Google a Microsoft (v abecedním pořadí).

Produktu Azure od Microsoftu je věnována samostatná kapitola, proto zde není uveden.

5.5.1 Amazon

Průkopník oblasti cloud computingu v podobě, jaké ho známe nyní. První služba cloud computingu vznikla již v roce 2002 pod názvem *Amazon Web Services* (AWS). *Amazon* je známý svým internetovým obchodem, který byl svým způsobem impulsem k vytvoření služby. Většinu času byly servery na kterých běžel internetový obchod kapacitně nevyužité a sloužily pro případ nárazového vytížení, např. v době Vánoc. Myšlenkou tedy bylo tuto nevyužitou kapacitu prodávat. První komerční služba (*Amazon Elastic Compute Cloud*, EC2) byla spuštěna v roce 2006.



Obr. 5.5 Tržní podíl cloudových poskytovatelů [22]

AWS je nabízena z šestnácti geografických lokalit nacházejících se v Americe, Asii, Austrálii a Evropě. Pestrá je také škála nabízených služeb, jako příklad lze uvést:

- *AWS Elastic Beanstalk*. Služba určená pro hostování aplikací napsaných v .NET, Go, Java, Node.js, PHP, Python a Ruby. Standardem je automatické škálování.
- *Amazon Elastic Compute Cloud (Amazon EC2)*. Služba umožňující běh virtuálních počítačů. Počítače je možno vytvářet z předem daných konfigurací. K dispozici jsou operační systémy Windows i linuxové distribuce. Nejvýkonnější instance může mít CPU se 128mi jádry a 1952 GB RAM.
- *Amazon S3*. Cloudové úložiště dat podporující objekty až do velikosti 5TB.
- *Amazon Aurora*. Cloudové databázové úložiště kompaktibilní s MySQL a PostgreSQL. Minimální velikost databáze je 10GB, maximální pak v přírůstcích po 10GB až do velikosti 64TB. [23]

5.5.2 Google

Google, proslavený především svým vyhledávačem, nabízí své cloudové služby pod značkou *Google Cloud Platform*. Služby poskytuje z patnácti lokalit umístěných v Americe, Asii a Evropě. [24] Google jako své hlavní plusy uvádí:

- *Nadčasovou infrastrukturu*. Bezpečná, celosvětová, výkonná, efektivní a neustále se zlepšující
- *Big data a analýza*. Možnost zapojení big dat a jejich analýzy za účelem zlepšování nabízených produktů.

- *Kód na prvním místě.* Není třeba řešit kapacitu, spolehlivost nebo výkon. [25]

Google nabízí své služby v kategoriích *Compute*, *Storage & Databases*, *Networking*, *Big Data*, *Machine Learning*, *Identity & Security*, *Management Tools* a *Developer Tools*.

Příklady nabízených služeb:

- *App Engine.* Platforma umožňující vytvářet a hostovat aplikace. Mezi podporované programovací jazyky patří C#, Go, Java, Node.js, PHP, Python a Ruby. App Engine podporuje automatické škálování aplikací.
- *Compute Engine.* IaaS služba umožňující vytvářet a spouštět virtuální počítače (VM) určené pro náročné a rozsáhlé výpočty. Na VM je možné provozovat linuxové distribuce, Windows Server, Red Hat či SQL Server. Výhodou compute engine je, kromě volby z předdefinovaných šablon, možnost plně konfigurovat VM. Nejvýkonnější konfigurace tak může obsahovat CPU s až 64 jádry, 416 GB RAM a až osm grafických karet NVIDIA Tesla K80.
- *Cloud Storage.* Cloudové úložiště dat poskytované jako IaaS.
- *Cloud SQL.* Služba umožňující provozovat MySQL a PostgreSQL databázi. Jde o tzv. „fully-manage“ službu - uživatel se stará pouze o databázi, správu serveru obstarává poskytovatel. [26]

6 Microsoft Azure

Tato kapitola je zaměřena na cloudové prostředí *Microsoft Azure*. První část je věnována historii platformy. Druhá část se věnuje představení nabízených služeb. Konec kapitoly si klade za cíl popsat způsoby práce s tímto prostředím.

6.1 Historie a současnost

Vývoj platformy započal v roce 2006. Na projektu nezávisle pracovaly tři týmy - *Windows*, *Windows Communication Foundation* a *SQL tým*. Projekt se nazýval *Microsoft Azure Service Bus* a v té době běžel na jednom počítači. Týmy začaly postupem času spolupracovat a v roce 2008 představily na *Professional Developers Conference* (PDC) první preview verzi. Tato verze obsahovala tři základní produkty - *.NET Services*, *Live Services* a *SQL Services*. Došlo též k uvolnění *software development kit*-u (SDK).

V roce 2009 se platforma otevřela a přibyla podpora PHP a Javy. Spuštěn byl také webový portál pro správu poskytovaných služeb. O rok později, 2010, Microsoft hlásí prvních 10.000 zákazníků. Maximální velikost databáze (SQL Azure) vzrostla na 50 GB. Došlo k oznámení prací na *Windows Azure Virtual Machine* (AVM).

V roce 2012 bylo AVM uvolněno jako preview, čímž se Microsoft začíná řadit do skupiny IaaS poskytovatelů. Windows Azure je dostupná z osmi lokací. Rok 2013 znamená roční tržby přesahující jednu miliardu dolarů. Rok 2014 přináší přejmenování na *Microsoft Azure*. [27, 28]

Po celou dobu, až do současnosti, dochází k rozšiřování poskytovaných služeb i dostupnosti. V současné době (2016/2017) došlo například k výstavbě datového centra v Německu. Toto nově vznikající datové centrum je speciálně navrženo pro splnění přísných norem Evropské unie. [29] Služba Microsoft Azure je nyní poskytována z 34 datacenter, přičemž další čtyři jsou ve výstavbě (viz Obr. 6.1). [30]

Obr. 6.1 Datacentra Microsoft Azure [30]

6.2 Služby

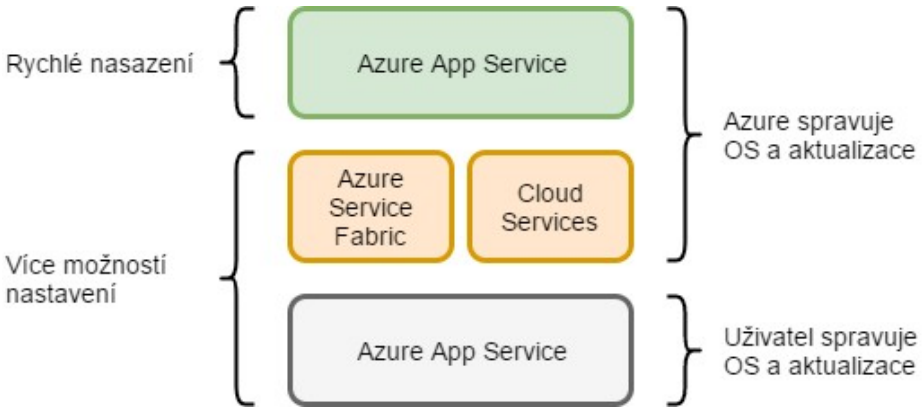
Microsoft Azure, či zkráceně jen *Azure*, je cloudová platforma s širokými možnostmi použití. Je možno ji např. využít k hostování existující aplikace, rozšířit stávající on-premise aplikaci (aplikaci poskytovanou z vlastní infrastruktury) či použít výpočetní kapacity dle potřeby a nároků aplikace. Azure také integruje nástroje potřebné pro vývoj, testování, nasazení a správy aplikací. Vše při zachování výhod cloudového řešení. [31, 32]

Platforma je složena ze služeb. V době psaní této práce již jejich počet výrazně převýšil hranici sta služeb (březen 2017: 115 služeb, 17 služeb v preview režimu [33]), proto jsou popsány jen ty nejzákladnější, použité v této práci.

Služby jsou vždy svázány s konkrétním datacentrem, proto zejména ty v režimu preview nemusí být dostupné na všech datacentrech. Společným parametrem u většiny služeb, kromě jejich lokace, je i plán - ten určuje výkon, maximální počet instancí a další konfigurace služby. Plány jsou cenově odstupňovány. [33]

6.2.1 Hostování aplikace

Pro hostování aplikací Azure poskytuje několik možností. Jedním z kritérií je volba úrovně, na které bude služba poskytována. V tomto případě hovoříme o úrovních PaaS a IaaS. Přehled služeb rozdělených podle úrovní je zobrazen na Obr. 6.2. Popis jednotlivých úrovní je uveden v kapitole 5.2. [34]



Obr. 6.2 IaaS a PaaS pro hostování aplikací v Azure [34]

Pro srovnání následuje popis dvou rozdílných možností - hostování aplikace pomocí *Azure App Service* (PaaS) a *Virtual Machines* (IaaS).

Azure App Service První možností pro hostování webových aplikací je *Azure App Service* (zkráceně *App Service*). Tato varianta slouží pro rychlé a jednoduché nasazení

webových aplikací. Odpadá zde nutnost zdlouhavého nastavování či správy infrastruktury. Webové aplikace, v rámci této služby, je možno tvořit za pomocí .NET, Java, Node.js, PHP a Python.

App Service podporuje principy DevOps. V rámci nich je možno využít kontinuální dodávku ze služeb jako jsou BitBucket, GitHub, Jenkins, TeamCity nebo Visual Studio Team Services. Aplikaci je možno podrobit výkonnostním testům či provádět monitoring v reálném čase. Aplikace též může být nasazena ve více verzích zároveň, tzv. *staging*. Tuto funkci lze využít např. k A/B testování či vytvoření akceptačního prostředí.

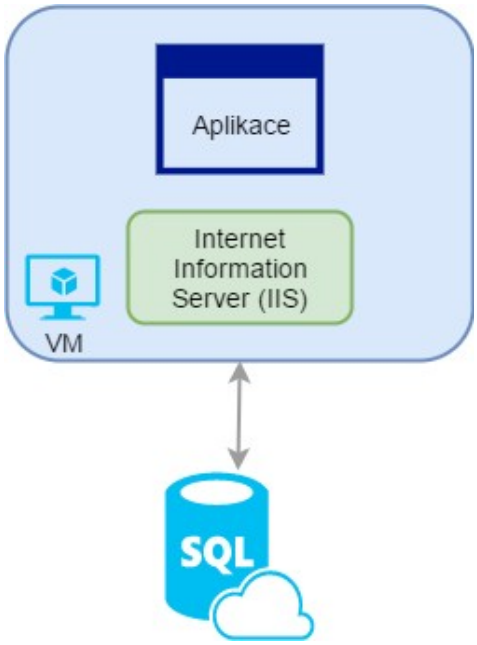
Prostředí, na kterém jsou aplikace spuštěny, je možno škálovat (vertikálně i horizontálně), aby odpovídalo výkonnostním požadavkům. Škálování je možno provádět manuálně i automaticky na základě předdefinovaných metrik (např. vytížení CPU).

App Service podporuje čtyři základní typy projektů:

- *Web Apps*. Slouží pro hostování webových stránek a webových aplikací. Příklad struktury projektu založeného na Web Apps je uveden na Obr. 6.3.
- *Mobile Apps*. Rozšiřuje Web Apps pro použití na mobilních zařízeních vytvořením potřebného zázemí (backend). Integruje například push notifikace (zasílání zpráv na mobilní zařízení) či autorizaci pomocí externích poskytovatelů.
- *API Apps*. Umožňuje hostovat RESTful API.
- *Logic Apps*. Podporuje automatizaci business procesů bez nutnosti psaní kódu. [32, 33, 34, 36]

Virtual Machines *Azure Virtual Machines* (Azure VM) představují jeden ze základních prvků samotného Azure. Azure VM nabízí možnost vytvoření virtuálního počítače konfigurovatelného po stránce hardware i software. Softwarová konfigurace začíná výběrem provozovaného operačního systému. Mezi podporované operační systémy patří Windows či Linux. Na výběr je z několika předpřipravených operační systémů - např. Red Hat Enterprise Linux, Ubuntu Server či Windows Server, ale též již sestavených kompletních řešení pro specifické platformy či úlohy (SAP Hana, IBM včetně WebSphere či podnikové aplikace založené na Oracle). Pokud by nevyhovovalo žádné z nabízených řešení je podporováno nahrání vlastního virtuálního disku (VHD) s vlastním řešením. Příklad řešení je uveden na Obr. 6.4.

Azure VM poskytuje flexibilitu virtualizace bez nutnosti nákupu a správy hardware, na rozdíl od App Service je ale nutno konfigurovat a spravovat software, který je



Obr. 6.3 Struktura Web App [35]

na něm provozován (např. aktualizace či konfigurace operačního systému a ostatních používaných aplikací). [36]

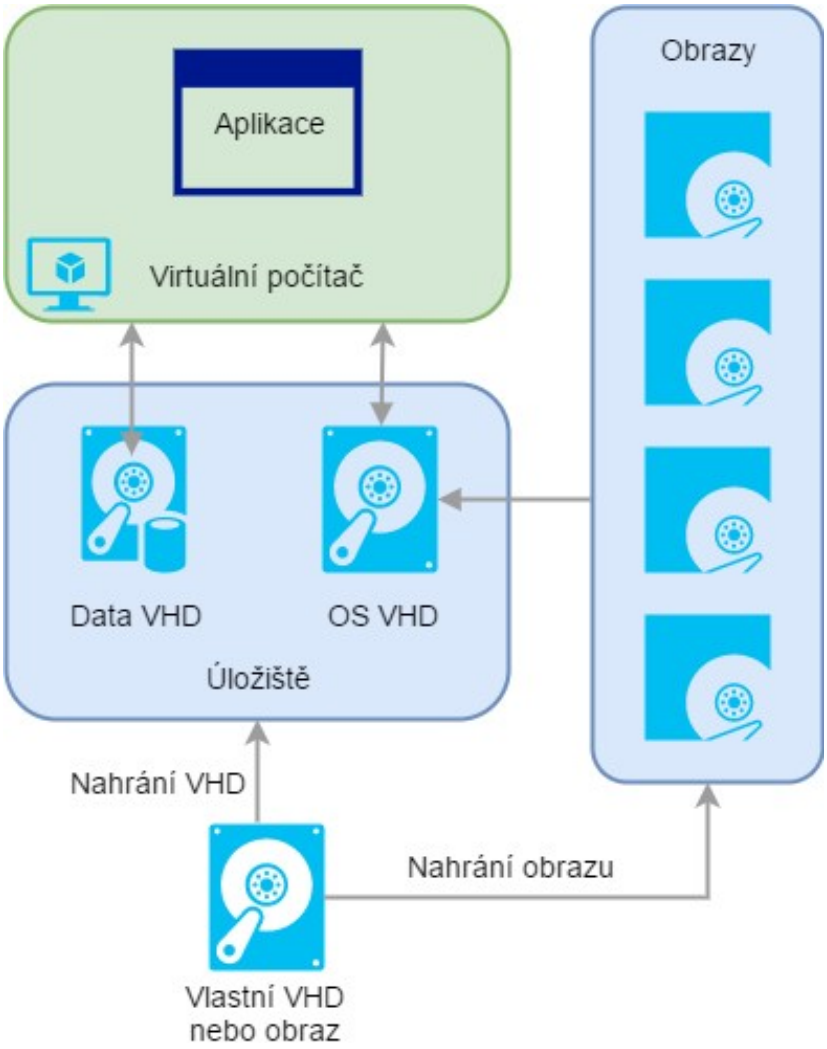
Stejně jako v případě App Service, i u virtuálních počítačů je k dispozici vertikální i horizontální škálování. Škálování může být spuštěno manuálně i automaticky dle definovaných pravidel. [33]

Použití Azure VM se neomezuje pouze na hostování aplikací, ale jejich použití je mnohem obecnější. Lze je použít třeba jako:

- *Vývojové a testovací prostředí (Dev/Test).* Azure VM může sloužit jako levné vývojové a testovací prostředí. Aplikace je nahrána, nakonfigurována, otestována a po dokončení je prostředí vypnuto.
- *Přesun aplikací do Azure (Lift-and-shift).* Aplikace jsou přesunuty z lokálního datacentra do Azure.
- *Rozšíření datacentra.* Přesunuta je pouze část aplikací a zbývající jsou dále obsluhovány z lokálního datacentra, ke kterému jsou virtuální počítače z Azure VM připojovány. [35]

6.2.2 Úložiště dat

Dalším krokem, který je obvykle třeba při vývoji aplikace učinit, je výběr úložiště dat. Výběr konkrétního typu úložiště je dán především typem ukládaných a zpracovávaných dat. Azure pro tyto účely nabízí dvě základní služby - *Azure SQL Database* a *Storage*.



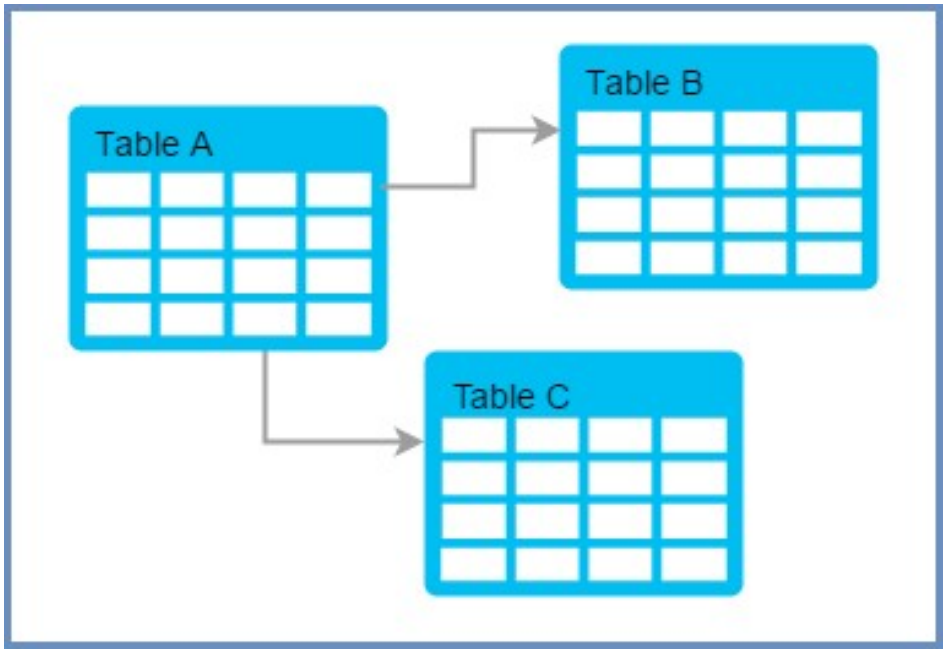
Obr. 6.4 Struktura Azure VM [35]

Obě zmíněné služby automaticky udržují data ve třech synchronizovaných kopiích v rámci jednoho datacentra nebo v šesti při povolené geo-redundanci. [35]

Azure SQL Database *Azure SQL Database* je služba nabízející relační databázové úložiště. Z hlediska funkčnosti poskytuje všechny klíčové vlastnosti relačních databázových systémů. Azure SQL Database podporuje atomické operace, současný přístup více uživatelů při zachování integrity dat či vykonávání standardizovaných ANSI SQL dotazů. [35, 36]

Azure SQL Database je poskytována jako PaaS. Data a přístup k nim určuje uživatel, přičemž o chod a údržbu se stará poskytovatel služby. Azure SQL Database dále nabízí vysokou dostupnost, automatické zálohy a replikaci do zvolených geografických lokací. [35, 36]

Model úložiště je zobrazen na Obr. 6.5.



