TECHNICKÁ UNIVERZITA V LIBERCI
www.tul.cz

Systém shromažďování a interpretace aplikačních metrik v rozsáhlém AIX prostředí

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Jan Paulík**
Vedoucí práce: Mgr. Milan Keršláger

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
www.tul.cz

The system of gathering and interpreting of application metrics in the large AIX environment

Diploma thesis

Study programme: N2612 – Electrotechnology and informatics
Study branch: 1802T007 – Information technology

Author: **Bc. Jan Paulík**
Supervisor: Mgr. Milan Keršláger

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2014/2015**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jan Paulík**
Osobní číslo:	**M14000175**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Systém shromažďování a interpretace aplikačních metrik v rozsáhlém IBM Power prostředí**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Seznamte se s možnostmi sběru a uložení dat pro rozsáhlé IBM Power systémy s použitím xCATu.
2. Navrhněte systém, umožňující sběr dat a jejich interpretací v rozsáhlém IBM Power prostředí, navržený systém implementujte a zvolte vhodný způsob uložení dat.
3. Ověřte správnou funkci navrženého systému a demonstrujte možnosti a omezení nasazení v reálném provozu, diskutujte další možnosti rozvoje vytvořeného systému.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **cca 50 stran**

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

Seznam odborné literatury:

[1] Gift Noah, Jones M. Jeremy: Python for Unix and Linux System Administratio, O'Reilly Media, 2008, ISBN: 978-0596515829
[2] Krosing Hannu, Roybal Kirk, Mlodgenski Jim: PostgreSQL Server Programming, Packt Publishing, 2013, ISBN: 978-1-84951-698-3
[3] Randal K. Michael: AIX 5L Administration, Mcgraw-hill, 2002, ISBN: 078-3254039063
[4] Payne J.: Beginning Python, Wiley Publishing, Inc., Indianapolis, Indiana, 2010, ISBN: 978-0-470-41463-7
[5] IBM Redbooks [online][2014-10-18] URL: http://www.redbooks.ibm.com/
[6] Perzl.org [online][2014-10-18] URL: http://www.perzl.org/aix/

Vedoucí diplomové práce: **Mgr. Milan Keršláger**
Ústav nových technologií a aplikované informatiky

Konzultant diplomové práce: **Ing. Petr Gänsel**
Síť, s.r.o.

Datum zadání diplomové práce: **20. října 2014**

Termín odevzdání diplomové práce: **15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

Technická univerzita v Liberci - 14 -

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 12.5.2015

Podpis: 

Abstrakt

Tato diplomová práce se zabývá návrhem systému, který umožňuje shromažďování a interpretaci aplikačních metrik z rozsáhlého IBM Power prostředí. Systém je vyvíjen pro společnost SÍŤ, spol. s r. o., jejíž hlavní činnost spočívá v návrhu řešení a realizaci sofistikovaných komunikačních systémů.
Systém se skládá ze dvou částí. První část zajišťuje získávání informací a jejich import do databáze a druhá část, v podobě webové aplikace, umožňuje jejich interpretaci. V rámci zpracování návrhu byly použity moderní metody programování. Webová aplikace byla navržena pomocí PHP frameworku, HTML a CSS a obsahuje jak uživatelské, tak administrační prostředí, kde je možné provádět nastavení systému. Pro uložení dat byla vybrána postgreSQL databáze.
Vzniklá webová aplikace byla instalována do izolovaného a produkčního prostředí, kde proběhlo její testování.

Klíčová slova: sběr dat, webová aplikace, MVC, IBM, framework, databáze

Abstract

This thesis is focused on a system which was designed to collect and interpret application metrics from the extensive IBM Power environment. This system is being developed for the company SÍŤ, spol. s r. o., which is focused on the design and realization of sophisticated communication systems. This system is composed of two parts. The first part obtains information and imports that information into the database. The second part works as a web application to interpret this information. Modern methods of programming were used to prepare this thesis. The web application was designed by using PHP framework, HTML and CSS. It contains an environment which can be setup by both the administrator and user. The PostgreSQL database was chosen to store the data. The web application was installed into an insulated and productive environment for testing.

Keywords: data collection, web applications, MVC, IBM, framework, database

Poděkování

Za odborné vedení mé diplomové práce bych rád poděkoval mému vedoucímu Mgr. Milanu Keršlágerovi. Velké poděkování patří také mému konzultantovi Ing. Petru Gänselovi, který mi ochotně poskytl cenné rady.

Obsah

Seznam tabulek

Seznam obrázků

Seznam zkratek

RPM	Red Hat Package Manager
LPAR	Logical partition
HMC	Hardware management console
VIO	Virtual I/O
SEA	Shared Ethernet adapter
MPIO	Multipath I/O
vSCSI	virtual Small Computer System Interface
LVM	Logical Volume Management
JFS	Journaling File System
HPC	High-performance computing
PHP	Hypertext Preprocessor
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
MVC	Model-View-Controller
OOP	Object-oriented Programming
XML	Extensible Markup Language
CRUD	Create,read,update and delete
SSH	Secure Shell
PSH	Parallel Remote Shell
CMDB	Configuration Management Database

1 Úvod

V současné době pracuji jako Linux a AIX administrátor ve společnosti SÍŤ, spor. s r.o., která je významným dodavatelem, implementátorem a poskytovatelem služeb pro systémy Power a operačního systému AIX běžícího na Hardware IBM. Významnými zákazníky této společnosti jsou automobilové závody v České republice a zahraničí. Firma SÍŤ, spor. s r.o. spravuje IT infrastrukturu, která zahrnuje stovky instancí operačního systému AIX a dalších serverů s jinými operačními systémy. Pro fyzický Hardware, virtuální servery a operační systémy jsou informace evidovány v centrální CMDB databázi, která je v určitém časovém intervalu pravidelně aktualizována.

Aktualizace je prováděna nejen automatickým, ale i manuálním sběrem informací, proto je nutné uchovávat dílčí záznamy aktuální. V současné době je uchovávání neefektivní a často nesrozumitelné. Na základě této skutečnosti mi byla nabídnuta možnost vývoje systému, který by usnadnil sběr těchto dat.

Cílem této diplomové práce je navržení systému pro shromažďování a interpretaci aplikačních metrik z rozsáhlého IBM Power prostředí. Ten zahrnuje proces načtení dat v podobě textových souborů, jež jsou generovány pomocí skriptů a vnitropodnikovou webovou aplikaci, které umožní export aktuálních dat.

Oba celky by měly splňovat základní požadavky, které se případně mohou rozšířit o další funkce, jako je např. přehled instalovaných balíčků na jednotlivých systémech nebo fyzické umístění diskového pole, ze kterého je přidělen virtuální disk. Webová aplikace by měla zahrnovat administraci pro nastavení systému a uživatelské prostředí pro interpretaci dat. Aplikace bude realizována pomocí PHP frameworku s využitím HTML, CSS a pro uchování dat bude využívat databázi postgreSQL. Načítání dat bude prováděno prostřednictvím bash skriptů a textových souborů.

