

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Nové komunikační rozhraní pro služby DCIx

Místo této strany bude
zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 23. června 2022

Přemysl Kouba

Abstract

The main topic of this master's thesis is a proposal for a new communication interface for system services of the application DCIx created by the AIMTEC a.s. Part of the thesis is research of accessible technologies for monitoring. However, none of the researched technologies were appropriate for the application DCIx. Therefore a custom solution is introduced. The thesis includes an implementation description of a given custom solution. The implementation is then manually tested on one of the system services and also with automated and unit tests. Performance testing of the whole solution is also included.

Abstrakt

Tématem předložené diplomové práce je monitoring stavu služeb v komunikačním rozhraní aplikace DCIx od společnosti AIMTEC a.s. Práce stručně seznamuje čtenáře s aplikací DCIx. Součástí práce je rešerše stávajících volně dostupných řešení pro monitoring. Žádné ze zkoumaných řešení není vhodné k využití v rámci aplikace DCIx, proto je výstupem diplomové práce vlastní návrh. Práce popisuje důležité části implementace prototypu navrženého řešení. Vzniklý prototyp je ozkoušen na jedné ze služeb a otestován automatickými a jednotkovými testy. Část práce se také věnuje performance testování celého řešení.

Obsah

1	Úvod	9
2	Aplikace DCIx	10
2.1	Aimtec	10
2.1.1	ASP	10
2.1.2	DCI	10
2.1.3	INT	11
2.1.4	SAP	11
2.1.5	SUP	11
2.2	DCIx	11
2.2.1	Přechod k nové architektuře	12
3	Architektura aplikace DCIx	14
3.1	Core	14
3.1.1	Alfa Framework	17
3.1.2	Datová vrstva	17
3.1.3	Business vrstva	18
3.1.4	Webová vrstva	18
3.2	Mikroslužby aplikace DCIx	19
3.2.1	ApiGateway	19
3.2.2	LogCollector	19
3.2.3	ServiceDiscovery	19
3.2.4	CoreGate	20
3.2.5	CoreServices	20
3.3	Rozhraní služby CoreServices	22
3.3.1	/execute	22
3.3.2	/handleMessage	22
3.3.3	/administrator	23
3.3.4	/serviceEvent	25
3.4	Protokol OPC	26
3.4.1	OPC Subskripce	26
3.4.2	OPC Node	27
3.5	DTO objekty pro správu služby	27
3.5.1	ServiceStatus	27
3.5.2	ServiceEventStatus	28
3.5.3	OPCSubscriptionItemStatus	29

3.6	Dokumentace	29
4	Testování aplikace DCIx	30
4.1	Jednotkové testy	31
4.1.1	T-SQL testy	31
4.1.2	JUnit testy	32
4.2	End-to-end testy	32
4.2.1	Selenium WebDriver	32
4.2.2	Jameleon	32
4.2.3	Vizuální testování	33
4.3	Performance testování	33
4.4	Manuální testování	33
4.5	Android testování	33
4.6	Pokrytí kódu	34
5	Existující technologie a nástroje pro monitoring běžících aplikací	35
5.1	Hodnotící kritéria pro výběr	35
5.1.1	Splnění nároků na funkčnost	35
5.1.2	Snadné použití v kontextu aplikace DCIx v clusteru .	36
5.1.3	HW nároky řešení	36
5.1.4	Licenční podmínky	36
5.2	Apache Karaf Decanter	37
5.2.1	Kolektory	37
5.2.2	Appendery	38
5.2.3	Alerting	38
5.2.4	Procesor	39
5.2.5	Shrnutí a splnění kritérií	39
5.3	Zabbix	41
5.3.1	Architektura	41
5.3.2	Shrnutí a splnění kritérií	43
5.4	Prometheus	44
5.4.1	Architektura	45
5.4.2	Shrnutí a splnění kritérií	46
5.5	Proprietární řešení	47
5.5.1	Shrnutí a splnění kritérií	48
5.6	Shrnutí	49
5.6.1	Vybraná technologie	49

6	Komunikační rozhraní pro vzdálenou správu a monitoring služeb DCIx	50
6.1	Obrazovka systémových služeb	50
6.2	Stávající řešení	52
6.2.1	Problémy stávajícího návrhu:	54
6.3	Navrhované řešení	55
6.3.1	SystemServicesMonitor	56
6.3.2	EventBus	56
6.3.3	Úpravy komunikace	56
6.3.4	Frontend	57
6.3.5	Výhody a nevýhody tohoto řešení	57
7	Implementace navrhovaného řešení	59
7.1	SystemServicesMonitor	59
7.2	EventBus	60
7.3	Úpravy komunikace	61
7.3.1	Vytvoření endpointů	62
7.3.2	Úprava DTO objektů pro přenos dat	62
7.3.3	Definování nové WebSocket konekce	64
7.4	Frontend	64
7.4.1	Javascriptový framework	64
7.4.2	Úpravy JSP šablon	65
7.5	Získávání dat o OPC subskripcích	66
7.6	Povýšení na nejnovější verzi DCIx	66
7.7	Testování implementace	67
7.8	Výkonnostní testování navrženého řešení	68
7.8.1	Zdroje pro testování	68
7.8.2	Způsob testování	69
7.8.3	Definice testovacích scénářů	70
7.8.4	Zjištění celkové propustnosti systému s výchozím nastavením HW nároků	71
7.8.5	Maximální výkon systému dosažitelný na testovacím nodu	72
7.8.6	Otestování propustnosti jednotlivých komponent navrženého řešení	74
7.8.7	Závěr	76
7.9	Navrhované vylepšení implementace	77
8	Závěr	78

9	Uživatelská příručka	79
9.1	Navigace na obrazovku Systémové služby	79
9.2	Nastavení monitorování stavu systémových služeb	81
	Literatura	84
A	Slovníček pojmů	87
B	Přílohy	88

1 Úvod

Aplikace DCIx umožňuje zobrazovat stav řídicích jednotek výrobních strojů (zařízení) na výrobní lince. Ilustrací může být příklad výrobní linky která je konstruovaná za účelem zajištění požadované produkce výroby.

„Webová aplikace DCIx od společnosti Aimtec je produkt z kategorie MOM (Manufacturing operations management), který integruje celý dodavatelsko-odběratelský řetězec a poskytuje přesný a okamžitý digitální obraz. Produkt nabízí komplexní řešení pro řízení logistiky a výroby.“ [4]

„Jedno z těchto řešení – Manufacturing Execution System (MES) – umožňuje sesbírat data z jednotlivých výrobních zařízení, montážních linek a od zaměstnanců. Tato data integruje do jednoho zdroje, který poskytuje data pro operativní řízení výroby, plánování výroby a nákupu, zvyšování produktivity a zpřesňování norem.“ [5]

Data lze získávat z jednotlivých výrobních zařízení různými protokoly. Ve většině případů je potřeba udržovat informace o aktivitě zařízení, případně o stavu spojení – k tomuto účelu je v aplikaci DCIx definována služba TCPIPListener. Data z této služby jsou zobrazována uživateli a kategorizována podle protokolů na obrazovce „Systémové služby“.

Cílem této práce je umožnit uživateli sledovat v změny výše zmíněných zařízení v definovaném intervalu. K dosažení tohoto cíle je nutné navrhnout proces, kterým budou data o zařízeních propagována mezi mikroslužbami a následně zobrazena uživateli. Velký důraz je kladen na zachování stávající funkčnosti a dostatečné otestování nového vývoje automatickými a jednotkovými testy.

Obsah této práce lze rozdělit na několik částí. První kapitola seznamuje čtenáře s aplikací DCIx od společnosti AIMTEC a.s. Druhá část se věnuje průzkumu existujících řešení pro monitoring běžících aplikací. Cílem rešerše je nalezení hotového řešení pro sledování stavu zařízení. Třetí část se zabývá návrhem řešení. Následující část popisuje samotnou implementaci řešení. Poslední část této práce je věnována testování aplikace DCIx a nově navržené funkcionality.

2 Aplikace DCIx

Tato kapitola se věnuje popisu aplikace DCIx od společnosti AIMTEC a.s. Cílem práce je rozšíření této aplikace o komunikační rozhraní pro služby. Je tedy vhodné nejprve se podrobně seznámit s aplikací (v kapitole 2) a její architekturou (v kapitole 3). Text kapitoly vychází z oficiálních stránek společnosti Aimtec [3] a z dříve publikovaných diplomových prací.

2.1 Aimtec

Plzeňská společnost AIMTEC a.s. (dále jen Aimtec) vznikla v roce 1996 a od svého založení je zaměřena na vývoj, podporu a konzultační expertizu v oblasti výroby a logistiky. Firma implementuje vize v oblasti Industry 4.0 a Digital factory. V poslední době se také ukázala důležitost Digital Delivery, kterou společnost realizuje. Zákazníky nejsou jen české společnosti, ale jsou to firmy po celém světě. Společnost Aimtec. má v současnosti celkem pět divizí:

2.1.1 ASP

Oddělení ASP (zkratka pro Asprova) se zabývá konzultací a vývojem produktu japonského plánovacího informačního systému Asprova pro pokročilé plánování výroby s možností optimalizace výrobního plánu.

2.1.2 DCI

Oddělení zabývající se vývojem informačního systému **DCIx**. DCIx je produkt z kategorie MOM (Manufacturing operations management), který integruje celý dodavatelsko-odběratelský řetězec a poskytuje přesný a okamžitý digitální obraz. Usnadňuje koordinaci a obchodní spolupráci se zákazníky, dodavateli a partnery [29].

2.1.3 INT

Toto oddělení (dříve EDI) se zabývá systémovou integrací a elektronickou výměnou dat (EDI). Stěžejní náplní tohoto oddělení je dodávka služeb v oblasti EDI. Hlavním produktem tohoto oddělení je ClouEDI jako služba založená na cloudovém systému. Dalšími produkty jsou konzultační služby a DCIxPortal [10].

2.1.4 SAP

Oddělení společnosti Aimtec, které podporuje, rozšiřuje a implementuje produkty německé společnosti SAP AG v produktu Sappy4x4, který implementuje SAP ERP ve specifických oblastech.

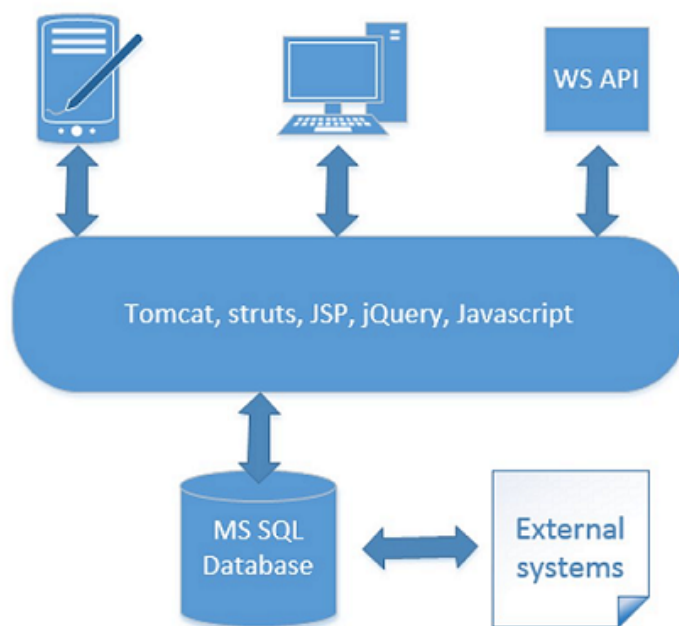
2.1.5 SUP

Oddělení nabízející podporu 24/7 pro všechny produkty společnosti Aimtec. Služby lze využít i externě, kdy zákazník využívá podporu od společnosti Aimtec ke svým produktům, nebo k internímu IT.

2.2 DCIx

Aplikace nabízí komplexní řešení pro řízení logistiky a výroby propojitelné s jinými informačními systémy a HW zařízeními a přednastavené řešení pro výrobní a distribuční společnosti [3].

První verze této aplikace vznikla v roce 1998 a od té doby je neustále vylepšována. Původní architektura je třívrstvá založená na technologiích Tomcat, Struts, JSP, JQuery, JavaScript. Využívá programovací jazyk Java. Databázovým systémem je Microsoft SQL Server [9]. Obrázek 2.1 ukazuje schéma této architektury. Původní třívrstvá architektura je pomalu nahrazována novou.



Obrázek 2.1: Diagram architektury aplikace DCIx. Přejato z prezentace „De-kompozice monolitické aplikace“. [13]

2.2.1 Přechod k nové architektuře

Třívrstvá architektura aplikace je postupně rozdělována na dílčí mikroslužby za použití technologií Kubernetes a Docker. Tyto technologie jsou použity v nové verzi architektury a jejich použití poskytuje zákazníkovi vyšší dostupnost, konfigurovatelnost, bezvýpadkové nasazení a škálovatelnost aplikace.[6]

Mikroslužby využívají framework Spring, který urychluje vývoj mikroslužeb. Další technologií pro usnadnění práce je Hibernate – oblíbený ORM framework¹.

Jedním z cílů přechodu na novou architekturu je nasazení a provozování aplikace v cloudu, které je díky těmto změnám jednodušší.

¹ORM - objektově relační mapování, programovací technika definující vztah mezi datovou vrstvou a aplikační logikou. Pomocí ORM je možné automaticky konvertovat data mezi relační databází a objektově orientovaným programovacím jazykem.

Další důvody pro přechod na architekturu mikroslužeb jsou [13]:

- vysoká dostupnost a škálovatelnost - služby mohou být snadno spuštěné ve více replikách
- technologická izolovanost mikroslužeb
 - služby umožňují zapouzdřit nové technologie a nejsou závislé na zbytku aplikace
 - nezávislost služeb na programovacím jazyce
 - nasazení služeb je jednodušší oproti monolitu
 - omezení zaměření funkcionality mikroslužby na jednu oblast
- organizační aspekty
 - rozdělení na menší týmy dle byznys logiky služeb
- menší nároky na testovací infrastrukturu
 - testovat je možné menší celky a ne vždy jen celou aplikaci

3 Architektura aplikace DCIx

V roce 2019 byl zahájen přechod z monolitické architektury na architekturu mikroslužeb. Důvody jsou popsány v kapitole 2.2.1. Přechod je pozvolný a v době zpracování této práce stále probíhá dekompozice monolitického jádra na jednotlivé oddělitelné části - mikroslužby.

Existující mikroslužby jsou napsány v jazyce Java s použitím frameworku *Spring Boot*¹, který usnadňuje jejich vývoj. Jednotlivé mikroslužby jsou zabaleny pomocí aplikace *Docker*² do obrazů (image) a ty jsou nasazovány do aplikace *Kubernetes*³.

Po dekompozici monolitické aplikace na jednotlivé mikroslužby bylo potřeba zajistit vzájemnou komunikaci mezi nimi. Aby bylo možné mezi službami komunikovat, je nejprve potřeba zjistit, které jednotlivé mikroslužby jsou spuštěné, na jaké adrese a portu komunikují a jakou mají verzi. K tomu slouží mikroslužba **ServiceDiscovery**.

Schéma 3.1 ilustruje architekturu aplikace po oddělení mikroslužeb.

3.1 Core

Text této podkapitoly vychází z diplomové práce Martina Kantoříka z roku 2017 [15]. **Core** je monolitickým jádrem celé aplikace. Jen některé služby jsou oddělené mimo toto jádro, ale větší část zůstává v jádře. Jádro dodržuje třívrstvou architekturu.

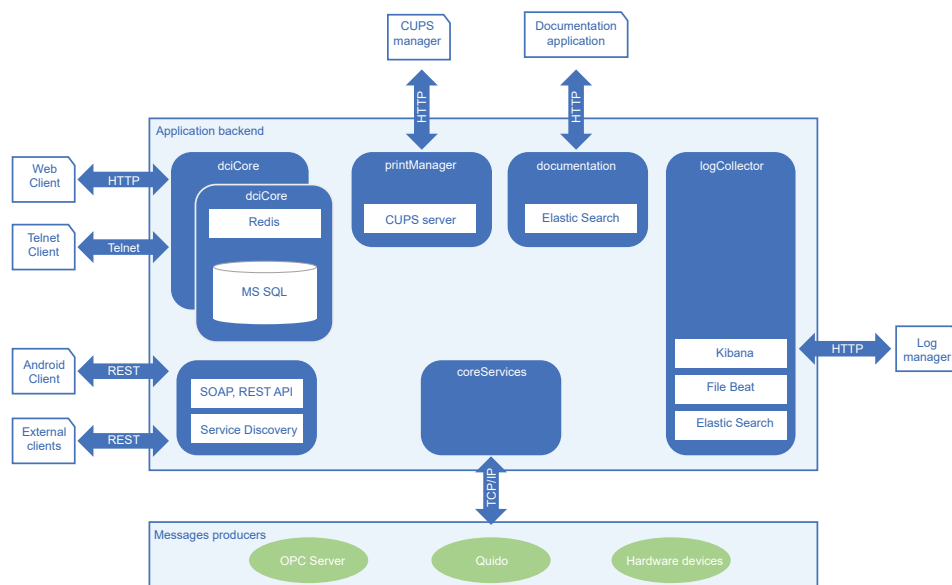
Architekturu komponenty **Core** (Obrázek 3.2) lze rozdělit na tři hlavní mezi sebou komunikující vrstvy:

- Datová vrstva
- Business vrstva
- Webová vrstva

¹Open-source aplikační framework pod licencí Apache 2.0. Stránka projektu <https://spring.io/projects/spring-boot>

²Docker je otevřený software (open source projekt) pod licencí Apache 2.0, jehož cílem je poskytnout jednotné rozhraní pro izolaci aplikací do kontejnerů (virtualizace). Stránka projektu <https://www.docker.com/>

³Open source software pod licencí Apache 2.0 původně vyvinutý společností Google určený pro orchestraci jednotlivých kontejnerů (orchestrace virtualizace). Stránka projektu <https://kubernetes.io/>

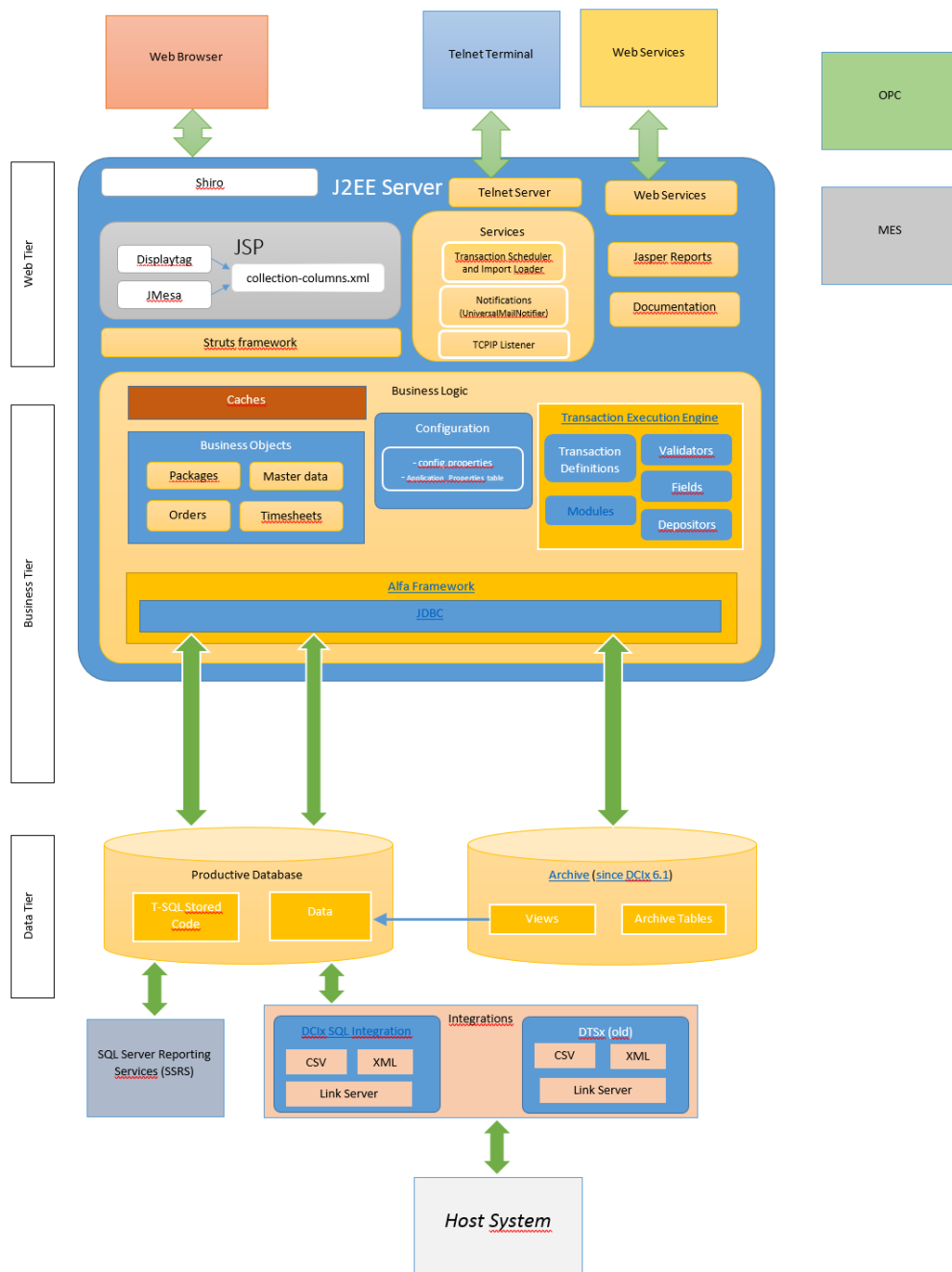


Obrázek 3.1: Schéma aplikace DCIx po oddělení mikroslužeb [2]

Klientem může být:

- **Webový prohlížeč** - doporučeným prohlížečem je Google Chrome a jeho deriváty postavené na jádru Chromia
- **DCIBrowser** - webový prohlížeč založený na prohlížeči Chromium a jeho rozšířeních. Je speciálně upravený tak, aby usnadnil operátorům práci s aplikací DCIx.
- **Android DCIBrowser** - alternativa DCIBrowseru pro operační systém Android.
- **DCIx TouchClient** - aplikace vyvíjená společností Aimtec pro spouštění jednotlivých transakcí na operačním systému Android. Využívá webových služeb.
- **Telnet** - Spouštění transakcí za pomoci protokolu Telnet.