Obr. 6.5 Struktura Azure SQL Database [35]

Storage *Storage* je služba poskytující škálovatelné a vysoce dostupné datové úložiště. Na rozdíl od Azure SQL Database není zaměřena na ukládání relačních dat. Dle typu ukládaných dat jsou dále rozlišovány následující typy úložiště:

- *Blob*. Nestrukturovaná data velkých objemů. Možnosti úložiště na bázi vrstev.
- *Disk*. Určeno pro aplikace náročné na vstupně-výstupní operace.
- *File*. Jednoduchý distribuovaný systém souborů.
- *Queue*. Fronty pro cloudové služby s velkými objemy dat.
- *Table*. Úložiště tabulek s dvojicí klíč-hodnota. [33, 35]

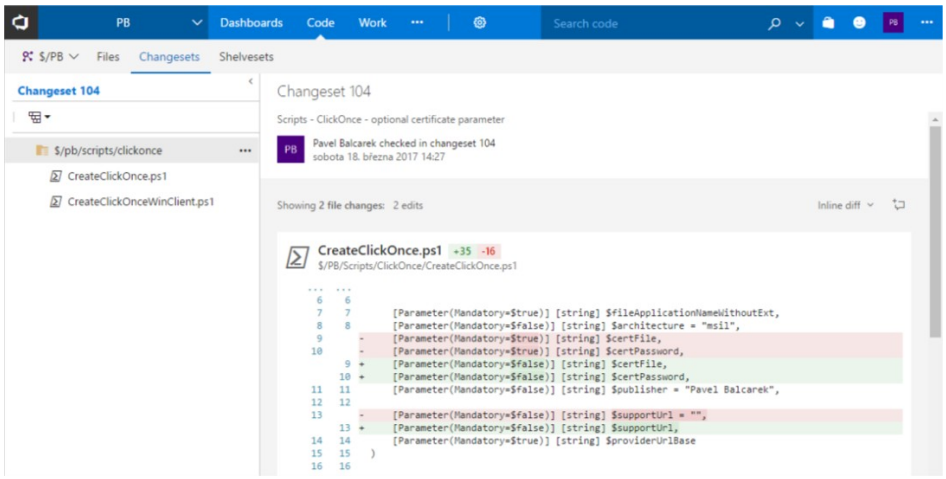
6.2.3 Nástroje pro vývojáře

Tato skupina služeb je zaměřena na podporu vývojářů.

Visual Studio Team Services *Visual Studio Team Services* (VSTS) je služba typu SaaS umožňující sdílení kódu, plánování práce, sledování práce a dodávky aplikací. K dispozici je i varianta instalovatelná na vlastní server, ta je dostupná pod názvem *Team Foundation Server* (TFS). [33, 37]

Pro sdílení kódu je využíváno verzovacího systému. Na výběr je z distribuovaných (Git) a centralizovaných (Team Foundation Version Control) verzovacích systémů. Vývojáři k nahrávání změn v kódu mohou použít jimi preferované *vývojové prostředí*

(Integrated Development Environment, IDE), za předpokladu podpory některého ze zmiňovaných verzovacích systémů. Kód i změny v něm provedené jsou k dispozici i prostřednictvím webového rozhraní. Příklad náhledu provedené změny na webovém rozhraní je uveden na Obr. 6.6. [38]



Obr. 6.6 Náhled změny v prostředí VSTS

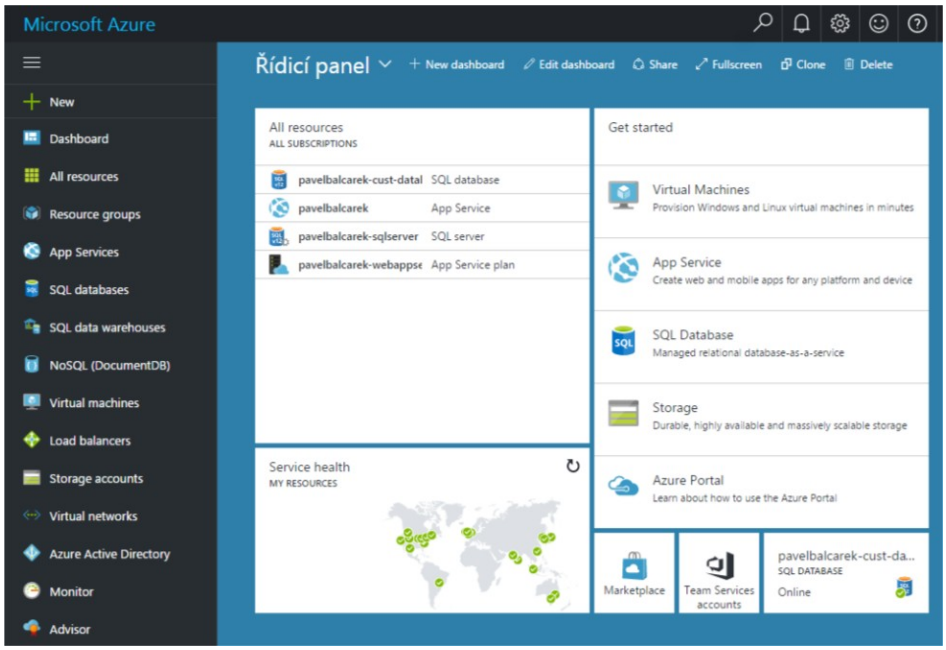
VSTS je mířen především na agilní metody vývoje, které jsou založeny na principu inkrementálního vývoje a dodávání. Pro ty VSTS poskytuje potřebné nástroje. Jedním z těchto nástrojů jsou nástroje pro plánování a sledování práce. Poskytnuté nástroje umožňují vytvářet pracovní položky (Work items). Pracovní položka obsahuje popis plánované práce, přiřazenou osobu, která má položku na starosti a odhad doby práce. Po dokončení práce je k pracovní položce přiřazen jeden či více *commit*-ů (v terminologii VSTS *check in*). Pracovní položce je přiřazen i její stav (např. to do, in progress, testing, done) a kategorie (product backlog item, feature, bug). [38, 37]

Druhým nástrojem na podporu agilních technik jsou nástroje pro kontinuální integraci (CI) a dodávku (CD). VSTS tyto nástroje označuje jako *Build* a *Release*. Build a Release nástroje umožňují vytvářet definice složené z jednotlivých úkolů (Tasks). Každý úkol představuje jednotlivou akci, např. sestavení či provedení unit testu. K dispozici jsou jak předdefinované úkoly, tak je možnost vytvářet vlastní. Příklad předdefinovaných úkolů je na Obr. PII.1, konkrétní úkol s možnostmi nastavení je na Obr. PII.2. Úkoly jsou členěny do kategorií *Build* (sestavení), *Utility* (nástroje), *Test* (testování), *Package* (balíčkování) a *Deploy* (nasazení). Úkoly lze použít v Build i Release definicích bez ohledu na jejich kategorii. [39]

6.3 Nástroje pro správu

Azure nabízí několik nástrojů pro správu služeb (zdrojů). Tyto nástroje nabízí možnosti jako vytváření, konfigurace, spouštění či zastavení služeb. Za všechny lze jmenovat:

- *Portál Azure*. Webové rozhraní, náhled je na Obr. 6.7.
- *Visual Studio a Azure SDK*. Vývojové prostředí které ve spojení s Azure SDK nabízí nástroj *Cloud Explorer*. Cloud Explorer je určen pro správu zdrojů.
- *Azure PowerShell*. Rozšíření skriptovacího jazyka umožňující provádět správu a operace se zdroji. Vhodné pro automatizaci opakujících se operací.
- *Azure Command-line interface (Azure CLI)*. Node.js aplikace umožňující podobné možnosti jako Azure PowerShell, jen je tento nástroj multiplatformní. [40]



Obr. 6.7 Webové rozhraní portálu Azure

6.4 Azure Resource Manager

Azure Resource Manager (Azure RM, ARM) poskytuje způsob nasazení zdrojů. Azure RM umožňuje definovat šablonu (ARM template) obsahující potřebné zdroje, jejich parametry a vzájemné závislosti. Šablona je uložena v souboru ve formátu *JavaScript Object Notation* (JSON). Šablona umožňuje vyčlenit parametry a ty ukládat do samostatného souboru, též ve formátu JSON. Vyčlenění parametrů umožňuje šablonu použít opakovaně s různými parametry. Nasazení zdrojů je provedeno v rámci jedné operace. Zdroje jsou sdružovány do skupin (resource group), což umožňuje následnou hromadnou správu. Šablonu je možno upravit a provést její opětovné nasazení, Azure RM zajistí aktualizaci alokovaných zdrojů. Díky zmiňovaným vlastnostem je vhodný pro automatizaci nasazení zdrojů. Příklad vytvoření úložiště: [40]

```
{
  "$schema":
    "http://schema.management.azure.com/schemas/2015-01-01/
       deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "newStorageAccountName": {
      "type": "string",
      "defaultValue": "mystorage",
      "metadata": {
        "description": "Unique DNS Name for the Storage
                        Account where the Virtual Machine's disks
                        will be placed."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "West US",
      "allowedValues": [
        "West US",
        "East US" ],
      "metadata": {
        "description": "Restricts choices to where premium
                        storage is located in the US."
      }
    }
  },
  "resources": [ {
    "type": "Microsoft.Storage/storageAccounts",
    "name": "[parameters('newStorageAccountName')]",
    "apiVersion": "2015-06-15",
    "location": "[parameters('location')]",
    "properties": {
      "accountType": "Standard_LRS"
    }
  } ]
}
```

II. PRAKTICKÁ ČÁST

7 Stávající řešení, motivace

Na začátek se podíváme jaký je současný stav, jaká byla motivace k vytvoření této práce a definujeme si jakých cílů chceme dosáhnout. Pracuji ve společnosti zabývající se tvorbou software. V současné době dodáváme softwarová řešení, která jsou z větší části postavena na on-premise architektuře. Větší část je hostována na infrastruktuře zákazníků, malá část na vlastní infrastruktuře. Hosting na zákazníkově infrastruktuře má původ především v historii, kdy se o cloudovém řešení ještě neuvažovalo. Hosting na vlastní infrastruktuře je spíše dočasným řešením.

Z hostingu u zákazníka vyplývá, že každý zákazník má svoji vlastní instanci - je tedy použit *single-tenant* model. Instalace na zákazníkově infrastruktuře z pohledu dodavatele software s sebou přináší jisté komplikace. Komplikovanější je především správa a aktualizace software či řešení problémů. Nezřídka jsme pak stavěni do role servisních techniků zákazníkovy sítě, i přes fakt, že s naším softwarem vzniklý problém nesouvisí. Řešením je tedy instalace do prostředí, které bude z pohledu správy dostupnější. Vlastní infrastruktura řeší problém správy, ale je nutno řešit kapacitní problémy (nákup a obnova hardware, síťové kapacity atd.). Cílem je tedy postupný převod zákazníků do cloudového prostředí.

Hlavní řešení, které dodáváme, je postaveno na .NET technologiích od společnosti Microsoft. Využíváme též ostatní produkty od této společnosti, zejména Team Foundation Server (TFS) a od nedávné doby i jeho cloudovou variantu Visual Studio Team Services (VSTS). Visual Studio slouží jako primární vývojové prostředí a SQL Server jako databázové úložiště. TFS/VSTS využíváme jako verzovací systém a zároveň jako prostředí pro sestavení. Toto sestavení je pak nutno konfigurovat a nainstalovat.

Typický projekt se skládá z datové vrstvy (databáze), logické vrstvy, komunikační vrstvy a aplikace v podobě desktopové nebo webové aplikace. Komunikační vrstva je postavena na *Windows Communication Foundation* (WCF) frameworku.

8 Zadání

Tato práce si klade dva cíle. Prvním očekávaným výstupem této práce je vytvořit systém kontinuální dodávky (CD) tak, jak byla definována v kapitole 4.1. Kontinuální dodávka bude distribuována do cloudových prostředí pro jednotlivé zákazníky v Azure. Každý zákazník bude mít (ve většině případů) dvě prostředí - jedno produkční a jedno akceptační. Do akceptačního prostředí bude dodávka provedena automaticky, do produkčního až po schválení. Vytvoření prostředí v Azure by mělo být automatické a součástí dodávky. Při vytváření zákazníků a jejich prostředí by měl být kladen důraz na maximální jednoduchost a minimalizaci chyb ze strany uživatele.

Druhým cílem je prozkoumat možnosti použitých služeb především z pohledu správy a monitoringu.

Součástí této práce je ukázkový systém, na kterém budou demonstrovány definované cíle. Jako téma systému jsem zvolil jednoduchý fakturační systém. Uživatel bude moci spravovat tři základní agendy: klienti, zboží a faktury. Agendy budou mezi sebou provázané - uživatel bude vytvářet faktury a přiřazovat jim zboží. Faktury bude uživatel přiřazovat jednotlivým klientům. Klient bude obsahovat kontaktní údaje (jméno, příjmení, adresa, e-mail, telefon). Zboží bude obsahovat identifikaci výrobku (výrobce, typ) a údaje o ceně a stavu skladových zásob. Faktura bude obsahovat informace o klientovi a seznam položek faktury. Položka faktury bude obsahovat výrobek, cenu a počet kusů výrobku.

Ukázkový systém bude postaven na vícevrstvé architektuře. Prezenční vrstva bude reprezentována tenkým klientem. S logickou vrstvou bude spojena přes komunikační kanál. Pro persistenci dat bude využita relační databáze.

8.1 Rozsah práce

Hlavním cílem práce je vytvořit systém kontinuální dodávky. Tento systém dodávky je předveden na vytvořeném ukázkovém systému. Ukázkový systém je tedy po obsahové a funkční stránce zjednodušen pouze pro potřeby demonstrace dodávky.

8.2 Požadavky

V této kapitole jsou rozebrány požadavky na ukázkový systém. Jejich specifikace slouží jako podklad pro další fázi vývoje a při porovnání s výsledným systémem v akceptačním procesu. Požadavky jsou děleny na *funkční* a *nefunkční*. Vzhledem k faktu, že jde o ukázkový systém pro demonstraci systému kontinuální dodávky, je zde větší důraz kladen na nefunkční požadavky.

8.2.1 Funkční požadavky

Funkční požadavky specifikují funkce, které má výsledný systém umět. Požadavky jsou rozděleny do kategorií.

Klienti

- vedení seznamu klientů
- vyhledání klientů
- nastavení kontaktních údajů
- zobrazení přiřazených faktur
- zobrazení součtu částek přiřazených faktur

Zboží

- vedení seznamu zboží

Faktury

- vedení seznamu faktur
- zobrazení nejčastěji nakupujících klientů (dle počtu nákupů)
- zobrazení nejvíce nakupujících klientů (dle součtu částek přiřazených faktur)

8.2.2 Nefunkční požadavky

Nefunkční požadavky v tomto případě kladou omezení a způsob řešení.

- vícevrstvý systém, minimálně obsahující vrstvy: datovou, logickou a prezentační
- pro persistenci dat bude využita relační databáze
- využití vhodného frameworku umožňující *objektově relační mapování* (ORM)
- ORM musí podporovat změny v datovém modelu
- prezentační vrstvu bude tvořit desktopová aplikace
- desktopová aplikace bude využívat architektury *tenký klient*
- desktopová aplikace bude podporovat možnost aktualizace
- desktopová aplikace bude komunikovat s logickou vrstvou pomocí WCF frameworku

9 Analýza

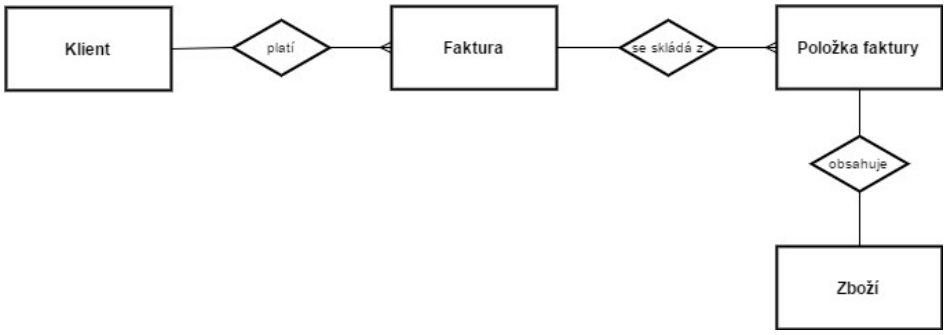
Tato kapitola obsahuje *datovou* a *funkční analýzu* demonstračního systému.

9.1 Datová analýza

Ze zadání jsme získali čtyři entity, které budou popisovat systém po datové stránce. Jsou to:

- *Klient.*
- *Zboží.*
- *Faktura.*
- *Položka faktury.*

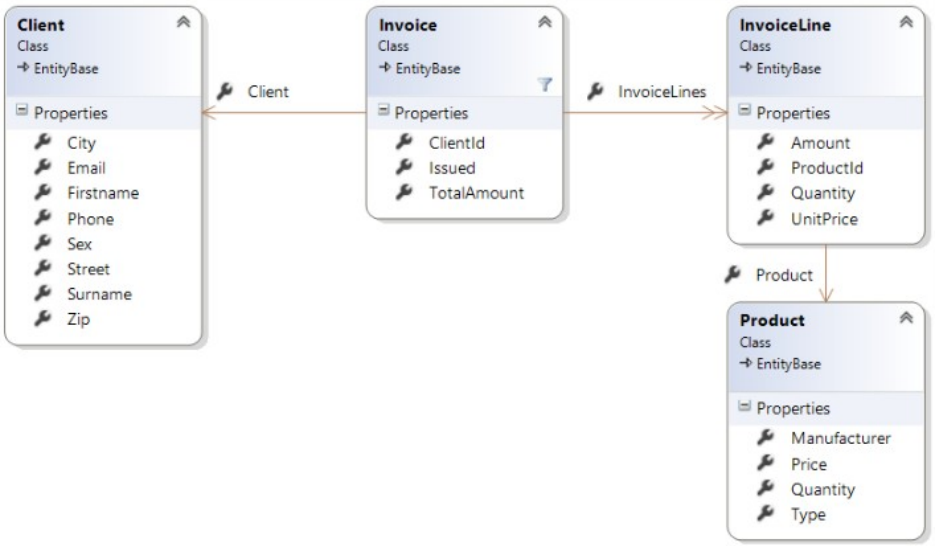
Vzájemný vztah entit je zobrazen na Obr. 9.1. Vzhledem k jednoduchosti zadání je na Obr. 9.2 uveden rovnou *diagram tříd.*



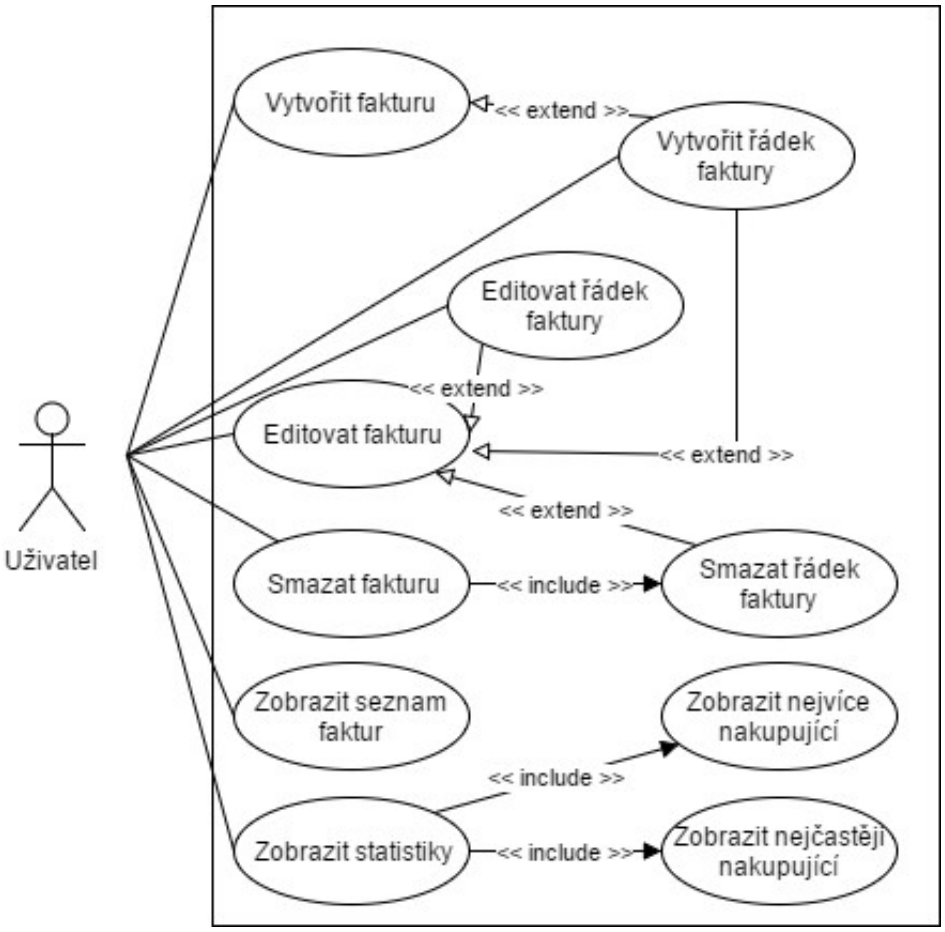
Obr. 9.1 ER diagram demonstračního systému

9.2 Funkční analýza

Pro popis funkcí systému si vystačíme s diagramem případů užití. V systému bude figurovat pouze jeden aktér - uživatel, který bude spravovat všechny tři agendy. Na Obr. 9.3 je zobrazen diagram případů užití pro práci s fakturami. Ostatní případy užití (klient, zboží) a jednotlivé scénáře jsou vzhledem k jednoduchosti a hlavnímu cíli práce vynechány.