Pro systém je nezbytně nutné stanovit základní funkce, které by měl systém splňovat. Hlavním požadavkem je zjištění aktuálního počtu serverů a na nich vytvořených virtuálních systémů. Další neméně důležitou součástí je vytvoření přehledu s možností exportu získaných dat. Do systému budou integrovány další funkce, které blíže specifikují jednotlivé systémy, způsob či podrobnosti implementace a jejich vzájemné vazby.

Diplomovou práci lze rozdělit do tří částí. V první části jsou představeny Power systémy s využitím virtualizace hardwaru. Dále je zde popsán operační systém AIX s jeho nadstavbou pro vytvoření clusteru za účelem vysoké dostupnosti aplikací.

Druhá část se dělí na analýzu možností řešení a samotný návrh systému. Dosavadní shromažďování dat probíhalo pouze prostřednictvím excelovských tabulek, kdy při aktualizaci centrální databáze bylo nejprve nutné ověřit a případně

doplnit jejich stav, aby odpovídal skutečnosti. V této části jsou dále popsány základní funkce systému včetně načítání dat pomocí skriptů, správy uživatelů a možnosti exportu dat. Dále jsou zde sepsány prostředky, pomocí nichž byl systém navržen.

Závěrečná část je věnována možnostem dalšího rozšíření systému. Je zde kladen důraz i na testování funkcí systému, které probíhalo nejen na vývojovém, ale i produkčním a na úrovni IP odděleném prostředí, kde byly zjišťovány určité rozdíly ve výsledcích.

Systém využívá moderní metody programování a při návrhu webové aplikace byly využity API jednotlivých doplňků.

Power systémy

V průběhu let se zvyšovaly nároky klientů, kteří očekávali efektivnější propojení hardwaru a softwaru, proto v roce 2001 společnost IBM představila systém s procesorem POWER4 umožňující vytvářet logické oddíly (LPAR) a tak položit základy pro virtualizaci. Z počátku se tento způsob zdál velmi radikální, ale v dnešní době je ve většině operačních systémech samozřejmostí. Postupem času se vývoj power systémů nadále rozvíjel až do dnešní podoby, kdy systémy disponují sdílenými prostředky v podobě až 12jádrovými procesory s 3.52 Ghz a 1TB paměti.

Řešení IBM Power Systémů urychluje získávání poznatků z „velkých dat“ i implementaci hybridních cloudů. Pod pojmem „velká data“ si lze představit strukturovaná a nestrukturovaná data, která pocházejí z nejrůznějších zdrojů a jejich objem narůstá velkou rychlostí. Power Systémy tak umožňují řešit náročné výpočetní úkoly - rozsáhlé businnes analýzy a databáze, zpracovávání velkého objemu transakcí a konsolidaci IT při exponenciálním snížení nákladů.

Tabulka 1.1: Přehled IBM hardwaru

Power7	Power8
	
Blade Center H	Storwize V7000
	

Nové systémy Power8 mimo jiné využívají technologie GPU[1] akcelerátoru od společnosti NVidia. Ty jsou schopné paralelně vykonávat miliony výpočetních operací a jsou navrženy k výraznému urychlení výpočetně intenzivních aplikací. Budoucí verze serverů Power Systems bude disponovat technologií NVidia NVLink, která umožní eliminovat potřebu, přenášet data mezi CPU a GPU přes PCI Express[2] rozhraní.

[1] grafický procesor, který zajišťuje vykreslování dat uložených v operačním paměti na zobrazovacím zařízení

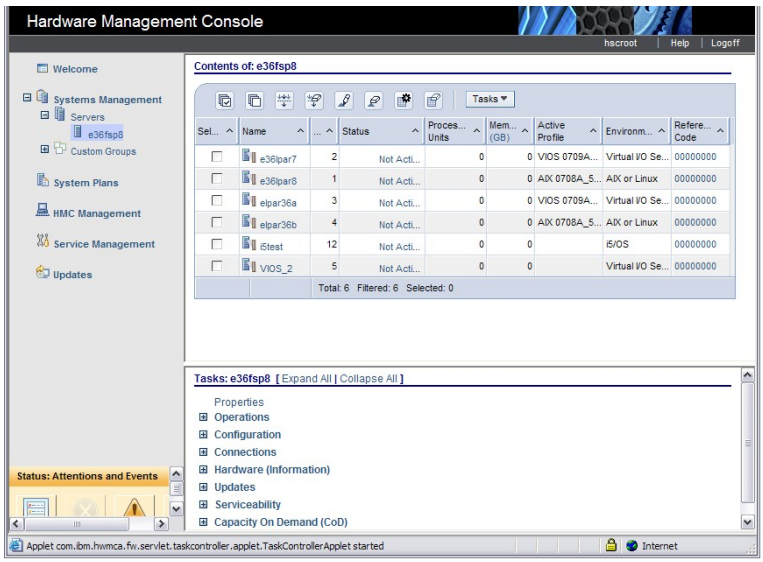
[2] Systémová sběrnice pro sériový přenos dat

1.1 HMC (Hardware management console)

HMC představuje „1 učkové“ nebo „stand-alone“ zařízení se sériovou nebo ethernetovou kartou, které se využívá k virtualizaci Power systémů. Sériová karta slouží především pro připojení starších serverů POWER4. Novější servery s procesory POWER5, POWER6, POWER7 a POWER8 vyžadují připojení po ethernetu.

Obrázek 1.1: HMC 7310-CR2

HMC má speciálně upravenou verzi operačního systému s uživatelem hscroot, kterému jsou přiřazena všechna potřebná oprávnění k řízení a používání, vycházejícího z Linuxu . HMC je užíváno zejména ke tvorbě a následné správě virtuálních strojů (LPARů) včetně možnosti dynamického přidělování a odebírání hardwaru, bez nutnosti restartu operačního systému běžícího na daném LPARu. Pro správu LPARů je dostačující zapojení jednoho zařízení, není vyžadována redundance.