Komponenta Core využívá server Apache Tomcat. Tomcat je založen na jazyce Java. Aplikace dále využívá javové servlety a JSP(Java Server Pages).



Obrázek 3.2: Detailní schéma mikroslužby Core [17]

3.1.1 Alfa Framework

Před tím, než bude popsána třívrstvá architektura mikroslužby **Core**, je vhodné seznámit čtenáře s **Alfa Frameworkem**. Tento framework postupuje vrstvami architektury mikroslužby **Core**.

Alfa Framework je interní framework společnosti Aimtec určený k ORM. Zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programováním. Pro získání dat z databáze framework automaticky vytvoří požadovaný dotaz a následně vrátí výsledek tohoto dotazu. Takto sestavený dotaz je zabezpečen proti útokům jako je například *SQL Injection* a navíc se předchází zbytečným chybám programátora (mohly by vzniknout např. v případě psaní dotazu manuálně).

3.1.2 Datová vrstva

Tato vrstva je vytvořena produktem SQL Server od Microsoftu, který používá rozšíření jazyka *SQL (Structured Query Language)* s názvem *T-SQL (Transact-SQL)*. Aplikace využívá pět databází:

- **hlavní**
- **kolektor**
- **monitor**
- **report**
- **archivní**

První databáze je nazývána **hlavní** či **aktivní** databáze. V této databázi se nachází všechna data ukládaná aplikací. V této databázi se nachází všechna potřebná data pro správný běh aplikace a každá akce provedená v aplikaci je snadno dohledatelná.

Databáze **kolektor** obsahuje data sesbíraná ze strojů připojených do aplikace DCIx pomocí klientů pro jednotlivé protokoly (OPC, TCP/IP, MQTT). Dále obsahuje data pohybu ve skladu. Databáze **monitor** sleduje a ukládá změny stavů kmenových dat v aplikaci DCIx. Mezi kmenová data patří například uživatelé, položky, důvodové kódy a další. V databázi **report** jsou shromažďována data určená pro reporting. Aimtec pro reporting využívá následující technologie:

- PowerBI
- SQL Server Reporting Services
- Ignition

Poslední zmíněnou databází je **archiv**. Do této databáze jsou ukládána data, která nejsou nutně potřeba v hlavní databázi. Díky tomuto mechanismu je možné udržovat hlavní databázi malou a rychlou. Pro přístup k datům v archivní databázi slouží buď speciální (archivní) aplikace, nebo

přímý přístup do databáze pomocí dotazů. Data v archivní databázi jsou k dispozici i za cenu delšího přístupu.

Datová vrstva komunikuje s business vrstvou pomocí pohledů a uložených procedur, kdy dochází k dotazování *Alfa Frameworkem* (kapitola 3.1.1). V databázové vrstvě se nenachází pouze uložená data, ale je zde i výkonný kód (procedury, funkce, pohledy a triggerry).

3.1.3 Business vrstva

Tato vrstva se stará o většinu funkcionality komponenty **Core**. Zajišťuje spojení mezi akcemi uživatele a daty u uživatele. V této třídě lze nalézt **Alfa Framework** a **Transaction Execution Engine**. Druhý zmíněný se zjednodušeně stará o zpracovávání transakcí a jejich úspěšné proběhnutí.

3.1.4 Webová vrstva

Poslední vrstva zobrazuje data klientům. K obluze klientských požadavků je využit framework Struts doplněný technologií Java Server Pages (JSP), EJB - Enterprise Java Beans. V šablonách je využit Javascript (převážně frameworky JQuery a RequireJS), HTML 4 a CSS 3 (respektive SCSS⁴). Tato část se stará o definici vzhledu stránek zobrazujících se v prohlížeči u uživatele. Zároveň zachycuje všechny akce prováděné uživatelem a předává je dál do business vrstvy.

Další komponentou je **Telnet** server, který se stará o komunikaci pomocí protokolu telnet. Při využití Telnet serveru se uživateli zobrazí jen dostupné transakce, které může spouštět. Transakce si volí tím, že píše identifikační čísla dané transakce. Pro přístup musí mít uživatel oprávnění.

Poslední možností je využití **Web Service**. Odpovědi na uživatelská volání jsou data, která si klient zpracuje dle potřeby.

⁴SCSS – „Sassy CSS“, je jazyk, který syntakticky vychází z CSS. Oproti CSS nabízí další funkcionalitu jako jsou například proměnné, vnořená pravidla, či funkce. Toto rozšíření je možné z toho důvodu, že tento jazyk je interpretován nebo kompilován pomocí Sass do výsledného CSS. Webové stránky projektu <https://sass-lang.com/>

3.2 Mikroslužby aplikace DCIx

Tato podkapitola popisuje mikroslužby, které již byly odděleny od jádra aplikace (3.1) v rámci přechodu na novou architekturu. K 13.06.2022 existují tyto mikroslužby:

- `ApiGateway`
- `LogCollector`
- `ServiceDiscovery`
- `CoreGate`
- `CoreServices`

3.2.1 ApiGateway

`ApiGateway` řídí komunikaci mezi klientskými zařízeními a mikroslužbami v clusteru. Přínosem této služby je zjednodušení přístupu klienta k aplikaci.

`ApiGateway` přesměrovává požadavky z jedné adresy na adresy a porty ostatních mikroslužeb v clusteru. Také slouží jako *load balancer* (přesměrování požadavků na méně vytížené instance komponent).

3.2.2 LogCollector

`LogCollector` je obecná logovací služba, která zajišťuje sběr logů ze všech kontejnerů (mikroslužeb) aplikace a ukládá je do společné databáze. Skládá se z jednotlivých nezávislých částí, které jsou v kontejnerech.

- **File beat** – sleduje všechny ostatní kontejnery aplikace a sbírá jejich logy, ty jsou dále ukládány do Elastic Search
- **Elastic Search** – databáze logů
- **Kibana** – samostatná webová služba sloužící k prohlížení logů

3.2.3 ServiceDiscovery

Tato mikroslužba slouží pro detekci a registrování jednotlivých mikroslužeb k sobě. Jedná se o centrální prvek pro komunikaci mezi mikroslužbami [30].

V aplikaci DCIx je k tomuto účelu využita knihovna `Eureka`⁵. Mikroslužby se k `ServiceDiscovery` hlásí pomocí anotace `@EnableEurekaClient`. Jedná se o Springovou anotaci, která zajistí, že se aplikace při startování

⁵Eureka Discovery je open source software pod licencí Apache 2.0 od společnosti Netflix. Jedná se o službu využívající REST API primárně užívanou v cloudu pro lokalizaci služeb (service discovery) a vyvažování zátěže (load balancing). Dokumentace a zdrojový kód dostupný na adrese <https://github.com/Netflix/eureka>

chová jako `ServiceDiscovery` klient. Registrované mikroslužby mají konfigurační soubor, ve kterém jsou následující informace:

```
service.name      = ${service.name}
service.version   = ${service.version}
service.server    = ${service.server}
service.port      = ${service.port}
service.path      = ${service.path}
```

Pro spojení se službou `ServiceDiscovery` navíc potřebují mít definovanou adresu této služby, ke které se mají registrovat

```
service.discovery.server = ${service.discovery.server}
service.discovery.port   = ${service.discovery.port}
service.discovery.path   = ${service.discovery.path}
```

Po zajištění identifikace jednotlivých mikroslužeb bylo potřeba připravit jednotný způsob komunikace mezi nimi a monolitickým jádrem. Řešením tohoto problému je `CoreGate`.

3.2.4 CoreGate

Tato mikroslužba zprostředkovává přístup k funkcionalitě aplikačního serveru `DCIx (Core)` pomocí REST API ostatním mikroslužbám v Kubernetes clusteru. Z jednotlivých mikroslužeb jsou volány koncové uzly (endpointy) a výsledná serializovaná data jsou přenášena ve formátu *JSON*. Komunikace je obousměrná.

3.2.5 CoreServices

V této podkapitole je třeba rozlišovat mikroslužbu a službu. Mikroslužba je komponenta, která má definované rozhraní a může běžet uvnitř Kubernetes clusteru. Systémovou službu lze definovat jako výkonný kód, který běží ve vláknech uvnitř komponenty `CoreServices` (v kódu lze systémové služby poznat tak, že implementují třídu `Service`).

`CoreServices` je mikroslužba implementující funkcionalitu, která nemůže běžet paralelně ve více replikách. Funkcionalitu lze rozdělit do jednotlivých systémových služeb. Účelem této mikroslužby je také správa těchto služeb (kontrola funkčnosti, validace spouštění, pozastavování a ukončování). Mikroslužba slouží také k přenosu informací o systémových službách a zpráv ze služeb do jádra.

Systémové služby lze rozdělit do dvou kategorií:

- „simple“ služba - skládá se z jednoho eventu (úloha – například kontrola tisků). Kód eventů běží v jednom vlákne.
- „multievent“ služba – skládá se z více eventů (např. kontrola úloh pro SQL se dělí na jednotlivé SQL úlohy). Eventy mohou mít rozdílný výkonný kód a platí, že 1 event = 1 vlákno.

Ze systémových služeb je pro tuto diplomovou práci stěžejní *TCPIP Listener*, na které bude demonstrována funkčnost navrhovaného řešení.

3.2.5.1 TCPIP Listener

Různá zařízení produkují zprávy v různých protokolech. Cílem služby *TCPIP Listener* je tyto zprávy zpracovat. Zprávy jsou zpracovávány paralelně a každý event představuje jedno klientské spojení s cizím serverem.

Nejvýznamnější protokoly (využívající protokol TCP/IP), pro které existují v aplikaci *DCIx* konektory (klienti):

- OPC - jednotný protokol pro sběr dat ze zařízení (kapitola 3.4)
- MQTT - standardní protokol používaný pro IoT⁶
- Modula - protokol určený primárně pro ovládání skladových zařízení od výrobce Modula⁷
- Kardex - protokol používaný stroji společnosti Kardex
- VNA - protokol užívaný pro komunikaci s vysokozdviznými vozíky a zakladači (Very Narrow Aisle)
- Quido - protokol pro komunikaci s I/O moduly od společnosti Papouch⁸
- FTP - protokol pro přenos souborů

Funkčnost této služby je možné shrnout jako *Message Bus*. V případě *DCIx* se jedná o *middleware*, který sjednocuje implementace jednotlivých komunikačních integrací do jednotného rozhraní. Obsahuje implementace klientů pro komunikaci se stroji a jejich protokoly. Zasílání zpráv přes protokol může probíhat přímo, ale také za pomoci front příchozích a odchozích zpráv.

TCPIP Listener poskytuje prostředky pro připojování a odpojování jednotlivých komunikačních konektorů (klientů) bez vlivu na ostatní (loosely coupled). Aimtec vytváří vlastní adaptéry pro jednotlivé komunikační konektory nebo rozšiřuje již existující. Příkladem takového rozšíření je *OPC Client*, který rozšiřuje funkčnost proprietární knihovny *prosys-opc-ua-sdk-client*⁹.

⁶<https://mqtt.org/>

⁷<https://www.modula.eu/products/modula-link/>

⁸<https://papouch.com/io-moduly/quido/>

⁹<https://www.prosysopc.com/products/opc-ua-java-sdk/>

Výše zmíněné adaptéry mají několik zásadních funkcí:

- sjednocují komunikační rozhraní
- udržují navázané spojení aktivní (pokud to protokol vyžaduje)
- kontrolují, zda-li je spojení aktivní (watchdog)

3.3 Rozhraní služby CoreServices

Tato podkapitola seznamuje čtenáře s jednotlivými endpointy mikroslužby **CoreServices**. Pro tuto diplomovou práci je významná podkapitola 3.3.3.1. Ta specifikuje endpoint, který vrací stav služeb v DTO objektu **StatusContainer**.

Při registraci mikroslužby **CoreServices** do **CoreGate** proběhne načtení konfigurace z jádra (databáze) pomocí volání endpointu `/coreServices/configuration`. Na základě této konfigurace jsou jednotlivé služby a jejich eventy postupně spouštěny. Pro práci se službami je definováno REST API. Jednotlivé endpointy pro správu a monitoring jsou popsány v následující podkapitole.

V mikroslužbě **CoreServices** je aplikován princip API-First¹⁰. API mikroslužeb (výčet endpointů, možné HTTP stavové kódy odpovědí, použité DTO objekty a dokumentace) je definováno v textovém formátu YAML. V mikroslužbě **CoreServices** je popis API obsažen v souboru `service-api.yaml` (ukázka 3.1).

3.3.1 /execute

Na tuto adresu jsou zasílány příkazy, které má mikroslužba vykonat v *JSON* formátu. Jedná se například o příkaz k restartu systémových služeb a jejich eventů.

3.3.2 /handleMessage

Tento endpoint má na starost zpracování příchozích a odchozích zpráv. Příchozí zpráva je přijata od mikroslužby **Core** – například srovnání hodnot v komunikačním adaptéru oproti poslední zprávě v databázi. Odchozí zpráva je zpráva, kterou **CoreServices** přijme od adaptéru a zašle do **Core** – například zpráva z OPC serveru.

¹⁰Moderní princip pro vývoj webových aplikací, který lze považovat jako návrhový vzor. Pro vývoj aplikací je nejprve specifikováno (a zdokumentováno) rozhraní a to je následně implementováno. Takto specifikované rozhraní (API) je možné využít jako kontrakt.

```

/administrator/status:
  get:
    tags:
      - Administrator
    description: All active services status .
    parameters:
      - name: companyId
        in: query
        required: true
        description: identifier of company which the service
          belongs to
        schema:
          type: integer
      - name: serviceCode
        in: query
        required: false
        description: service code which we want to get
        schema:
          type: string
    responses:
      200:
        description: All services status container .
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/StatusContainer'

```

Ukázka 3.1: Ukázka definice rozhraní přístupem API First v mikroslužbě CoreServices . Rozhraní je popsáno v textovém formátu YAML.

3.3.3 /administrator

Skupina příkazů obsahující tento endpoint zprostředkovává obecné informace o službách.

3.3.3.1 /status

Volání tohoto endpointu vrací stav všech aktivních služeb v `CoreServices`. Odpověď je serializovaný objekt obsahující informace o službách (viz kapitola 3.5). Tato data jsou deserializována v jádře a propagována do webového prohlížeče (klienta). Odpověď na požadavek na tento endpoint lze vidět v následující ukázce 3.2.

```
{
  "firstTime": "",
  "companyId": "1",
  "scheduleType": "IMPULSE",
  "serviceCheckTimeout": "",
  "serviceCode": "TransactionScheduler",
  "interval": "0",
  "enabled": "false",
  "events": []
},
{
  "firstTime": "",
  "companyId": "1",
  "scheduleType": "IMPULSE",
  "serviceCheckTimeout": "1000",
  "serviceCode": "TCPIPLListener",
  "interval": "1000",
  "enabled": "true",
  "events": [
    ...
    ...
    ...
  ]
},
```

Ukázka 3.2: Výsledek volání endpointu /configuration

3.3.3.2 /allowedCommands

Endpoint vrací informace o tom, jaké příkazy je možné zasílat jednotlivým službám.

3.3.3.3 /watchdogTypes

Na této adrese mikroslužba vrací názvy a typy jednotlivých watchdogů. Watchdog (hlídací pes) je kód, který v určitém časovém intervalu kontroluje dostupnost hlídaného objektu. Tento princip je v aplikaci DCIx pevně spjat se systémovou službou `TCPIPListener`. V případě, že tato služba (nebo některá z jejích částí) není dostupná, tento kód provede automatické kroky za účelem zotavení z chyby. Typické použití je zotavení po chybě komunikace s výrobními stroji na protokolu OPC.

3.3.4 /serviceEvent

V této podskupině jsou příkazy pro manipulaci s jednotlivými službami.

3.3.4.1 /pause

Zastaví službu (service) a její eventy, nebo pouze adaptér (event). Vyžaduje parametry *serviceCode*, *companyId*, *eventCode*.

3.3.4.2 /restart

Volání endpointu restartuje službu. Je možné restartovat nejen celou službu, ale také jednotlivé adaptéry. Vyžaduje parametr *eventCode*.

3.3.4.3 /sendCommand

Zašle příkaz (zprávu) konkrétnímu eventu. Event je definován v parametru a příkaz je součástí těla volání.

3.3.4.4 /readCommandResponse

Přečte návratovou hodnotu volání `/sendCommand`.

3.3.4.5 /hasCommandResponse

Vrací HTTPStatus kód 200 (OK), pokud event má odpověď na zaslaný příkaz.

3.4 Protokol OPC

Tato kapitola popisuje protokol OPC. Na systémové službě TCPIPLister lze nastavit event typu OPC, na kterém bude demonstrován výsledek této práce. Je vhodné čtenáře seznámit s tímto protokolem pro snadnější pochopení dalšího textu.

OPC¹¹ je sada standardů a specifikací definující problematiku komunikačního rozhraní při řízení a monitorování technologických procesů v rámci automatizace průmyslu využívající komunikaci M2M (machine to machine).

V roce 1994 pět předních výrobců HMI a SCADA systémů založilo konsorcium OPC Foundation, které v roce 1996 definovalo specifikaci OPC založenou na technologii OLE¹². V roce 2009 vznikla specifikace OPC UA (Universal Architecture) [19][21], která již není postavena primárně na technologii OLE od společnosti Microsoft a není tedy závislá na platformě. Nyní je členem konsorcia již více než 500 společností. Vzniká tak univerzální komunikační platforma, která se dokáže napojit na data stovek různých typů zařízení od rozmanitých výrobců. Tato data dokáže převést do jednotné OPC komunikace srozumitelné mnoha nadřazeným aplikacím (např. ERP, MES, SCADA).

OPC UA obecně podporuje dva protokoly:

- binární protokol využívající TCP/IP – tento protokol je využíván v aplikaci DCIx
- protokol Web služby využívající HTTP a SOAP

3.4.1 OPC Subskripce

Protokol OPC definuje princip zvaný OPC subskripce. Tento princip definuje rozdílný (a sofistikovanější) přístup pro získávání dat v porovnání s pollingem¹³. OPC klient umožňuje zaregistrovat se pomocí subskripce k odebrání požadovaných hodnot z OPC serveru (k tomu slouží „OPC Node“). Server tyto hodnoty monitoruje a pouze v případě, že nastane změna této hodnoty, notifikuje klienta o změně. Tento mechanismus výrazně snižuje množství dat, která jsou přenášena. [1]

¹¹Ole for Process Control, nebo také Open Connectivity via Open Standards, nebo také Open Platform Communications[22]

¹²Object Linking and Embedding – technologie klient/server od firmy Microsoft

¹³Polling – periodické dotazování za účelem získání dat v daném intervalu

3.4.2 OPC Node

Node je základní objekt pro adresování dat na protokolu OPC, který umožňuje serveru reprezentovat objekty pro klienty. Tyto objekty se skládají z atributu, hodnoty a referencí na další OPC Nody. OPC Nody mají unikátní identifikátory a datové typy [11]. OPC Nodů může být definováno libovolné množství a mohou jich být v aplikaci DCIx připojeny stovky až tisíce.