Obr. 9.2 Class diagram demonstračního systému



Obr. 9.3 Případy užití - faktury

10 Návrh implementace

V této části životního cyklu aplikace dochází k řešení systémového návrhu a vlastního návrhu implementace.

10.1 Systémový návrh

Ukázkový systém bude naprogramován v jazyce C# za využití *.NET Framework*-u 4.5. Pro spuštění desktopové aplikace bude tedy vyžadován HW a SW kompatibilní s uvedeným frameworkem.

Kromě desktopové aplikace bude zbytek ukázkového systému nasazen v cloudovém prostředí Azure. V něm je nutno specifikovat použité služby. Z nefunkčních požadavků vyplývá, že budeme potřebovat relační databázi - ta bude tvořit jednu z využitých služeb. Pro mapování objektů bude využit *Entity Framework* (EF). Tyto objekty jsou v terminologii EF nazývány *entitami*. Jednotlivé entity jsou reprezentovány třídami. Soubor těchto entit je nazýván *datovým modelem*.

EF podporuje tři základní přístupy k datovým modelům. Liší se především výchozím bodem a použitými nástroji. Pokud máme již existující databázi, je vhodné použít přístup *database first*. Při tomto přístupu jsou z databáze získány všechny potřebné informace a z nich dochází k vygenerování entit. Druhým způsobem je návrh modelu v designeru Visual Studia. Z něj jsou poté vytvořeny jednotlivé entity a na jejich základě je vytvořena odpovídající databáze. Tento přístup je nazýván *model first*. Poslední možností je přístup *code first*. U code first jsou nejprve napsány třídy popisující entity. Z nich je poté vytvořena databáze. V demonstračním systému bude z uvedených přístupů použit poslední zmiňovaný - code first. EF podporuje také změnu datového modelu. Tyto změny jsou reprezentovány *migracemi*.

Dále máme specifikovanou desktopovou aplikaci hostovanou jako tenký klient. Tento požadavek nepřímo definuje i existenci logické vrstvy systému, kterou je též třeba někde hostovat. Z požadavku na podporu aktualizací desktopové aplikace plyne nutnost hostovat i aktualizační data.

Použité cloudové služby:

- *Azure App Service - Web Apps*. Logická vrstva společně s hostováním aktualizačních dat.
- *Azure SQL Database*. Relační databáze.

Vybrané služby jsou nabízené v několika výkonnostních a cenových kategoriích. Obě služby nabízejí bezplatnou základní variantu, která je dostačující pro vývojové a testovací účely. [33]

Pro komunikaci desktopové aplikace s logickou vrstvou byl v zadání specifikován WCF framework. [42] Zbývá vyřešit požadavek na podporu aktualizace desktopové aplikace, resp. na systém distribuce aktualizací. Pro splnění tohoto požadavku bylo zvoleno využití technologie *ClickOnce*. ClickOnce mj. umožňuje vytvářet automaticky aktualizovatelné aplikace. [43]

10.2 Vlastní návrh implementace

V této fázi dochází k návrhu jednotlivých funkčních modulů. Význačné moduly a jejich zařazení do vrstev je zakresleno na Obr. 10.1. Následuje výpis všech použitých modulů s krátkým popisem.

Obr. 10.1 Rozdělení funkčních modulů do vrstev

10.2.1 Prezentační vrstva

WinClient Desktopová aplikace, slouží k prezentaci dat a interakci s uživatelem.

10.2.2 Komunikační vrstva a moduly sdílené mezi více vrstvami

Contracts Rozhraní s definovanými kontrakty, tedy operacemi které jsou implementovány v *Services*. Definicí konktraktu se rozumí metoda (GetClients), předávané parametry (GetClientsRequest) a návratová hodnota (GetClientsResponse).

Interfaces Rozhraní sdílené napříč projektem.

Services Obálka pro implementace služeb. Tímto modulem projdou veškeré požadavky než dojde k jejich zpracování v aplikační vrstvě. Zde je např. možnost autorizovat autora požadavku.

Services.Host Logika pro hostování služby a její konfigurace (WCF služba).

10.2.3 Aplikační vrstva

DataServices Aplikační logika - zde je samotná implementace nedefinovaných kontraktů. Jinými slovy - je zde sestaven dotaz, který bude následně vykonán v databázi.

10.2.4 Datová vrstva

DataContext Tento modul umožňuje přístup k namapovaným entitám. Využívá Entity Framework. V tomto modulu je možno provádět operace před uložením entit do persistentní vrstvy - např. specifické validace entit, které nelze jednoduše provést v samotné databázi.

Entities Mapované objekty.

10.2.5 Základní moduly

Core Jádro systému, podpora pro vytváření spojení s WCF službou, práce s výjimkami.

Database Připojení k databázi, správa transakcí.

10.2.6 Pomocné nástroje

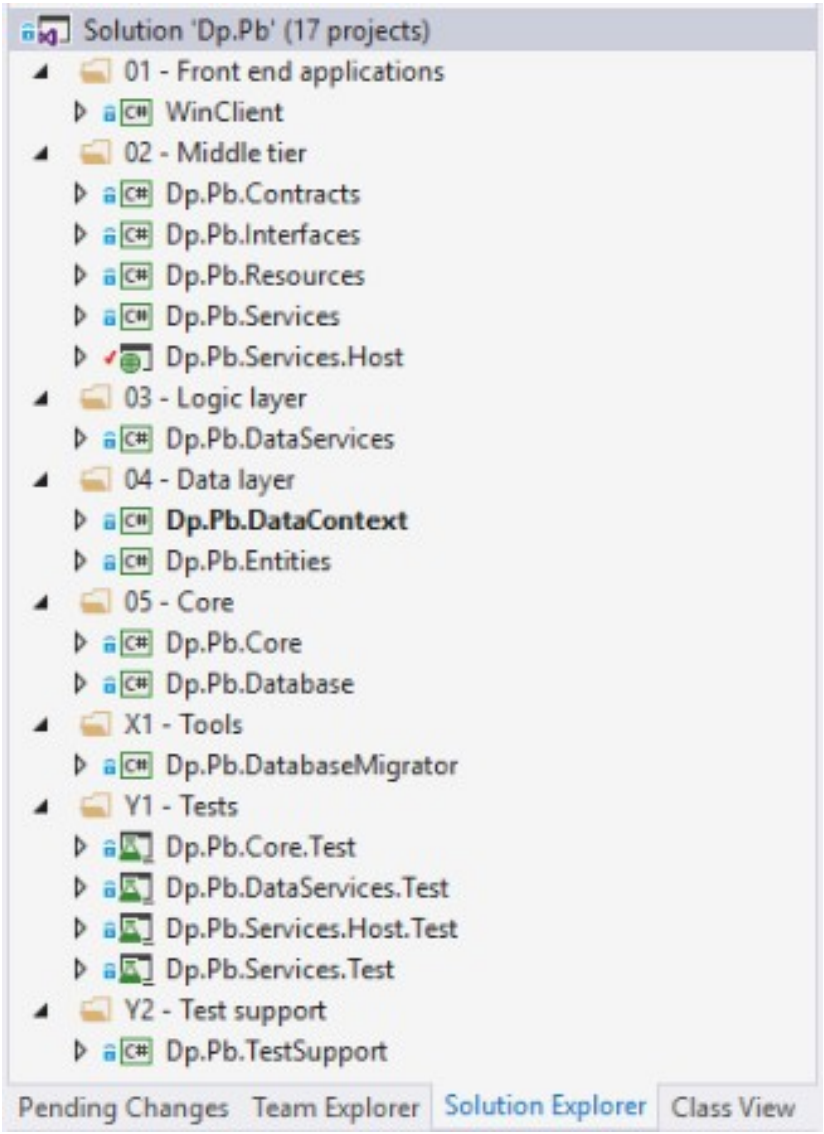
DatabaseMigrator Nástroj pro migraci databáze (např. po změně atributu entity).

10.2.7 Testy a podpora pro testování

Tato kategorie obsahuje jednotkové a integrační testy. Testy jsou rozděleny do modulů dle příslušnosti testované části (DataServices.Test testuje modul DataServices).

11 Implementace

Jako vývojové prostředí bylo zvoleno *Visual Studio 2015 Community Edition* (Visual Studio, VS). Jednotlivé moduly navržené v kapitole 10.2 odpovídají projektům vytvořeným ve Visual Studiu. Všechny projekty jsou seskupeny do jednoho *solution* - soubor s koncovkou *.sln*. Náhled všech projektů v panelu *Solution Explorer* ve Visual Studiu je na Obr. 11.1.



Obr. 11.1 Náhled projektů v panelu Solution Explorer

Při vývoji bylo třeba řešit použití databáze. Při práci v jednom člověku je možno už při vývoji použít *Azure SQL Database*. Praktičtější je však využít lokální databázi, v tomto případě hostovanou v *Microsoft SQL Server 2014 Express*. Lokální databáze poskytuje lepší odezvy a při týmovém vývoji umožňuje vývojáři provádět úpravy v databázi, aniž by při práci nad sdílenou databází ovlivnil ostatní členy v týmu.

V této fázi byla využita jiná služba poskytovaná v rámci Azure - Visual Studio Team Services (VSTS, více viz kapitola 6.2.3). Z této služby byl použit verzovací systém (Team Foundation Version Control), pracovní položky pro plánování práce a nástroje pro sestavení (Build).

Vývojové prostředí a testovací databázový server byly zvoleny především dle osobních preferencí. Tyto volby nemají vliv na zbytek práce a je možno je nahradit adekvátními nástroji dle preferencí konkrétního vývojáře.

VSTS bylo zvoleno především díky tomu, že je již v současné době využíváno v rámci společnosti pro kterou práce vznikala. To znamená, že není nutno současné vývojáře přeškolovat a učit novým věcem. Díky tomu by také měla být snazší adaptace na navržený systém dodávky.

12 Testování

Souběžně s implementací bylo průběžně prováděno testování. Z počátku bylo testování omezeno jen na jednotkové testy, s přibývajícími moduly došlo k rozšíření o integrační testy. V práci jsou především demonstrovány přístupy k testování a jednotlivé typy testů. Testy využívají knihoven *Microsoft.VisualStudio.QualityTools.UnitTestFramework* (Unity) a *Moq*.

12.1 Jednotkové testy

Jako příklad jednotkového testování může sloužit následující test. Součástí tohoto testu je i demonstrace použití *mocks*. Vybraný test také demonstruje, že jednotkový test nemusí být založen jen na volání metody a porovnání výsledku. Před testem samotným nejprve jeho inicializace:

```
[TestInitialize]
public void TestInitialize()
{
  /* Mock<IPbConnectionTransactionContext>,
     Mock<IServiceBaseTestableImplementation> */
  _pbConnectionTransactionContextMock =
    new Mock<IPbConnectionTransactionContext>();
  _serviceImplementationMock =
    new Mock<IServiceBaseTestableImplementation>();
  /* ServiceBaseTestable */
  _SUT = new ServiceBaseTestable(
              _pbConnectionTransactionContextMock.Object,
              _serviceImplementationMock.Object);
}
```

Jsou vytvořeny dvě instance typu *Mock<IPbConnectionTransactionContext>* a *Mock<IServiceBaseTestableImplementation>*. Tyto instance umožňují simulování reálných objektů. Chování těchto objektů je možno v průběhu testu kontrolovat a přizpůsobit potřebám testu. V tomto konkrétním testu slouží jen k vytvoření instance testované třídy, protože jsou očekávány jako parametry konstruktoru třídy *ServiceBaseTestable*.

Instance *_SUT* (zkratka System Under Test - testovaný systém) je testovaná instance třídy *ServiceBaseTestable*, která je potomkem třídy ServiceBase. Tento test ověřuje, že při volání protected metody *ExecuteRequestInternal* abstraktní třídy *ServiceBase* dojde k volání metody *VerifyConsumer* (abstraktní metoda třídy *ServiceBase*).

Třída *ServiceBaseTestable* je implementována následovně (zkráceno):

```
public class ServiceBaseTestable : ServiceBase
{
  public int VerifyConsumerCallsCount { get; set; }
  protected override void VerifyConsumer()
```

```
    {
      VerifyConsumerCallsCount++;
    }
    public virtual TResponse CallExecuteRequestInternal(/* ... */)
    {
      return base.ExecuteRequestInternal(/* ... */);
    }
  }
```

Metoda *CallExecuteRequestInternal* volá metodu *ExecuteRequestInternal* z bázové třídy (tedy metodu, kterou chceme testovat). Ve třídě *ServiceBaseTestable* je přepsána metoda *VerifyConsumer*, která inkrementuje *VerifyConsumerCallsCount* při jejím volání.

Výsledný test pak vypadá následovně:

```
  [TestMethod]
  public void ExecuteRequestInternal_VerifyConsumerShouldBeCalled()
  {
    // assign
    var request = string.Empty;
    Func<string, string> func = (s) => { return s; };
    // act
    var count = _SUT.VerifyConsumerCallsCount;
    _SUT.CallExecuteRequestInternal(func, request, () => "xxx");
    // assert
    Assert.AreEqual(count + 1, _SUT.VerifyConsumerCallsCount);
}
```

Test tedy poté již jen zavolá *CallExecuteRequestInternal* a porovná hodnotu *VerifyConsumerCallsCount*.

12.1.1 Moduly třetích stran

Většina systémů není složena pouze z vlastních naprogramovaných modulů, ale integruje též cizí moduly - moduly, které naprogramoval někdo jiný. Takové moduly nazýváme *moduly třetích stran*. V tomto demonstračním systému to je například Entity Framework, Unity (dependency injection - vkládání závislostí mezi moduly bez nutnosti reference v době sestavení) či log4net (framework pro logování).

Tyto moduly mohou být testovány autory samotnými (u projektů s dostupným zdrojovým kódem je tato skutečnost jednoduše zjistitelná). Nabízí se tedy otázka, zda a do jaké míry tyto moduly testovat. Je pravděpodobně zbytečné snažit se duplikovat již existující jednotkové testy napsané autorem samotným. Jak již ale bylo zmíněno (viz kapitola 1.5.1), testy mohou sloužit také jako funkční dokumentace. Z toho důvodu je minimálně na zvážení (především u nově používaných modulů), zda se nevyplatí provést alespoň základní jednotkové testy.

12.2 Integrační testy

V integračních testech dochází k testování systému jako celku - tedy vzájemné interakci modulů. Některé integrační testy vyžadují pro svoji funkčnost například připojení k databázi či jinou důležitou komponentu, která je dostupná až po nasazení systému. V případě lokálního testování (na počítači vývojáře) jsou testy spouštěny a jsou použity komponenty, které jsou dostupné během vývoje (např. lokálně nainstalovaný SQL Server). V případě provádění testů automaticky je nutno zajistit dostupnost těchto komponent.

Jako příklad jsou zvoleny dva testy. Oba provádí testy na metodě *CreateClient* definované v kontraktovém rozhraní *IClientContractService*. Účelem této metody je vytvoření klienta. Nejprve se zaměřme na společné body obou testů. V obou případech je vytvořeno simulované spojení za pomoci *ChannelFactory* s *IClientChannel* na straně klienta a *ServiceHost* na straně hostované služby (WCF). Ačkoliv mohou být parametry spojení stejné jako jsou používány při provozu demonstračního systému, nejsou předmětem těchto testů.

Podívejme se nyní na tělo prvního zmiňovaného testu:

```
// arrange
var request = new CreateClientRequest();
// act
try
{
  /* _SUT is instance of IClientContractService */
  _SUT.CreateClient(request);
  Assert.Fail("Exception should be thrown");
}
catch (Exception e)
{
  /* check if e is expected type */
}
```

Tento zdánlivě jednoduchý test ověřuje, že není možné uložit klienta bez vyplněného příjmení (modul *Entities* obsahuje v entitě *Client* u vlastnosti *Surname* atribut *Required*). V našem případě tedy požadavek putoval pomocí vytvořeného simulovaného spojení k modulu *Services* a dále do *DataServices*. Vedlejším efektem testu je prokázaní, že parametr metody *GetClients* se podařilo úspěšně serializovat. *DataServices* obsahuje aplikační logiku pro metodu *CreateClient*. Dochází k vytvoření entity *Client* (bez vyplněného příjmení) a přidání této nové entity do seznamu entit typu *Client* nacházejícího se v modulu *DataContext*. Následuje pokus o uložení přidané entity voláním *SaveChanges* (nad *DataContext*-em). Dalším krokem řetězce volání je *DbContext.SaveChanges*, tedy metodou z modulu Entity Framework. Zde dojde k ověření validity entity dle definovaného datového modelu a vyvolání výjimky. Tato validační výjimka je propagována zpět až k testu samotnému a zkontrolována, zda byla vrácena

očekávaná výjimka. Při propagování výjimky je zapojen také modul *Core*, který se stará o transformaci výjimek do přenositelného tvaru (do tvaru, který je možné přenést zpět na klienta). Zapojen byl také modul *Database*, který vytvořil připojení k databázi, které však v tomto případě nebylo nutno pro ukládání entity použít.

Druhý, zde uvedený test, je obdobou prvního, jen s tím rozdílem že jsou poskytnuty všechny potřebné údaje a dojde k uložení entity do databáze. Podívejme se blíže na na třídu *CreateClientRequest*, která slouží pro předání parametrů při vytváření klienta:

```
[DataContract]
public class CreateClientRequest : RequestBase
{
  [DataMember]
  public string Firstname { get; set; }
  [DataMember]
  public string Surname { get; set; }
}
```

Nyní na samotnou entitu *Client*:

```
[DataContract(IsReference = true)]
[Table("Client")]
public class Client : EntityBase
{
  [DataMember]
  public string Firstname { get; set; }
  [Required(ErrorMessage = "Surname is required")]
  [DataMember]
  public string Surname { get; set; }
  [DataMember]
  public Sex Sex { get; set; }
}
```

Na první pohled vidíme, že vlastnost *Sex* není možno metodou *CreateClient* přiřadit. Tuto potencionální chybu je minimálně obtížné testováním objevit - už jen z toho důvodu, že ji zde nemusíme chtít nastavovat (a vytvořit např. metodu *SetClientSex*). Co však lze snadno testovat je správné vytvoření entity *Client* - tedy zda *CreateClient-Request.Firstname* je rovno *Client.Firstname* (a obdobně u *Surname*). Po uložení by tedy mělo dojít k načtení uložené entity a k porovnání s ukládanými hodnotami.