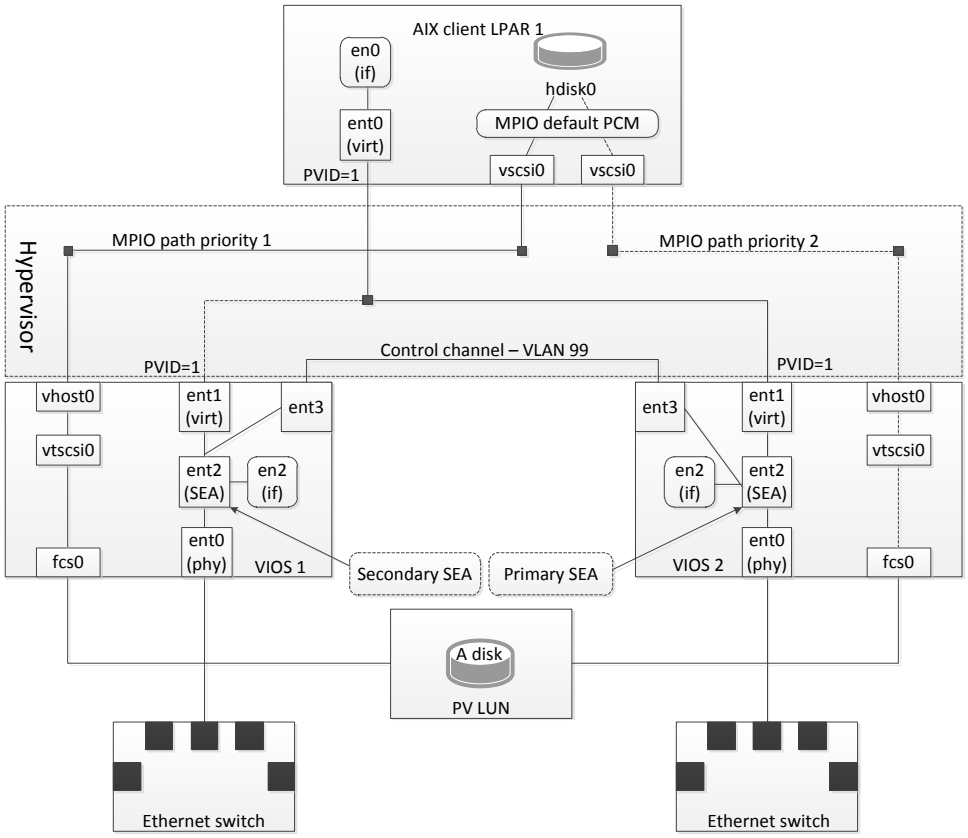


Obrázek 1.2: HMC GUI

Jedním HMC může být ovládáno až 1024 LPARů.[3] Jestliže je spravován pouze jeden Power systém, je možné využít přímé Ethernet propojení bez přítomnosti LAN switche. Přítomnost HMC je nutné pouze pro konfiguraci a spuštění Power systémů, nadefinované systémy budou fungovat nadále i po vypnutí HMC. Mimo správy systémů, HMC poskytuje přístup ke konzoli každého virtuálního stroje na každém spravovaném serveru.

Pokud je konfigurován moderní Power systém pomocí FSP (Flexible Service Processor)[4], informace o LPARech jsou uloženy jak v FSP, tak i v HMC. V případě výměny HMC, lze tak získat informace o stávající konfiguraci z FSP v každém spravovaném systému, ke kterému je HMC připojeno.

1.2 VIO server



Obrázek 1.3: Návrh propojení fyzických zdrojů s virtuálním systémem za pomocí dvou VIO serverů

[3] v případě HMC V7.7.3.0

[4] firmware, který poskytuje diagnostiku, inicializace, konfigurace, odhalování chyb run-time a korekce, spojuje spravovaný systém do HMC

Virtuálná I/O server je součástí hardwarových funkcí IBM Power Systémů. Umožňuje sdílení fyzických zdrojů mezi `LPARy` včetně virtuálních SCSI a síťových adaptérů, což má za následek efektivnější využití fyzických zdrojů a usnadnění konsolidace serverů. Díky tomu lze na jednom serveru provozovat více izolovaných operačních systémů ve stejnou dobu. Prostřednictvím VIO serveru tak může být každému `LPARu` přidělen méně než jeden procesor (automatické vyrovnávání zátěže), sdílené disky, síťové adaptéry a optická zařízení.

Na obrázku 1.3 je znázorněna architektura Power systému s využitím dvou VIO serverů s aktivními MPIO (Multi Path I/O) cestami. Ty jsou konfigurovány prostřednictvím sdílených ethernetových adaptérů SEA (Shared Ethernet adapter), u nichž je nastavena rozdílná priorita. Primární SEA je konfigurována na VIO serveru s neaktivními MPIO cestami.

V případě použití dvou VIO serverů je rozděleno zatížení náročných aplikací mezi dostupné zdroje (virtuální adaptéry, vSCSI adaptéry) a tím zaručena jejich dostupnost při výpadku jedné „větve“.

2 Virtualizace hardwaru

Virtualizací hardwaru je myšleno oddělení operačního systému specifickou vrstvou od samotného hardwaru, na kterém běží operační systém. Softwarová vrstva zajišťující toto oddělení je nazývána **hypervisor**.

Hypervisor na softwarové úrovni není jediný způsob jak virtualizovat hardwarové prostředky, tedy běh několika instancí operačního systému na jednom hardwaru. Existuje i tzv. hardware partitioning, kdy je na jednom serveru vytvořeno několik virtuálních strojů, kterým jsou přiřazeny hardwarové prostředky serveru. Příkladem tohoto řešení jsou například IBM LPAR (logical partitioning) nebo HPnPar, na kterých běží převážně unixové operační systémy.

2.1 Typy virtualizace

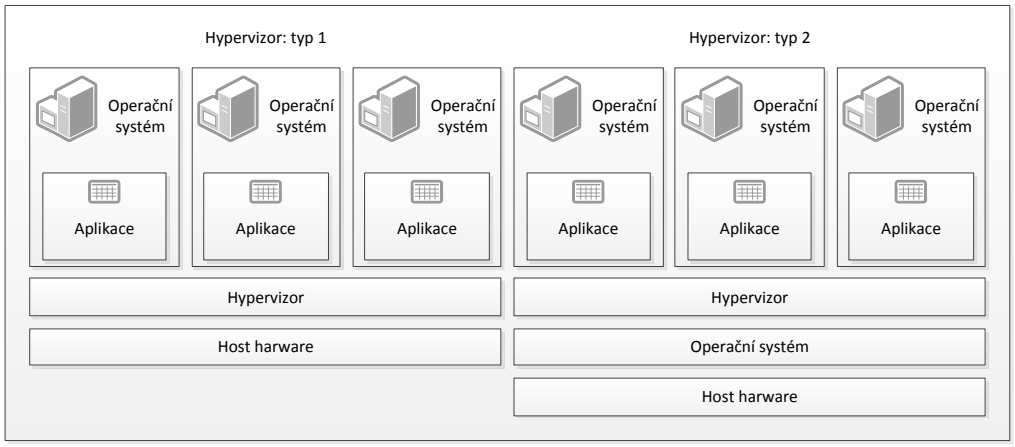
Hypervizor je systém, který má na starost přístup jednotlivých virtualizovaných počítačů/systémů (guest k hardwaru počítače (host), řídí jejich běh a zároveň je od sebe odděluje. Hypervizory byly rozděleny v roce 1973 dle Roberta P. Goldberga z Hardvardské univerzity na dva základní typy.

2.1.1 Hypervizor typu 1

Hypervizor prvního typu je instalován jako samostatný software, a tak nevyžaduje ke svému běhu žádný operační systém. Poskytuje všechny nutné prostředky pro běh virtuálních systémů.