3.5 DTO objekty pro správu služby

V předchozí kapitole bylo zmíněno, že zprávy jsou zasílané serializované ve formátu *JSON*. V této kapitole jsou popsány jednotlivé DTO¹⁴ objekty, které se používají pro serializaci a deserializaci. Pro serializaci objektů se v aplikaci používá knihovna *Jackson*, ale vzhledem ke zpětné kompatibilitě aplikace je zachováváno také rozhraní *JsonSerializable*.

3.5.1 ServiceStatus

Stav služby (ukázka 3.3) je definován pomocí následujících atributů:

- *serviceCode* je název služby
- *companyID* je kód společnosti¹⁵
- *lastTime* definuje, kdy byla služba naposledy spuštěna
- *lastTimeScheduled* definuje, na kdy byla služba naposledy naplánována
- *paused* udává, zdali je služba zastavená
- *active* udává, zdali je služba aktivní
- *interval* definuje, jak často se má služba spouštět
- *events* je seznam jednotlivých eventů pro danou službu. Seznam je složen z objektů typu *ServiceEventStatus*

¹⁴Data Transfer Object - objekty využívané v objektově orientovaných jazycích, které slouží pro přenos dat mezi systémy (např. přenos dat mezi klientem a serverem).

¹⁵Struktura společností je v aplikaci DCIx používána převážně kvůli právům

```

public class ServiceStatus implements JsonSerializable {
    private String                serviceCode;
    private Integer               companyId;
    private DateTime              lastTime;
    private DateTime              lastTimeScheduled;
    private boolean               paused;
    private boolean               active;
    private long                  interval;
    private List<ServiceEventStatus> events;

```

Ukázka 3.3: Definice třídy ServiceStatus

3.5.2 ServiceEventStatus

Stav eventu (ukázka 3.4) je definován pomocí následujících atributů:

- *code* je unikátním identifikátorem eventu
- *isScheduled* popisuje, zda se jedná o naplánovaný event (běh)
- *lastExecution* definuje, kdy byl event naposledy spuštěn
- *period*, *timeUnit*, *periodPattern* - tyto atributy definují, v jakém intervalu se má event periodicky spouštět.
- *state* definuje stav eventu pomocí výčtu (třída `ServiceEventState`). Používány jsou stavy *running*, *down*, *error*.

Speciálně pro event OPC definujeme také atribut *opcSubscriptions*, který je seznamem OPC subskripcí (kapitola 3.4.1).

```

public class ServiceEventStatus {
    private String                code;
    private boolean               isScheduled;
    private DateTime              lastExecution;
    private int                   period;
    private String                timeUnit;
    private String                periodPattern;
    private ServiceEventState     state;
    private List<OPCSubscriptionItemStatus> opcSubscriptions;

```

Ukázka 3.4: Definice třídy ServiceEventStatus

3.5.3 OPCSubscriptionItemStatus

Tato třída obsahuje informace o jednotlivých OPC Subskripcích. Jsou definovány tyto atributy¹⁶ (ukázka 3.5):

- *nodeID* je název uzlu OPC serveru
- *attributeId* definuje, která z hodnot (proměnných) OPC serveru nás pro tuto danou subskripci zajímá (například číslo 13 = value)
- *transactionCode* udává, která transakce (akce, či posloupnost akcí v aplikaci DCIx) má být spuštěna při přijetí hodnoty
- *lastRunTime* definuje, kdy byl event naposledy spuštěn
- *publishingInterval* je interval, v jakém jsou posílány dotazy na hodnotu přes subskripci

```
public class OPCSubscriptionItemStatus {  
    private String    nodeId;  
    private String    attributeId;  
    private String    transactionCode;  
    private DateTime  lastRunTime;  
    private String    publishingInterval;  
}
```

Ukázka 3.5: Definice třídy OPCSubscriptionItemStatus

3.6 Dokumentace

Aplikace DCIx je plně dokumentována. Dokumentace je součástí jádra 3.1. V budoucnu by měla vzniknout dokumentační služba se dvěma kontejnery:

- Documentation - poskytuje REST API pro odkazy na dokumentaci z jádra aplikace. Obsahuje server tak, aby mohl být dostupný jako samostatná aplikace.
- Elastic Search - jedná se o kontejner pro podporu fulltextového vyhledávání v databázi dokumentace

¹⁶Některé z těchto atributů jsou definovány v knihovně OPC UA ([20]), další byly přidány pro potřeby aplikace DCIx

4 Testování aplikace DCIx

V této kapitole je popsán proces testování ve společnosti Aimtec, kterým projde veškerý vývoj. Výsledky testování nového vývoje jsou popsány v kapitole 7.7 a 7.8.

Společnost Aimtec má vlastní testovací oddělení, které má na starosti vývoj a údržbu testů a celé infrastruktury pro testování. Cílem je mít aplikaci co nejlépe pokrytou testovacími scénáři. Je také nutné, aby testování aplikace trvalo co nejkratší dobu a bylo co nejméně chybové.

Testování aplikace je spouštěno na dvou oddělených virtuálních strojích. První z nich má spuštěný databázový server, spouštějí se na něm testy a ukládají se zde výsledky. Na druhém stroji (aplikačním) je Kubernetes Cluster, ve kterém běží jednotlivé aplikace dle verzí.

Testování probíhá automaticky. Průběh je definován testovacím profilem v nástroji Apache Maven, který připraví všechny závislosti a postupně pustí všechny testy. Výstup testů je poté kopírován do speciálního úložiště. Toto úložiště je sdílené pro různé branchy aplikace a s tímto úložištěm pracují další nástroje, které přehledně zobrazují informace o průběhu testování. Na obrázcích 4.1 a 4.2 je vidět zobrazení výsledků testů.

Branch ↕	Testování	Výsledky ↕	Jameleon ↕	JUnit ↕	JS ↕	T-SQL ↕	API [?] ↕	Visual ↕
Filtr: <input type="text"/>	MES							
BRANCH_AFP506	[?] Vybrat	2021-11-27-03-34	0/1248/8	0/893	0/66	0/265		
BRANCH_AIMDEV_COPYUSER	[?] Vybrat	2022-04-13-09-12	1/333/5	0/1246/88	0/52/0	0/531/0	0/6/0	
BRANCH_AIMDEV_CTXMENU	👁 [?] Vybrat	2022-03-02-15-28	1/332/5	0/1239/86	0/57/0	0/531/0	0/4/0	
BRANCH_AIMDEV_DOCKER_BUILD	[?] Vybrat	2022-03-29-19-41	2/328/5	0/1240/86	0/52/0	0/531/0	0/4/0	
BRANCH_AIMDEV_DUOH	[?] Vybrat	2022-06-09-08-53	4/170/6	0/1250/83	0/50/0	1/540/0	5/8/0	7/103/0
BRANCH_AIMDEV_DUOH_2	[?] Vybrat	2022-03-14-11-29	1/331/5	0/1239/86	0/56/0	0/531/0	0/4/0	
BRANCH_AIMDEV_HROM	👁 [?] Vybrat	2022-05-25-18-14	0/94/5	0/1248/87	0/52/0	1/540/0	0/6/0	25/53/0
BRANCH_AIMDEV_HROM3	[?] Vybrat	2022-04-20-18-34	1/70/5	0/1215/73	0/52/0	4/531/0	0/6/0	
BRANCH_AIMDEV_KOUP	👁 [?] Vybrat	2022-06-20-18-07	5/1917/37	0/1250/83	0/54/0	0/540/0	5/8/0	21/214/0
BRANCH_AIMDEV_KVAJ	👁 [?] Vybrat	2022-06-16-15-26	6/99/5	6/1250/83	0/54/0	2/541/0	5/8/0	10/19/0
BRANCH_AIMDEV_KVAJ1	[?] Vybrat	2022-05-30-02-03	1/89/5	0/1248/87	0/52/0	0/540/0	0/6/0	19/35/0
BRANCH_AIMDEV_MES	[?] Vybrat	2022-06-21-10-10	2/105/5	1/1251/83	0/52/0	0/541/0	0/8/0	5/33/0

Obrázek 4.1: Zobrazení výsledků testů pomocí interního nástroje DciIndex. Zdroj: Aimtec

Home > LIVE > CTL.DEV > 0146 Failing Tests

Branch Name	%	Test Type	%
Branch Manager	%	Project Manager	%
Programmer	%	Team	MES
AIMDEV Branch	%	Show DOC-en Tests	No
Show Sonar	Yes	Tester	%
Show random tests	No	Show Only Not Analysed Branches	No

1 of 2 Find | Next

Branch	Manager	Test Name	Days	PGM (DCIx)	PGM (Deployed libraries)	State
BRA	kvaj/zizj/smap					
CLM	pliv/zajv/pola					
	Visual	eChartCreateTest (eChartAfterUpdate)	21	qkanm	dcijenkins	(upravit)
	Visual	eChartCreateTest (eChartAfterUpdate2)	21	qkanm	dcijenkins	(upravit)
	Visual	eChartCreateTest (OneDBShowsInChart)	21	qkanm	dcijenkins	(upravit)
	Visual	eChartCreateTest (ThreeDBShowsInChart)	21	qkanm	dcijenkins	(upravit)
	Visual	userTest (rolesWindowMax)	20	/		(upravit)
	JUnit	DciCoreVersusProjectOriginalsFilesChangesTest	155	hrom, qkanm	dcijenkins	nutno opravit, doplnit soubor - pgm (upravit)
	JUnit	AppDescriptorsSecretsTest	22	qkanm, zajv		pliv (upravit)
	JUnit	AllJmesaPagesTestedTest	6	kmr		(upravit)

Obrázek 4.2: Zobrazení výsledků testů pomocí reportu pro SQL Server Reporting Services. Zdroj: Aimtec

DCIx proto vytvořilo vlastní nástroj integration-tests-maven-plugin pro testování. Schéma B.1 popisuje jak funguje testování aplikace DClx.

Společnost Aimtec se snaží při vývoji aplikace DClx dodržovat zásady „Test Driven Developmentu“.

4.1 Jednotkové testy

Testy aplikace DClx jsou psány pomocí knihovny JUnit4 v programovacím jazyce Java. Hlavní větev v době psaní textu obsahuje přibližně 1400 testovacích scénářů. Dále je definováno asi 500 databázových jednotkových testů, které využívají frameworku T-SQL.

4.1.1 T-SQL testy

Pro otestování logiky psané v jazyce SQL je využíván tento nástroj¹. Definiuje API podobné ostatním knihovnám pro jednotkové testování. Výstup z testování je v HTML. Testovací scénáře lze libovolně do sebe zanořovat nebo seskupovat. Podmínkou pro běh testování je pouze SQL server, ve kterém vznikne databáze TST, ve které jsou jednotlivé funkce knihovny. Testy po dokončení provedou rollback, pokud je to možné.

¹T-SQL, licence: Eclipse Public License - v 1.0, stránky projektu <https://github.com/ladimolnar/TST>. Projekt je v současné době neudržovaný

Aimtec tuto knihovnu dále rozšířil o svoje vlastní procedury, které připravují data pro testy, a tím urychlují vytváření testovacích scénářů.

4.1.2 JsUnit testy

Testování aplikace obsahuje také nástroj pro testování kódu v programovacím jazyce Javascript. Aplikace DCIx obsahuje 60 takových testovacích scénářů. Licence - Apache License 2.0. Stránky projektu <https://code.google.com/archive/p/js-test-driver/>

4.2 End-to-end testy

Pro testování end-to-end testy je využíván framework Jameleon² společně s další knihovnou Selenium WebDriver³. V době psaní textu obsahuje hlavní větev kódu přibližně dva tisíc testovacích scénářů. Každý testuje jinou část aplikace. Aplikace je testována přes webový prohlížeč Chrome, který je jedním ze dvou podporovaných. Podporované prohlížeče jsou Chrome a Microsoft Edge.

4.2.1 Selenium WebDriver

Knihovna vytváří nadstavbu nad klasickým webovým prohlížečem, který simuluje uživatelskou interakci. Pro funkčnost je potřeba mít kromě prohlížeče také „ovladač“ tohoto prohlížeče (Chromedriver). Testovací scénáře lze psát v jazyce Java. Tato knihovna není využívána v testovacích scénářích, protože je zde velká závislost na technologických znalostech (jazyk Java). Proto je pro psaní scénářů využívána knihovna Jameleon.

4.2.2 Jameleon

Pro odstranění závislosti na znalosti jazyku je využívána další knihovna Jameleon. Cílem této knihovny je odstranit závislost na technických znalostech a nabídnout uživateli jednoduchý jazyk pro definici scénářů. Výsledkem je možnost zapisovat testovací scénáře v jazyce XML. Pro definici nabízeného jazyka je využíváno jazyku dtd.

²Jameleon, licence: GNU Lesser General Public License, stránky projektu dostupné na adrese <http://jameleon.sourceforge.net/>

³Selenium WebDriver, licence: Apache 2.0 license, stránky projektu dostupná na adrese <https://www.selenium.dev/>

4.2.3 Vizuální testování

Nadstavbou pro knihovnu Selenium je Ashot⁴. Jedná se o rozšíření třetí strany, která umožňuje využití knihovny Selenium ke snímání obrazovky. Výsledný snímek je porovnán s minulým snímkem (pokud existuje) a pokud se neshodují o více procent, než je chybovost, je test vyhodnocen jako chybný.

4.3 Performance testování

K performance testování aplikace je využívána knihovna JMeter⁵. Cílem performance testování je:

- posoudit, zda-li je DCIx použitelné pro dané množství uživatelů při daném výkonu serveru
- posoudit výkonnostní dopad nově vyvíjené funkcionality
- nalézt úzká hrdla v aplikaci a optimalizovat je

4.4 Manuální testování

V rámci aplikace DCIx je definováno i několik scénářů pro čistě manuální testování. Jsou to scénáře, které by šlo jen obtížně automatizovat. Proces manuálního testování je využíván většinou před uvolněním nové hlavní (major) verze aplikace. Hlavním cílem manuálního testování je ověřit funkčnost a vzhled aplikace. Funkčnost aplikace je otestována v nejčastěji používaných modulech. V současné době je potřeba manuálního testování nahrazována vizuálním testováním, které dokáže odhalit změny ve vzhledu aplikace.

4.5 Android testování

Společnost Aimtec dodává k užívání aplikace DCIx také další software pro zařízení využívající operační systému Android (DCIBrowser, Aimtec TouchClient). Funkčnost tohoto software je testována pomocí testovací knihovny Espresso a pomocí JUnit testů pro jednotkové testování. Knihovna Espresso⁶ je alternativa výše zmíněné knihovny Selenium pro operační systém

⁴Ashot, licence: Apache License 2.0, stránky projektu dostupné na adrese <https://github.com/pazone/ashot>

⁵JMeter, licence: Apache License 2.0, stránky projektu dostupné na adrese <https://jmeter.apache.org/>

⁶Espresso, licence: Apache License 2.0, stránky projektu dostupné na adrese <https://developer.android.com/>

Android. Testovací scénáře jsou psány v jazyce Java.

4.6 Pokrytí kódu

Aplikace DCIx je pokrytá jednotkovými testy z 69 % metod a ze 79 % co do rozhodovacích větví. K zjištění pokrytí testovacími scénáři je využívána knihovna JaCoCo⁷.

⁷Jacoco, licence: Eclipse Public License, stránky projektu dostupné na adrese <https://github.com/jacoco/jacoco>

5 Existující technologie a nástroje pro monitoring běžících aplikací

Kapitola se zabývá průzkumem existujících technologií a nástrojů pro monitoring běžících aplikací. Zkoumaná řešení jsou:

- Apache Karaf Decanter 5.2
- Zabbix 5.3
- Prometheus 5.4
- Řešení s využitím současných technologií v aplikaci DCIx 5.5

5.1 Hodnotící kritéria pro výběr

Cíle průzkumu jsou dva. Prvním cílem je zkusit nalézt volně dostupnou technologii, která by umožnila rozšířit stávající rozhraní o sledování stavu služeb. Druhým cílem je prozkoumat, jak monitoring funguje a jaké funkcionality obsahují volně dostupná řešení a následně z nabízených funkcionalit identifikovat ty, které by měly přínos pro budoucí použití v DCIx.

Kritéria výběru nástroje pro sledování stavu spuštěných služeb:

- splnění nároků na funkčnost
- snadné použití v kontextu aplikace DCIx v clusteru
- minimální HW nároky
- licenční podmínky
- (volitelné) další nabízená funkcionalita, kterou by bylo možné použít v DCIx

5.1.1 Splnění nároků na funkčnost

Je třeba, aby použité řešení umělo sbírat informace o službách z mikroslužby CoreService. Tyto informace je třeba vizualizovat uživateli v rámci aplikace DCIx na obrazovce *Systémové služby*.

5.1.2 Snadné použití v kontextu aplikace DCIx v clusteru

Výběr technologie pro monitoring je omezený tím, že je potřeba, aby nově vzniklé řešení zapadalo do konceptu nové architektury. Je kladen důraz na znovupoužitelnost. Zkušenost vývojářského oddělení s danou technologií je v tomto případě příjemným bonusem.

5.1.3 HW nároky řešení

Abychom splnili při výběru i přání zákazníka, je nutné myslet i na HW nároky zvoleného řešení. Aplikace DCIx cílí primárně na provozování v cloudu. Výhodou cloudového řešení je, že je snadné přidávat zdroje podle potřeby. Není ale vhodné zbytečně rozšiřovat stávající, poměrně vysoké nároky na HW. Navíc společnost stále umožňuje instalaci do prostředí zákazníka (on premise) a zákazníci současné HW požadavky považují za poměrně vysoké (většinou v porovnání oproti aplikaci DCIx se starší architekturou). Pro aplikaci DCIx hledáme řešení s optimálními (nízkými) HW nároky.

5.1.4 Licenční podmínky

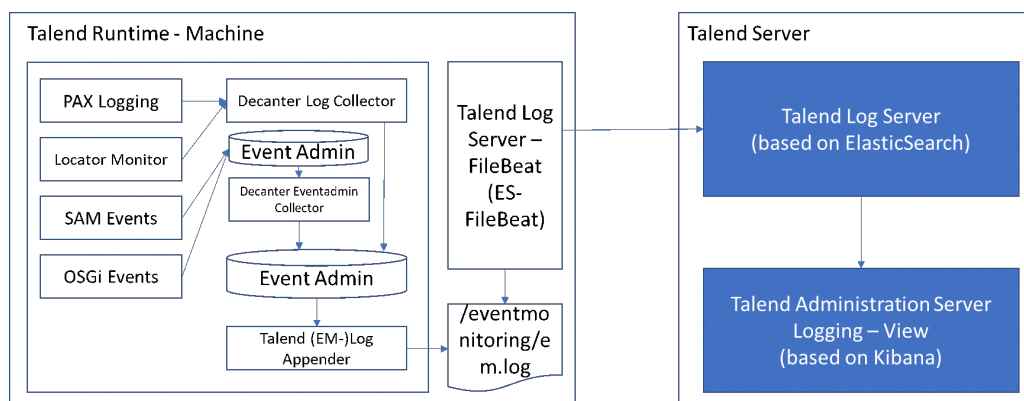
Společnost Aimtec preferuje pro svoji aplikaci DCIx knihovny třetí strany s otevřenou licencí. Použití placené licence pro proprietární software musí být velmi dobře odůvodněné (např. jedná-li se o kritický proces či neexistence ekvivalentní volně dostupné technologie)

5.2 Apache Karaf Decanter

licence	Apache Licence 2.0
dokumentace	https://dist.apache.org/repos/dist/release/karaf/documentation/decanter/
git	https://github.com/apache/karaf-decanter/
poslední aktivita	15.02.2022 (den kontroly 13.06.2022)
nejnovější verze	2.9.0 (15.02.2022)

Apache Decanter je hotové, volně dostupné řešení určené pro monitoring, které k běhu využívá modulární runtime knihovnu Apache Karaf. Architektura je ilustrována obrázkem 5.1. Řešení se skládá ze čtyř částí:

- Kolektory
- Appendery
- Alerting
- Procesory



Obrázek 5.1: Architektura nástroje Decanter [26] dostupné z URL

5.2.1 Kolektory

Kolektory sbírají monitorovací data. Knihovna Decanter nabízí kolektory pro jednotlivé typy dat. K tomu využívá dva druhy kolektorů:

- kolektory řízené událostmi (Event Driven Collectors) - reagují na události a poté vysílají (broadcast) data do appenderů
- časované kolektory (Polled Collectors) – periodicky získávají data z aplikace a ty dále zasílají do appenderů. Periodické spouštění těchto kolektorů řídí **DecanterScheduler**.