12.3 Automatizace testů a databáze

V kapitole 1.5.5 byla zmíněna kritéria, která by měl splňovat test vhodný k automatizaci testování. K uvedeným kritériím patřila opakovatelnost a nezávislost. V případě testů využívajících datovou vrstvu (přístup k databázi) je nutno tato kritéria též splnit. Vezměme si například test zapisující nastavení aplikace do tabulky. Jistě je rozumným požadavkem, aby každá položka nastavení (například adresa poštovního serveru) byla v databázi pouze jednou. Vytvořme následující fiktivní test:

```
// arrange
var smtpSetting = new Setting {
                        Key = "SMTP",
                        Host = "smtp.example.com" };
// first time should be without problems
DataContext.Settings.Add(smtpSetting);
DataContext.SaveChanges();
// act, assert
DataContext.Settings.Add(smtpSetting);
try
{
  DataContext.SaveChanges();
  Assert.Fail("Exception expected.");
}
catch
{
  // exception was thrown, test passed
}
```

Test proběhl jednou, v prvním případě položka neexistovala, a proto byla přidána, v druhém případě došlo k očekávanému chování. Byla vyvolána výjimka, a k uložení nedošlo. Při druhém volání testu nad stejnou databází dojde k vyvolání výjimky už při prvním pokusu o uložení, a test skončí neúspěšně. Test tedy není opakovatelný. Pokud bude před tímto testem spuštěn jiný test, který již toto nastavení vytvoří, nebude tento test úspěšný ani při prvním pokusu o uložení. Takový test je závislý na ostatních testech.

Řešením by jistě bylo před každým testem nejprve zjistit, zda již je hodnota uložena a pokud ano, tak ji smazat. V tomto jednoduchém případě to jistě možné je, ale při složitějších testech mohou nároky na „úklid“ značně narůstat. Ani tento úklid by však neřešil situaci, kdy by došlo ke spuštění testů paralelně (více testů spuštěných najednou). Správným řešením je tedy provádět tento úklid před každým testem. Toho je dosaženo využitím atributů metod *[TestInitialize]* a *[TestCleanup]* poskytované frameworkem *Unity*. Nabízí se několik možností, jak tento úklid provádět, například:

- *Smazání a znovu vytvoření databáze.*
- *Smazání všech dat.*
- *Využití transakcí.*

Každý z uvedených příkladů má své výhody i nevýhody. Při *smazání a znovu vytvoření databáze* nemůže dojít k situaci, kdy by databáze obsahovala neočekávaná data. Před začátkem testování je vždy ve stejném stavu. Vytváření databází a jejich struktury je však časově náročné.

Smazání všech dat s sebou přináší jinou komplikaci. Pokud chceme mazat data, musíme kvůli dodržení referenční integrity znát pořadí v jakém data mazat. Druhou mož-

ností je nejprve referenční integritu odstranit (smažeme veškeré definice cizích klíčů), smazat data a následně obnovit referenční integritu databáze.

V ukázkovém systému byla zvolena poslední zmiňovaná metoda - *využívání transakcí*. Pokud situaci velmi zjednodušíme, tak každá kontraktová třída (např. *ClientContractDataService*) jako parametr konstruktoru očekává připojení k databázi. Při otevření připojení dojde zároveň k vytvoření transakce. Připojení (společně s otevřenou transakcí) je použito při přístupu k data kontextu (k datům), který využívá kontraktová třída. Stejného přístupu je využito i při testování - na začátku každého testu (*TestInitialize*) je vytvořeno spojení a započata transakce. Po dokončení testu (*TestCleanup*) je proveden rollback - databáze je vrácena do stavu před vytvořením transakce.

13 Manuální nasazení

Tato kapitola se zabývá manuálním nasazením systému do zvoleného prostředí. Způsobů manuálního nasazení je několik. Cílem není uvést všechny způsoby, ale položit základ pro analýzu automatizovaného nasazení.

Využitý přístup kombinuje využití portálu *Azure* a vývojového prostředí *Visual Studio*. Portál slouží k *vytvoření a konfiguraci služeb*, Visual Studio je použito pro *sestavení* systému a jeho *nasazení*.

13.1 Portál Azure

Začneme s vytvořením služeb na portálu Azure (dále jen portál). Pro navigaci v portálu je možno využít levé menu. Menu obsahuje odkazy na nejběžněji používané zdroje (služby) a také tlačítko *Přidat*. Po kliknutí na tlačítko *Přidat* je k dispozici vyhledávání i seznam dostupných služeb rozdělených do kategorií. Zde vyhledáme nebo nalistujeme požadovanou službu a po stisku na tlačítko *Vytvořit* se otevře krátké nastavení služby.

13.1.1 Resource Group

Nejprve tímto způsobem vytvoříme *Skupinu zdrojů* (služba Resource group), do které budou zdroje sdružovány (viz Obr. PI.1). Skupině je nastaven *název*, defaultní *lokace* a *předplatné* (Subscription).

13.1.2 SQL database

V dalším kroku je třeba vytvořit *databázi* (služba SQL Database). Databázi je nastaven její *název*, *předplatné*, *skupina* a je vybrán existující nebo vytvořen nový *SQL server* na kterém má být databáze umístěna. Důležitou volbou je také výběr plánu, který určuje mj. cenu, výkon a maximální velikost databáze. Kompletní náhled obrazovky se všemi možnostmi konfigurace je na Obr. PI.2. Po potvrzení je zahájeno vytváření SQL serveru a databáze. Tato operace může trvat několik minut. Po dokončení je možno začít databázi používat. Na Obr. PI.3 jsou zobrazeny základní informace o nově vytvořené databázi. Po kliknutí na odkaz *Show database connection strings* jsou zobrazeny informace o připojení k databázi. Tyto informace budou třeba, aby se demonstrační systém mohl k této databázi připojit.

13.1.3 App service

Nyní vytvoříme službu *Web App* pro hostování komunikační a aplikační vrstvy. Postup je obdobný jako při vytváření databáze. Před vytvořením je uživatel opět dotázán na několik základních nastavení (viz Obr. PI.4).

13.2 Visual Studio

Následují kroky vykonané za pomoci *Visual Studia*.

13.2.1 Spuštění testů

Před každým nasazením systému je třeba se ujistit, že funguje bez chyb a tak jak bylo zamýšleno. Pro tyto případy byly vytvořeny jednotkové a integrační testy. Ty by měly být spuštěny před každým nasazením, v ideálním případě ještě před vytvořením zdrojů v prostředí Azure. Testy spustíme z panelu *Test Explorer* (viz Obr. PI.5). Pokud byly testy úspěšné, můžeme pokračovat dále. V opačném případě je třeba nejprve chybu nalézt a opravit, teprve potom je možno pokračovat.

13.2.2 Migrace

Databáze vytvořená v jednom z předchozích kroků je prázdná. Je třeba v ní vytvořit databázovou strukturu (tabulky entit), se kterou bude demonstrační systém pracovat. K vytvoření použijeme projekt *Dp.Pb.DatabaseMigrator* (DM). DM využívá vytvořené migrace *Entity Framework*-em. Ve složce projektu je umístěn souboru *ConnectionStrings.config*. V tomto souboru jsou informace o připojení k databázi. Tento soubor je třeba upravit a nastavit v něm připojení k databázi získané v předchozím kroku. Projekt sestavíme a výslednou aplikaci spustíme. Po spuštění aplikace se automaticky začne provádět migrace. Po dokončení migrace je připravena datová vrstva.

13.2.3 Publikování

V tomto kroku pracujeme s projektem *Dp.Pb.Services.Host*. Stejně jako u předchozího projektu, i zde je nutno provést konfiguraci souboru *ConnectionStrings.config*. Po konfiguraci nahrajeme projekt do služby *Web App*. Proces nahrání projektu se nazývá *publikování*. Po pravém kliku na projekt v panelu *Solution Explorer* se rozbalí kontextové menu, ve kterém zvolíme položku *Publish*. Je otevřen průvodce procesem publikování. Pokud byl již projekt někdy v minulosti publikován přeskočí průvodce na poslední krok. Průvodce je tvořen čtyřmi kroky:

- Výběr cíle publikování, v tomto případě *Microsoft Azure App Service* (viz Obr. PI.6). Po tomto výběru se otevře nové okno. Zde je na výběr z existujících služeb nebo možnost založit novou (viz Obr. PI.7). Vybereme vytvořenou v minulé kapitole.
- Rekapitulace připojení ke službě na kterou chceme publikovat.
- Konfigurace publikování. Zde volíme konfiguraci sestavení a možnosti publikování souborů.

- V posledním kroku je zobrazen náhled souborů, které budou publikovány s možností některý odvybrat. Ponecháme a volíme publikovat.

Projekt je před publikováním sestaven. Průběh sestavení i publikování je dostupný v panelu *Output*. Po dokončení publikování se v internetovém prohlížeči otevře úvodní stránka právě publikovaného webu. Projekt žádnou webovou stránku neobsahuje, je proto zobrazena defaultní stránka vytvořená při zakládání služby. Tu je možno později nahradit vlastní stránkou. Pro nás je důležitější stránka `/WinAppService.svc` (v ukázkovém případě je tedy celá adresa `http://demogroup.azurewebsites.net/WinAppService.svc`). Tato stránka tvoří bod, ke kterému se bude aplikace připojovat. Pokud bylo publikování úspěšné, měli bychom vidět podobnou stránku jako je na Obr. PI.8.

13.2.4 Desktopová aplikace

Posledním krokem je zprovoznění desktopové aplikace. Před jejím použitím je třeba nastavit bod, ke kterému se má připojovat (viz 13.2.3) - tzv. endpoint. Nastavení je provedeno editací *xml* souboru *App.config*. Získaný bod vyplníme do *atributu* address uzlu *configuration/system.serviceModel/client/endpoint*. Nyní již stačí aplikaci sestavit a spustit.

14 Kontinuální dodávka

Stejně jako demonstrační systém, tak i systém kontinuální dodávky prochází fázemi vývoje.

14.1 Analýza požadavků

Kapitola 8 obsahovala zadání systému kontinuální dodávky demonstračního systému jednotlivým zákazníkům. Předchozí kapitola (Kapitola 13) obsahovala nutné kroky pro nasazení demonstračního systému. Informace získané v těchto kapitolách jsou dále shrnuty a vytvořeny jednotlivé požadavky na systém kontinuální dodávky.

Požadavky získané ze zadání doplněné o krátký komentář:

- *Více zákazníků a více prostředí u každého zákazníka.* Tento požadavek mj. určuje použití *single-tenant* architektury. Každý zákazník bude mít svoji instanci systému, počet instancí bude roven počtu prostředí. Multi-tenancy by bylo komplikované i v rámci jednoho zákazníka, jelikož v každém prostředí může být nasazena jiná verze demonstračního systému a tedy i jiný datový model. Konsolidace by byla možná až na úrovni SQL Serveru.
- *Možnost automatického dodání demonstračního systému.* Ihned po vytvoření nové verze demonstračního systémů může dojít k jeho dodání do vybraných prostředí.
- *Schvalování dodávek.* Některé dodávky budou dodány až po schválení - opak předchozího.
- *Vytvoření prostředí je součástí dodávky.* Prostředí není nutno vytvářet předem; pokud neexistuje, bude vytvořeno během dodávky.
- *Jednoduchost a odolnost vůči chybám.* Nefunkční požadavek, pro jeho splnění si lze představit například situaci, kdy je demonstrační systém dodán s minimální interakcí uživatele.

Požadavky získané během nasazení:

- *Vytvoření odpovídajících služeb v Azure.* Tento bod odpovídá bodu *Vytvoření prostředí je součástí dodávky.* V tomto případě se jedná o *Resource Group*, *SQL database* a *App service.*
- *Sestavení demonstračního systému.*
- *Spuštění jednotkových a integračních testů.* Další kroky jsou podmíněny úspěšným dokončením tohoto kroku.

- *Konfigurace demonstračního systému.* Před spuštěním migrace i publikováním je nutno obojí konfigurovat. Konfigurace se provádí zapsáním hodnot do konfiguračních souborů (soubory s příponou *config*).
- *Migrace databáze.*
- *Publikování demonstračního systému.* Tento požadavek představuje publikování všech zbývajících vrstev, tedy i publikování - distribuování desktopové aplikace.

14.1.1 Datová analýza

Jako v každém systému, i zde je nutno určitým způsobem pracovat s daty. Data v tomto systému jsou vztažena k prostředí a je možno je dělit do dvou skupin:

- *Konfigurační data prostředí v Azure.* Obsahuje data o tom, jak je který zdroj konfigurován v prostředí Azure. Součástí této konfigurace je např. název služby *Web app* (resp. její adresy, která je z názvu služby tvořena), cenový plán či geografická lokalita. Obdobné informace jsou použity i pro službu *SQL database*. Pro *Resource group* je to její název a lokalita.
  Pro tuto skupinu dat bude využita šablona pro *Azure Resource Manager*.
- *Konfigurační data demonstračního systému.* V demonstračním systému jde především o nastavení endpointů (pro připojení k databázi, desktopová aplikace ke službě), ale mohou být lehce rozšířeny o data vztahující se obecně k danému prostředí zákazníka. Komponenty demonstračního systému čtou tyto hodnoty z konfiguračních souborů (soubory formátu *xml* s příponou *config* - *app.config* pro desktopové aplikace a *web.config* pro webové aplikace). Zákazník například bude mít požadavek, aby u desktopové aplikace bylo na první pohled patrno ke kterému prostředí (akceptační, produkční) patří. Do konfigurace se přidá barva okna desktopové aplikace a ta se podle této hodnoty bude podbarvovat.
  Formát a způsob uložení těchto dat je specifikován v kapitole 14.2.2.

14.2 Návrh implementace

Tato kapitola obsahuje *návrh implementace*. Návrh implementace je složen ze *systémového návrhu* a *vlastního návrhu implementace*.

14.2.1 Systémový návrh

V této části je nejprve třeba zvolit nástroje, které budou využity pro dodávku. V kapitole 11 již bylo nastíněno použití *VSTS* především jako verzovacího systému a systému

pro sestavení. Vytvořené sestavení sloužilo především pro kontrolu, zda verzovací systém obsahuje vše potřebné a že sestavení je možné provést i jinde než na počítači, na kterém byl demonstrační systém vyvíjen. Nic nebrání tomu vytvářet sestavení jinde. V případě nedostatků v použitém verzovacím systému je možno nahradit i jej, je však preferováno na tomto řešení setrvat.

Cílem práce není „znovu objevovat kolo“. Systémy pro kontinuální dodávku existují, jde tedy spíše o to, který bude nejvíce vyhovovat hodnotícím kritériím. Výběr takového systému je komplexní záležitost a hodnotí se vícero kritérií. Mezi hodnotící kritéria patří nejen splnění definovaných požadavků, odhady nákladů a úspor, ale také např. uživatelská přívětivost, aktuálnost, podpora či dokumentace. Na tomto místě bych se rád odkázal na moji bakalářskou práci, která se touto problematikou podrobněji zabývá.

Následují příklady systémů, které podporují kontinuální dodávku. Všechny zmíněné systémy umožňují propojení s VSTS, nasazení do cloudových prostředí včetně Azure a konfiguraci nasazovaných systémů (projektů). Systémy jsou seřazeny v abecedním pořadí:

- *BuildMaster*. Nástroj pro automatizaci sestavení a kontinuální dodávku.
  Domovská stránka: `https://inedo.com/buildmaster`

- *Octopus Deploy*. Systém určený pro integrační úlohy a správu nasazení. Neobsahuje nástroje pro sestavení. Cílen na platformu .NET.
  Domovská stránka: `https://octopus.com/why`

- *TeamCity*. Integrační systém podporující široké spektrum nástrojů pro sestavení projektů v Java (Ant, Maven, Gradle), .NET (MSBuild, Poweshell, NAnt), Ruby (Rake, Bundler) a dalších, které jsou dostupné přes rozšíření. Umožňuje pokročilé konfigurace projektů včetně vytváření jejich hierarchie (konfigurace předka může být využita v potomkovi). Dovoluje seskupování úkolů sestavení (build tasks) do znovupoužitelných skupin.
  Domovská stránka: `https://www.jetbrains.com/teamcity/features/`

- *VSTS*. Popis a základní vlastnosti tohoto systému byly zmíněn již v kapitole 6.2.3.

Pro použití byl zvolen *VSTS*, který již obsahuje většinu potřebných funkcí bez nutnosti dodatečné konfigurace. Požadované funkce jsou pokryty nabízenými úkoly (Tasks) v nástrojích *Build* a *Release*.

Celému procesu *kontinuální dodávky* předchází proces *kontinuální integrace*. Takto jsou i rozděleny použité nástroje - *Build* slouží pro kontinuální integraci, *Release* pro kontinuální dodávku. Výsledkem *Build* je sestavená a otestovaná aplikace, výsledkem

Release je pak její nasazení na konkrétní prostředí zákazníka. Vhodné úkoly pro použití v build definici:

- *Get sources.* Zkopíruje potřebné soubory z verzovacího systému do zadané složky (checkout).
- *NuGet Installer.* Instalace balíčků referencovaných projekty (např. Entity Framework).
- *Visual Studio Build.* Sestavení celého *Solution* či jednotlivých projektů.
- *Visual Studio Test.* Spuštění testů.
- *Publish Build Artifacts.* Publikování výsledků build definice do zadaného umístění.

Mezi vhodné úkoly pro release definici patří:

- *Azure Resource Group Deployment.* Umožní vytvořit zdroje v Azure podle zadané šablony (viz kapitola 6.4).
- *Azure App Service Deploy.* Slouží k nasazení aplikace do Web App.

Pokud by úkoly nevyhovovaly, je možno vytvořit vlastní, či použít obecné úkoly jako spuštění powershellového skriptu či dávky.

14.2.2 Vlastní návrh implementace

V této kapitole je uveden návrh implementace pro uchovávání *konfiguračních dat prostředí* a *konfiguračních dat systému.*

Konfigurační data prostředí V této fázi je nutno vyřešit jakým způsobem uchovávat konfigurační data zákazníků. Šablony pro Azure RM jsou uchovávány v souboru JSON. Úkol *Azure Resource Group Deployment* kromě šablony umožňuje též nastavení parametrů šablony ze souboru. Šablony jsou sdíleny mezi zákazníky a obsahují pouze obecné parametry potřebné pro vytvoření skupiny a služeb, které do ní patří (SQL Database a Web apps). Konkrétní parametry jsou dosazeny z parametrů šablony. Šablony se liší například plánem poskytovaných služeb či přednastaveným škálováním. Parametry šablon obsahují například adresu *Web apps.*

Konfigurační data systému Druhý typ dat, která je třeba uchovávat jsou konfigurační data (demonstračního) systému, tedy ta která budou použita v konfiguračních souborech. Jednou z možností je použití transformací. [44] Aby bylo možno tyto transformace využít je třeba definovat novou *Solution Configuration*. Každá Solution Configuration obsahuje vlastní konfigurační soubory ve kterých jsou specifikovány konfigurační hodnoty. Hodnoty nemusí být specifikovány přímo, ale je možno použít transformací s využitím Solution Configuration. Při sestavení je specifikován název Solution Configuration a podle něj dojde k nahrazení hodnot v konfiguračním souboru. V tomto řešení však spatřuji několik nevýhod:

- Konfiguraci je nutno provádět již ve vývojovém prostředí (což znesnadňuje použití pro ne-vývojáře).
- Při větším počtu konfigurací se může stát orientace mezi jednotlivými konfiguracemi nepřehledná (konfigurace jsou vedeny jako seznam).
- Transformace jsou primárně určeny pouze pro *web.config* (na jiné je možno je rozšířit pomocí doplňků do VSTS [1]),
- Transformace se provádí již při sestavení.