Příklady hypervizorů:

- **VMWare ESX Server** - podnikový softwarový hypervizor, který běží přímo na hardwaru serveru, bez nutnosti dalšího základního operačního systému
- **Xen** - umožňuje několika ”hostům” operační systémy spustit na stejném hardwaru současně
- **KVM** - původně pro infrastruktury s linuxovým jádrem, KVM podporuje nativní virtualizace na procesorech s rozšířením virtualizace hardwaru. Podporuje velkou škálu operačních systému (distribuce Linuxu, BSC, Solaris, Windows, Hiku), upravená verze KVM umožňuje spustit Mac OS X



Obrázek 2.1: Porovnání hypervizorů

- **PowerVM** - základ pro IBM POWER5, POWER6, POWER7 a POWER8 servery, s podporou operačních systémů AIX a LINUX
- **z/VM** - běží na systémech IBM zSeries a podporují velký počet (tisíce) linuxových virtuálních strojů

2.1.2 Hypervizor typu 2

Hypervizor druhého typu je instalován na běžící operační systém, a tudíž funguje jako další „program“. Umožňuje tedy spouštět další oddělené instance operačního systému. Jeho využití je především v klientské oblasti pro testování, vývoj nebo zajištění zpětné kompatibility softwaru.

Příklady hypervizorů:

- VMware Workstation, VMware Server
- VirtualBox
- Virtual PC

2.1.3 Výhody/Nevýhody hardwarové virtualizace

Využití hardwarové virtualizace přináší v serverovém prostředí bezesporu mnoho výhod, především pro využití vysokého výkonu moderních serverů a uspokojení požadavků na instalaci oddělených instancí pro jednotlivé aplikace. Instalace na jednotlivé servery by byla nejen hardwarově, ale i ekonomicky nevýhodná.

Bezesporu největší výhodou je vyšší efektivita využití výkonu, které je dosaženo pomocí sdílených prostředků. Z výkonného hardwaru je každé instanci přidělena potřebná část pro její běh. Jestliže je třeba, lze snadno navýšit výkon

(jsou-li systémové prostředky k dispozici) systému (CPU,RAM) a tím tak dosáhnout maximálního využití hardwaru.

Mezi další výhody patří :

- **vyšší bezpečnost** plynoucí z jednoduché separace aplikací na úrovni operačního systému
- **kompatibilita s novým hardwarem** - jestliže není zaručena kompatibilita s novým hardwarem, ale je zapotřebí stále využívat starý, je virtualizace optimálním řešením
- **rychlá obnova z chyb HW** - využití on/off line migrace virtuálního systému mezi jednotlivými hypervizory
- **úspora nákladů** na infrastrukturu, prostor, elektřinu a licence

Přes tyto výhody přináší virtualizace i několik nevýhod, mezi které patří větší nároky na spolehlivost a redundanci jednotlivých částí hardwarové infrastruktury. Jestliže by se vyskytly hardwarové problémy, jejich následky by byly zřejmé na více virtuálních systémech.

3 AIX

Představuje propracovaný UNIXový operační systém vyvíjený společností IBM. Byl navržen pro RISCové[1] pracovní stanice řady IBM 6150, na které bylo možné instalovat více OS platforem. Power systémy v současnosti umožňují nejen běh platformy OS AIX, ale i Linux, IBM System a zOS.

AIX disponuje oproti ostatním UNIXovým systémům unikátními vlastnostmi, které jsou většinou skryté a navenek se projevují spolehlivostí a robustností. Jedna z mnoha „featur“ je Object Database Model (ODM) databáze, která představuje souhrn všech konfiguračních souborů systému. Tato databáze se spravuje pomocí konzolového GUI rozhraním SMIT.

Příklad smitty mksysb :

```
                Back Up This System to Tape/File or UDFS capable media

Type or select values in entry fields.
Press Enter AFTER making all desired changes.
                                                                [Entry Fields]
 WARNING:  Execution of the mksysb command will
           result in the loss of all material
           previously stored on the selected
           output medium. This command backs
           up only rootvg volume group.
 Backup DEVICE or FILE                                                  []
 Create MAP files?                                                      no
 Create backup using snapshots?                                         no
 EXCLUDE files?                                                         no
 Exclude WPAR file systems?                                             no
 Location of File System Exclusion List                                 []
 List files as they are backed up?                                      no
 Verify readability if tape device?                                     no
 Generate new /image.data file?                                         yes
 EXPAND /tmp if needed?                                                 no
 Disable software packing of backup?                                    no
 Backup extended attributes?                                            yes
 Number of BLOCKS to write in a single output                           []
    (Leave blank to use a system default)
 Location of existing mksysb image                                      []
 File system to use for temporary work space                            []
   (If blank, /tmp will be used.)
 Backup encrypted files?                                                yes
 Back up DMAPI filesystem files?                                        yes

 F1=Help    F2=Refresh    F3=Cancel    F4=List    F5=Reset    F6=Command
 F7=Edit    F8=Image      F9=Shell     F10=Exit   Enter=Do
```

[1] označuje procesory s redukovanou instrukční sadou

Další výjimečnou vlastností, kterou se liší operační systém oproti konkurenci, je zabudovaný Logical volume Manager (LVM) uvnitř jádra samotného systému. Zatímco u ostatní systémů je k dispozici jako placený balíček instalovaný v uživatelském prostoru.

Největší předností, kterou má AIX od svého vzniku, je s největší pravděpodobností zálohovací systém tzv. System Backup Manager. Ten umožňuje vytvořit zálohu celého operačního systému na pásku jedním příkazem (včetně konfigurací a dat). Z této pásky lze pak „nabootovat“ a obnovit celý systém do původního stavu bez provedení dalších konfigurací (zálohu lze obnovovat i na jiný typ serveru). Mimo zálohovacího systému lze využít i tzv. klonování, které je užitečné při migrování na nový hardware.

V případě, že výkonnostní nároky převyšují stávající architekturu sítě, musí být proveden „upgrade“ na výkonnější hardware, lze to provést bez nutnosti instalace nového operačního systému, opětovné konfigurace, instalace aplikací a přenášení uživatelských účtů a dat. Postačí „nabootovat“ systém z instalačního cd AIX a zvolit obnovu z pásky nebo jiného zálohovacího média. Systém si načte konfiguraci LVM, nadefinuje partition tabulku disků a poté „přelije“ zálohovací pásky na disk. V závěrečném kroku se automaticky nainstaluje podpora pro nový hardware, provede se rekonfigurace a vytvoří se odpovídající boot image. Při následném „rebootu“ je systém plně funkční (stejné aplikace, data i uživatelské účty jako před upgradem). Na závěr celého `backup/recovery` procesu je provedena úprava konfigurace s ohledem na výkon nového systému. Místo zálohování na pásky lze také využít síťový server - **NIM** (Network Installation Management), případně CD či DVD.

3.1 Network Installation Management

NIM představuje nástroj pro instalaci, aktualizaci a správu AIX systémů z jednoho specifického místa, kdy na jednom systému běží NIM server a na ostatních systémech NIM klienti. Existují tři základní typy zdrojů NIM serveru, které lze využít k instalaci systému na klienty: `mksysb, SPOT a LLP-sources`.