Knihovna disponuje velkým počtem (23) předem připravených implementací kolektorů. Každá z těchto implementací obsahuje již připravené nastavení pro komunikaci s danou technologií (například JMS, MQTT, SOAP, Rest Servlet atd.). Decanter umožňuje napsání vlastního časovaného nebo událostmi řízeného kolektoru při dodržení společného rozhraní.

5.2.2 Appendery

Appendery získávají data z kolektorů a ukládají je do datového úložiště. Stejně jako v případě kolektorů. Tato knihovna nabízí velký počet appenderů připravených k použití (16) a umožňuje také vytvoření vlastních. Jednotlivé implementace se od sebe liší cílovým úložištěm, se kterým aplikace komunikuje (Elasticsearch, JDBC, JMS, MQTT atd.).

5.2.3 Alerting

Apache Decanter také nabízí výstražná upozornění. K tomu se využívá služba **Alerting Service**, kde je možné pomocí konfiguračního souboru nastavit pravidla, podle kterých budou zasílány výstrahy. Tato pravidla jsou nastavena v JSON formátu a k vyhodnocení podmínek se využívá Apache Lucene Query. Je možné nastavit podmínku výstrahy (condition), severitu (level), periodu validování podmínky (period) a příznak, definující možnost přes výstrahu pokračovat v běhu aplikace (recoverable). Tento poslední příznak definuje, jak často bude zasílána výstraha. Pokud je nastavený na *false*, výstraha přijde kdykoliv, když bude podmínka vyhodnocena jako pravdivá. Pokud je nastavený na *true*, výstraha přijde nejprve při prvním výskytu a tehdy, když je vše uvedeno do původního stavu a daná podmínka již nenastává (back to normal).

Decanter definuje několik možností zasílání výstrah a umožňuje vytvořit své vlastní. Definované možnosti jsou:

- Log – jednotlivé výstrahy jsou zapsány do logovacího souboru
- E-mail – při výskytu výstrahy jsou zaslány e-mailové zprávy dle adres definovaných v konfiguračním souboru. Mimo adres příjemce je možné také nastavit SMTP server, odesílatele, přihlašovací údaje na SMTP server a formát zprávy.
- Camel – zasílá výstrahy na definovaný endpoint. Tento druh zasílání výstrah využívá framework Camel a kromě zprávy jsou na endpoint předávány detailní informace o výstraze pomocí mapy (např. alertLevel, alertAttribute, alertPattern).
- použití stávajícího appenderu – Výstraha je speciálním typem obecného

appenderu. Je tedy možné pro zpracování výstrah využít jakýkoliv jiný stávající appender. K použití stávajícího appenderu je nutné dodržet správné nastavení.

5.2.4 Procesor

Procesory dat slouží k použití aplikační logiky nad daty získanými z kolektorů před tím, než jsou předána na appendery. Procesory kontrolují komunikaci (vznikající události) na konkrétní adrese a zpracované zprávy posílají na jiný endpoint. Zdali se má konkrétní zpráva zpracovat procesorem je dáno atributem **topic**. pomocí tohoto atributu je možné procesory zřetěžit za sebe.

Jsou rozlišovány tyto druhy procesorů:

- Pass Through – tento druh neobsahuje logiku a jedná se jen o příklad implementace procesoru
- Aggregate – procesor typu *Aggregate* spojuje několik událostí do jedné která je zasílána periodicky
- GroupBy – sjednocuje události obsahující stejné hodnoty zaslané během určitého časového intervalu, výsledkem je zpráva s „unikátními“ hodnotami, která je posílána periodicky
- Apache Camel – pomocí frameworku Apache Camel je možné implementovat vlastní procesor

5.2.5 Shrnutí a splnění kritérií

5.2.5.1 Splnění nároků na funkčnost

Tento framework splňuje funkční požadavky, které na monitorovací aplikaci máme. Jeho výhodou (i nevýhodou) je možnost využití další knihovny *Apache Camel*, která je integračním frameworkem pro middleware orientovaný na messaging se rozhraním pro komunikaci. Při použití Apache Decanter v aplikaci DCIx by bylo potřeba domyslet klientskou část - zpracování zpráv na straně DCIx a vizualizace klientovi. Decanter nabízí k zasílání zpráv například JMS broker, Apache Kafka (JMS) nebo Network Socket. Na straně DCIx by bylo potřeba zprávy zpracovat, poslat na klienta a tam zobrazit.

5.2.5.2 Snadné použití v kontextu aplikace DCIx v clusteru

Framework Apache Decanter je součástí většího celku Apache Karaf a nelze jej samostatně užívat. Apache Karaf je runtime knihovna podobná Springu

se zaměřením na enterprise segment. Je možné také tyto dvě knihovny spojit. Na Karaf lze nahlížet jako na OSGi prostředí. Pro aplikaci DCIx by použití tohoto frameworku znamenalo rozšíření architektury o další prvek (OSGi).

5.2.5.3 Minimální HW nároky

Aplikace DCIx nevyužívá žádné prvky OSGi prostředí. Využití Apache Decanter by znamenalo zvýšení nároků o minimální požadavky pro Apache Karaf. To znamená nejméně 20 MB volného místa, 128 MB RAM (ale reálně 2 GB).

5.2.5.4 Licenční podmínky

Tato technologie splňuje stanovené požadavky na licenční podmínky.

5.2.5.5 Další funkcionalita

Decanter nabízí navíc kromě samotného monitoringu:

- alerting – logování, e-mail, integrace na Apache Camel
- processing – agregace, integrace na Apache Camel

5.3 Zabbix

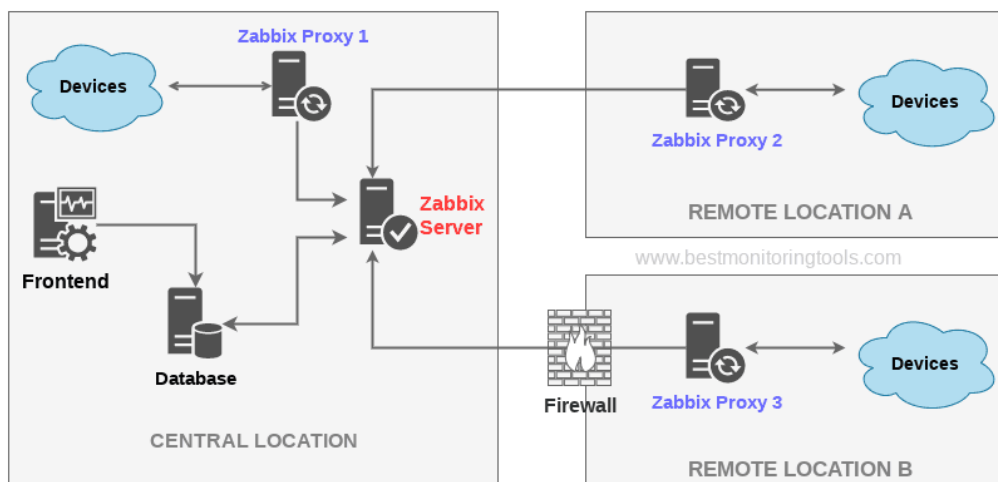
licence	GPLv2
dokumentace	https://www.zabbix.com/documentation/current
git	https://github.com/zabbix/zabbix
poslední aktivita	13.06.2022 (den kontroly 13.06.2022)
poslední verze	6.2.0rc1 (ze dne 09.06.2022)

Zabbix [8] [18] [28] je volně dostupný nástroj pro vzdálené monitorování IT komponent pod licencí GPLv2. O vývoj a podporu tohoto nástroje se stará společnost Zabbix SIA. Nástroj se skládá z několika základních komponent:

- Server
- Proxy
- Agent
- Webové uživatelské rozhraní
- Databáze

5.3.1 Architektura

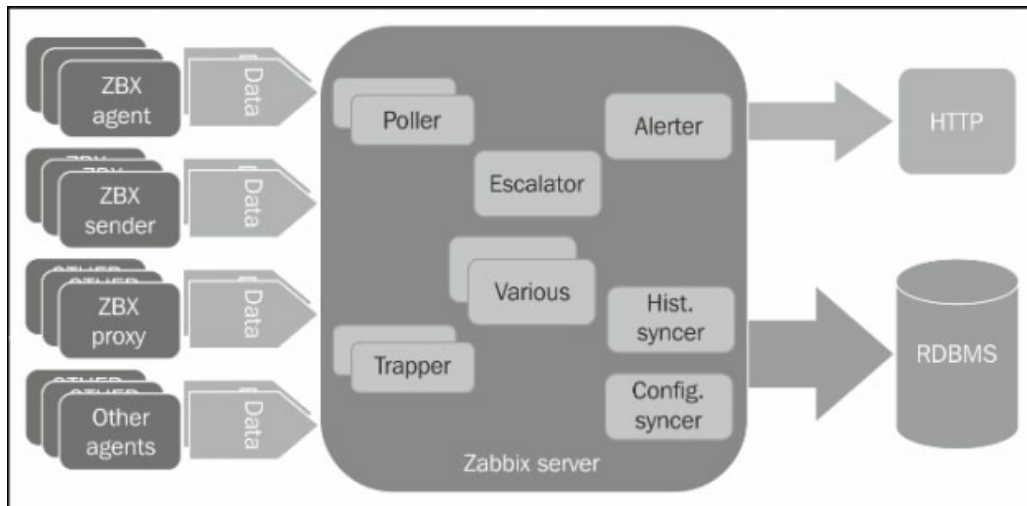
Architektura technologie Zabbix je znázorněna na obrázku 5.2. Komunikace jednotlivých prvků architektury nastíněna na obrázku 5.3.



Obrázek 5.2: Architektura nástroje Zabbix [23]

5.3.1.1 Server

Server je hlavní komponenta nástroje. Udrží nastavení aplikace, historická a operační data. Dále umožňuje notifikaci uživatele (alerting). Server umožňuje přímé monitorování jednotlivých zařízení. Skládá se z webového rozhraní a databázového systému¹.



Obrázek 5.3: Zabbix Data Flow [8]

5.3.1.2 Webové rozhraní

Je součástí Zabbix serveru a většinou je také spouštěno na stejném serveru, kde běží Zabbix. Webové rozhraní zasílá dotazy přímo do databáze¹ a tím pomáhá snižovat zátěž serveru. Zabbix Server nefunguje bez webového rozhraní. Webové rozhraní je vytvořeno v jazyce PHP.

5.3.1.3 Proxy

Zabbix Proxy sbírá data určená ke zpracování na serveru. Data jsou sbírána do dočasné cache a pak jsou zasílána na server. Funguje také jako load balancer - část zpracování se uskuteční na úrovni proxy. Výsledkem jsou nižší nároky na přenos dat a výpočetní výkon serveru. Vzdálená komunikace se serverem prochází přes firewall.

¹Zabbix umožňuje komunikaci s databázovými systémy MySQL, PostgreSQL, TimescaleDB, Oracle, SQLite

5.3.1.4 Agent

Je komponenta, která má za úkol získávat hodnoty z cílového zařízení. Většinou se jedná o monitorování systémových zdrojů fyzického či virtuálního stroje. Podmínkou funkcionality agenta je jeho instalace na fyzické zařízení, ze kterého jsou data získávána. Získávání dat z jednotlivých zařízení je efektivnější díky využití systémových volání.

Zabbix nabízí dvě varianty sběru dat - pasivní a aktivní. Pasivní varianta je aktivována vždy, když je zavolán definovaný endpoint (request). Aktivní kolekce dat vyžaduje prvotní získání informací o datech, která se mají sbírat, a ta jsou pak periodicky zasílána ke zpracování. Speciálním případem Agentu je JMX agent, ten získává data z aplikací napsaných v jazyce Java, respektive z virtuálního stroje (JVM).

5.3.2 Shrnutí a splnění kritérií

5.3.2.1 Splnění nároků na funkčnost

Zabbix umožňuje zasílání zpráv přes komponentu *Trapper* pomocí API. Další možností zasílání zpráv je využitím Zabbix proxy a jejích konfigurací. Zabbix neumožňuje zasílání jiných zpráv kromě alertingu.

5.3.2.2 Snadné použití v kontextu aplikace DCIx v clusteru

Zabbix se skládá z několika komponent – server, databáze, proxy. Každá komponenta je jeden obraz v prostředí Kubernetes. Výhodou je, že tato technologie je již ve společnosti Aimtec využívána a zaměstnanci s ní mají zkušenosti.

5.3.2.3 Minimální HW nároky

Nejmenší možné HW nároky jsou 128 MB RAM a 256 MB volného místa pro databázi. S počtem sledovaných strojů rostou i HW nároky. Zabbix na svých stránkách uvádí, že v závislosti na zátěži mohou být nároky na CPU „významné“.

5.3.2.4 Licenční podmínky

Tato technologie splňuje stanovené požadavky na licenční podmínky.

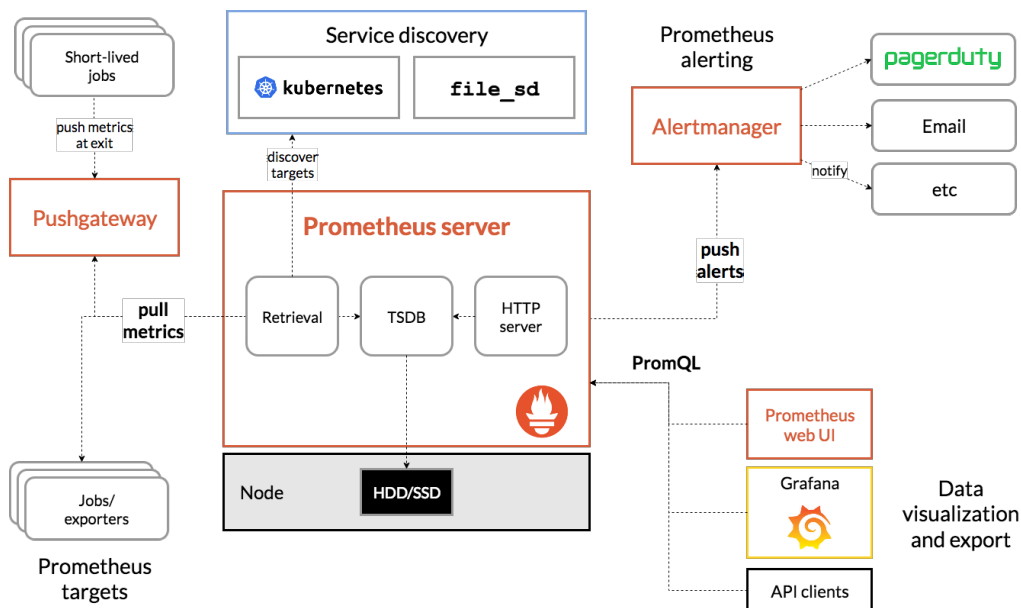
5.3.2.5 Další funkcionality

Zabbix nabízí mnoho další funkcionality (webové rozhraní, API, proxy komponenty, alerting, atd.), ale nesplňuje požadavky na základní funkcionality.

5.4 Prometheus

licence	Apache License 2.0
dokumentace	https://prometheus.io/docs/introduction/overview/
git	https://github.com/prometheus
poslední aktivita	13.06.2022 (den kontroly 13.06.2022)
poslední verze	2.36.1 (ze dne 09.06.2022)

Prometheus [25] [7] [27] je volně dostupný nástroj pro monitorování událostí a zpracování výstrah (alerting), který vznikl nejprve při tvorbě služby SoundCloud, ale později byl otevřen a přijalo jej větší množství organizací. Architektura technologie je ilustrována na obrázku 5.4. Projekt je stále aktivní a má velkou komunitu vývojářů i uživatelů.



Obrázek 5.4: Architektura nástroje Prometheus [24]

5.4.1 Architektura

Nástroj získává data (metriky) v časové řadě z aplikací pomocí klientských knihoven nebo pomocí tzv. exporterů z koncových uzlů pomocí protokolu HTTP metodou typu **pull** nazvaným „scrape“. Požadavky jsou volány periodicky a data z odpovědí jsou následně zpracována serverem a uložena do DB nebo zpracována Alert Managerem. Prometheus používá komponentu *push gateway*. Pomocí této komponenty lze ukládat malé množství dat z cílů, které nelze snadno dosáhnout (například aplikace chráněné branou firewall).

5.4.1.1 Klientské knihovny a exportery

Klientské knihovny slouží k získávání metrik v rámci kódu, ke kterému máme přístup. Prometheus nabízí klientské knihovny pro hlavní programovací jazyky jako je Java, C#, .NET, Go, Python a některé knihovny třetích stran jako jsou Node.js, Haskell, Erlang či Rust. Hlavní výhodou těchto knihoven je odstínění od detailů implementace. Není vynucován formát zpráv, které jsou zasílány serveru.

Prometheus definuje exportery pro kód, kterého není uživatel tvůrcem a nemá k němu přístup. Takový software většinou obsahuje rozhraní pro získání metrik. Exporter je software, který tato data získává a zasílá na server. Komunita stojí za vznikem velkého množství těchto exporterů.

5.4.1.2 Service Discovery

Data jsou získávána z endpointů, které odpovídají jednotlivým službám, procesům či aplikacím. Prometheus vyžaduje nejen informace o tom, jak se k jednotlivým uzlům připojit, ale také další nastavení (např. frekvence žádostí o data).

Způsoby získávání zdrojů k monitorování:

- vytvoření statického seznamu uživatelem
- získávání zdrojů pomocí souboru - může se jednat o konfigurační soubor, jehož obsah se automaticky načte do aplikace
- automatické vyhledávání - například pomocí dotazu nebo pomocí nástrojů třetích stran jako Amazon, Google

5.4.1.3 Prometheus Server

Server nástroje získává data z kolektorů a komunikuje s databázovým úložištěm. Klientská komunikace se serverem je možná pomocí API, nebo dotazy v jazyce „PromQL“.

5.4.1.4 Ukládání dat

Prometheus využívá vlastní zabudovaný databázový systém pro každou instanci serveru. Nespoléhá se na distribuované systémy.

5.4.1.5 Dashboard

Pro náhled dat je k dispozici velmi bohaté API pro přímý přístup a vyhodnocení dotazů v jazyce „PromQL“. Volání adres API umožňuje zobrazení informací v grafech či na „řídících deskách“². Pro detailnější vizualizace je doporučeno využít software Grafana³. Grafana umožňuje zobrazení dat z více instancí serverů Prometheus.

5.4.1.6 Agregace a správa výstrah

Server umožňuje také agregaci záznamů a zpracování výstrah. Agregace je řízená pomocí pravidel a nejčastější agregace jsou již připravené (počítání míry, sumy). Kromě pravidel pro agregaci je možné definovat pravidla i pro zpracování výstrah dle kritérií a severity.

Zpracování výstrah není součástí serveru, ale jedná se o samostatnou službu (server). Tato služba převádí výstrahy na notifikace uživatelům (např. emailová notifikace, integrace na další služby jako je Slack nebo PagerDuty). Uvnitř komponenty je možné také agregovat jednotlivé výstrahy, nebo je potlačit.

5.4.2 Shrnutí a splnění kritérií

5.4.2.1 Splnění požadavků na funkčnost

Prometheus je na tom velice podobně jako Zabbix. Umožňuje zasílání zpráv a metrik mnoha různými způsoby. Nenabízí kromě alertingu žádnou možnost, jak zprávy přeposílat dále.

5.4.2.2 Snadné použití v kontextu aplikace DCIx v clusteru

Pro funkčnost nástroje Prometheus je nutné nainstalovat Prometheus Server. Nasazuje se pomocí deskriptoru do vlastního namespace⁴. Deskriptor pro Kubernetes umožňuje nastavení aplikace - obsahuje například definici

²dashboard

³Multiplatformní open source webová služba pro analýzu a interaktivní vizualizaci dat.

⁴jmenný prostor v Kubernetes prostředí

metrik, či časování „scrape“ požadavku. Výhodou je, že tato technologie je ve společnosti Aimtec již využívána a zaměstnanci s ní mají zkušenosti.