Vzhledem k uvedeným nevýhodám byl zvolen jiný přístup. Základem tohoto přístupu je tzv. *šablona konfiguračního souboru* (soubor s příponou .TEMPLATE.config). Její struktura je totožná se samotným konfiguračním souborem, ale místo hodnot obsahuje pouze zástupné hodnoty. Soubor vypadá např. takto:

```
<?xml version="1.0" encoding="utf-8"?>
<connectionStrings>
  <add name="[DataContext]"
       connectionString="[ConnectionString]"
       providerName="System.Data.SqlClient" />
</connectionStrings>
```

Zástupné hodnoty zde tvoří řetězce *[DataContext]* a *[ConnectionString]*. Hodnoty jsou získány ze souborů *defaultConfig.xml*, resp. *customerConfig.xml*. V obou případech jde o xml soubory tvořené páry zástupná hodnota - skutečná hodnota. První zmiňovaný soubor (*defaultConfig.xml*) obsahuje defaultní hodnoty, tedy hodnoty společné pro většinu zákazníků, druhý soubor obsahuje specifické hodnoty pro konkrétního zákazníka. Příklad souboru *defaultConfig.xml* obsahující hodnotu pro *[DataContext]*:

[1] Je možno využít např. doplňků:
`https://marketplace.visualstudio.com/items?itemName=WillBuikMSFT.SlowCheetah-XMLTransforms`
`https://marketplace.visualstudio.com/items?itemName=GolanAvraham.ConfigurationTransform`

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <!-- [DataContext] = zastupna hodnota -->
  <!-- PbDataContext = skutecna hodnota -->
  <config find="[DataContextName]" replace="PbDataContext" />
</configuration>
```

Spojovací článek tvoří powershellové skripty *CallUpdateAllTemplateConfigs.ps1* a *UpdateConfig.ps1*. Dosazení hodnot za zástupné hodnoty je provedeno voláním prvního powershellového skriptu, který očekává tři parametry:

- Cestu ve které hledat šablony konfiguračních souborů.
- Cestu k souboru *defaultConfig.xml*.
- Cestu k souboru *customerConfig.xml*.

Tyto skripty prohledají zadanou složku a najdou všechny šablony konfiguračních souborů. Dále vytvoří seznam zástupných hodnot a skutečných hodnot získaných z *defaultConfig.xml*. Stejný seznam je vytvořen i ze souboru *customerConfig.xml*. Tyto seznamy jsou spojeny, pokud existují stejné zástupné symboly v obou souborech, jsou použity ty z *customerConfig.xml*. Do každé šablony konfiguračního souboru jsou dosazeny skutečné hodnoty za zástupné a z koncovky souboru je odstraněno *.TEMPLATE* (nový název je např. *ConnectionStrings.config*). Takto vzniká z šablony konfigurační soubor.

Tento přístup přináší výhody v tom, že sestavení je „generické“ a konkrétní hodnoty je možno dosadit až při nasazení. Další výhodou je možnost hierarchického členění hodnot - *default*, *customer*, přičemž není problém tuto hierarchii rozšířit o další stupeň na *default*, *customer*, *environment*. Nevýhodou je, že při dosazení může vlivem špatné hodnoty dojít k narušení xml struktury konfiguračního souboru. Tuto situaci lze částečně řešit pokusem o načtení konfiguračního souboru po každé dosazené hodnotě. Pokud by hodnota soubor poškodila, nebyla by použita a zástupná hodnota by nebyla nahrazena. Některé hodnoty ale musí být nahrazeny (např. informace o připojení k databázi).

Struktura uložení dat Projekt bude mít následující strukturu:

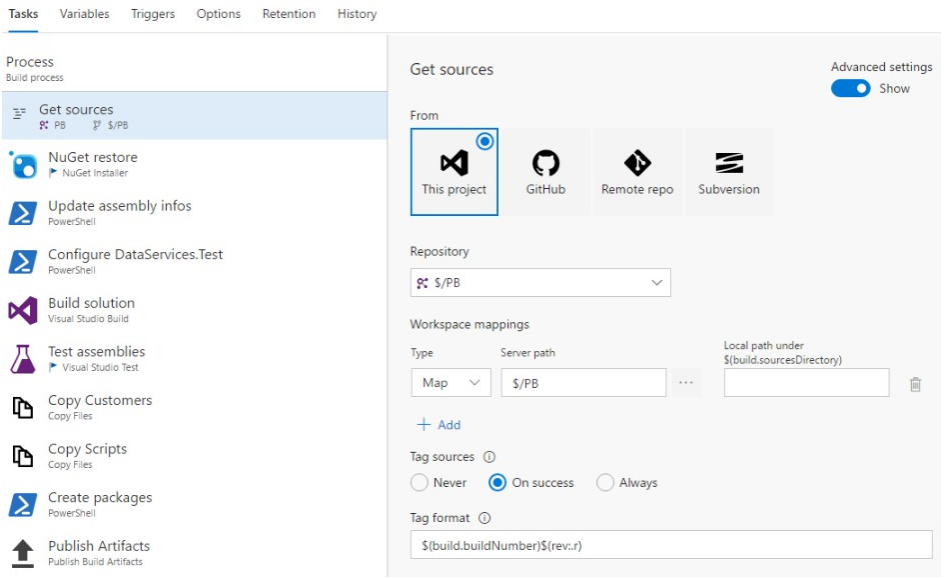
```
Customers   Konf. data prostředí a konfigurační data dem. systému
  _Build    Data pro integrační testy
  _Common   Defaultní a sdílená data
  CustomerA Data pro zákazníka "CustomerA"
    DEV     ... DEV prostředí
    PROD    ... PROD prostředí
  CustomerB Data pro zákazníka "CustomerB"
    DEV     ...
    PROD    ...
    ...     Data dalších zákazníků
Net         Zdrojové soubory
Scripts     Podpůrné skripty
```

14.3 Implementace

V této kapitole je popsána *Build definice* sloužící pro kontinuální integraci a *Release definice* sloužící pro kontinuální dodávku.

14.3.1 Build definice

Build definice má za úkol sestavit demonstrační systém, otestovat jej a v případě úspěchu umožnit provést dodávku. Náhled build definice ve VSTS je na Obr. 14.1. Definice je složena z úkolů, které jsou uvedeny na kartě *Tasks*. Kromě úkolů lze nastavit proměnné přístupné z jednotlivých úkolů (karta *Variables*). V tomto případě jsou nastaveny proměnné *VersionMajor* a *VersionMinor* určující verzi sestavení. Dalším nastavením je specifikace, kdy se má definice spouštět. Definici lze spouštět automaticky po provedené změně (Continuous Integration), v naplánovaný čas (Scheduled) či pouze manuálně. Výstup z build definice je obecný a není vázán na konkrétního zákazníka. Následuje stručný popis úkolů.

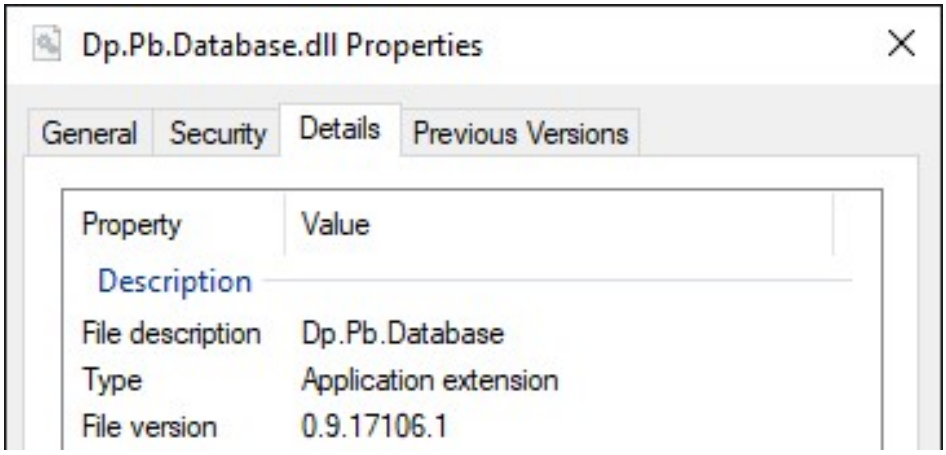


Obr. 14.1 Stránka build definice

Get sources Při sestavení se nepracuje přímo se soubory v repozitáři, ale jsou staženy (checkout) do pracovní složky. Repozitář není třeba stahovat celý, ale lze zvolit jeho část.

NuGet restore Repozitář neobsahuje NuGet balíčky referencované v *Solution*. Tento krok stáhne všechny balíčky, aby šlo *Solution* sestavit. Pokud by byly balíčky již součástí repozitáře, bylo by možné tento krok přeskočit.

Update assembly infos Úkol volá powershellový skript. Skript aktualizuje údaje o verzi v souborech *AssemblyInfo.cs*. Po sestavení je nastavená verze čitelná ve vlastnostech souboru (viz Obr. 14.2). Verze se skládá ze čtyř čísel oddělených tečkou. První dvě čísla jsou získána z proměnných definice (*VersionMajor*, *VersionMinor*). Třetí číslo udává den sestavení ve formátu *yyDoy* (*yy* - poslední dvě číslice roku, *Doy* - den od začátku roku). Poslední číslo udává pořadí sestavení v daném dni.



Obr. 14.2 Náhled verze ve vlastnostech souboru

Configure DataServices.Test V tomto úkolu dojde ke konfiguraci dat systému způsobem popsaným v kapitole 14.2.2. Konfigurován není celý systém, ale pouze modul *DataServices.Test*. Tento modul slouží pro integrační testování a vyžaduje připojení k databázi.

Build solution Provede sestavení *Solution*.

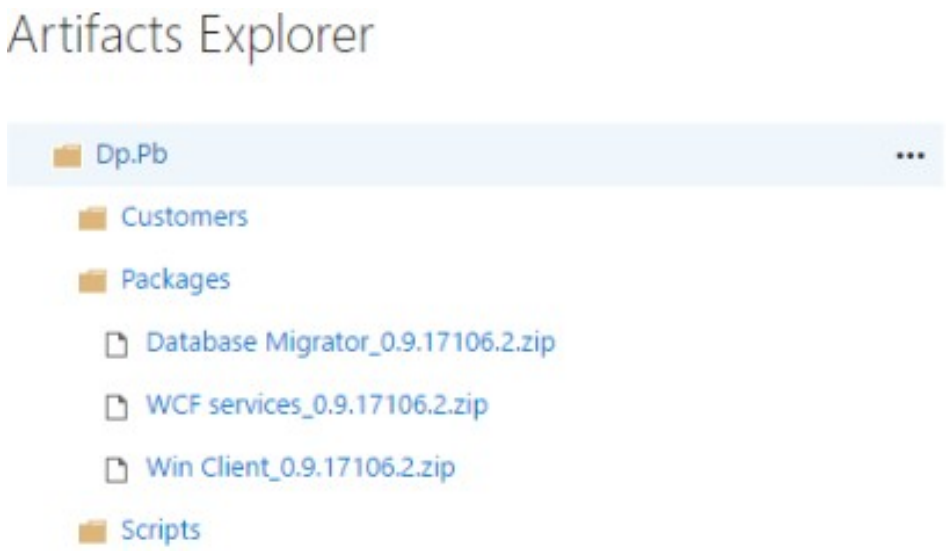
Test assemblies Spustí jednotkové a integrační testy. Integrační testy využívají *Azure SQL Database*. Databázi není díky způsobu testování třeba vytvářet automaticky při každém spuštění testů, byla tedy vytvořena manuálně před zavedením integračního testování.

Copy Customers, Copy Scripts Pomocné úkoly, které zkopírují složky *Customers* a *Scripts* do výstupní složky. Přiložení těchto složek usnadňuje následné použití v *release definici*. Pokud by tyto složky nebyly přiloženy, bylo by nutno při *Release* z repozitáře stahovat verze souborů odpovídající použitému sestavení. Obsah složek (např. skripty) se mohl změnit a se starší verzí sestavení nemusí správně fungovat.

Create packages Komprimuje soubory a složky podle daného předpisu a umístí je do výstupní složky. V tomto kroku jsou komprimovány sestavené moduly demonstračního systému. Příklad předpisu pro komprimaci:

```
<?xml version="1.0" encoding="utf-8" ?>
<packages>
  <package name="Database Migrator">
    <file path="Dp.Pb.DatabaseMigrator\*.dll" />
    <file path="Dp.Pb.DatabaseMigrator\*.config" />
    <file path="Dp.Pb.DatabaseMigrator\Dp.Pb.DatabaseMigrator.exe" />
  </package>
</packages>
```

Publish Artifacts Zveřejní výstup definice (obsah výstupní složky), aby byla dostupná pro použití v *Release*. V našem případě jsou výstupem složky *Customers*, *Packages* a *Scripts*. *Customers* a *Scripts* jsou kopií složek z repozitáře. *Packages* obsahují výsledky sestavení rozdělených do jednotlivých aplikačních modulů. Příklad výstupu je na Obr. 14.3.



Obr. 14.3 Výstup z úkolu *Publish Artifacts*

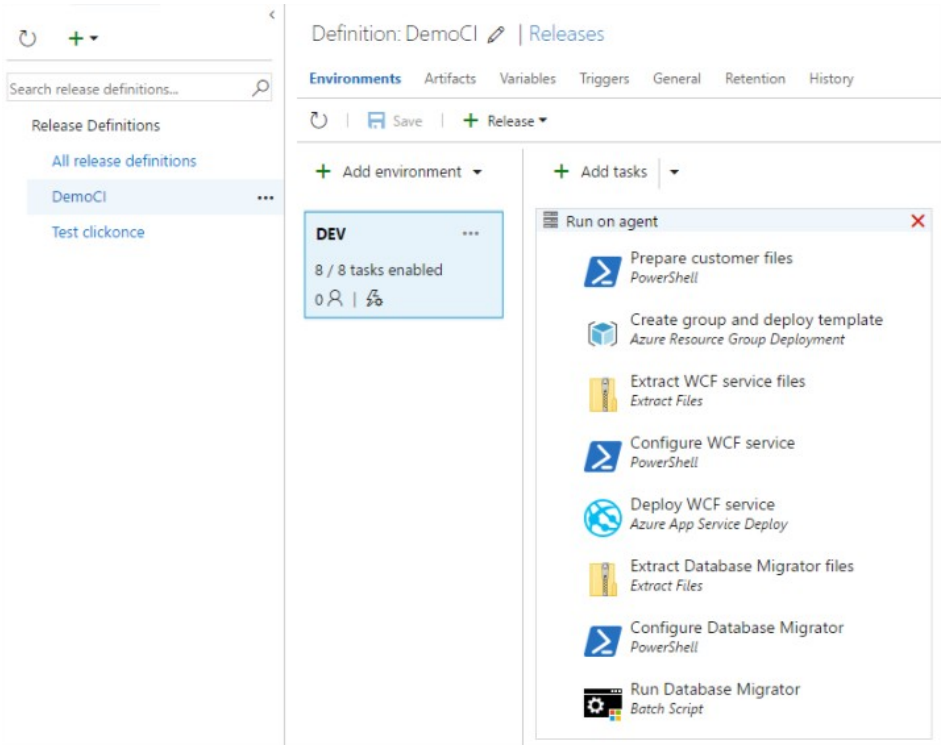
14.3.2 Release definice

Release definice slouží v tomto případě pro dodávku. *Release definice* je velmi podobná *build definici*, liší se však v několika podstatných bodech:

- Vstupem mohou být výsledky z úkolu *Publish Artifacts* (tento vstup je stejně jako v případě *Get sources* stažen do pracovní složky).
- Release definice obsahuje seznam prostředí (Environments), teprve ony obsahují seznam úkolů.

- Je specifická pro konkrétního zákazníka a prostředí (úkoly mohou být pro všechny stejné, lišit se mohou jen parametry úkolů).

Příklad *release definice* je na Obr. 14.4. Tato definice je určena pro zákazníka *DemoCI* a prostředí *DEV*. Název zákazníka odpovídá názvu definice. Konfigurační data tohoto zákazníka jsou uložena ve složce *Customers/DemoCI/DEV*. Obecný předpis pro cestu s konfiguračními daty prostředí daného zákazníka tedy je *Customers/[ReleaseName]/[EnvironmentName]*. Následuje seznam úkolů pro prostředí *DEV*.



Obr. 14.4 Náhled *Release definice*

Prepare customer files Tento úkol spouští powershellový skript *ReleaseInit.ps1*. Skript má za úkol zjednodušit vytváření a práci ostatních úkolů. Úkol sestaví, z obecného předpisu definovaného v předchozím odstavci, konkrétní cestu ke konfiguračním datům a konfigurační data zkopíruje do složky *CustomerSpecific*. Zbylé úkoly nemusí při přístupu ke konfiguračním datům konstruovat celou cestu, ale data vždy najdou ve složce *CustomerSpecific*.

Konfigurační data obsahují i hodnoty, které jsou využívány jako parametry úkolů (např. název *Web app* v úkolu *Deploy WCF Service*). Tyto hodnoty skript nastaví jako proměnné.

Create group and deploy template Úkol nejprve vytvoří *Skupinu* (Resource Group) a v ní vytvoří zdroje definované v šabloně (dle jejich parametrů) v prostředí Azure.

Extract, Configure & Deploy WCF service files Rozbalí, konfiguruje a nasadí systém do služby App Service.

Extract, Configure & Run Database Migrator Rozbalí, konfiguruje a spustí aplikaci, která má za úkol provést migraci databáze. Entity Framework umožňuje provést migraci automaticky při prvním připojení k databázi. Při migraci ale může dojít ke komplikacím (např. existují data se kterýma migrace nepočítá) a migrace se nezdaří. Tuto skutečnost je tedy mnohem lepší zjistit již při dodávce systému.

Extract, Configure WinClient files, Create & Deploy WinClient ClickOnce Rozbalí a zkonfiguruje soubory komponenty *WinClient*. Z těchto souboru vytvoří *ClickOnce* balíček. Balíček je nasazen na Web app.

Instalace se provede stažením a spuštěním souboru `[webappname].azurewebsites.net/ClickOnceWinClient.exe.application`. V demonstračním systému je tedy adresa: `pavelbalcarek.azurewebsites.net/ClickOnce/WinClient.exe.application`. Při každém spuštění aplikace dochází ke kontrole aktualizace.

15 Provoz a údržba

Poslední fází vývoje je provoz a údržba. Tato poslední fáze zabírá až 80% procent životního cyklu. [1] Portál Azure nabízí nástroje, které jsou v této fázi nápomocny. Nástroje jsou členěny do sekcí. Tato kapitola obsahuje seznam těchto nástrojů doplněný o krátký popis. Tento seznam nelze brát jako kompletní výčet, ale jeho cílem je spíše nastínit nabízené nástroje a možnosti.

15.1 Web app

Každá služba v portálu Azure poskytuje úvodní obrazovku se základními informacemi a operacemi. Tato obrazovka se nazývá *Overview* (viz Obr. 15.1). Odtud je možno službu zastavit, restartovat či smazat. Součástí je i graf poskytující rychlý přehled o požadavcích, vytížení CPU, přenesených datech, počtu chyb a dalších informací dle nastavení. *Activity log* zobrazuje seznam událostí - např. spuštění, zastavení, zálohování ale též nové nasazení či incidenty (výpadky, plánované nedostupnosti) služby.

Druhá sekce je věnována nasazení a jejímu základnímu nastavení. Je zde taktéž možnost nastavit *kontinuální dodávku* (Continuous Delivery). Pokud je kontinuální dodávka nastavena, je zde přehled o aktuálně nasazené verzi a verzích dřívějších (viz Obr. 15.2). Tyto informace jsou provázány s VSTS a je zde možno zjistit po jaké změně bylo nasazení provedeno, verzi sestavení a další.

Třetí sekce je věnována nastavení. Tato sekce pokrývá širokou škálu od nastavení aplikačního prostředí, cenového plánu, záloh, vertikálního a horizontálního škálování přes přiřazení domén, ssl certifikátů, až po autentifikaci a autorizaci.

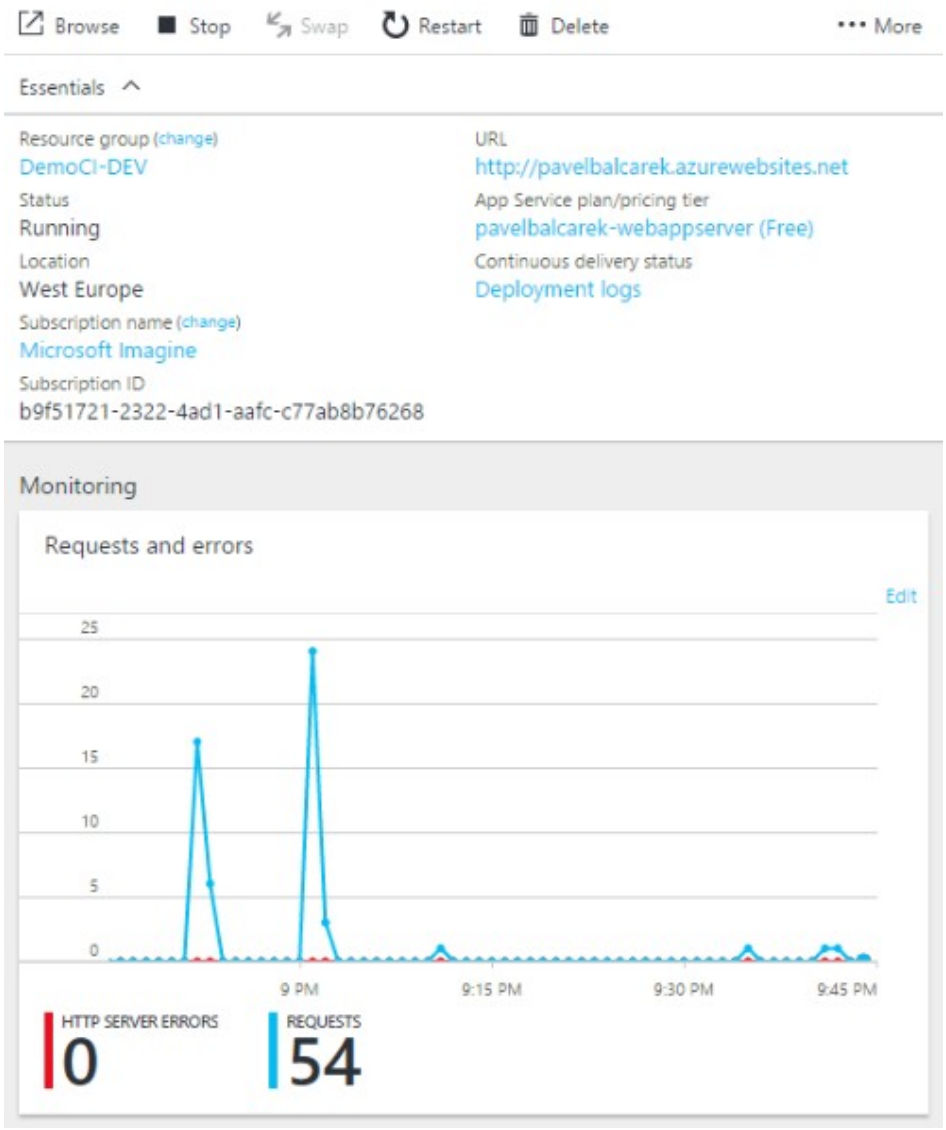
Sekce vývojářské nástroje poskytuje připojení ke vzdálené konzoli (Console), přístup k RESTovému rozhraní (Advanced Tools) umožňující provádět některé operace bez nutnosti přístupu k portálu (vhodné pro automatizaci), testování výkonnosti systému (Performance testing), A/B testování (Testing in production) a správu rozšíření.

Monitoring seskupuje nástroje jako jsou Application Insights, zobrazení běžících procesů (Process explorer), sledování HTTP požadavků (Live HTTP trafic) či nastavení notifikací (Alerts).

15.2 SQL Database

První sekce je opět věnována základním informacím a operacím. K základním operacím zde patří kopírování, smazání, obnova a export databáze. Odtud je také možnost nastavit firewall či získat informace o připojení k databázi (connection strings).

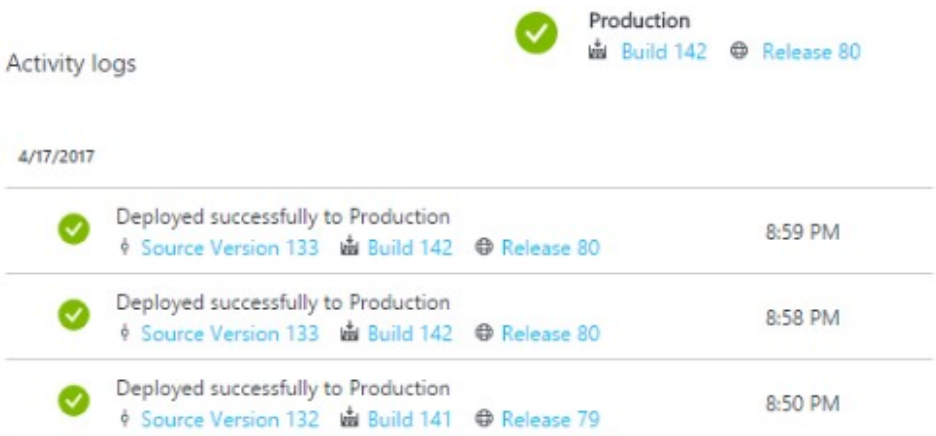
Nastavení kromě volby cenového plánu umožňuje konfigurovat geo-replikaci, šifrování databáze či dynamické maskování dat. Dynamické maskování dat funguje tak,



Obr. 15.1 Web app - *Overview*

že dotazy vykonané nad databází budou vracet maskovaná data. Dynamické maskování lze nastavit pro jednotlivé uživatele, což je možno použít například v situaci, kdy vývojář potřebuje pracovat s databází v produkčním prostředí a produkční prostředí obsahuje data, která by mu měla zůstat skryta. Sekce monitorování obsahuje nastavení diagnostiky a podobně jako u Web app také nastavení notifikací.

Poslední sekce je podpora a řešení problémů. Tato sekce obsahuje část věnovanou výkonnostní otázce. Kromě přehledu o využití zdrojů (Performance overview) obsahuje též výkonnostní doporučení (Performance recommendation) a ladění dotazů (Query Performance Insight). Výkonnostní doporučení obsahuje rady jak optimalizovat výkon úpravou indexů v databázi a parametrizací dotazů. Ladění dotazů poskytuje přehled o nejnáročnějších dotazech. Těmto dotazům by pak měla být věnována větší pozornost a měly by být optimalizovány.



Obr. 15.2 Web app - *Continuous delivery*

16 Rozšíření systému

Tato kapitola je věnována rozšíření demonstračního systému pro širší použití v cloudovém prostředí. Jde především o taková rozšíření, která jsou lehce a rychle implementovatelná či aplikovatelná a přitom přináší nesporné výhody či šetří čas. Nemusí se tak nutně jednat o implementaci nové funkcionality, ale i vhodné použití nabízených služeb v prostředí Azure přinášející podobné výsledky.

16.1 Logování a sběr informací

Jakmile je aplikace „vypuštěna do světa“ je jednou z výzev logování, efektivní sběr a následná analýza těchto informací. V současných projektech využíváme především logovací framework *log4net*. Log4net umožňuje přidávat celou řadu výstupů do kterých budou logované záznamy umístěny. Záznamy mohou být např. ukládány do databáze, souboru, systémového logu či posílány v e-mailových zprávách.

Jedním z výstupů může být také zde již zmíněná služba - *Application Insights* (AI). AI podporuje několik typů profilů přizpůsobených dle aplikace pro kterou zde budou informace shromažďovány. Založení lze provést přes portál (viz kapitola 13.1), ale je možno např. využít i šablonu (ARM template). Aby bylo možno službu využívat, je třeba získat tzv. *Instrumentation Key*. Ten má formát *GUID* a je dostupný v základních informacích o službě na portálu nebo pomocí powershellového skriptu:

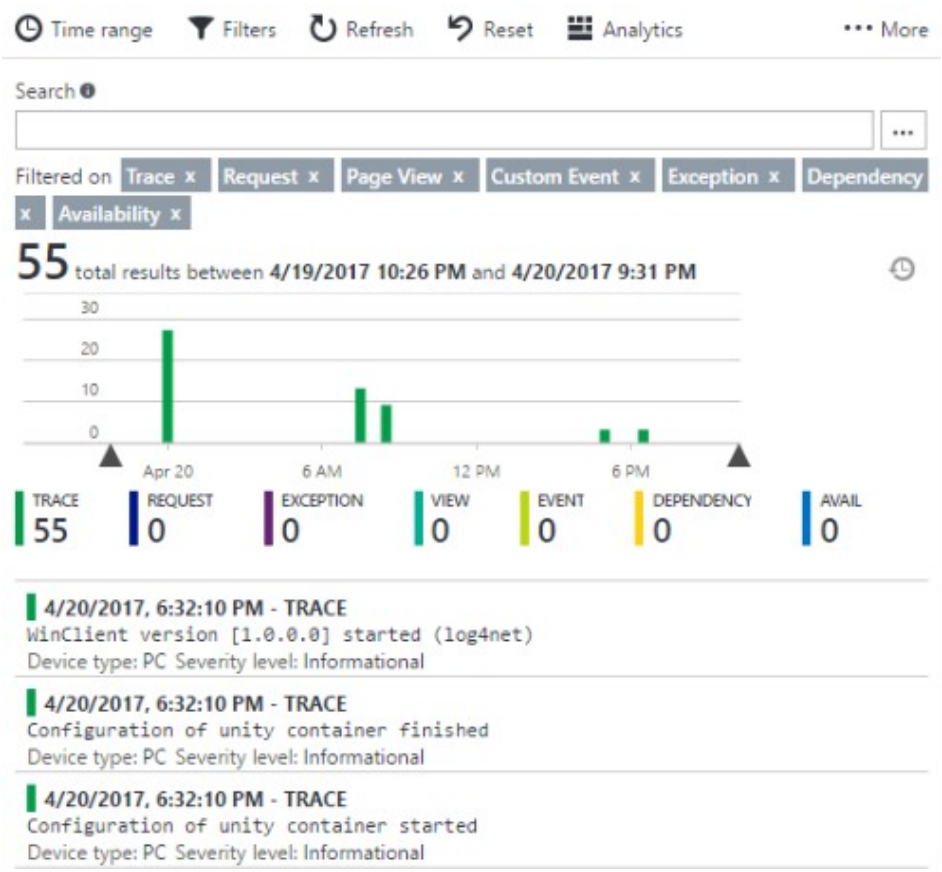
```
$resource = Find-AzureRmResource -ResourceNameEquals "<AI NAME>"
$details = Get-AzureRmResource -ResourceId $resource.ResourceId
$ikey = $details.Properties.InstrumentationKey
```

V demonstračním systému se poté nainstalují NuGet balíčky:

- *Microsoft.ApplicationInsights*,
- *Microsoft.ApplicationInsights.Log4NetAppender*,
- *log4net*.

Vytvoří se konfigurační soubory *ApplicationInsights.config*, *log4net.config* a upraví se *App.config*. Ukázky těchto souborů jsou součástí elektronické přílohy diplomové práce. Příklad logování, jak jsou tyto logy reprezentovány na portále, je pak na Obr. 16.1:

```
// initialization
private static readonly ILog Log =
  LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);
// logging
Log.InfoFormat("WinClient version 1.0.0.0 started (log4net)");
```



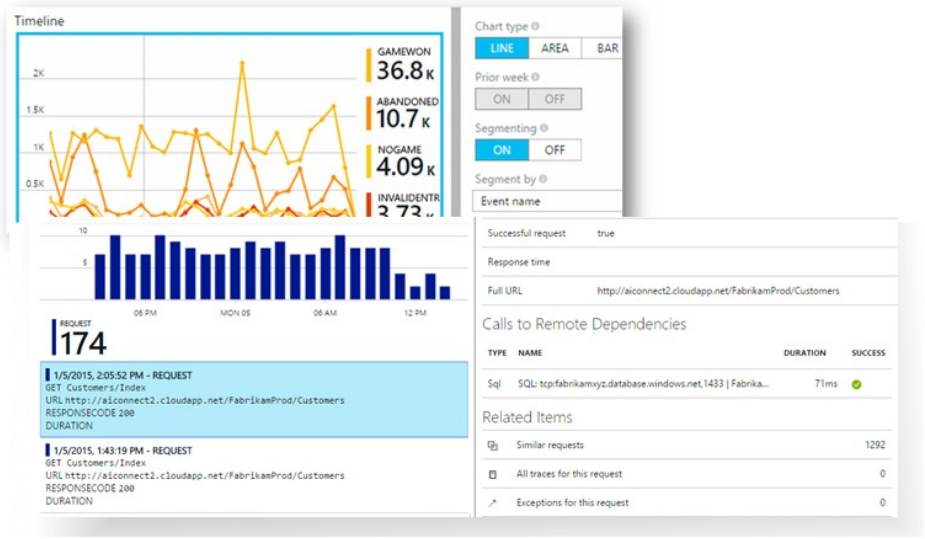
Obr. 16.1 Náhled logů v *Application Insights*

AI ale neslouží pouze pro ukládání a prohlížení logů. Balíček *Microsoft.Application-Insights* obsahuje mj. třídu *TelemetryClient*, která podstatně více rozšiřuje možnosti využití. *TelemetryClient* napomáhá při shromažďování nejrůznějších metrik jako jsou např. počty shlédnutí stránek, délky trvání operací, počty událostí, počty výjimek či počty požadavků. V demonstračním systému se nabízí např. logování délky jednotlivých požadavků (GetClients, GetInvoices,...). Nad získanými metrikami lze provádět dotazy (podobně jako u SQL) za účelem analýzy. Výsledky dotazů lze vizualizovat (viz Obr. 16.2).

16.2 Bezpečnost

Již při návrhu jakéhokoliv systému by mělo dojít na řešení bezpečnostních otázek. Ty zde, vzhledem k zadání a účelu demonstračního systému. řešeny nebyly. Tato kapitola se snaží aspoň částečně tento nedostatek odstranit. Cílem této kapitoly je poskytnout základní návod jak lze při zabezpečení tohoto demonstračního systému postupovat. Dle [46] lze do oblasti počítačové bezpečnosti zahrnout:

- ochranu před neoprávněnou fyzickou manipulací,



Obr. 16.2 Možnosti využití *Application Insights* [45]

- zabezpečení přístupu k datům (autentizace a autorizace),
- ochranu před narušením celistvosti, důvěrnosti a dostupnosti,
- zajištění zálohování,
- ochranu autorského práva,
- zabezpečení komunikace při přenosu dat.

Tato kapitola je zaměřena na dva body: *zabezpečení přístupu k datům* a *zabezpečení komunikace při přenosu*. Oba dva tyto body spolu úzce souvisí. Aby bylo možné zabezpečit přístup k datům je nejprve třeba určit jakým způsobem bude uživatel prokazovat svoji identitu a kdo tuto identitu bude ověřovat. Desktopový klient komunikuje se zbytkem systému přes Internet, je nutno také určit jakým způsobem bude jejich vzájemná komunikace zabezpečena. Pro ukázku je zvolena varianta ověření uživatele pomocí uživatelského jména a hesla. Ověření těchto údajů je v režii demonstračního systému. Komunikace je zabezpečena pomocí protokolu *HTTPS*.

Nejprve je třeba upravit soubor *web.config* modulu *Host.Services*. Do něj je přidáno nastavení chování služby (pod uzel *serviceBehaviors*). Součástí tohoto nastavení je i uzel *userNameAuthentication*. Tento uzel určuje třídu (*PbUserNamePasswordValidator*) a její namespace, která je zodpovědná za autentizaci uživatele. Třída implementuje rozhraní *UserNamePasswordValidator* obsahující metodu *Validate*. Zde dochází k autentizaci uživatele. Pokud uživatel neprošel validací, je vyvolána výjimka. K autorizaci uživatele dochází v třídě *ServiceBase* voláním metody VerifyConsumer. Princip je stejný jako u *Validate* - pokud uživatel není autorizován, je vyvolána výjimka. V demonstračním systému jsou obě metody prázdné a je tedy úspěšně ověřen každý uživatel.

V konkrétní implementaci může dojít např. k vyhledání a ověření záznamu o uživateli v databázi či může být zapojena jiná autorita zodpovědná za toto ověření.

Zbývá vyřešit zabezpečení komunikace. Pro zabezpečení komunikace je využit protokol *HTTPS*. Protokol *HTTPS* vyžaduje pro svoji funkčnost *SSL certifikát*, který je automaticky přiřazován všem *Web app*. Jedná se o tzv. *wildcard certifikát* zabezpečující všechny subdomény na hlavní doméně. Je tedy vystaven pro **.azurewebsites.net*. Toto řešení je dostačující pro testovací prostředí.

V případě použití vlastní domény je nutno přidat vlastní *SSL certifikát*. Certifikát lze zakoupit přímo přes portál Azure nebo importovat vlastní. Alternativně lze využít certifikační autoritu *Let's Encrypt*[1)]. Tato certifikační autorita nabízí SSL certifikáty zdarma. Zdánlivou nevýhodou může být jejich krátká platnost - 90 dní. [47] Pro tyto případy lze využít možnosti *Rozšíření* (Extensions), které *Web app* nabízí. Mezi nabízenými rozšířeními je i doplněk *Azure Let's Encrypt*[2)], které obnovu tohoto certifikátu provede automaticky.

16.3 Vzdálené ladění

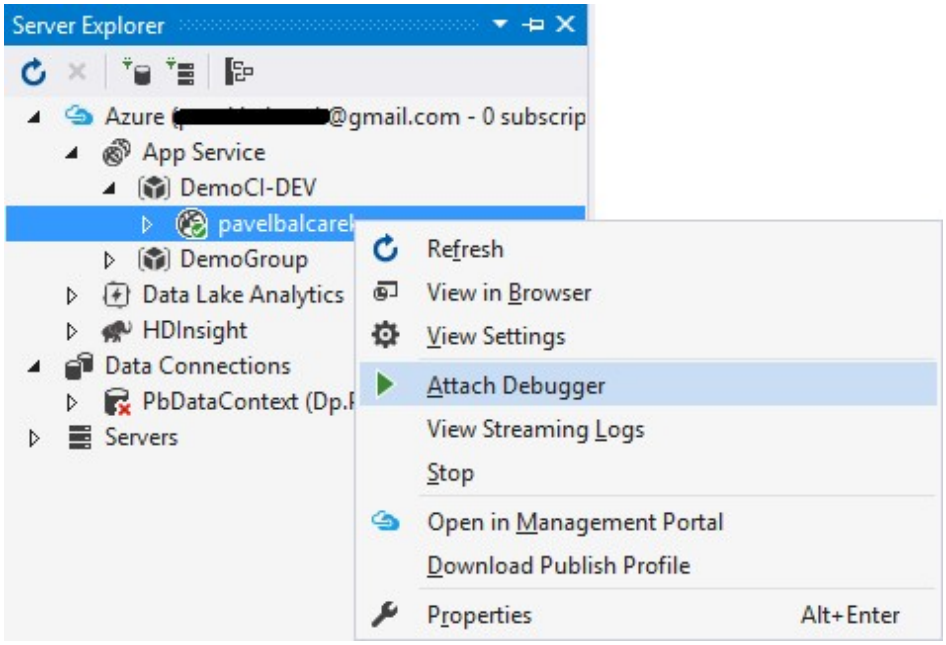
Při provozu aplikací občas může dojít k situacím, kdy se nasazený systém chová nestandardně a toto chování se nedaří nasimulovat na počítači, kde probíhal vývoj. Řešením obvykle bývá zvýšit četnost logování. V některých případech však ani to nemusí pomoci a je nutno přistoupit k jiným řešením. Takovým řešením může být vzdálené ladění. Tuto možnost nabízí i *Web app*. Defaultně bývá vypnuta, je proto nutno ji povolit v *Nastavení* (Application settings). V nastavení je i volba verze Visual Studia, ze které bude vzdálené ladění prováděno.

Ve Visual Studiu se k *Web app* připojuje z panelu *Server Explorer* (viz Obr. 16.3). Připojení by mělo být možné i z panelu *Cloud Explorer*. Přes něj se však připojení nepodařilo uskutečnit. Po připojení probíhá ladění stejným způsobem jako při ladění pouze v rámci lokálního počítače.

Při tomto způsobu ladění je nutno mít na paměti následující:

- *Po dokončení práce vždy vypnout.* Je zbytečné a potencionálně nebezpečné nechávat tuto funkci zapnutou.
- *Pozor na produkční prostředí.* Toto vzdálené ladění by nemělo být prováděno na produkčním prostředí, protože ovlivňuje všechny uživatele aktuálně využívající laděnou *Web app*.

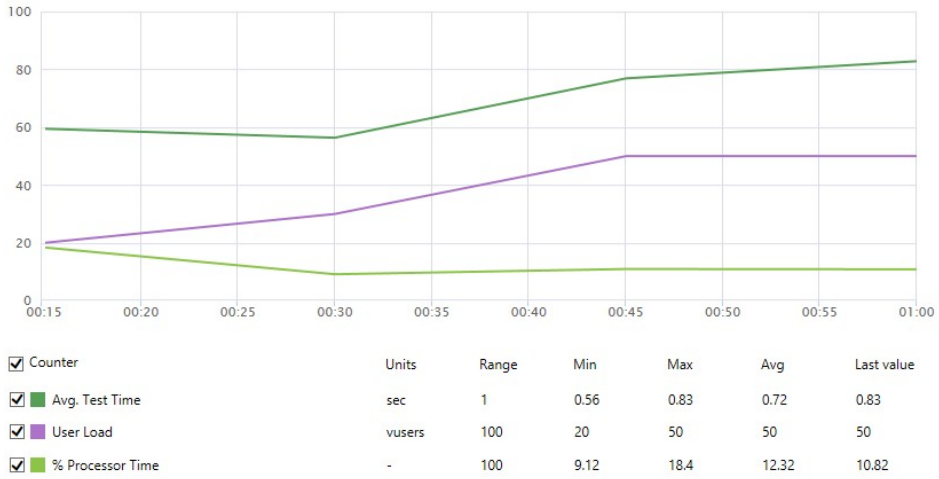
[1)] `https://letsencrypt.org/`
[2)] `https://github.com/sjkp/letsencrypt-siteextension`



Obr. 16.3 Vzdálené ladění Web app ve Visual Studiu

16.4 Výkonnostní testy

Výkonnostní testy jsou další skupina testů, kterou lze s daným systémem provádět. Slouží k simulování chování systému pod zátěží. Visual Studio v edici *Enterprise* obsahuje podporu pro tuto skupinu testů. Jelikož tato edice není v současné době v rámci společnosti dostupná, věnuje se těmto testům práce jen velmi okrajově.



Obr. 16.4 Výstup zátěžového testu ve Visual Studiu

V demonstračním systému jde především o testování WCF služby. Pro testování je vytvořen projekt typu *Web Performance and Load Test Project*. Tomuto projektu je přidána *Service Reference* s adresou nasazeného systému (např. `https://pavelbalcarek.azurewebsites.net/WinAppService.svc`). Dalším krokem je vytvoření jednotlivých

testů, které chceme během zátěžového testu spouštět. Tyto testy jsou vytvářeny stejným způsobem jako v případě jednotkových či integračních testů, jen je možno využít přidané *Service Reference*. Jednotlivý test může vypadat např. takto:

```
[TestMethod]
public void GetClients()
{
  var clientService = new PbAggregateContractServiceClient();
  var response = clientService.GetClients(new GetClientsRequest());
}
```

V zátěžovém testu jsou definovány parametry samotného testu. K základním parametrům patří spouštěné testy, kolik uživatelů má být v testu simulováno a jakou dobu má test běžet. Výstup testu je zobrazen na Obr. 16.4.

ZÁVĚR

Tato práce se zabývala životním cyklem aplikací a cloudovým prostředím *Azure* od společnosti *Microsoft*. V praktické části došlo k navržení a vytvoření demonstračního systému. Příslušné části tohoto demonstračního systému pak byly nasazeny do služeb *Web app* a *SQL Database* nabízených v prostředí *Azure*. Proces nasazení se podařilo automatizovat do takové míry, aby splňoval atributy *kontinuální dodávky*. Hlavní část kontinuální dodávky tvoří služba *Visual Studio Team Services* patřící též k portfoliu nabízených služeb v rámci *Azure*. Služba *Visual Studio Team Services* sloužila nejen jako verzovací systém, ale také jako nástroj pro kontinuální integraci a kontinuální dodávku. Jednotlivé kroky kontinuální integrace a kontinuální dodávky jsou složeny z vytvořených powershellových skriptů a předpřipravených *Úkolů* (pro většinu těchto Úkolů jsou též předpřipraveny ve složce *Scripts* alternativní powershellové skripty).

Pro kontinuální dodávku byl navržen systém uchovávání konfiguračních dat. Tato konfigurační data obsahují nejen konfiguraci demonstračního systému, ale též konfiguraci služeb v prostředí *Azure*. Díky poslednímu jmenovanému není třeba vytvářet tyto služby ručně, ale jsou automaticky vytvořeny (případně upraveny) před samotným nasazením demonstračního systému. Součástí kontinuální integrace jsou též jednotkové a integrační testy. Práce proto obsahuje základní vysvětlení principů těchto testů a možnosti jakým způsobem tyto testy provádět. Demonstrační systém pak obsahuje základní sadu těchto testů.

V závěru práce jsou zmíněny některé možnosti rozšíření demonstračního systému.

Na závěr krátké zhodnocení výsledků této práce. Záměrem této práce bylo zjednodušit nasazení aplikací do produkce, což je často velmi komplikovaný proces. Cíl byl díky specifické znalosti programových nástrojů daného prostředí splněn a proces nasazení se podařilo automatizovat. Automatizace v tomto případě přináší nejen podstatné zrychlení procesu, ale též vede k výraznému snížení chybovosti. Jinak velmi komplikované nasazení se díky navrženému systému změnilo na jednoduchý proces nahrání změny.

SEZNAM POUŽITÉ LITERATURY

[1] ŠARMANOVÁ, Jana. *Databázové a informační systémy* [online]. Ostrava: Vysoká škola báňská - Technická univerzita, 2008 [cit. 2017-03-29]. ISBN 9788024814995. Dostupné z: http://www.elearn.vsb.cz/archivcd/FEI/DAIS/DAIS.pdf

[2] BUREŠ, Zbyněk. *Databázové systémy 2: studijní opora.* Jihlava: Vysoká škola polytechnická Jihlava, 2014. ISBN 9788087035894. Dostupné z: http://www.vspj.cz/ISBN/Skripta%20-%20V%C5%A0PJ/Datab%C3%A1zov%C3%A9%20syst%C3%A9my%202%20-%20Zbyn%C4%9Bk%20Bure%C5%A1.pdf

[3] ALERYANI, Arwa. Comparative Study between Data Flow Diagram and Use Case Diagram. *International Journal of Scientific and Research Publications* [online]. Saba University, Yemen, 2016 [cit. 2017-03-15]. ISSN 2250-3153. Dostupné z: http://www.ijsrp.org/research-paper-0316/ijsrp-p5122.pdf

[4] CHROMEK, Ondřej. Testování podnikových aplikací. *SystemOnLine.cz* [online]. 2016, (7-8/2016) [cit. 2017-03-15]. ISSN ISSN 1802-615X. Dostupné z: https://www.systemonline.cz/sprava-it/testovani-podnikovych-aplikaci-1.htm

[5] ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. *Řízení kvality softwaru: průvodce testováním.* Brno: Computer Press, 2013. ISBN 9788025138168.

[6] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu.* Praha: Grada, 2016. Profesionál. ISBN 9788024755946.

[7] BRUCKNER, Tomáš. *Tvorba informačních systémů: principy, metodiky, architektury.* Praha: Grada, 2012. Management v informační společnosti. ISBN 9788024741536.

[8] Třívrstvá architektura (Three-tier architecture). In: *ManagementMania.com* [online]. Wilmington (DE) 2011-2017, 05.12.2015 [cit. 2017-03-17]. Dostupné z: https://managementmania.com/cs/trivrstva-architektura-three-tier-architecture

[9] KLEINE, Matthias, Robert HIRSCHFELD a Gilad BRACHA. *An Abstraction for Version Control Systems.* Potsdam: Universitätsverlag Potsdam, 2012. ISBN 978-3-86956-158-5.

[10] FAIGL, Jan. *Úvod do verzovacích systémů: (informativní)* [online]. 2016 [cit. 2017-03-17]. Dostupné z: https://cw.fel.cvut.cz/wiki/_media/courses/a0b36pr2/lectures/lecture11-slides.pdf

[11] PETŘÍK, Michal. Nové pohledy na populární metody projektového řízení (2. část). *SystemOnLine.cz* [online]. 2016, (11) [cit. 2017-03-19]. ISSN 1802-615X. Dostupné z: https://www.systemonline.cz/rizeni-projektu/zacnete-s-devops.htm

[12] DevOps Learning Guide for Agile, Git & CI. *Visual Studio* [online]. [cit. 2017-03-19]. Dostupné z: https://www.visualstudio.com/devops/

[13] BASS, Len., Ingo M. WEBER a Liming ZHU. *DevOps: a software architect's perspective*. Pearson Education, 2015. ISBN 978-0-13-404984-7.

[14] Cloud. In: *ManagementMania.com* [online]. Wilmington (DE) 2011-2017, 05.12.2015 [cit. 2017-03-21]. Dostupné z: https://managementmania.com/cs/cloud-computing

[15] MELL, Peter a Timothy GRANCE. *The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology* [online]. National Institute of Standards and Technology, 2011 [cit. 2017-04-30]. DOI: 800-145. Dostupné z: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[16] PORTNOY, Matthew. *Virtualization essentials*. Indianapolis, Indiana: John Wiley & Sons, 2012. Serious skills. ISBN 978-1-118-17671-9.

[17] BURIAN, Pavel. *Internet inteligentních aktivit*. Praha: Grada, 2014. Průvodce (Grada). ISBN 978-80-247-5137-5.

[18] Cloud computing: Co ty pojmy znamenají? *Cloud.cz* [online]. [cit. 2017-03-22]. Dostupné z: http://www.cloud.cz/cloud/158-cloud-computingco-ty-pojmy-znamenaji.html

[19] Scaling out with Azure SQL Database. *Microsoft Azure* [online]. 2016 [cit. 2017-03-23]. Dostupné z:https://docs.microsoft.com/en-us/azure/sql-database/sql-database-elastic-scale-introduction

[20] MORSY, Mohamed, John GRUNDY a Ingo MÜLLER. *An Analysis of the Cloud Computing Security Problem* [online]. Swinburne University of Technology, Hawthorn, Victoria, Australia [cit. 2017-03-24]. Dostupné z: https://arxiv.org/ftp/arxiv/papers/1609/1609.01107.pdf

[21] HOLUBOVÁ, Irena, Jiří KOSEK, Karel MINAŘÍK a David NOVÁK. *Big Data a NoSQL databáze*. Praha: Grada, 2015. Profesionál. ISBN 978-80-247-5466-6.