3.1.1 Mksysb

Je instalovaný a bootovatelný image pro rootvg [2], který obsahuje všechny důležité souborové systémy (`/, /usr, /etc, ..`). Lze jej snadno použít k instalaci nového systému. Mksysb je využíván především dvěma způsoby. Jednak k vytvoření a udržování základního image, ze kterého budou prováděny instalace nových systémů, které budou obsahovat pouze absolutní minimum. Takto nainstalované systémy, je pak nutno doplnit instalací dalších balíčků a aplikací. Druhou možností je využít mksysb k vytvoření jednoho kompletního snímku z běžícího systému (včetně uživatelů, aplikací, balíčků) a tento mksysb rozdistribuovat na další systémy. Tuto kopii je možné použít pro rychlou obnovu systému.

[2] výchozí volume groupa v operačním systému AIX

3.1.2 LPP source

Dalším zdrojem je lppsource, což je sbírka balíků installp, které budou NIM serveru poskytnuty pro instalaci. NIM zjistí jaké balíky daný lppsource obsahuje a nabídne je k instalaci na příslušný systém.

3.1.3 SPOT (Shared Product Object Tree)

SPOT je v podstatě `/usr` souborový systém, představuje vše co systém vyžaduje, jako je jádro systému AIX, spustitelné příkazy (příkazy pro instalaci systému), knihovny a aplikace. Během instalace si klientský systém připojí přes NFS [3] potřebné zdroje pro instalaci.

SPOT je zodpovědný za vytvoření zaváděcího obrazu, který se odešle do klientského počítače v síti. Lze jej vytvořit z mksysb i lpp source. SPOT vytvořený z mksysb je však možné použít pouze se zdrojem, ze kterého byl vytvořen.

3.2 Filesystém JFS

JFS je moderní 64bitový žurnálovací souborový systém vyvinutý firmou IBM pro použití na serverech. Žurnálovací souborový systém zapisuje změny, které mají být v systému provedeny do speciálního záznamu tzv. žurnál, obvykle realizovaného jako cyklický buffer. Jeho účelem je ochrana dat na pevném disku v případě neočekávaných situacích. Hlavní předností jfs je použití extendů, díky kterým dochází ke zrychlení práce filesystému produkujícího efektivní a malé struktury pro mapování souborů.

Dalšími výhodami jsou :

- různé velikosti bloků (512, 1024, 2048, 4098 bytů) - pro optimalizaci výkonných systémů
- dynamické alokování i-nod[4], čímž se zamezí rezervování fixního místa na disku pro inody v průběhu vytvoření filesystému
- dva způsoby organizace adresářů - první způsob především pro malé adresáře (do 8 položek), obsah adresáře je uložen v i-nodě příslušného adresáře; druhý způsob je B+ strom setříděný dle jména, toto řešení poskytuje rychlejší možnost přístupu v porovnání s předchozím způsobem
- podpora řídkých souborů
- podpora velkých souborů a souborových systémů

[3] Network File System - internetový protokol pro vzdálený přístup k souborům přes počítačovou síť

[4] datová struktura uchovávající metadata o souborech a adresářích používaná v UNIXových souborových systémech

3.3 Cluster

`Cluster` představuje seskupení volně vázaných serverů, které spolu úzce spolupracují po rychlé datové síti. Mohou se tvářit jako jeden počítač (HPC). Jejich úloha spočívá ve velké výpočetní rychlosti pro náročné početní úlohy (faktorizace na prvočísla, simulace proudění vzduchu, analýza statistických dat) nebo zajištění vysoké dostupnosti určité služby (databáze, SMS centrum) s větší efektivitou než by mohl poskytnout jediný server.

3.3.1 Výpočetní cluster

Výpočetní cluster (High-performance computing - HPC) slouží k náročným početním úkolů, kdy se úloha rozdělí mezi několik výpočetních jednotek propojených vysokorychlostní sítí. Minimální konfigurací pro takový cluster je využití zapojení jednoho hlavního serveru (master node), který přiděluje výpočty jednotlivým výpočetním uzlům (compute node) přes síťový switch.

3.3.2 Cluster s vysokou dostupností

Cluster s vysokou dostupností (High-availability cluster - HACMP) poskytuje několika serverům nepřetržité poskytování dané služby i při výpadku počítače z důvodu hardwarové závady nebo plánované údržby. Tuto službu poskytne v případě výpadku automaticky druhý počítač zadefinovaný v clusteru.

3.3.3 Cluster s rozložením zátěže

Cluster s rozložením zátěže (Load ballancing cluster) umožňuje snížit míru zátěže. Výpočet je poskytnut několika serverům, které mají stejný obsah. Ten je zajištěn jeho replikací mezi ostatní propojené servery nebo lokální úložiště.

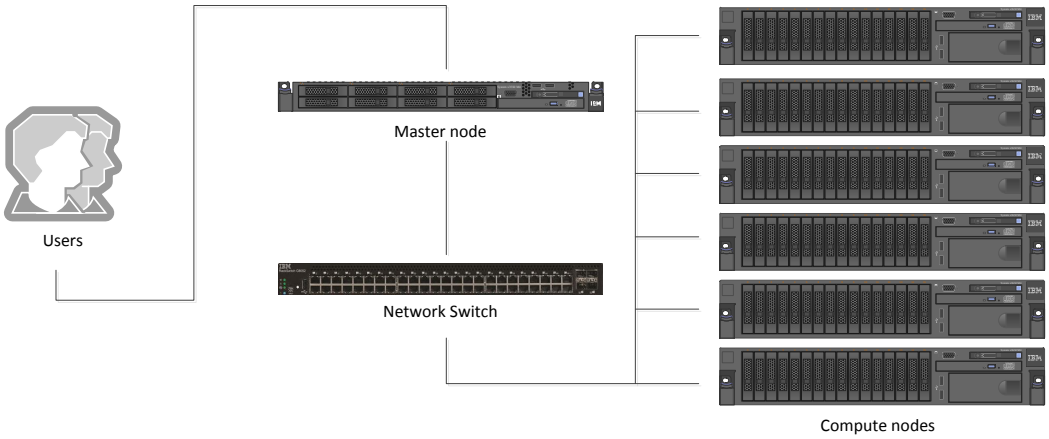
3.3.4 Úložný cluster

Úložný cluster (Storage cluster) zprostředkovává přístup k diskovému poli, které je poskytnuto pro dosažení vyššího toku dat mezi více serverů nebo pro zajištění vyšší spolehlivosti. K tomu jsou zapotřebí doprovodné služby v podobě speciálních souborových systémů umožňujících rozložení zátěže, ochrany redundance dat, pokrytí výpadků jednotlivých uzlů, mechanismu pro zamykání souborů a dalších podpůrných služeb.