5.4.2.3 Minimální HW nároky

Nejmenší možné HW nároky jsou 390 MB RAM a 600 MB volného místa pro databázi. S počtem sledovaných strojů rostou i HW nároky. Prometheus na svých stránkách uvádí, že v závislosti na zátěži se mění i HW nároky (disk, RAM i CPU). Výpočet využívaných systémových prostředků není triviální.

5.4.2.4 Licenční podmínky

Tato technologie splňuje požadavky na stanovené licenční podmínky.

5.4.2.5 Další funkcionalita

Prometheus nabízí i jiné, než požadované funkčnosti:

- vývoj měřených hodnot v čase
- jazyk PromQL pro dotazování pomocí API
- service discovery
- dashboard
- alerting

5.5 Proprietární řešení

Jedná se o řešení využívající kód, který již v aplikaci DCIx existuje.

Součástí aplikace DCIx je komunikace mezi jednotlivými mikroslužbami. Limitací této komunikace je její synchronita. Ideální by bylo, aby zasílání zpráv ze systémových služeb do jádra bylo asynchronní (aby nebylo potřeba čekat na odpověď všech služeb aplikace DCIx). Není však nutné komunikaci příliš upravovat - doménové objekty je možné využít a v případě nutnosti rozšířit.

Vývojářský tým připravil prototyp mikroslužby `EventBus`, která umožňuje asynchronní komunikaci mezi komponentami.

Sbírání zpráv ze systémových služeb zatím není implementováno. Je třeba vyvinout kód pro sběr dat. Pro tento účel je možné využít framework systémových služeb, který umožňuje spouštět časované úlohy. Úlohou by v tomto případě bylo zaslání zprávy o stavu služeb z komponenty `CoreServices` do `Core`.

Na straně komunikace s klientem je možné využít WebSocket serveru. Ten je již implementován v mikroslužbě **Core**. V aplikaci DCIx je využito jak zasílání zpráv, tak jejich zpracování na straně klienta v Javascriptu.

5.5.1 Shrnutí a splnění kritérií

5.5.1.1 Splnění nároků na funkčnost

Proprietární řešení s využitím stávajících komponent aplikace DCIx nesplňuje plně požadavky na funkčnost. Chybějící funkcionalitou je sběr informací ze systémových služeb o změně jejich stavu. Vzhledem k tomu, že kód je v držení společnosti Aimtec, není složité tento nedostatek odstranit.

5.5.1.2 Snadné použití v kontextu aplikace DCIx v clusteru

Téměř všechny komponenty jsou dostupné v aplikaci DCIx. Výjimkou je mikroslužba **EventBus**. Tuto mikroslužbu bude nutné doplnit mezi stávající mikroslužby. V tomto případě jsou všechny součásti řešení přímo využívané vývojáři aplikace DCIx a každý by měl mít alespoň minimální znalost jednotlivých technologií.

5.5.1.3 Minimální HW nároky

Minimální nároky zatím nelze plně analyzovat. Mikroslužba **EventBus** vyžaduje zdroje navíc. Další nároky na zdroje plynou z rozšíření kódu aplikace DCIx a budou analyzovány. Dá se předpokládat, že nároky budou nižší než u předchozích řešení.

5.5.1.4 Licenční podmínky

Toto řešení splňuje stanovené požadavky na licenční podmínky.

5.5.1.5 Další funkcionalita

Toto řešení neumožňuje další funkcionalitu. Umožňuje kód snadno spravovat a upravovat.

5.6 Shrnutí

Kritéria pro výběr technologie				
název	funkčnost	nasaditelnost (kapitola 5.1.2)	HW nároky	další funkčnost
Decanter	ano	těžká*	střední	ano
Zabbix	ne	střední	vysoké	ano
Prometheus	ne	střední	vysoké	ano
DCIx	ne**	lehká	nízké	ne

Tabulka 5.1: Kritéria pro výběr technologie pro monitoring systémových služeb v aplikaci DClx

** Pro aplikaci DClx by nasazení frameworku Decanter znamenalo využití jiné runtime knihovny (respektive její rozšíření o Apache Karaf, který využívá OSGi, jiná mikroslužba OSGi nevyužívá a není to zatím v plánu).*

*** V současné verzi aplikace DClx tato funkčnost není. Implementace funkčnosti však nepředstavuje zásadní technologický problém.*

5.6.1 Vybraná technologie

Technologie, která je nejvhodnější pro monitoring systémových služeb, je proprietární řešení využívající již existující kód aplikace DClx.

Požadavky na funkčnost splňují jen dvě řešení – proprietární a Apache Decanter. Decanter nabízí další funkčnosti pro monitoring jako je alerting, agregace zasílaných zpráv a integrace na Apache Camel. Jeho velkou a zásadní nevýhodou je vazba na runtime knihovnu Apache Karaf. Pokud by vznikala nová služba pro komunikaci se stroji, která by měla funkčně nahradit `CoreServices` a službu `TCPIPListener`, bylo by toto řešení vhodné.

Architekti aplikace DClx společnosti Aimtec neuvažují o nahrazení vlastního řešení (mikroslužba `CoreServices`) implementací Apache Camel (využívající runtime framework OSGi).

Nevýhodou proprietárního řešení je neexistence kódu pro sběr zpráv z jednotlivých služeb.

6 Komunikační rozhraní pro vzdálenou správu a monitoring služeb DCIx

Cílem této práce je navrhnout nové komunikační rozhraní pro komunikaci a správu služeb v aplikaci DCIx. Stávající verze je již nedostačující z více důvodů. Hlavním důvodem je nutnost uživatelské interakce pro zobrazení nových údajů (aktualizace stavu). Následující kapitola (6.1) popisuje jak jsou data prezentována uživateli.

6.1 Obrazovka systémových služeb

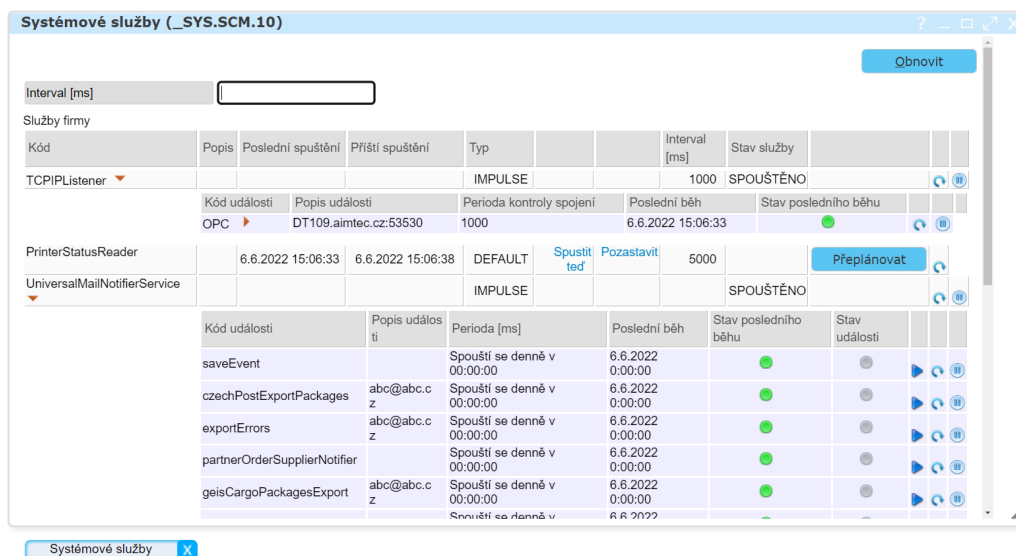
Obrazovka systémových služeb zobrazuje informace o systémových službách. Tato data jsou získávána z komponenty **CoreServices** (kapitola 3.2.5).

Obrázek 6.1 zobrazuje obrazovku „Systémové služby“. Je zde vidět tlačítko pro obnovu informací v pravém horním rohu, dále informace o jednotlivých službách¹. z těchto dat je nejvíce zajímavá část zobrazující informace o systémové službě „TCPIP Listener“² typu OPC³. Tato služba obsahuje největší množství vlastností (jsou zde až dvě úrovně zanoření - jednotlivé eventy služby a dále OPC subskripce viz obrázek 6.2).

¹Většinou se jedná o kód, plánované spouštění, interval a akce, které lze nad jednotlivými službami provádět.

²Informace o TCPIP listeneru je v kapitole 3.2.5.1

³Protokol OPC popisuje kapitola 3.4



Obrázek 6.1: Detail obrazovky systémových služeb

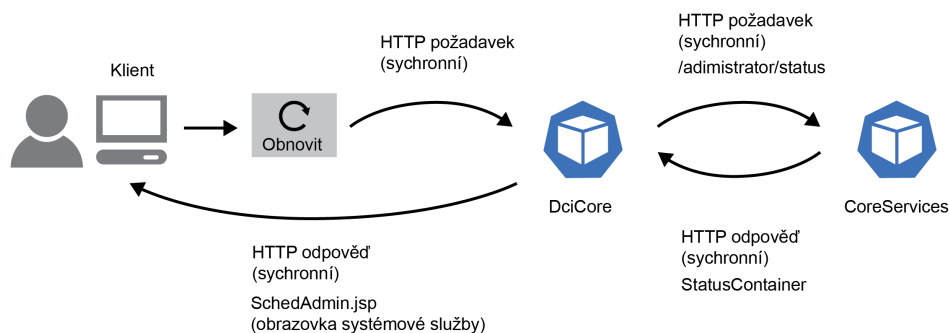
Detailnější obrázek 6.2 zobrazuje informace o TCPIP Listeneru a jednotlivých OPC nodech.

Kód	Popis	Poslední spuštění	Příští spuštění	Typ	Interval [ms]	Stav služby		
TCPIListener				IMPULSE	1000	SPOUŠTĚNO		
	Kód události	Popis události	Perioda kontroly spojení	Poslední běh	Stav posledního běhu			
	OPC	DT109.aimtec.cz:53530	1000	6.6.2022 15:05:35				
	Kontrola odběru	Odebíraný NodeId	Odebíraný AttributId	Spust' transakci	Stav	Poslední běh	Interval příjmu zpráv	
		ns=3;s=String	Value	subTrn1		6.6.2022 12:48:49	100.0	
		ns=3;s=String	Value	subTrn1		6.6.2022 12:48:49	100.0	
		ns=3;s=String	Value	subTrn1		6.6.2022 12:47:22	100.0	
		ns=3;s=String	Value	subTrn1		6.6.2022 12:48:49	100.0	
		ns=3;s=String	Value	subTrn1		6.6.2022 12:48:49	100.0	

Obrázek 6.2: Detail TCPIP Listeneru zobrazující OPC subskripcce

6.2 Stávající řešení

Stávající řešení je ilustrováno na obrázku 6.3.



Obrázek 6.3: Stávající řešení - schéma komunikace

Uživatel klikne na tlačítko „Obnovit“, nebo nově otevře stránku „Systémové služby“. Obsluhou uživatelské interakce je provedení akce ve třídě `cz.aimtec.dci.admin.action.SchedulerAdmin` v komponentě `Core`. V rámci obsluhy je volána metoda `getActiveServices` v mikroslužbě `CoreServices` (ukázka 6.1). Tato metoda pošle *synchronní* požadavek komponentě `CoreServices` na endpoint `administrator/status` (kapitola 3.3.3.1).

```

// get active services from CoreServices
ServicesStatusContainer servicesStatusContainer =
    ServiceList.coreServices.getActiveServices(workspace);
// add instance dependent services
Collection<Service> companyServices =
    servicesStatusContainer.getCompanyServices();
companyServices.addAll(
    companyDispatcher.getInstanceDependentTaskList()
);
Collection<Service> applicationServices =
    servicesStatusContainer.getApplicationServices();
applicationServices.addAll(
    applicationDispatcher.getInstanceDependentTaskList()
);

xForm.setSqlJobs(SqlJobsList.getList());
xForm.setCompanyServices(companyServices);
xForm.setApplicationServices(applicationServices);
xForm.setLocalizer(workspace.getMessageResourcesLocalized());
// forward control to the success page
return (mapping.findForward("success"));

```

Ukázka 6.1: Kód třídy `cz.aimtec.dci.admin.action.SchedulerAdmin`, který komunikuje s `CoreServices`

Logika pro obsluhu tohoto požadavku v `CoreServices` zkontroluje všechny aktivní služby a informaci o jejich stavu serializuje pomocí DTO objektů (kapitola 3.5). Tyto objekty jsou pak zaslány jako odpověď na volání mikroslužby Core ve formátu JSON. Poté následuje deserializace odpovědi do objektu `StatusContainer`.

Objekt `StatusContainer` je přiřazen na `SchedulerAdminForm` a následuje přesměrování uživatele na obrazovku `schedAdmin.jsp` (ukázka 6.2). Tato šablona vykreslí uživateli informace o systémových službách proiterováním všech služeb, které obsahuje třída `SchedulerAdminForm` v atributu typu `StatusContainer`.

```
<action
  path="/schedulerAdmin"
  type="cz.aimtec.dci.admin.action.SchedulerAdmin"
  name="schedulerAdminForm"
  scope="request"
  input="/adm/apl/schedAdmin.jsp">
  <set-property property="authClass"
    value="cz.aimtec.dci.action.DciActionAuthentication"/>
  <forward
    name="success"
    path="/adm/apl/schedAdmin.jsp" />
  <forward
    name="runSqlJobInfo"
    path="/adm/apl/schedAdminRunJobInfo.jsp" />
</action>
```

Ukázka 6.2: Nastavení přesměrování akcí `/schedulerAdmin` v souboru `struts-config.xml`

Pro zobrazení dat uživateli jsou užity dvě šablony. Správné zobrazení na webovém klientovi umožňuje závislost na dalších šablonách (ty v sobě nesou vzor návrhu stránky). Informace o systémových službách obsahují tyto šablony:

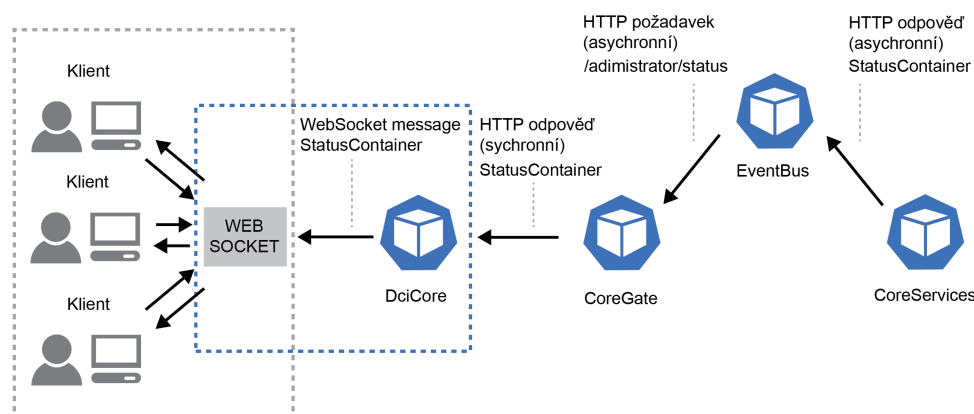
- `/adm/apl/schedAdmin.jsp`
- `/adm/apl/TCPIPListenerAdditionalProperties`

6.2.1 Problémy stávajícího návrhu:

- nutnost uživatelské interakce k zobrazení informací
- uživateli jsou zobrazovány neaktuální informace
- uživateli není při aktualizaci zaslána pouze změna, ale celá stránka
- technologická závislost na JSP
- každý klient zasílá HTTP požadavek z mikroslužby `Core` do `CoreServices`

6.3 Navrhované řešení

Technologický základ navrhovaného řešení je popsán v kapitole 5.5. Schéma návrhu je ilustrováno obrázkem 6.4.



Obrázek 6.4: Navrhované řešení - schéma komunikace

Po otevření obrazovky „Systémové služby“ jsou (některá) data zobrazená na této obrazovce periodicky obnovována bez interakce uživatele. Tato obnova probíhá i na pozadí v případě, že uživatel minimalizuje okno systémových služeb a pracuje s jinou obrazovkou. Uživatel si může periodu obnovy dat nastavit na obrazovce „Nastavení aplikace“ a na stejném místě může tuto funkčnost zakázat (nastavení je popsáno v kapitole 9.2).

Úpravy kódu v navrhovaném řešení lze rozdělit do několika kroků:

- vytvoření systémové služby (**SystemServicesMonitor**) pro sběr dat v mikroslužbě **CoreServices**
- využití asynchronního volání pomocí mikroslužby (**EventBus**)
- úpravy komunikace mezi mikroslužbami
- zobrazení změn dat uživateli (**Frontend**)

Funkčnost tohoto řešení bude ověřena primárně na službě TCPIP Listener konkrétně na eventu typu OPC a to z toho důvodu, že tato služba je nejsložitější a její informace mají největší hloubku zanoření v hierarchii JSP šablon. Řešení bude zpětně kompatibilní a nenaruší současnou funkčnost aplikace DCIx. Tento bod bude splněn otestováním aplikace standardní sadou testů (standardní sada testů je popsána v kapitole 4. Výsledky testování touto sadou jsou detailněji rozepsány v podkapitole 7.7). Testovací oddělení

společnosti Aimtec v budoucnosti vytvoří nové testovací scénáře pokrývající funkčnost tohoto řešení.

6.3.1 SystemServicesMonitor

V mikroslužbě **CoreServices** existuje systémová služba **SystemServiceMonitor**, která sleduje změny stavu v systémových službách (sleduje i sama sebe). Tato služba zasílá asynchronní zprávy do mikroslužby **Core** v intervalu, který je nastavitelný. Zpráva obsahuje informace o stavu pouze těch služeb, které byly v tomto období změněny. Uživateli je umožněno nastavit interval, s jakým jsou data zasílána. Implementace musí dovolit uživateli tuto funkčnost vypnout. Důvodů pro vypnutí může být více – hardwarové nároky nového řešení, zpětná kompatibilita, nepotřebnost v prostředích, kde není dostatek systémových služeb a další. Ve výchozím stavu je tato funkčnost vypnutá a uživatel ji zapíná pouze v případě potřeby.

6.3.2 EventBus

Návrh komunikačního rozhraní počítá s využitím komponenty **EventBus** pro asynchronní komunikaci mezi mikroslužbami **CoreServices** a **Core**. Jedná se o framework, který odstiňuje vývojáře od nastavování asynchronní komunikace. Prototyp této komponenty byl již vytvořen architektem společnosti Aimtec - Stanislava Horáčka, ale nebyl nikde použit. Pro asynchronní komunikaci tato komponenta využívá framework RabbitMQ.

6.3.3 Úpravy komunikace

Řešení definuje několik nových endpointů v jednotlivých mikroslužbách, kterých se komunikace týká (**Core**, **CoreGate**, **CoreServices** a **EventBus**). Speciálním případem je **EventBus**, který nedefinuje endpoint, ale frontu zpráv.

Při komunikaci je nutné částečně upravit DTO objekty. Každá mikroslužba totiž definuje ve svém API objekty, které umí přijímat a pracovat s nimi. Nově navržené řešení vychází ze stávajících objektů mikroslužby **CoreServices** (kapitola 3.5).

Další úpravy jsou provedeny při zasílání zpráv na klienta. Data jsou zasílána z mikroslužby **Core** pomocí WebSocket serveru. Při komunikaci je vhodné poslat takové hodnoty, které umožní nahradit celé stávající DOM objekty (může se tedy jednat například o celé html tagy).

6.3.4 Frontend

Javascriptový klient zpracuje zprávu z WebSocketu a postará se o aktualizaci obsahu pomocí manipulace s DOM na obrazovce systémových služeb. Tomuto klientovi je nutné předat možnost manipulace se službami, eventy (akce) a ikonami indikujícími stav služby. Klient udržuje otevřené spojení s WebSocket serverem a postará se o validní ukončení tohoto spojení po uzavření obrazovky „Systémové služby“.

WebSocket server je již používán v aplikaci DCIx. Součástí této práce není implementace tohoto serveru, ale jen jeho rozšíření o zasílání a zpracování zpráv o stavu systémových služeb.

Pro správnou funkčnost řešení bude upravena struktura JSP šablon tak, aby bylo možné hodnoty příchozí z WebSocket serveru mapovat na elementy webové stránky (DOM). Řešení bude diskutováno s týmem pracujícím na tvorbě nových frontend klientů (v současné době je plánován přechod od JSP šablon na klienta vytvořeného pomocí frameworku Angular⁴).