[22] COLES, Cameron. AWS vs Azure vs Google Cloud Market Share 2017. *Skyhigh* [online]. [cit. 2017-04-30]. Dostupné z: `https://www.skyhighnetworks.com/cloud-security-blog/microsoft-azure-closes-iaas-adoption-gap-with-amazon-aws/`

[23] Cloud Products. *Cloud Products & Services - Amazon Web Services (AWS)* [online]. [cit. 2017-03-24]. Dostupné z: `https://aws.amazon.com/products/`

[24] Data center locations. *Google Data Centers* [online]. [cit. 2017-03-24]. Dostupné z: `https://www.google.com/about/datacenters/inside/locations/index.html`

[25] *Google Cloud Platform: Google Cloud Computing, Hosting Services & APIs* [online]. [cit. 2017-04-30]. Dostupné z: `https://cloud.google.com/`

[26] Products & Services. *Google Cloud Platform* [online]. [cit. 2017-03-25]. Dostupné z: `https://cloud.google.com/products/`

[27] TULLOCH, Mitch. *Introducing Windows Azure: For IT Professionals*. Redmond, Washington 98052-6399: Microsoft Press, 2013. ISBN 978-0-7356-8288-7.

[28] Upcoming Name Change for Windows Azure. *Microsoft Azure* [online]. 2014 [cit. 2017-03-27]. Dostupné z: `https://azure.microsoft.com/en-us/blog/upcoming-name-change-for-windows-azure/`

[29] Azure Germany Cloud Computing. *Microsoft Azure* [online]. [cit. 2017-03-27]. Dostupné z: `https://azure.microsoft.com/en-us/overview/clouds/germany/`

[30] Azure regions. *Microsoft Azure* [online]. [cit. 2017-03-27]. Dostupné z: `https://azure.microsoft.com/en-us/regions/`

[31] Microsoft Azure: Cloud Computing Platform & Services. *Microsoft Azure* [online]. [cit. 2017-03-28]. Dostupné z: `https://azure.microsoft.com/en-us/`

[32] GAILEY, Glenn. *Get started guide for Azure developers* [online]. Microsoft, 2016 [cit. 2017-03-30]. Dostupné z: `https://docsmsftpdfs.blob.core.windows.net/guides/azure/azure-developer-guide.pdf`

[33] Directory of Azure Cloud Services. *Microsoft Azure* [online]. [cit. 2017-03-28]. Dostupné z: `https://azure.microsoft.com/en-us/services/`

[34] *Understanding Azure: Guide for Developers* [online]. One Microsoft Way Redmond, Washington 98052-6399: Microsoft Corporation, 2016 [cit. 2017-03-29]. Dostupné z: http://download.microsoft.com/download/2/C/F/2CF7401A-B9D7-4828-917D-199E0896BFE5/Azure_Developer_Guide_eBook.pdf

[35] Introducing Microsoft Azure. *Microsoft Azure* [online]. 2015 [cit. 2017-03-29]. Dostupné z: https://docs.microsoft.com/en-us/azure/fundamentals-introduction-to-azure

[36] Microsoft Azure Documentation. *Microsoft Azure* [online]. [cit. 2017-03-30]. Dostupné z: https://docs.microsoft.com/en-us/azure/

[37] Project Tracking for Software Teams. *Visual Studio Team Services* [online]. [cit. 2017-04-01]. Dostupné z: https://www.visualstudio.com/team-services/

[38] Team Services & TFS documentation. *Visual Studio IDE, Code Editor, Team Services, & Mobile Center* [online]. [cit. 2017-04-02]. Dostupné z: https://www.visualstudio.com/en-us/docs/overview

[39] Continuous integration and deployment (CI/CD). *Team Services & TFS* [online]. [cit. 2017-04-05]. Dostupné z: https://www.visualstudio.com/en-us/docs/build/overview

[40] COLLIER, Michael a Robin SHAHAN. *Fundamentals of Azure: Second edition.* 2.vydání. Redmond, Washington 98052-6399: Microsoft Press, 2016. ISBN 978-1-5093-0296-3.

[41] Introduction to Entity Framework. *Microsoft Developer Network* [online]. [cit. 2017-04-04]. Dostupné z: https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx

[42] What Is Windows Communication Foundation. *Microsoft Developer Network* [online]. [cit. 2017-04-04]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx

[43] ClickOnce Security and Deployment. *Microsoft Developer Network* [online]. [cit. 2017-04-06]. Dostupné z: https://msdn.microsoft.com/en-us/library/t71a733d.aspx

[44] How to: Transform Web.config When Deploying a Web Application Project. *Microsoft Developer Network* [online]. [cit. 2017-04-10]. Dostupné z: http://msdn.microsoft.com/en-us/library/dd465318(VS.100).aspx

[45] What is Application Insights? *Visual Studio IDE, Code Editor, Team Services, & Mobile Center* [online]. [cit. 2017-04-10]. Dostupné z: `https://www.visualstudio.com/en-us/docs/insights/application-insights`

[46] Počítačová bezpečnost (Computer security). In: *ManagementMania.com* [online]. Wilmington (DE) 2011-2017, 12.01.2016 [cit. 2017-04-12]. Dostupné z: `https://managementmania.com/cs/pocitacova-bezpecnost`

[47] Why ninety-day lifetimes for certificates? *Let's Encrypt - Free SSL/TLS Certificates* [online]. 2015 [cit. 2017-04-12]. Dostupné z: `https://letsencrypt.org/2015/11/09/why-90-days.html`

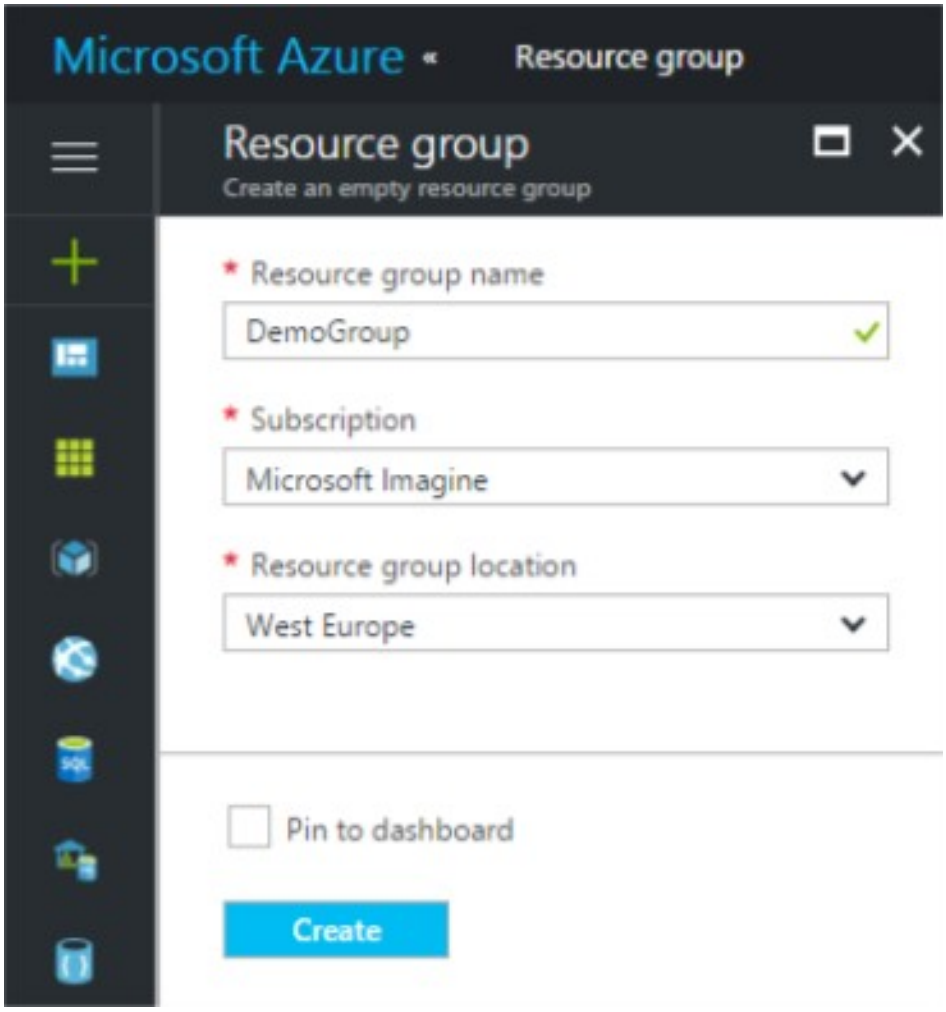
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ANSI	American National Standards Institute
API	Application Programming Interface
ARM	Azure Resource Manager
AWS	Amazon Web Services
CD	Continuous Delivery
CI	Continuous Integration
CPU	Central Processing Unit
DevOps	Development & Operations
DFD	Data Flow Diagram
EC2	Amazon Elastic Compute Cloud
EF	Entity Framework
GUID	Globally Unique Identifier
HW	Hardware
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
ORM	Object-Relational Mapping
PaaS	Platform as a Service
PHP	Hypertext Preprocessor
RAM	Random-Access Memory
REST	Representational State Transfer
SaaS	Software as a Service
SCM	Source Code Management
SDK	Software Development Kit
SQL	Structured Query Language
STD	State Transition Diagram
SUT	System Under Test
SW	Software
TFS	Team Foundation Server
TFVS	Team Foundation Version Control
UCD	Use Case Diagram
UML	Unifed Modeling Language
VHD	Virtual Hard Disk
VM	Virtual Machine
VSTS	Visual Studio Team Services
WCF	Windows Communication Foundation
XML	Extensible Markup Language

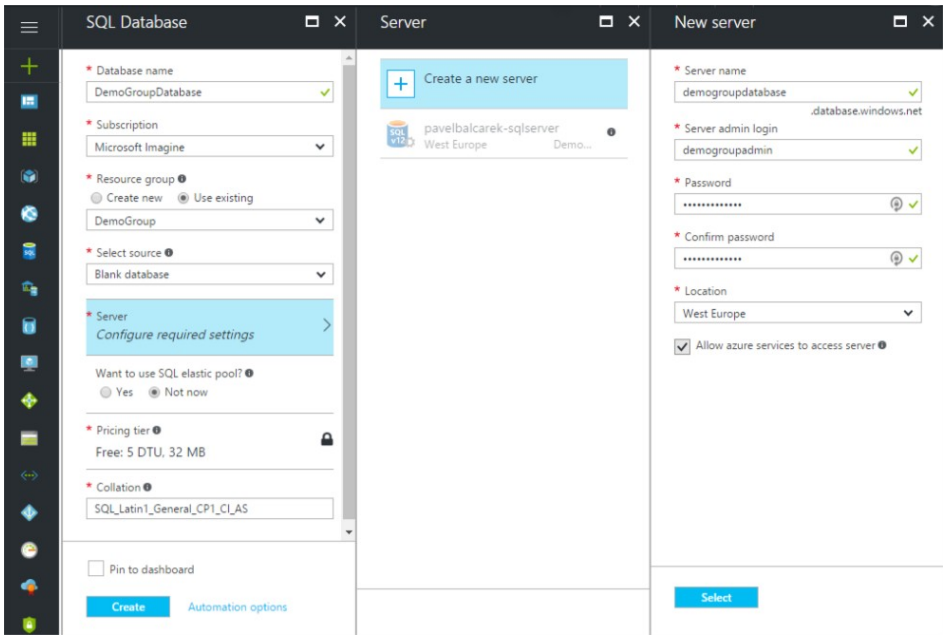
SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

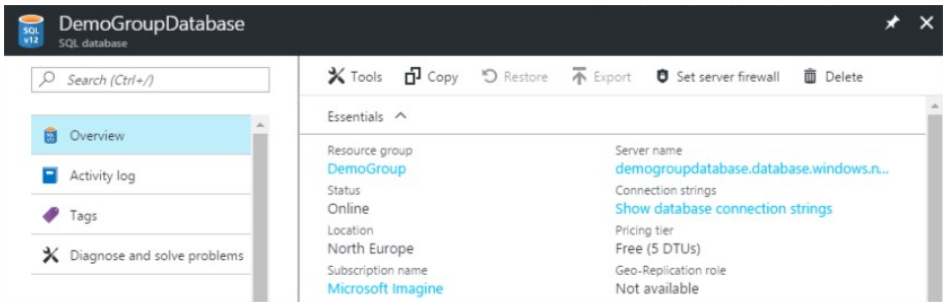
PŘÍLOHA P I. MANUÁLNÍ NASAZENÍ



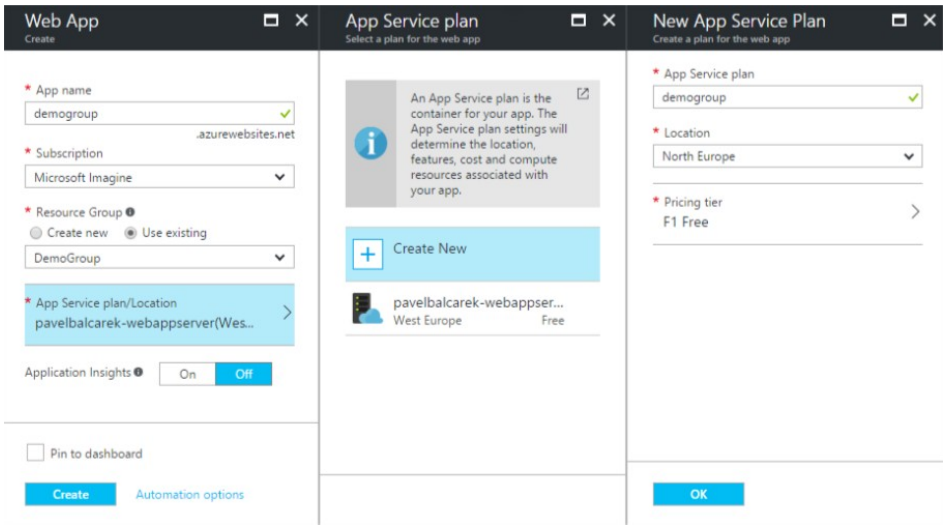
Obr. PI.1 Vytvoření *Skupiny zdrojů*



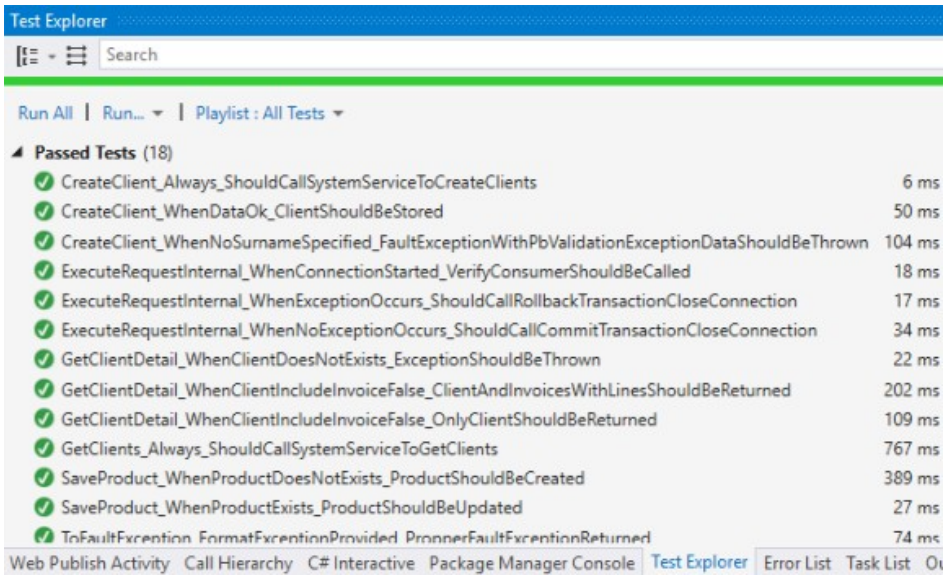
Obr. PI.2 Vytvoření *SQL databáze*



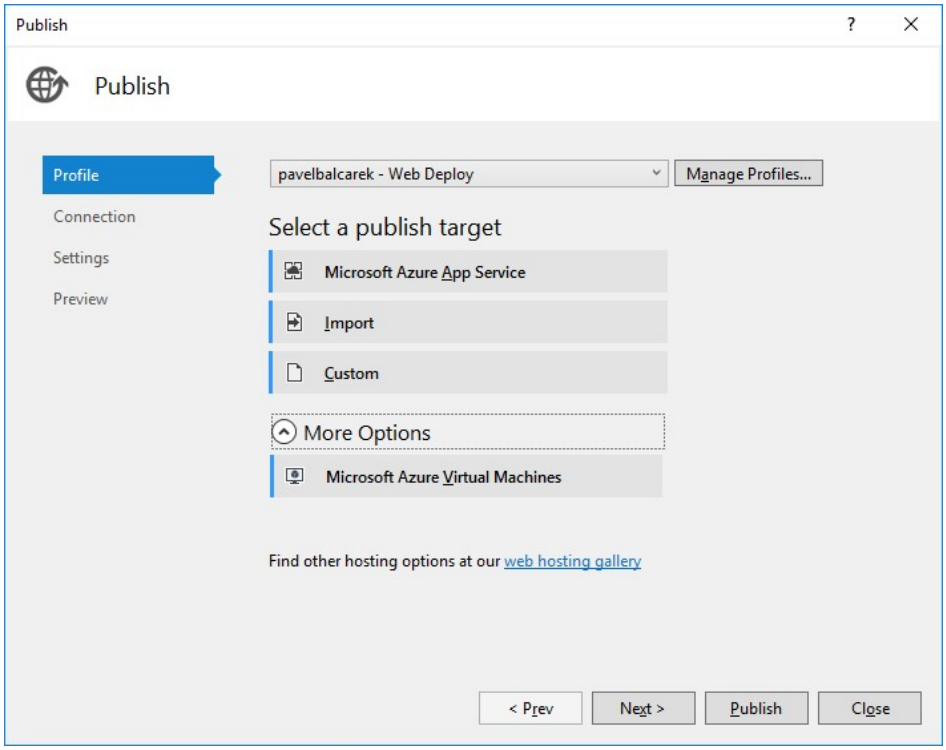
Obr. PI.3 Přehled informací o *SQL databázi*



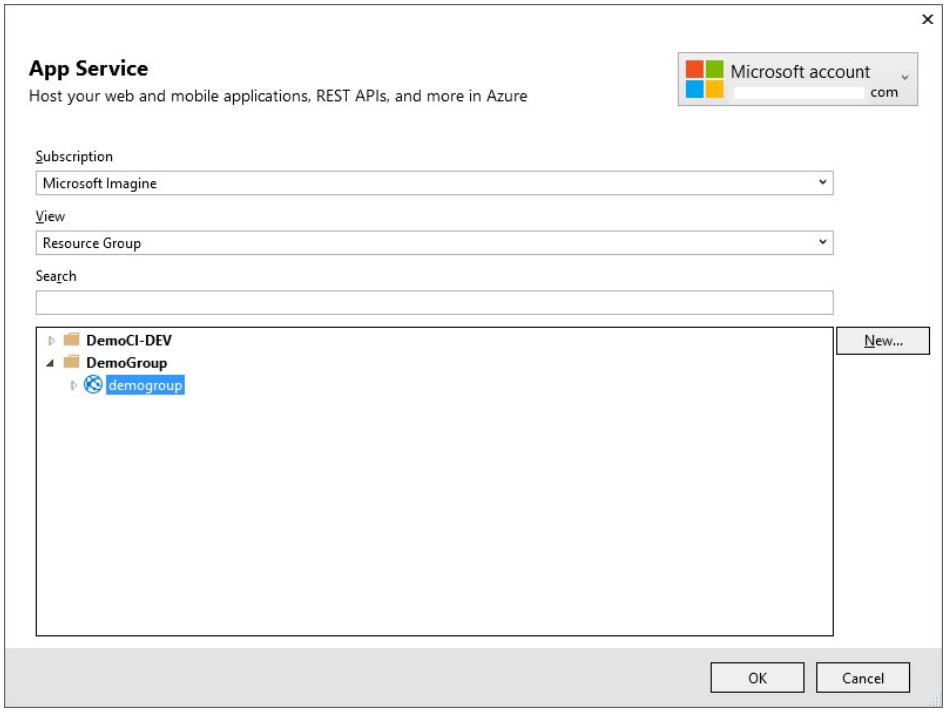
Obr. PI.4 Vytvoření *Web App*



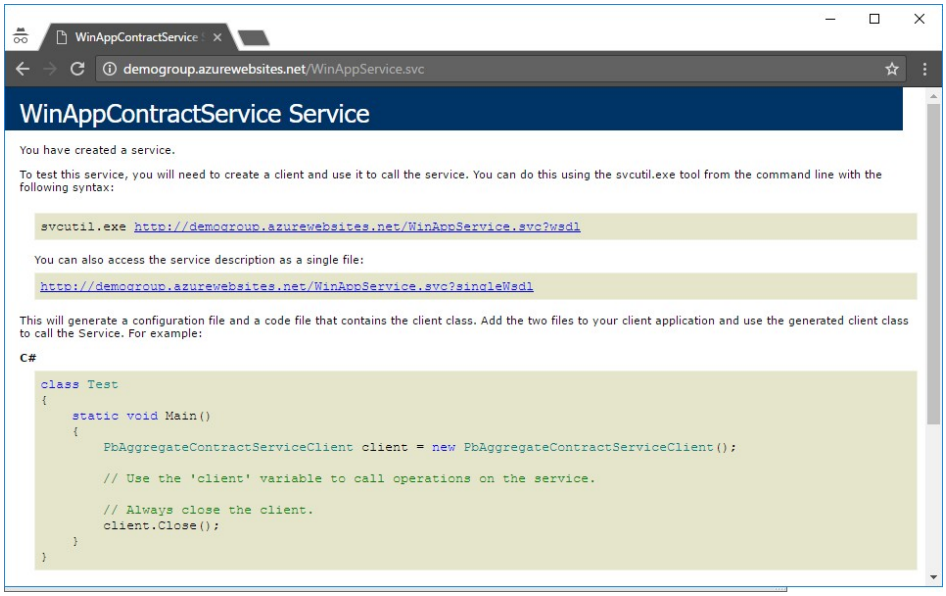
Obr. PI.5 Přehled testů v panelu *Test Explorer*



Obr. PI.6 Průvodce publikováním - první krok

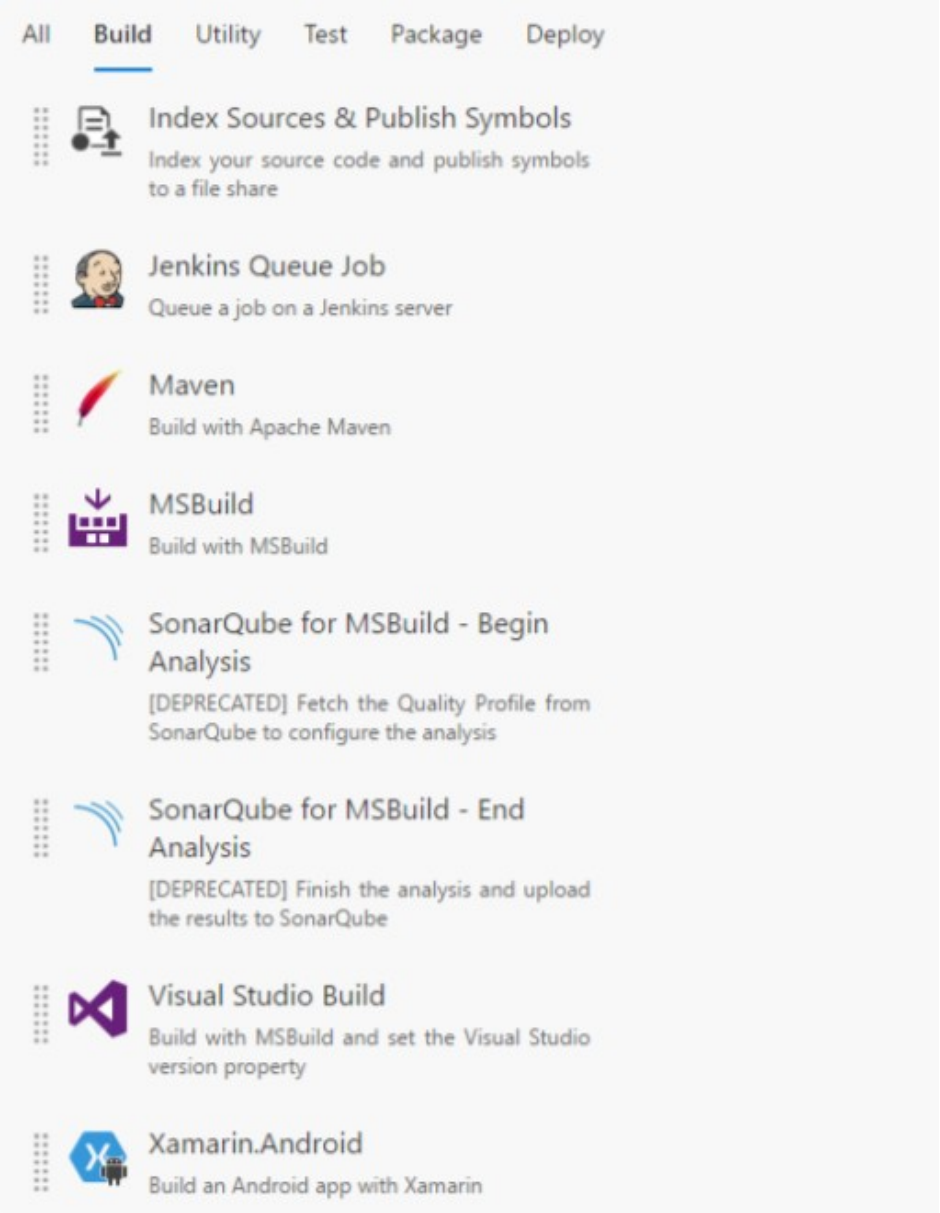


Obr. PI.7 Průvodce publikováním - druhý krok

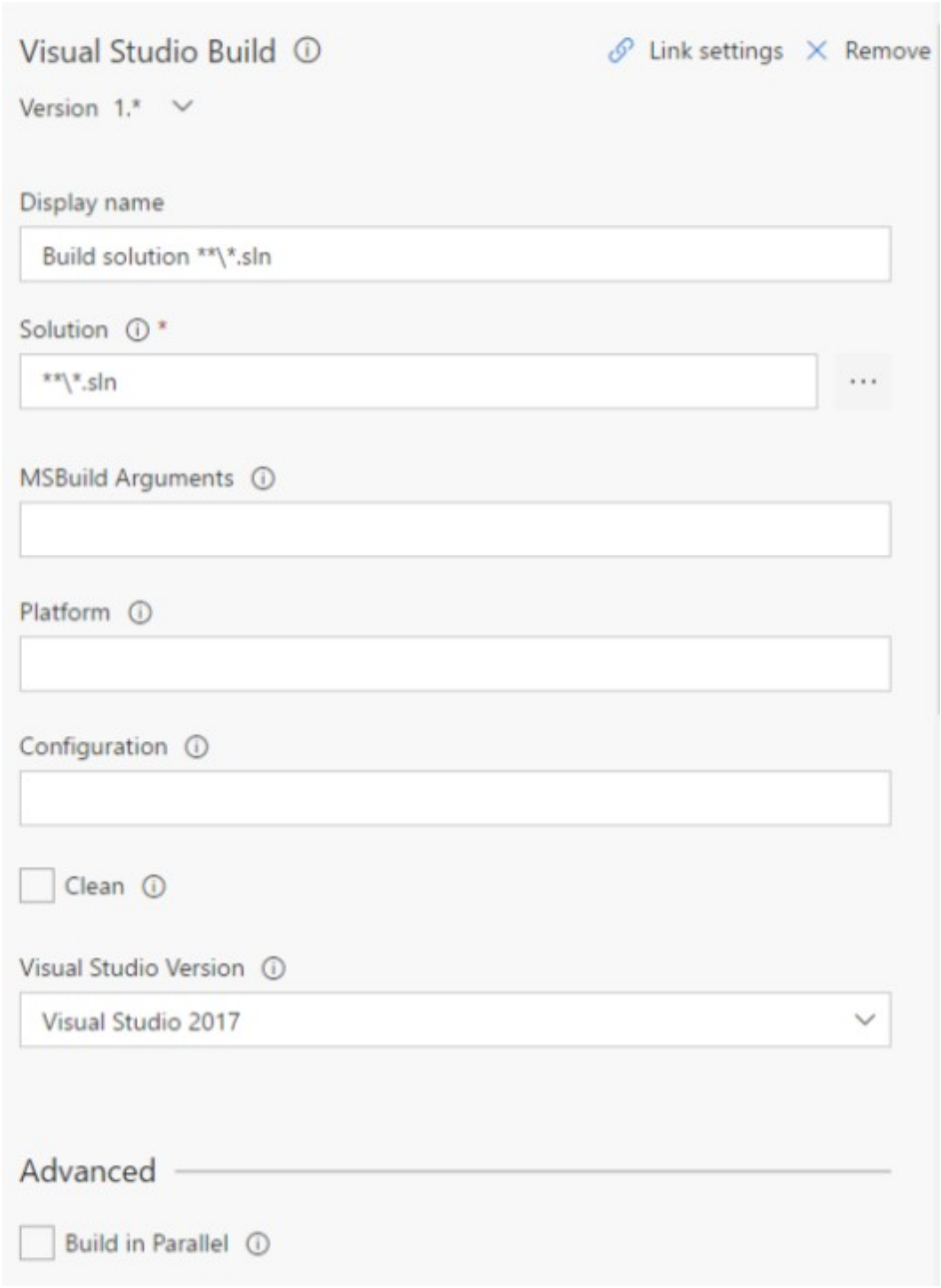


Obr. PI.8 Stránka publikované služby

PŘÍLOHA P II. VISUAL STUDIO TEAM SERVICES



Obr. PII.1 Příklad Build úkolů



Obr. PII.2 Příklad nastavení úkolu