3.3.5 Gridový cluster

Gridové clustery jsou primárně určeny pro jinou činnost než poskytnutí výpočetního výkonu ve prospěch clusteru. Mohou se rozprostírat po celém světě, jeho uzly mohou být výkonné servery nebo i osobní počítače. Frontend celého gridu musí obsahovat

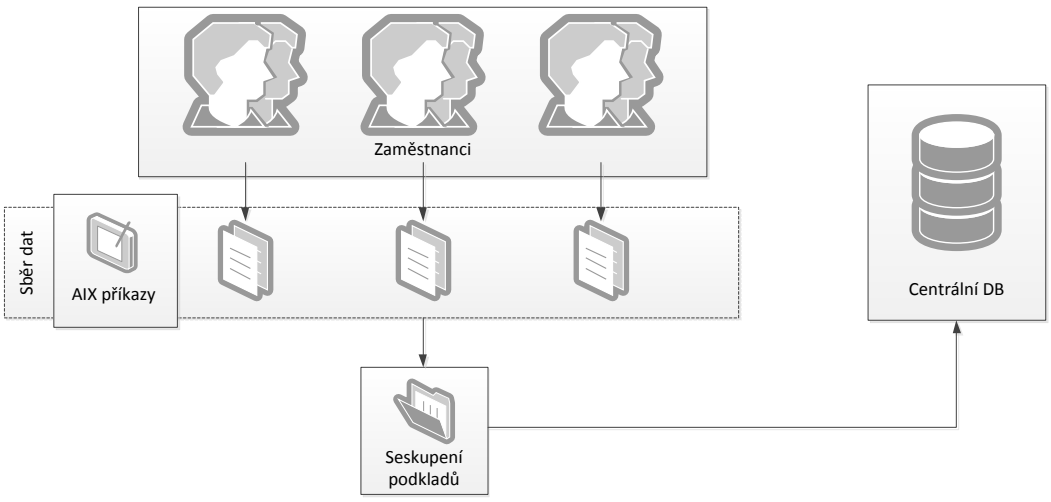
komplexní nástroje pro spuštění a řízení úloh, které stanoví, na kolika počítačích z gridu může být úloha spuštěna.



Obrázek 3.1: Výchozí architektura clusteru

Návrh systému CMDB pro SÍŤ, spol. s r. o

Před návrhem samotného systému bylo nutné provést analýzu současného stavu shromažďování dat, které není ideální. Nyní jsou data shromažďována v excelovských tabulkách, které jsou uloženy u různých zaměstnanců. Případně je možné využít aix příkazy ke zjištění doplňujících informací z příslušných serverů. Tento způsob je při sestavování podkladů pro centrální databázi velmi zdlouhavý a v době předání již nemusí odpovídat aktuálnímu stavu.



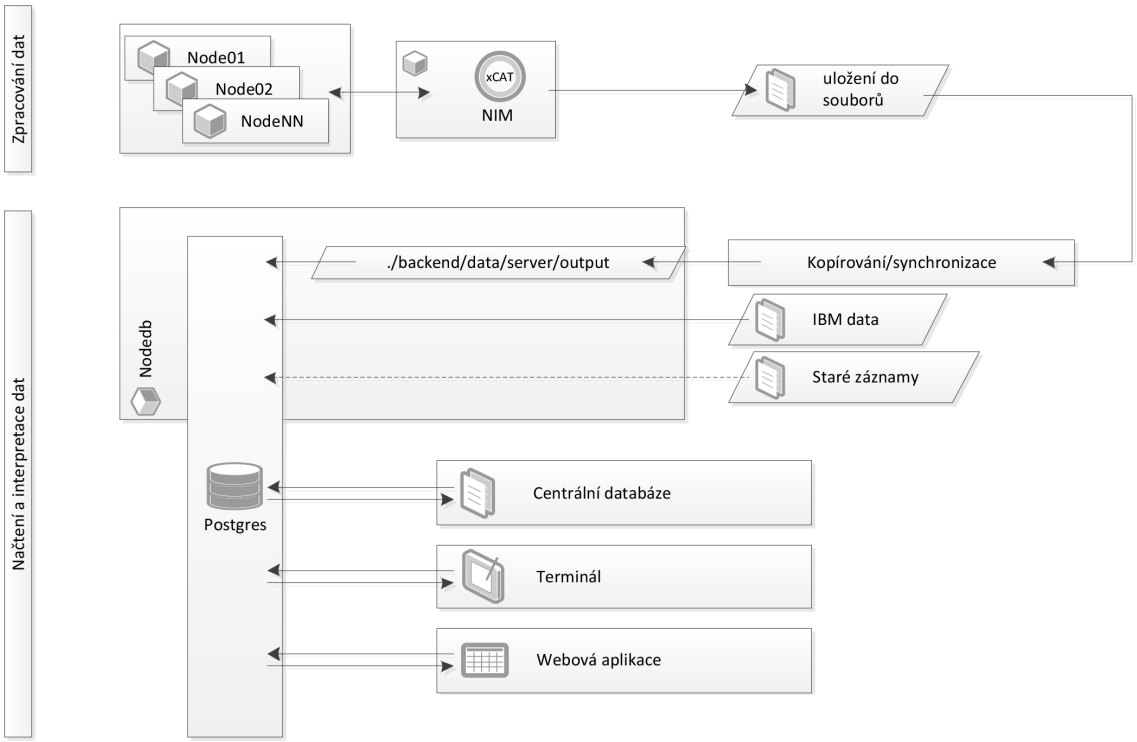
Obrázek 3.2: Současný stav

Největší nedostatek současného stavu je absence záznamů fyzického umístění jednotlivých serverů, což při hardwarových poruchách vede ke složitému dohledávání. S fyzickým umístěním souvisí i další evidence. Zejména pak evidence umístění běžících LPARů na konkrétním serveru. Dalším nedostatkem je absence upozornění na vypršení časového období hardwarové podpory.

Při návrhu bylo uvažováno využití opensource nástroje `Ganglia`, který umožňuje u monitorovaných systémů sledovat aktuální či historické statistiky systémových metrik. Tento nástroj však neřeší problém s evidencí systémů, proto bylo nutné navrhnout vhodný způsob pro shromažďování a interpretaci informacích o jednotlivých systémech. Získané informace musejí být koncipovány tak, aby je bylo možné předat co nejsnazším způsobem do centrální databáze.

Na obrázku 3.3 je znázorněn návrh řešení systému, který je rozdělen do několika kroků. V prvním kroku jsou vytvářeny soubory obsahující data získaná z jednotlivých nodů. Tento proces využívá jednoho nodu, kde je instalován xCAT (více v kapitole 4.0.14), který umožní získat potřebné informace. Další fází je synchronizace nebo manuální přenos souborů s nodem, kde je vytvořena databáze, kam se s využitím připravených skriptů načtou získaná data. Do databáze jsou dále importována data o jednotlivých hardwarových zařízeních a staré záznamy.

Výsledné záznamy je tak možné zobrazovat prostřednictvím terminálu nebo webové aplikace.