6.3.5 Výhody a nevýhody tohoto řešení

6.3.5.1 Nevýhody

Hlavní nevýhodou tohoto řešení je potřeba zavedení nové mikroslužby - EventBus pro asynchronní komunikaci. Tím se zvedne celková HW náročnost aplikace DCIx.

Nové řešení je potřeba řádně otestovat v produkčním prostředí, aby bylo možné ověřit jeho správnost.

⁴Framework založený na jazyce Type-Script (jazyk přímo vycházející ze známého jazyka Javascript) určený pro tvorbu webových klientů.

6.3.5.2 Výhody

Díky navrhovaným úpravám bude možné bez interakce uživatele obnovovat informace o službách.

Další výhodou je, že toto řešení používá WebSocket k broadcastingu změn stavu služeb.

Klienti nebudou nuceni pro každé obnovení stránky volat obsluhu z **Core** až do **CoreServices** (a čekat na odpověď).

Řešení by mělo být snadno použitelné při změně technologie klienta. Nový klient (založený na frameworku Angular) může využívat přístup k WebSocketu a pracovat s objekty, které jsou zasílány.

Vedlejším produktem práce je skutečnost, že řešení poslouží jako technologické demo pro asynchronní komunikaci v aplikaci DCIx.

7 Implementace navrhovaného řešení

V rámci implementace návrhu řešení komunikačního rozhraní pro služby DCIx bylo modifikováno několik mikroslužeb – `CoreServices`, `CoreGate`, `Core` a `EventBus`. Samotný vývoj lze rozdělit do pěti částí:

- Monitorovací služba
- `EventBus`
- Úpravy komunikace
- Front-End
- Získávání dat o OPC subskripcích

7.1 SystemServicesMonitor

Komponenta pro monitoring systémových služeb je implementována ve třídě **SystemServicesMonitor**. Tato třída dědí od předka **Service** (viz kapitola 3.2.5). Díky tomu je možné tuto komponentu dle potřeby vypnout, zapnout a periodicky spouštět kód (ukázka 7.1). Konfiguraci této komponenty je možné nastavit na obrazovce „Nastavení aplikace“ nebo v databázi.

```
@Override
public void run() {
    StatusContainer actualStatusContainer =
        getStatusContainerFromServicesSet();
    //příprava seznamu zmen od posledního behu
    actualStatusContainer.getCompanyServices().removeIf(s ->
        lastStatusContainer.getCompanyServices().contains(s));
    lastStatusContainer = actualStatusContainer;
    //zaslání zpravy do mikroslužby Core (pokud je co poslat)
    if(!actualStatusContainer.getCompanyServices().isEmpty() &&
        !actualStatusContainer.getApplicationServices().isEmpty())
    {
        coreGate.sendServicesStateToDCI(actualStatusContainer);
    }
}
```

Ukázka 7.1: Výkonný kód služby `SystemServicesMonitor`

Na předchozí ukázce (7.1) lze vidět zasílání zpráv do mikroslužby **Core**. Načtení informací o systémových službách je implementováno v metodě **getStatusContainerFromServicesSet()**. Kód této metody je k nahlédnutí v ukázce 7.2. Lze zde vidět, jak jsou informace o systémových službách získávány ze správce služeb **ServiceDispatcher** (kapitola 9.2 popisuje jak tuto systémovou službu zapnout / vypnout).

```
private StatusContainer getStatusContainerFromServicesSet() {
    StatusContainer servicesStatus = new StatusContainer();
    servicesStatus.setCompanyServices(new ArrayList<>());
    servicesStatus.setApplicationServices(new ArrayList<>());

    //read company services
    for (Service service :
        dispatcher.getActiveServices(ADMIN_COMPANY_ID)) {
        servicesStatus.getCompanyServices().add(
            service.createServiceStatus()
        );
    }
    // read application services
    for (Service service : dispatcher.getActiveServices()) {
        servicesStatus.getApplicationServices().add(
            service.createServiceStatus()
        );
    }
    return servicesStatus;
}
```

Ukázka 7.2: Načtení informací o službách

Součástí implementace nové systémové služby bylo také zaregistrování této systémové služby do aplikace DCIx doplněním do statického seznamu služeb a definování výchozího nastavení služby pomocí SQL skriptu.

7.2 EventBus

Pro umožnění asynchronní komunikace mezi mikroslužbami **CoreServices** a **CoreGate** vznikla nová fronta zpráv v komponentě **EventBus** s adresou *coreServices/monitorSystemServices*. Na tuto adresu mikroslužba **CoreServices** odesílá změny stavu systémových služeb (ukázka 7.3) a **CoreGate** je odebírá a zasílá do mikroslužby **Core** (obrázek 6.4).

Dále bylo nutné upravit mikroslužby tak, aby mohly využívat již existujícího klienta pro komunikaci s mikroslužbou `EventBus`. Jedná se o dvě zásadní změny:

- úprava klientské části mikroslužeb tak, aby bylo umožněno zasílat asynchronní požadavky
- vložení klienta pro komunikaci do mikroslužeb do mikroslužeb pomocí knihovny *Spring boot* (anotace `@autowired`)

```
public void sendServicesStateToDCI(StatusContainer
servicesContainer) {
    ClientResponseHandler responseHandler = (pResponse) -> {
        this.logger.debug("RESPONSE from EventBus:" +
            pResponse);
    };
    this.callMethodAsync(HttpMethod.POST,
        "coreServices/monitorSystemServices", (Map)null,
        servicesContainer, (Map)null, responseHandler);
}
```

Ukázka 7.3: Zasílání změny stavu systémových služeb z mikroslužby `CoreServices` (součást třídy `SystemServiceMonitor`)

Vzhledem k tomu, že tato mikroslužba (`EventBus`) není zatím využívána v aplikaci `DCIx`, bylo nutné spolupracovat s jejím autorem Stanislavem Horáčkem a společně opravit její chyby (nefunkční zasílání asynchronních zpráv, umožnění nasazení atd.).

7.3 Úpravy komunikace

Komunikaci bylo potřeba upravit na několika místech. Změny se týkaly převážně vytvoření nových endpointů pro komunikaci a následně úpravy DTO objektů pro přenos dat.

7.3.1 Vytvoření endpointů

V rámci úprav komunikace bylo potřeba vytvořit nové endpointy v komponentách, ve kterých se komunikace odehrává. Na ukázce 7.4 lze vidět založení nového endpointu pro mikroslužbu **CoreGate**¹. Definice je v textovém formátu YAML a lze jí zpracovat frameworkem Swagger². V komponentě **Core** je definována nová akce *servicesMonitor*, která zpracovává data přijatá z komponenty **Core**. Následně tato data zasílá pomocí WebSocket serveru klientovi, který s nimi dále pracuje na frontendu aplikace (obrázek 6.4).

```
/coreServices/monitorSystemServices:
  post:
    tags:
      - Core Services
    description: Services state monitoring
    requestBody:
      description: Information about company/system services
        wrapped in StatusContainer
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/StatusContainer'
    responses:
      200:
        description: OK
```

Ukázka 7.4: Vytvoření nového endpointu v mikroslužbě **CoreGate**

7.3.2 Úprava DTO objektů pro přenos dat

Do API mikroslužby **CoreGate** byl doplněn objekt **StatusContainer**, který nese informace o službách. Součástí úprav je také přidání chybějících identifikátorů pro objekty, jejichž vlastnosti se mění (příkladem může být objekt **OPCSubscriptionItem**, který nebylo možné rozlišit od ostatních objektů stejného typu). Výsledkem je umožnění úprav těchto měnících se objektů

¹V mikroslužbě je využíván přístup API first

²Swagger je open source framework pro návrh, tvorbu, dokumentaci a konzumaci RESTful web API.

na straně webového klienta (je možné tyto objekty vyhledávat v DOM pomocí identifikátoru).

7.3.3 Definování nové WebSocket konexce

Do stávajícího interního frameworku pro vytváření WebSocket serveru (třída `WebSocketManager`) byl přidán nový typ WebSocket spojení s názvem *SERVICES*. Pomocí tohoto typu WebSocketu jsou připojeným klientům zasílány změny systémových služeb z mikroslužby *Core*.

7.4 Frontend

Úpravy zobrazení dat klientovi lze rozdělit na dvě části, které spolu úzce souvisí:

- úpravy JSP šablon
- javascriptový framework

7.4.1 Javascriptový framework

Javascriptový framework je implementován v souboru `cz.aimtec.dci.systemServices`. Součástí souboru je připojení na WebSocket server aplikace DCIx (ukázka 7.5). Po přijetí zprávy ze serveru je tato zpráva zpracována frameworkem (ukázka 7.6). Nejprve je text (ve formátu JSON) převeden na objekt a následuje procházení jednotlivých atributů objektu. V případě, že je procházený atribut typu pole, jedná se o další úroveň zanoření³. Pokud je atribut objektu součástí webové stránky a je mapován pomocí HTML značek (viz následující podkapitola), je text příslušné značky přepsán novou hodnotou přijatou ve zprávě. Hodnota může být před zapsáním upravena podle datového typu.

```
function openSocket(userId, semaphoreCode) {
    var semaphoreStorage = $(window);
    var websocketParams = '?code=' + semaphoreCode + '&userid='
        + userId;
    var websocketURL =
        websocketFramework.getWebSocketServerPath(SOCKET_ENDPOINT)
        + websocketParams;
    return websocketFramework.openSocket(semaphoreStorage,
        WEB_SOCKET_STORAGE_KEY, websocketURL, onMessage);
}
```

Ukázka 7.5: Ukázka připojení klienta k WebSocket serveru typu SERVICES

³Nejvýše v hierarchii stojí service, následuje event a nejpodrobnějším (nejhlubším) prvkem jsou tzv. „additionalProperties“.

```

/**
 * Event handler callback
 * Continue in transaction if the semaphore has been released
 */
function onMessage(event, websocket) {
    var message = event.data;

    var currentWindow = windowUtils.getCurrentWindow();
    console.log(message);
    switch (message) {
        case PING_MESSAGE:
            // send pong message back
            websocket.send(PONG_MESSAGE);
            break;
        default:
            var statusContainer = JSON.parse(message);
            var companyServices =
                statusContainer['companyServices'];
            updateServices(companyServices);
            break;
    }
}

```

Ukázka 7.6: Obsluha zpracování zpráv přijatých z WebSocket serveru

7.4.2 Úpravy JSP šablon

Pro potřeby adresace prvků, které se mají dynamicky měnit v Javascriptovém frameworku, bylo potřeba upravit stávající JSP šablony. Do těchto šablon jsou zaneseny nové HTML značky. Pro navigaci přes objekty a jednotlivé úrovně zanoření jsou využívány značky ve formátu:

<úroveň zanoření>-code=<hodnota>

Příkladem takové značky může být například **service-code=TCPIPListener**, **event-code=OPC** nebo **additional-properties-code**.

Dalšími značkou je značka:

attribute=<jméno atributu>

Tato značka definuje v šabloně prvek, kterému je možné přiřadit hodnotu z příchozí zprávy.

Poslední důležitou značkou je:

`data-type=<typ hodnoty>`

Značka **data-type** popisuje, v jakém formátu se mají data do tohoto prvku ukládat. V současné jsou používány dva datové typy:

- **date** – hodnota pro tento prvek je typu datum (je nutné udělat konverzi do cílového datového typu)
- **indicator** – hodnota indikuje stav objektu (běží/neběží)

Dále byl do JSP šablony **schedAdmin.jsp** přidán výše zmíněný javascriptový framework pomocí funkce **require()**.

7.5 Získávání dat o OPC subskripcích

Funkčnost řešení má být otestována na nejsložitější systémové službě: **TCPIPListener**. Nejhlubší (ve smyslu zanoření JSP šablon) kategorií na obrazovce systémových služeb jsou přídatné informace – informace o OPC subskripcích (zanoření informací je ilustrováno na obrázku 6.2).

Původní řešení neumožňovalo aktualizovat informace o stavu OPC subskripcí. V rámci implementace bylo změněno získávání dat o těchto subskripcích. Do třídy **OPCSubscriptionManager** byla přidána mapa, která v sobě drží informace o stavu jednotlivých subskripcí. Další změnou oproti původnímu stavu je, že mapa udržuje historii subskripcí. Je tedy možné ukazovat stav i těch subskripcí, které nejsou připojené.

7.6 Povýšení na nejnovější verzi DCIx

Vzhledem k poměrně dlouhému vývoji bylo nutné udržovat vývojovou větev na nejnovější verzi. Z tohoto důvodu bylo nutné přijmout větší množství změn z hlavní větve vývoje. Některé z těchto změn přinesly nekompatibilní změny vůči starším verzím a dokonce se částečně změnila architektura aplikace. Proběhl přechod k API first vývoji a další významnou změnou bylo rozdělení služeb na komponenty klient, server, model.

Důsledkem zahrnutí výše zmíněných změn v architektuře do řešení diplomové práce bylo zvýšení pracnosti. Definice koncových uzlů a DTO objektů bylo nutné přesunout do definice rozhraní (implementace principu API first). Dále bylo nutné rozdělit samotný kód do komponent klient, server, model v závislosti na jejich příslušnosti (zapracování rozdělení služeb na komponenty klient, server, model).

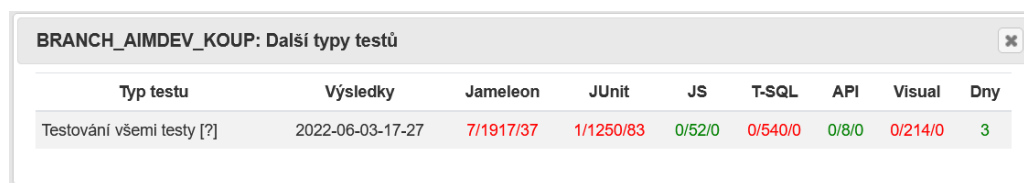
7.7 Testování implementace

Řešení bylo otestováno standardní sadou testů (kapitola 4). Tato sada testů obsahuje:

- cca 1900 integračních testů využívajících framework Jameleon
- cca 1250 jednotkových JUnit testů
- cca 60 JsUnit testů
- cca 540 T-SQL testů
- cca 200 vizuálních testů

Testování první verze nové implementace zjistilo malou chybu, které zapříčinily chybu v 7 integračních testech a 1 chybu v JUnit testu (obrázek 7.1).

Chyba byla při manipulaci s JSP šablonami. V šabloně `schedAdmin.jsp` byl omylem odstraněn identifikátor HTML elementu, který byl součástí XPATH testovacích scénářů. Chyba byla odstraněna a je možné konstatovat, že vývoj nerozbíjí stávající funkčnost aplikace tak, jak bylo definováno v požadavcích na řešení (kapitola 6.3).

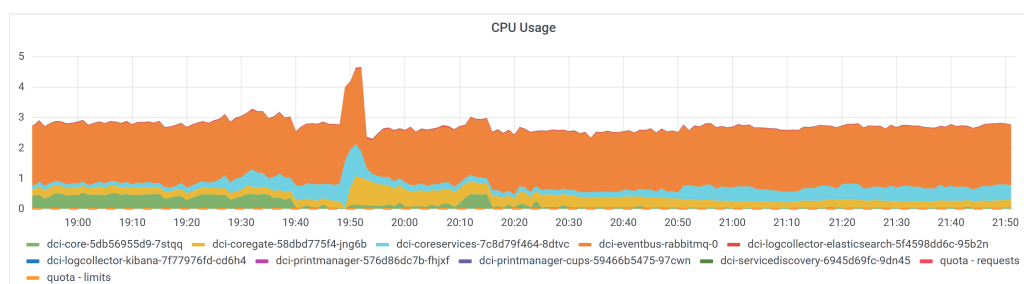


Typ testu	Výsledky	Jameleon	JUnit	JS	T-SQL	API	Visual	Dny
Testování všemi testy [?]	2022-06-03-17-27	7/1917/37	1/1250/83	0/52/0	0/540/0	0/8/0	0/214/0	3

Obrázek 7.1: Souhrn výsledků testování implementace navrhovaného řešení (zdroj: interní nástroj Aimtec)

7.8 Výkonnostní testování navrženého řešení

Tato kapitola popisuje výkonnostní testování implementovaného řešení, které bylo nasazené do interním vývojářském Kubernetes clusteru (monitorován nástrojem Prometheus). Při testování je využito nástrojů Grafana, Prometheus (nástroji byla věnována kapitola 5.4) a RabbitMQ Management. Nástroj Grafana je využit pro vizualizaci dat naměřených pomocí monitorovacího nástroje Prometheus. Na obrázku 7.2 lze vidět jak vypadá graf využití CPU v testovacím nodu během výkonostního testování.



Obrázek 7.2: Nástroj Grafana – grafové znázornění využití CPU v testovacím nodu Kubernetes clusteru

7.8.1 Zdroje pro testování

Vývojářské oddělení společnosti Aimtec má k dispozici sdílený testovací cluster. Cluster se skládá ze 7 nodů – jeden je master. Master node má k dispozici výrazně nižší HW zdroje a nelze v něm spouštět aplikace.

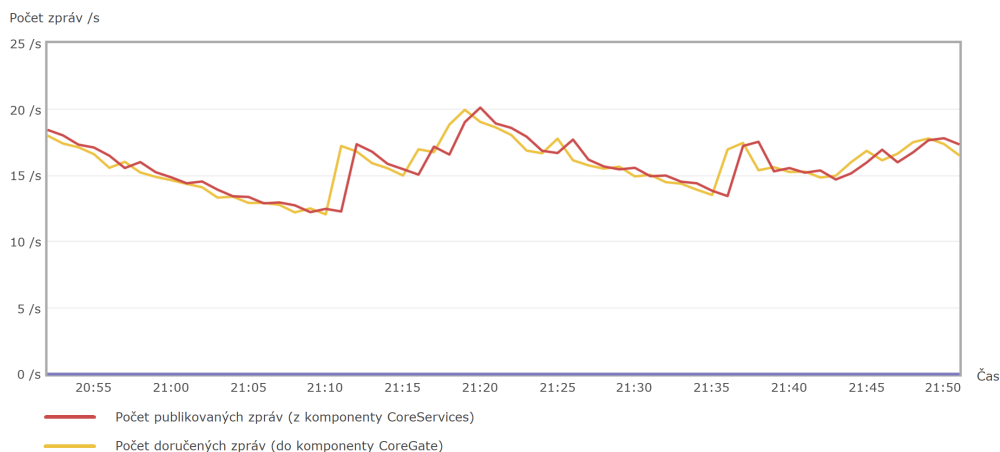
Vývojářům mají k dispozici 6 nodů. Každý node má obsahuje 8 CPU (procesory jsou virtualizované a nelze snadno zjistit jejich specifikace), 32 GB RAM a 50 GB volného místa na disku.

Testování navrženého řešení je omezené na užití pouze jednoho nodu. K tomuto nodu však není výlučný přístup - mohou v něm běžet i další aplikace a tím ovlivňovat měření výkonu systému.

Nastavování zdrojů pro jednotlivé kontejnery je prováděno pomocí pluginu Kubernetes pro aplikaci Visual Studio Code.

7.8.2 Způsob testování

Výkonnostní testování navrženého řešení probíhá na výše zmíněném clusteru. V první řadě je nutné nasadit aplikaci DCIx (s implementací navrženého řešení) do tohoto clusteru. Ke sledování výkonu komponenty **EventBus** je možné využít RabbitMQ management plugin⁴, který je součástí této komponenty. Výstup z nástroje RabbitMQ management lze vidět na obrázku 7.3 – v grafu je vidět, že počet publikovaných zpráv nemusí být vždy totožný s počtem doručených zpráv.



Obrázek 7.3: Vizualizace zpracování zpráv v komponentě EventBus.

Podmínky pro běh aplikace DCIx by v ideálním případě měly být stejné jako u některého ze zákazníků. Tyto podmínky však téměř nelze realizovat. Za účelem částečné simulace zátěže aplikace jsem aplikaci připojil k odběru dat z rychlovarné konvice pomocí protokolu MQTT (služba TCPIP Listener) a k OPC serveru s definicí několika OPC nodů. Zátěž aplikace je potřebná pro generování zpráv s větším počtem znaků (v rámci komunikace probíhá serializace DTO objektů pro reprezentaci změn stavu systémových služeb – více informací k DTO objektům je v kapitole 3.5).

Aby každý scénář měl stejné výchozí podmínky, je po každém dokončeném scénáři vyčištěn kubernetes namespace pomocí příkazu⁵:

```
kubectl delete --all pods --namespace=dev-koup
```

Tento příkaz zajistí, že všechny komponenty jsou spuštěny znovu.

⁴Informace o nástroji a jeho možnostech jsou dostupné na URL adrese <https://www.rabbitmq.com/management.html>

⁵Příkaz využívá nástroj kubectl – jedná se o nástroj pro ovládání Kubernetes pomocí příkazové řádky

Konfigurace se mění nastavením intervalu (milisekundy), s jakým se zasílají data z komponenty **CoreServices** uživateli. Popis, jak nastavit tuto hodnotu, je v uživatelské příručce v kapitole 9.2. Typická hodnota, která bude pro toto řešení nastavena u zákazníka je 1000 ms (1 s). Cílem výkonnostního testování je zjistit výkon systému i při menším intervalu, než bude u zákazníka (větší frekvence obnov).

V závislosti na podmínkách jsou definovány dva scénáře:

- minimální zátěž – systémové služby se mění minimálně
- simulovaná zátěž - zapnutá systémová služba TCPIP Listener, která je připojená k mnoha zařízením pomocí různých protokolů (OPC, MQTT).

Výstupem z výkonnostního testování jsou hodnoty propustnosti. Propustnost definuji jako počet zpráv za sekundu. Hodnoty jsou zprůměrovány za určitý interval – 10 minut, není-li definováno jinak. Výkonnost je také ovlivněna velikostí zasílaných souborů (počet znaků po serializaci DTO objektu), která se odvíjí od počtu definovaných systémových služeb.

7.8.3 Definice testovacích scénářů

Před definicí testovacích scénářů je vhodné seznámit čtenáře s jednotkami, které jsou v Kubernetes cluteru využívány pro definici HW zdrojů [16].

7.8.3.1 CPU

Kubernetes cluster používá pro přidělení množství CPU jednotlivým službám (kontejnerům) jednotky ve tvaru:

- celé číslo - jedná se o počet CPU, které může kontejner využít
- hodnota ve tvaru <číslo>m - **m** je takzvaná „milijednotka“ a platí:
 $1000\ m = 1\ CPU$

7.8.3.2 RAM

Nároky na paměť jsou definovány v bytech a je možné je definovat jako integer, nebo číslo s pevnou řádovou čárkou (fixed-point) s jednotkami E(exa), P(eta), T(era), G(iga), M(ega), k(ilo). Další možností je použít jednotky Ei, Pi, Ti, Gi, Mi, ki. Následující hodnoty jsou totožné:

$$128974848 = 129e6 = 129\ M = 128974848000\ m = 123\ Mi$$

7.8.3.3 Request, limit

Kubernetes cluster vyžaduje dva typy nastavení čerpaní HW zdrojů:

- request - definuje množství HW zdrojů, které kontejner požaduje (reálné čerpání může být vyšší i nižší)
- limit - definuje limitní množství HW zdrojů, které může kontejner využít (reálné čerpání může být jen nižší)

7.8.4 Zjištění celkové propustnosti systému s výchozím nastavením HW nároků

První scénář si klade za cíl ověřit propustnost navrženého řešení ve výchozím nastavení HW zdrojů (definováno v aplikaci DCIx). Omezení HW zdrojů je poměrně značné. Toto nastavení čerpání HW zdrojů jednotlivých kontejnerů (mikroslužeb) lze vidět v tabulce 7.1.

Mikroslužba	CPU limit	CPU request	RAM limit	RAM request
Core	3	815 m	2 Gi	2 Gi
CoreServices	2	652 m	512 Mi	512 Mi
CoreGate	2	815 m	384 Mi	384 Mi
EventBus	0.3 m	0.3 m	256 Mi	256 Mi

Tabulka 7.1: Výchozí HW limity mikroslužeb aplikace DCIx

7.8.4.1 Simulovaná zátěž

Výsledky testovacího scénáře jsou v tabulce 7.2.

interval (ms)	průměrná propustnost (počet zpráv/s)	stabilita
1000	0,77	stabilní
100	7.1	stabilní
10	22	stabilní
5	32	stabilní
1	?	nestabilní*

Tabulka 7.2: Výsledky testování propustnosti systému při simulované zátěži a výchozích HW nárocích komponent

* využití CPU komponentou **EventBus** dosáhlo 200 % limitu (nevalidní stav). Následkem byla nedostupnost komponenty **EventBus** a její restart. Nebylo tedy možné změřit propustnost systému.

7.8.4.2 Minimální zátěž

Výsledky testovacího scénáře jsou v tabulce 7.3

interval (ms)	průměrná propustnost (počet zpráv/s)	stabilita
1000	0,8	stabilní
100	8,8	stabilní
10	22	stabilní
5	34	stabilní
1 s	?	nestabilní*

Tabulka 7.3: Výsledky testování propustnosti systému při minimální zátěži a výchozích HW nárocích komponent

** využití CPU komponentou **EventBus** dosáhlo 200 % limitu (nevalidní stav). Následkem byla nedostupnost komponenty **EventBus** a její restart. Hodnota byla naměřena před restartem komponenty. Dalším problémem se ukázala dostupnost mikroslužby **CoreGate**, která přestala přijímat zprávy. Následoval narůst fronty, přetečení limitu pro RAM a následné selhání komponenty.*

7.8.4.3 Komentář k naměřeným hodnotám

Při intervalu monitorování <10 ms bylo využití CPU komponenty **EventBus** na 99 % limitu (300 m). Lze se domnívat, že při zvýšení HW limitů by mohla být propustnost větší.

Naměřené výsledky jsou částečně překvapením. Propustnost systému je poměrně malá. Při intervalu monitorování 10 ms by propustnost v ideálním případě měla být 100 zpráv za sekundu.

Vzhledem k celkem nízkým HW limitům a jejich využití jednotlivými komponentami jsem usoudil, že je vhodné ověřit propustnost systému při nastavení vyšších hodnot limitu HW nároků.

Z naměřených hodnot lze vidět rozdíl výkonu systému v závislosti na zátěži (tzn. počet systémových služeb, frekvence změn systémových služeb).

7.8.5 Maximální výkon systému dosažitelný na testovacím nodu

Druhým scénářem je otestování systému bez nízkých HW limitů. Cílem tohoto scénáře je určit, zda HW limity výrazně ovlivňují propustnost systému. Nastavené HW limity komponent jsou vidět v tabulce 7.4 – v tabulce lze

vidět výrazné navýšení CPU limitu pro komponentu **EventBus** a také zvýšení velikosti HW zdrojů pro komponentu **CoreGate**.

Na obrázku 7.2 v úvodu kapitoly lze vidět, že komponenta EventBus (červená barva) využívá většinu všech CPU zdrojů pro jednotlivé kontejnery – využití CPU u této komponenty dosahuje hodnoty 2 (= 2 virtuální procesory).

Mikroslužba	CPU limit	CPU request	RAM limit	RAM request
Core	3	815 m	2 Gi	2 Gi
CoreServices	2	652 m	512 Mi	512 Mi
CoreGate	2	815 m	1 Gi	512 Mi
EventBus	4	0.5 m	1 Gi	512 Mi

Tabulka 7.4: Zvětšené HW limity mikroslužeb aplikace DCIx

7.8.5.1 Simulovaná zátěž

První sadou dat jsou výsledky z testování při simulované zátěži. Výsledky provedených testů jsou vidět v tabulce 7.5.

interval (ms)	CoreServices - odeslané (počet zpráv/s)	CoreGate - přijaté (počet zpráv/s)	stabilita
1000 ms	0,92	0,92	stabilní
100 ms	9	9	stabilní
10 ms	45	45	stabilní
5 ms	82	77	stabilní
1 ms	73	69	stabilní

Tabulka 7.5: Výsledky testování propustnosti systému při simulované zátěži a zvětšených HW nárocích komponent (v mezích HW zdrojů nodu)

7.8.5.2 Minimální zátěž

Druhá sada testů běžela bez výrazné zátěže (vypnutá systémová služba TCPIPListener). Výsledky jsou vidět v tabulce 7.6.

7.8.5.3 Komentář k naměřeným hodnotám

Při monitoringu využití HW zdrojů nodu nebylo zjištěno využití všech zdrojů daného nodu. Během scénáře bylo nejvíce využito množství CPU (naměřené hodnoty ve špičce byly až 6 CPU)

interval (ms)	CoreServices - odeslané (počet zpráv/s)	CoreGate - přijaté (počet zpráv/s)	stabilita
1000 ms	0,92	0,92	stabilní
100 ms	9,3	9,3	stabilní
10 ms	56	56	stabilní
5 ms	83	78	stabilní
1 ms	77	69	stabilní

Tabulka 7.6: Výsledky testování propustnosti systému při minimální zátěži a zvětšených HW nárocích komponent (v mezích HW zdrojů nodu)

Z výsledků testování bez HW omezujících limitů vyvstala otázka, která část systému blokuje propustnost systému. Počet zpráv za sekundu stále neodpovídá intervalu, jakým je monitoring systémových služeb spouštěn.

Propustnost systému je sice vyšší, než s omezením, ale stále zaostává za očekáváním. Navíc si lze z výsledků povšimnout, že naměřené hodnoty byly stabilní a komponenty přestaly selhávat na nedostatek HW zdrojů (oproti předchozímu scénáři).

Na výsledcích si lze také všimnout, že při velmi nízkém intervalu pro obnovu (1 ms) je propustnost systému nižší, než při vyšší hodnotě. To je patrně dáno režii vláken při spouštění služby s tak velkou frekvencí.

Na základě naměřených výsledků jsem se rozhodl jako poslední scénář ozkoušet jednotlivé komponenty komunikačního rozhraní a zjistit tak, kde by mohlo být úzké hrdlo.

7.8.6 Otestování propustnosti jednotlivých komponent navrženého řešení

V tomto scénáři jsem se rozhodl otestovat propustnost mezi jednotlivými prvky komunikace. Zaměřil jsem se primárně na komponenty, kterých se týkal můj vývoj:

- CoreGate
- CoreServices
- EventBus

V rámci tohoto scénáře jsem netestoval WebSocket spojení a komunikaci mezi komponentami CoreGate a Core, protože pevně doufám, že tato funkčnost je již v aplikaci DCIx používána a také vhodně otestována.

7.8.6.1 EventBus

První testovanou komponentou je **EventBus**. Před samotným performance testováním jsem provedl krátkou rešerši předpokládaných výsledků měření (komponenta EventBus využívá knihovnu RabbitMQ, která již byla otestována). Na základě této rešerše lze očekávat, že výkon komponenty EventBus se bude odvíjet od velikosti zasílané zprávy [14] a počtu účastníků komunikace [12]. V tomto případě jsou účastníci jen 2 - producent a konzument. Počet zpracovaných zpráv by měl být větší než 500 zpráv za sekundu.

K testování výkonu komponenty EventBus jsem využil specializovaný nástroj pro testování RabbitMQ - RabbitMQ PerfTest⁶. Nástroj umožňuje zasílat velké množství zpráv pomocí jednoduchých příkazů. Příklad takového příkazu může vypadat následovně⁷

```
bin\runjava com.rabbitmq.perf.PerfTest
```

s parametry

```
-x 1 -y 1 -u "test" --id "test" -s 950 -z 60 --rate 1500
```

Propustnost komponenty jsem otestoval na 4 scénářích. Testovací scénář vždy běžel minutu a výsledky jsou průměrné hodnoty za danou dobu. HW zdroje jsou definované jako v předchozích scénářích. Velikost zprávy používá jako jednotku počet znaků. Výsledky testování komponenty jsou vidět v tabulce 7.7.

HW zdroje	velikost zprávy (počet znaků)	EventBus – přijaté (počet zpráv/s)	EventBus – odeslané (počet zpráv/s)
minimální	950	1483	1483
minimální	11569	573	545
maximální	950	11965	11490
maximální	11569	1476	1261

Tabulka 7.7: Výsledky měření počtu zpracovaných zpráv pomocí komponenty EventBus v závislosti na velikosti zprávy (využívá RabbitMQ)

⁶Stránky projektu jsou dostupné na URL
<https://rabbitmq.github.io/rabbitmq-perf-test>

⁷Příkazy jsou kvalitně zdokumentovány na adrese
<https://rabbitmq.github.io/rabbitmq-perf-test/stable/htmlsingle/>

7.8.6.2 CoreServices a CoreGate

Vzhledem k výsledkům předchozích scénářů jsem očekával, že frekvence zasílání zpráv z komponenty CoreServices do EventBus bude vyšší, než frekvence zpracování těchto zpráv komponentou CoreGate (to znamená, že zprávy se budou hromadit v komponentě EventBus).

Testovací scénář jsem definoval následovně (uvedeno v bodech):

- znemožnění přijímání zpráv z EventBusu komponentou CoreGate
- generování (zasílání) zpráv z komponenty CoreServices do EventBusu maximální možnou rychlostí po dobu 10 minut
- zastavení generování zpráv z komponenty CoreServices
- obnovení přijímání zpráv komponentou CoreGate, měření rychlosti zpracování nahromaděných zpráv

Sloupce CoreServices a CoreGate obsahují jako hodnotu průměr propustnosti zpráv za dobu trvání testu – 10 minut

Výsledky tohoto testování jsou vidět v tabulce 7.8.

HW zdroje	velikost zprávy (počet znaků)	CoreServices - odeslané (počet zpráv/s)	CoreGate - přijaté (počet zpráv/s)
maximální	950	80	75
maximální	11569	80	75

Tabulka 7.8: Výsledky měření propustnosti komponent CoreServices a CoreGate

7.8.7 Závěr

Na základě výkonostního testování lze tvrdit, že propustnost systému zcela postačí pro typické použití u zákazníka, který bude využívat obnovu informací nejčastěji 1 za sekundu ($interval = 1000\ ms$)

Lze také tvrdit, že při využití výchozího (minimálního) nastavení HW limitů, má systém rezervy a komponenta EventBus zvládne obsloužit i další požadavky, než jen zasílání stavu systémových služeb.

Na základě výsledků testování si dovoluji tvrdit, že obě komponenty, které využívají komponentu EventBus, nejsou schopné plně naplnit potenciál této komponenty. Vycházím hlavně z výsledků v tabulce 7.8, kde lze vidět, že rozdílná velikost zprávy neměla žádný vliv na propustnost mikroslužeb.

Bylo by vhodné prověřit správnost implementace klienta pro asynchronní komunikaci. Každá nová zpráva v současném stavu navazuje nové spojení a registruje `ResponseHandler`. Je k zamyšlení, zdali čekat na odpověď serveru při zasílání zpráv.

Jednou z příčin nižší propustnosti systému může být také fakt, že logika pro sběr stavu systémových služeb (třída `SystemServiceMonitor`) musí při každém běhu prozkoumat všechny definované služby a zjistit, u kterých z nich proběhla změna.

Bylo by vhodné provést výkonnostní testování také na straně klienta. Nabízí se testování počtu klientů připojených k WebSocket serveru typu *SERVICES*. Dalším možným scénářem by mohlo být testování komunikace mezi komponentami CoreGate a Core, kde jsou zasílány synchronní požadavky (rychlost této komunikace může být jedna z příčin nízké propustnosti systému).

7.9 Navrhované vylepšení implementace

Na tuto diplomovou práci lze z určitého hlediska nahlížet jako na technologické demo. V současné době aplikace DCIx nepracuje s asynchronním voláním metod (ačkoliv je pro tento účel aplikace částečně připravena). Na vzoru této práce bude možné využívat asynchronní volání i v jiných částech aplikace.

Dále je možné na tuto práci pohlížet jako na Proof of concept - jedná se o první řešení, které využívá komunikaci mezi komponentami ke změně vzhledu stránky na straně klienta.

Před tím, než se kód dostane do produkčního prostředí k zákazníkovi, by bylo vhodné práci rozšířit o následující funkčnost:

- doplnit funkčnost pro takzvané „multicompany“⁸ prostředí. Implementace zobrazuje pouze administrátorské služby. Aktuálně platí:
`companyId = AimtecId`.
- ke komunikaci využít stávající WebSocket konekci, která je používána pro zasílání zpráv uvnitř aplikace DCIx (implementace definuje novou konekci WebSocket serveru a klienta).
- upravit funkcionalitu javascript frameworku tak, aby bylo možné při otevření obrazovky „Systémové služby“ definovat obsluhu, která by po zavření okna ukončila WebSocket spojení (řešení by mělo umožňovat registrovat libovolnou obsluhu, která bude zavolána po zavření okna obrazovky).

⁸Definovaná společnost (company) využívající aplikaci DCIx může mít různá práva na služby a tím pádem se přihlášeným klientům mohou zobrazovat různé služby

8 Závěr

Cílem práce bylo prozkoumat možnosti monitorovacích nástrojů a následně vytvořit nové komunikační rozhraní pro monitorování stavu systémových služeb.

Zkoumané nástroje nenabídly tolik přidané hodnoty, aby jejich užití bylo obhajitelné oproti nárokům na vývoj. Analýza mi však otevřela cestu k dalšímu rozšiřování komunikačního rozhraní a přinesla inspiraci v oblasti funkcionality. V rámci práce jsem si také mohl celkem podrobně prohlédnout architekturu mikroslužeb (vývoj, použití, komunikace) a věřím, že tyto znalosti využiji.

Součástí zadání bylo (kromě analýzy a vývoje) dodržení architektury aplikace DCIx a byly kladeny nároky na testování – nový vývoj nesmí ovlivnit stávající aplikaci. Zadání bylo splněno i s těmito nároky. To potvrzuje fakt, že výsledkem spuštění testovacích scénářů aplikace DCIx jsou pouze úspěšné běhy.

Pro nasazení funkčnosti do produkčního prostředí je potřeba vykonat ještě několik kroků, které ale přesahují zadání (téma) této práce. Jedná se například o dokončení kontejnerizace mikroslužby EventBus či implementace navrhovaných vylepšení z kapitoly 7.9.

Pevně věřím, že vzniklé rozhraní poslouží nejen pro svůj účel (vizualizace stavu a správa systémových služeb aplikace DCIx), ale také jako vzor pro další komponenty využívající asynchronní volání služeb. Přidanou hodnotou je fakt, že kód byl implementován s vědomím vývoje nového webového klienta, který bude moci tento kód využít.

Asi největší překážkou při tvorbě práce bylo probíhající převedení aplikace na novou architekturu. Vzhledem k architektonickým změnám bylo nutné udržovat vývojovou větev vývoje na nejnovější verzi – zahrnutí častých změn přicházejících z hlavní větve vývoje. Typicky je vývoj v aplikaci DCIx výrazně kratší a nepromítne se do něj tolik změn.

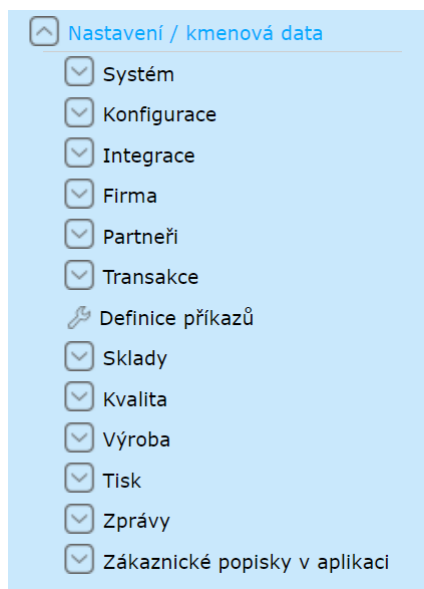
9 Uživatelská příručka

V této kapitole je nastíněno, jak je možné vyzkoušet implementaci nového komunikačního rozhraní aplikace DCIx

9.1 Navigace na obrazovku Systémové služby

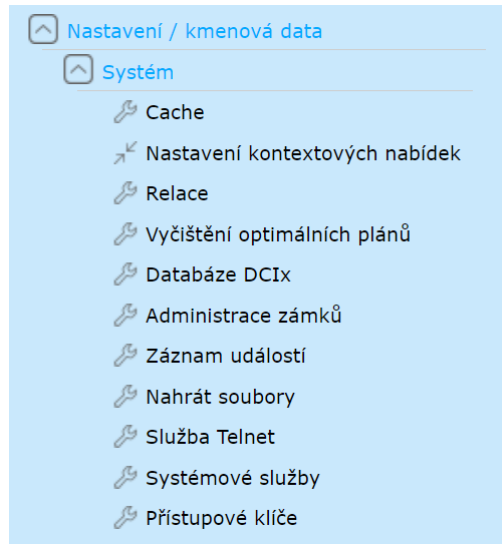
Na obrazovku **Systémové služby** je možné se dostat buď pomocí levého menu, nebo pomocí vyhledávací lišty v navigační liště aplikace.