Obrázek 3.3: Návrh řešení

Během zpracovávání návrhu byla zvažována možnost využití pouze jednoho „admin“ nodu, který by obsahoval xCAT, databázi a webovou aplikaci. Do databáze by pak byly všechny informace načítány automaticky prostřednictvím webové aplikace. Toto řešení je však díky omezeným prostupům nerealizovatelné, protože by nezahrnovalo celou infrastrukturu a tudíž by výsledná databáze neobsahovala kompletní seznam serverů a jejich nodů.

Funkční požadavky :

- Systém bude umožňovat registraci uživatelů
- Systém bude mít dvě základní role uživatelů (administrator, user)
- Každý uživatel systému bude muset být přihlášen

- Administrátor bude mít možnost upravovat veškeré záznamy
- Uživatel bude moci přidávat poznámky k jednotlivým serverům a nodům
- Administrátor a uživatel si bude moci vyexportovat příslušné informace
- Administrátor bude mít přehled o jednotlivých změnách

Nefunkční požadavky :

- Hlavní část systému bude realizována za pomocí PHP, HTML a CSS
- Skripty pomocí nichž se budou získávat data, budou napsány v Bash Shellu
- Všechny záznamy budou ukládány do databáze PostgreSQL

4 Použité technologie

4.0.6 Yii Framework

Yii (Yes, it is!) je open-source framework vydaný pod BSD licencí, který je poměrně mladý, ale velmi výkonný. Na jeho vývoji se podílí mezinárodní tým v čele s Qiang Xue, který se podílel na vývoji a udržování frameworku Prado. Aby yii předčilo výkonem ostatní frameworky a i přesto nabízelo bohatou sadu funkcí, inspirovalo se různými projekty. Především z frameworku Prado, ze kterého přejal komponentní a událostmi řízené paradigma, vrstvy databázové abstrakce a mnoho dalších již zaběhlých funkcí. I z dalších projektů byly implementovány do frameworku další přednosti jako jsou jQuery (integrace základní knihovny pro podporu JavaScriptu), Joomla (modulární architektura), Symfony (zásuvné moduly, návrh filtrů) a Ruby on Rails (návrhový vzor ActiveRecord).

Yii umožňuje maximální znovupoužitelnost kódu, což zkracuje vývojový proces. Pro běh webové aplikace je zapotřebí webový server s podporou PHP 5.1 +. Pro svůj vývoj využívá komponent a objektově orientované programování. Je navržen s ohledem na třívrstvou MVC architekturu [**?**].

4.0.7 HTML a xHTML

`HMTL`(HyperText Markup Language) je aplikací jazyka SGML(Standard Generalized Markup Language) a představuje hlavní značkovací jazyk, kterým se vytváří struktura webových stránek. Značkovacím jazykem je označován jazyk pro vytváření dokumentů obsahující nejen text, ale i instrukce pro jeho zpracování, které se provádí prostřednictvím webového prohlížeče.

4.0.8 PHP

`PHP`(Hypertext Preprocessor) je programovací jazyk, jehož syntaxe je odvozena od několika programovacích jazyků (Perl, C, Pascal a Java) a je nezávislá na platformě. Kód se zahrnuje do struktury `HTML` či `xHTML` pomocí syntaxe `<?php ?>`. `PHP` scripty fungují bez nutnosti úprav na různých operačních systémech.

Nejnovější využívanou verzí je 5.6, která podporuje oproti starším verzím upload souborů větších než 2 GB, disponuje integrací debuggeru `phpdbg` a možností využití skalárních výrazů nebo variadických funkcí s proměnným počtem parametrů. Starší verze 5 nabízejí využití ZEND Enginu II. generace obsahujícího vylepšenou podporu

pro objektově orientované programování (OOP). Jsou zde implementovány doplňky pro explicitní konstruktory a destruktory, pro klonování objektů a abstraktních tříd, které zlepšují využití `OOP v PHP`. Tyto verze také nabízejí dokonalejší podporu `XML`, založenou na knihovně *libxml2*, a zavedení *Simple XML* pro usnadnění čtení a manipulaci s `XML`.

Scriptovací jazyk `PHP` je vyvíjen pod Open Source licencí a je podporován na většině hostingů. Primárně je určen pro vývoj webových stránek.

4.0.9 CSS

`CSS`(Cascading Style Sheet) kaskádové styly jsou nadstavbou značkovacího jazyka HTML, xHTML nebo XML a slouží ke grafické úpravě webových stránek. I když k samotnému zobrazení stránek nejsou styly potřebné, lze s nimi vytvořit jedinečný vzhled stránky a aplikovat tak návrh grafika.

Výhody použití kaskádových stylů :

- možnost nastavení jedné webové stránky pro více výstupních zařízení (monitor, tiskárna, mobilní telefon)
- oddělení stylů od obsahu (větší přehlednost)
- soubor `css` se načítá do paměti CACHE, dokument má tak menší velikost a zobrazovaná stránka se načítá rychleji
- využívá dědičnost při úpravách vnořených stylů

4.0.10 JavaScript a jQuery

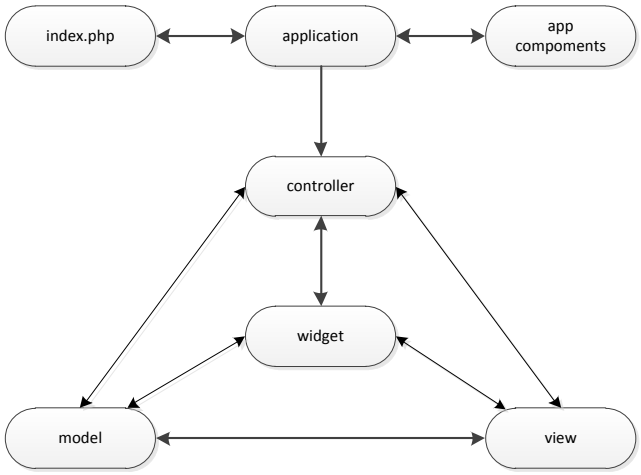
`JavaScript` je interpretovaný programovací jazyk disponující základními OOP schopnostmi. I přesto, že v `JavaScriptu` nemusí být specifikovány proměnné svými programovatelnými konstrukcemi, tak příkazy if a smyčka while připomínají jádro programovacích jazyků C, C++ a Javu.

Umožňuje nadstavbu v podobě frameworku jQuery, který usnadňuje práci s elementy DOM, k jehož výběru se využívá `CSS a XPath`. `jQuery` je přesná a rychlá knihovna JavaScirptu, která jednoduše prochází HTML elementy a zpracovává události či animace. Hlavní výhodou je malá velikost, podpora webových prohlížečů a `CSS` stylů.