První postup vyžaduje v levém menu najít položku **Nastavení / kmenová data**(obrázek 9.1).



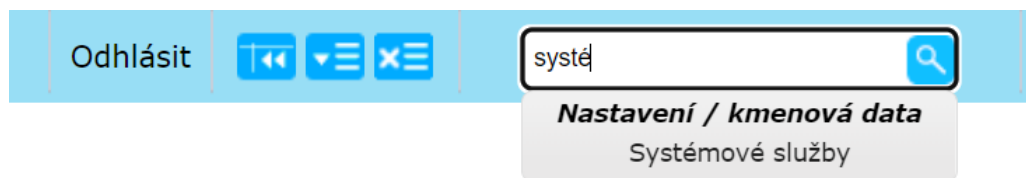
Obrázek 9.1: Otevření obrazovky **Systémové služby** - navigace v levém menu

V podmenu vybrat možnost **Systém** a zde lze nalézt hledanou položku **Systémové služby** (obrázek 9.2). Po kliknutí se na hlavní ploše aplikace objeví nové modální okno s danou obrazovkou.



Obrázek 9.2: Otevření obrazovky **Systémové služby** - navigace v levém menu (detail)

Druhá možnost obnáší vyplnění alespoň části hledaného textu (Systémové služby) do vyhledávací lišty (obrázek 9.3). Funkce autocomplete by měla za-fungovat a měla by uživateli nabídnout kliknutím otevřít danou obrazovku.



Obrázek 9.3: Otevření obrazovky **Systémové služby** – využití vyhledávací lišty aplikace

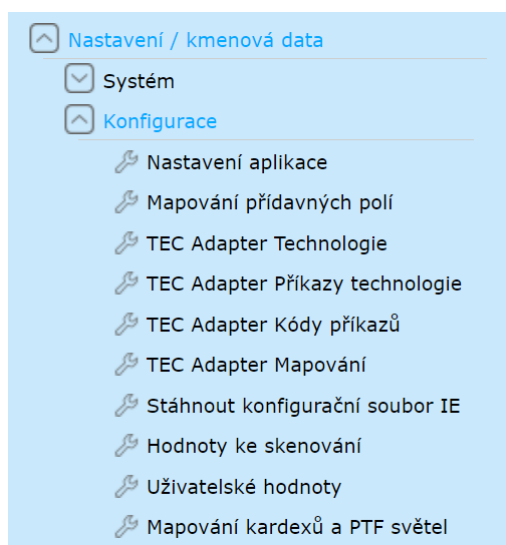
Na obrazovce Systémové služby je možné sledovat výsledek implementace nového komunikačního rozhraní a to nejlépe na eventu typu OPC a jeho OPC subskripcích.

Pro detailnější analýzu je zde možné spustit vývojářské nástroje a zkontrolovat funkčnost komunikace s WebSocket serverem.

Je zde také možné provádět akce nad danými službami - pozastavovat je a spouštět (pomocí příslušných ikoněk). Výsledek těchto akcí by se měl projevit dle nastavení intervalu (nastavení intervalu je popsáno v kapitole 9.2).

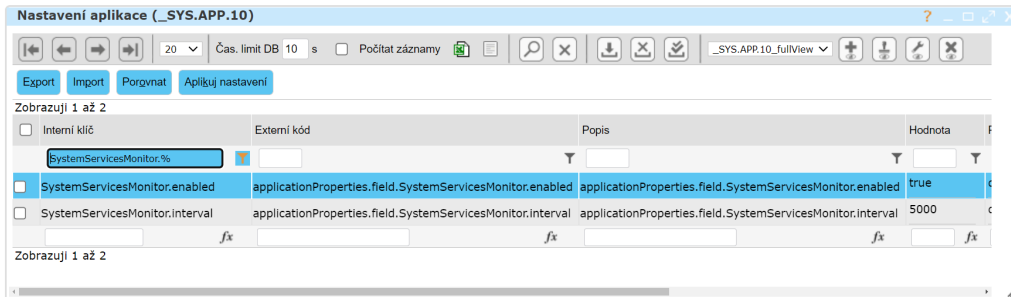
9.2 Nastavení monitorování stavu systémových služeb

Monitoring systémových služeb je možné vypnout v aplikaci DCIx. Slouží k tomu obrazovka **Nastavení aplikace**. Na tu je možné se dostat pomocí levého menu přes položku **Nastavení / kmenová data** (obrázek 9.4). Následně **Konfigurace** a zde už je možné kliknout na položku **Nastavení aplikace**.



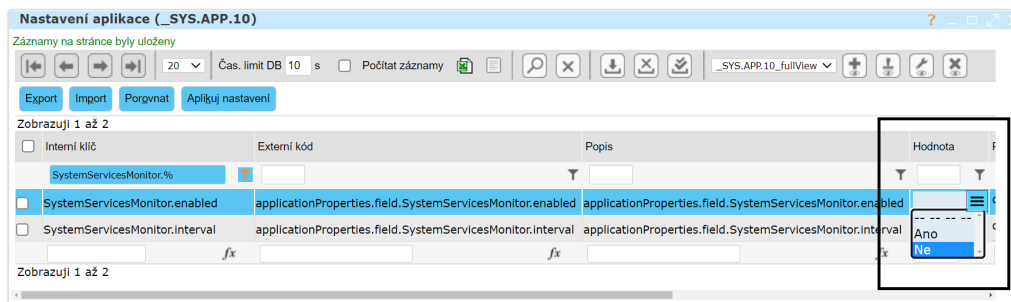
Obrázek 9.4: Otevření obrazovky **Nastavení aplikace** – levé menu

Na této obrazovce je poté nutné vyhledat klíč `systemServices.monitor.enabled` a to tak, že se tento řetězec vloží do políčka **kód** a poté se zmáčkne ikonka lupy (obrázek 9.5).



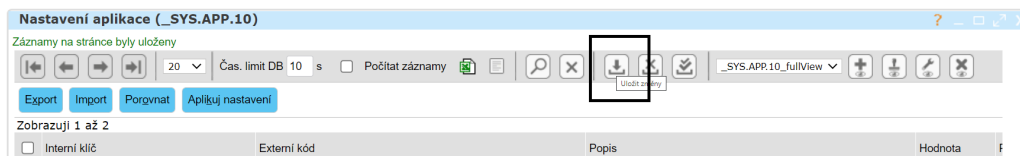
Obrázek 9.5: Nastavení služby monitorování systémových služeb – vyhledání klíče

Kliknutím na hodnotu v kolonce **hodnota** se objeví výběr možností, které je možné zadat. Pro vypnutí je nutné nastavit hodnotu na **Ne** a naopak **Ano** pro zapnutí (obrázek 9.6).

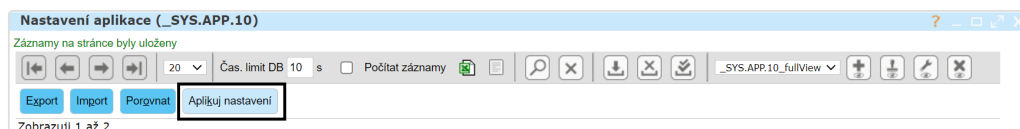


Obrázek 9.6: Nastavení služby monitorování systémových služeb – nastavení hodnoty

Nastavení je nutné nejprve uložit kliknutím na ikonku uložení (obrázek 9.7) a následně kliknout na tlačítko s textem **Aplikuj nastavení** (obrázek 9.8). Nastavení je následně aplikováno (může se stát, že aplikace nezareaguje okamžitě a bude nějakou dobu trvat, než se nastavení aplikuje).



Obrázek 9.7: Nastavení služby monitorování systémových služeb – uložení hodnoty



Obrázek 9.8: Nastavení služby monitorování systémových služeb – aplikování hodnoty

Druhým nastavením, které je možné upravit, je interval, s jakým se zasílají zprávy o službách mezi komponentami `CoreServices` a `Core`¹.

Nastavení je v tomto případě obdobné, jen místo klíče `systemServices.monitor.enabled` je použit klíč `systemServices.monitor.interval` a hodnotou je interval mezi zprávami v milisekundách.

¹V kapitole 7.8 bylo ozkoušeno, že nastavení nižší, než 10 ms nemusí být stabilní, pokud nejsou správně nastaveny HW zdroje aplikace.

Literatura

- [1] AG, P. OPC Ua, 2022. Dostupné z:
<https://www.paessler.com/it-explained/opc-ua>.
- [2] AIMTEC. Schéma aplikace DCIx po oddělení mikroslužeb. Dostupné z:
https://aimtec.sharepoint.com/sites/1dci.internalprojects/_layouts/15/Doc.aspx?sourcedoc={d4d53366-83a3-4014-83f1-002caeeec24e}.
Interní dokument společnosti AIMTEC a.s. – citováno 27.08.2021, 2021.
- [3] BENEŠ – MICHL. *Aimtec* [online]. Aimtec, 2021. [cit. 2021/06/05]. O společnosti Aimtec. Dostupné z: <https://www.aimtecglobal.com/>.
- [4] BENEŠ – MICHL. *DCIx* [online]. Aimtec, 2021. [cit. 2021/06/05]. O produktu DCIx. Dostupné z: <https://www.aimtecglobal.com/dcix/>.
- [5] BENEŠ – MICHL. *DCIx-MES* [online]. Aimtec, 2021. [cit. 2021/06/05]. O produktu DCIx-MES. Dostupné z:
<https://www.aimtecglobal.com/dcixmes/>.
- [6] BENEŠ – MICHL. *Jak inovujeme náš systém DCIx? Porcujeme ho jako slona!* [online]. Aimtec, 2021. [cit. 2021/06/15]. O společnosti Aimtec. Dostupné z: <https://kariera.aimtecglobal.com/jak-inovujeme-nas-system-dcix-porcujeme-ho-jako-slona-/>.
- [7] BRAZIL, B. *Prometheus: Up & Running 1st edition: 9781492034148, 9781492034094*. O'Reilly Media, 2018. Dostupné z:
<https://books.google.be/books?hl=cs&lr=&id=QW1jDwAAQBAJ&oi=fnd&pg=PT17&dq=prometheusmonitoring&ots=5udfyILaC4&sig=yNjUTwfEa5KLobqHjs8xqD2oY5o#v=onepage&q=prometheusmonitoring&f=false>. ISBN 9781492034094.
- [8] DALLE., V. A. – LEE, S. K. *Zabbix network monitoring essentials*. Packt Publishing Limited, 2015. Dostupné z: <https://www.packtpub.com/product/zabbix-network-monitoring-essentials/9781784399764>. ISBN 9781784399764.
- [9] DOBŘIČKA, P. Implementace generátoru datových pohledů pro vytváření sestav ve webové aplikaci DCIx. Diplomová práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, 2018.
- [10] DRDA, M. Datová analýza v dodavatelském řetězci. Diplomová práce, Západočeská univerzita v Plzni, Fakulta ekonomická, 2019.

- [11] GMBH, U. A. OPC UA Subscription Concept, 2022. Dostupné z: <https://documentation.unified-automation.com/uasdkc/1.4.0/html/L2UaSubscription.html>.
- [12] HONG, X. J. – SIK YANG, H. – KIM, Y. H. Performance Analysis of RESTful API and RabbitMQ for Microservice Web Application. In *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, s. 257–259, 2018. doi: 10.1109/ICTC.2018.8539409.
- [13] HORÁČEK, S. Dekompozice monolitické aplikace. Hřiště inovací – přednáška, 9 2016. citováno 15.06.2021.
- [14] IONESCU, V. M. The analysis of the performance of RabbitMQ and ActiveMQ. In *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, s. 132–137, 2015. doi: 10.1109/RoEduNet.2015.7311982.
- [15] KANTOŘÍK, M. Komunikace s řídicí jednotkou pro ovládání poloautomatických VNA vozíků. Bakalářská práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, 2017.
- [16] KUBERNETES. Resource Management for pods and containers, Apr 2022. Dostupné z: <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>.
- [17] KVAPIL, J. DCIx programátorský manuál, DCIx Architecture Diagram. Dostupné z: https://aimtec.sharepoint.com/sites/1DCI/_layouts/15/Doc.aspx?sourcedoc={c46658a7-7202-491b-ba38-b0f3f1040acf}. Interní dokument společnosti AIMTEC a.s. – citováno 27.08.2021.
- [18] OLUPS, R. *Zabbix 1.8 network monitoring monitor your networks hardware, servers, and Web performance effectively and efficiently*. Packt Pub., 2010. Dostupné z: https://books.google.cz/books?hl=cs&lr=&id=fsjNquHrQfYC&oi=fnd&pg=PT13&dq=zabbixarchitecture&ots=EcC-FqUxoY&sig=PjgF3rYDRrZs4YxtEsayllfCA4E&redir_esc=y#v=onepage&q=zabbixarchitecture&f=false. ISBN 9781783283507.
- [19] OPCCONNECT.COM. OPC Unified Architecture, 2022. Dostupné z: <https://www.opcconnect.com/ua.php>.
- [20] OPCFOUNDATION. OPC 10000-4 Unified Architecture Part 4 Services Subscription model, 2022. Dostupné z: <https://reference.opcfoundation.org/v104/Core/docs/Part4/5.13.1/>.
- [21] OPCFOUNDATION. OPC UA Specifications, 2022. Dostupné z: <https://opcfoundation.org/developer-tools/specifications-unified-architecture>.

- [22] OPCFOUNDATION. What is OPC?, Jun 2017. Dostupné z: <https://opcfoundation.org/about/what-is-opc/>.
- [23] OSMANAGIC, A., 2020. Dostupné z: <https://bestmonitoringtools.com/install-zabbix-proxy-on-raspberry-pi/>.
- [24] PROMETHEUS. Overview: Prometheus, 2021. Dostupné z: <https://prometheus.io/docs/introduction/overview/>.
- [25] PROMETHEUS. Overview: Prometheus, 2021. Dostupné z: <https://prometheus.io/docs/>.
- [26] TALEND. Event Monitoring architecture with Decanter, 2022. Dostupné z: <https://help.talend.com/r/lg54vtLQtA5FkzJek2SwcQ/3h8IvM0ktT8Jm0vAGSYlnA>.
- [27] TURNBULL, J. *Monitoring with Prometheus*. Turnbull Press, Jun 2018. Dostupné z: https://books.google.com/books/about/Monitoring_with_Prometheus.html?id=Et1fDwAAQBAJ. ISBN 9780988820289.
- [28] ZABBIX. The Enterprise-Class Open Source Network Monitoring Solution, 2022. Dostupné z: <https://www.zabbix.com/>.
- [29] ČASTA, T. Nové uživatelské rozhraní a navigace pro DCIx. Diplomová práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, 2013.
- [30] ŽUFÁNEK, J. Podnikový informační systém postavený na architektuře microservice. Diplomová práce, Univerzita Pardubice, Fakulta elektrotechniky a informatiky, 2019.

A Slovníček pojmů

DTO (Data transfer object) - objekt pro přenos dat mezi procesy (v kontextu této práce – přenos dat mezi mikroslužbami)

HMI (Human Machine Interface) - rozhraní mezi člověkem a strojem, typicky se jedná o zobrazovací zařízení

IoT (Internet of things) – v informatice označení pro síť fyzických zařízení, vozidel, domácích spotřebičů a dalších prvků

MES (Machine Execution System) – systémy tvořící vazbu mezi podnikovými informačními systémy a systémy pro automatizaci výroby

MOM (Manufacturing Operation Management) – systémy pro správu end-to-end výrobních procesů s cílem optimalizovat efektivitu

ORM (Object Relation Mapping) – technika, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem

cluster - skupina spolupracujících počítačů, v souvislosti s Kubernetes se jedná o prostor v aplikaci Kubernetes který poskytuje fyzické nebo virtualizované počítače

endpoint - označení pro síťový uzel, který slouží jako zdroj nebo cíl komunikace, ale nezajišťuje propojení sítě

kontejner - základní manipulační jednotka pro virtualizace na úrovni OS (může obsahovat spustitelnou aplikaci)

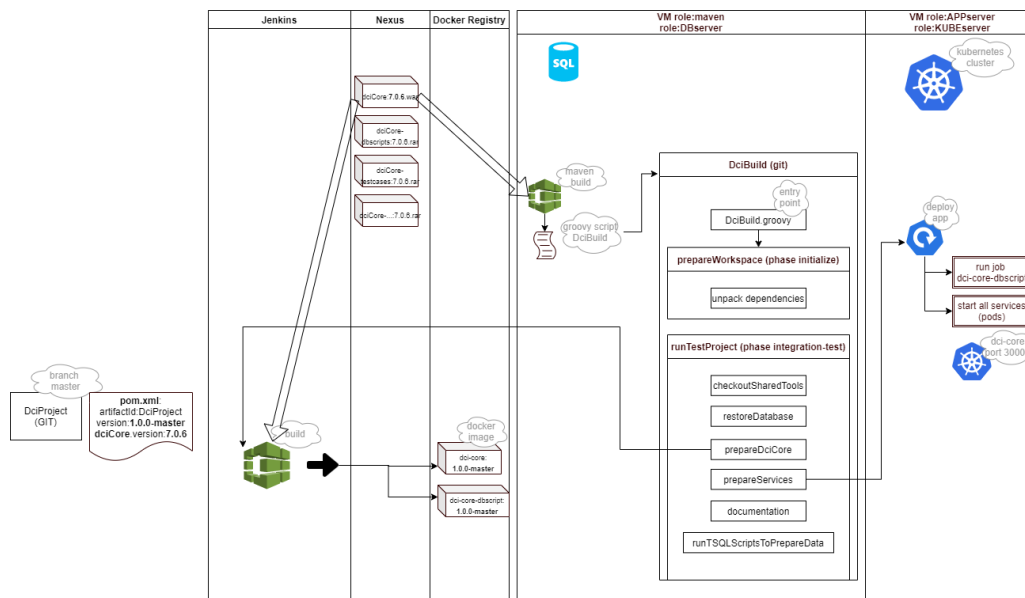
mikroslužba - architektonický přístup k sestavování aplikací, kdy se každá základní funkce, nebo služba, sestavuje a nasazuje nezávisle jako tzv mikroslužba

on premise - řešení, představuje software či hardware, který je uložen v interní infrastruktuře a prostoru společnosti

reporting - přehledné zpracování dat za účelem zobrazení informací o aktuální situaci nebo trendech

node - počítač v aplikaci Kubernetes který může být fyzický nebo virtuální

B Přílohy



Obrázek B.1: Schéma spouštění testů aplikace DCIx. Zdroj: Aimtec

Seznam obrázků

2.1	Diagram architektury aplikace DCIx. Přejato z prezentace „De-kompozice monolitické aplikace“. [13]	12
3.1	Schéma aplikace DCIx po oddělení mikroslužeb [2]	15
3.2	Detailní schéma mikroslužby Core [17]	16
4.1	Zobrazení výsledků testů pomocí interního nástroje DciIndex. Zdroj: Aimtec	30
4.2	Zobrazení výsledků testů pomocí reportu pro SQL Server Reporting Services. Zdroj: Aimtec	31
5.1	Architektura nástroje Decanter	37
5.2	Architektura nástroje Zabbix [23]	41
5.3	Zabbix Data Flow [8]	42
5.4	Architektura nástroje Prometheus [24]	44
6.1	Detail obrazovky systémových služeb	51
6.2	Detail TCPIP Listeneru zobrazující OPC subskripce	51
6.3	Stávající řešení - schéma komunikace	52
6.4	Navrhované řešení - schéma komunikace	55
7.1	Souhrn výsledků testování implementace navrhovaného řešení (zdroj: interní nástroj Aimtec)	67
7.2	Nástroj Grafana – grafové znázornění využití CPU v testovacím nodu Kubernetes clusteru	68
7.3	Vizualizace zpracování zpráv v komponentě EventBus.	69
9.1	Otevření obrazovky Systémové služby - navigace v levém menu	79
9.2	Otevření obrazovky Systémové služby - navigace v levém menu (detail)	80
9.3	Otevření obrazovky Systémové služby – využití vyhledávací lišty aplikace	80
9.4	Otevření obrazovky Nastavení aplikace – levé menu	81
9.5	Nastavení služby monitorování systémových služeb – vyhledání klíče	82
9.6	Nastavení služby monitorování systémových služeb – nastavení hodnoty	82
9.7	Nastavení služby monitorování systémových služeb – uložení hodnoty	83
9.8	Nastavení služby monitorování systémových služeb – aplikování hodnoty	83
B.1	Schéma spouštění testů aplikace DCIx. Zdroj: Aimtec	88