4.0.11 Návrhový vzor MVC (Model View Controller)

Návrhový vzor rozděluje aplikace na tři základní části: `model, view a controller`. Datová vrstva `model` obsahuje logiku aplikace, strukturu dat a potřebné přístupy k nim. Do uživatelského rozhraní `view` jsou zahrnuty výstupy v grafické podobě, které řídí poslední vrstva `controller`, vytvářející aplikační logiku celé aplikace. Jednotlivé části mezi sebou komunikují a plní svoji úlohu, lze je tedy modifikovat samostatně.

V Yii frameworku se k těmto vrstvám zavádí `front-controller`,který shromažďuje informace o požadavku uživatele a odešle je do příslušného `controlleru` k dalšímu zpracování.



Obrázek 4.1: Statická struktura Yii aplikace

Pracovní postup Yii frameworku při požadavku uživatele

1. Uživatel odešle požadavek s URL `www.example.com/index.php?r=post/show&id=1`, webový server zpracuje žádost zaváděcím skriptem index.php.
2. Bootstrap skript vytvoří instanci aplikace a spustí ji.
3. Aplikace získá z komponenty `request` detail požadavku uživatele.
4. Aplikace řídí požadavek `controlleru` pomocí komponenty `urlManager`.
5. Aplikace vytvoří instanci požadavku daného `controlleru` k dalšímu zpracování. `Controller` rozhodne jaké další činnosti se mají provést a kam se mají zobrazit.
6. Provedená akce čte pomocí metody POST z `modelu` id 1 z databáze.
7. Příslušné záznamy s id=1 se předají opět metodou POST do `view`, kde se zobrazí.
8. Vykreslení záznamů se může provádět pomocí `widgetů` - dataView, dataTable
9. Výsledek je dále zobrazen v předem definovaném `layoutu`
10. ... a uživatel může zadat další požadavek ke zpracování.

Model

`Model` zastupuje množinu tříd a funkcí, které primárně vykonávají operace s daty (načítání, zpracování, ukládání). Mimo manipulace s daty a zajištění jejich přístupů (např. komunikace s databází), je zde zastoupena veškerá logika systému, která zodpovídá za chod celého aplikace. Celek jako takový neví nic o `view` a `controlleru`. Pokud je navržen dle všech pravidel, měl by být bez problémů využitelný v jakékoli aplikaci.

View

`View` představuje data získaná z `modelu`. Jeho úkol je vykreslovat výstupy aplikace např. v grafické či textové podobě. Přístup k těmto datům zajišťuje `controller` či jiné rozhraní nabízející modelem. Výstup dat může být znázorněn v různých podobách, jak ve značkovacích jazycích HTML, XHTML či XML, tak i v podobě obyčejného textu.

Controller

Funkci řídící jednotky v tomto návrhovém vzoru představuje `controller`, který reaguje na uživatelské požadavky a zabezpečuje změny ve `view` nebo `modelu`. Na základě žádostí od dalších dvou vrstev rozhoduje, které funkce modelu se vykonají a jaká data se zobrazí. U webových aplikací je `controller` v podobě skriptů, ke kterým aplikace přistupuje metodami POST a GET.

Výhody x Nevýhody

Výhody MVC proti běžnému návrhu:

- **Opětovné využití kódu, který je součástí modelu** - oddělené komponenty `model` a `view` umožňují využívat logiku modelu v různých zobrazení. Jelikož `model` je nezávislý, mohou se snáze modifikovat, testovat a udržovat jeho části.
- **Snadná rozšířitelnost** - do aplikace se mohou neomezeně přidávat další `controllery` a `view` bez ovlivnění dalších celků. Jestliže je zachováno společné rozhraní, je možné využívat funkcionalitu komponent.
- **Přehlednost návrhu** - zajišťuje lepší orientaci v návrhu a tak minimalizuje duplicity.
- **Nezávislost při vývoji komponent**

Nevýhody :

- při větších projektech nárůst zdrojových kódů
- jestliže platforma neumožňuje vestavěné řešení, musí jej navrhnout sám programátor
- vývojář musí znát principy MVC pro správnou implementaci systému

4.0.12 Bash

`BASH`(Bourne Again Shell) představuje standardní sadu příkazů v Linuxu založenou na Bourne shellu, která reprezentuje rozhraní mezi uživatelem a systémem. Je součástí GNU projektu, proto se mohl šířit i na ostatní unixové systémy.

Obsahuje tyto základní funkce. První z nich je interaktivní režim, který čeká na zadání příkazů od uživatele, jež mohou být zahrnuty přímo v shellu nebo jako samostatné programy napsané téměř v libovolném programovacím jazyce. Dále systémové proměnné umožňující pohodlnou práci v pracovním prostředí. Některé mohou být přednastaveny systémem, ostatní nastavuje uživatel v inicializačních souborech při spuštění shellu.

4.0.13 AWK

`AWK` je univerzální počítačový jazyk určený pro zpracování textových dat. Data mohou být zpracovávána jak v podobě textových souborů, tak proudů. Jazyk značně využívá datové typy, asociativní pole a regulární výrazy. Jeho síla, stručnost a omezení programů v `AWK` a skriptů v sedu [1] byly inspirací k vytvoření jazyka Perl.

`AWK` je standardní součástí většiny operačních systémů unixového typu. Obecně jsou mu předávány dva druhy dat, příkazový soubor a primární vstupní soubor. V příkazovém souboru je obsažena série příkazů, která `AWK` říká, jak má být vstupní soubor zpracován. Oproti tomu primární vstupní soubor je formátovaný text, kterým může být jednak existující soubor nebo jej `AWK` čte ze standardního vstupu. Program `AWK` se sestavuje z posloupnosti řádků ve tvaru:

```
/vzor/ { akce }
```

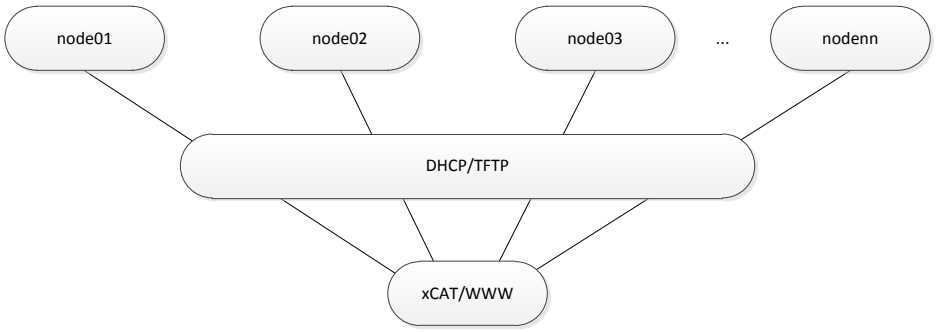
Kde vzor představuje regulární výraz a akce zastupuje příkaz nebo sekvenci příkazů, `AWK` prochází vstupní soubor a jestliže najde řádek s vyhovujícím vzorem, provede příkaz uvedené akce.

4.0.14 xCAT (Extreme Cloud Administration Toolkit)

xCAT je open source řešení, které umožňuje řídit a rozšiřovat thing provisioning - nástroj pro ovládání a vyhledávání vlastností hardwaru. Využívá se pro správu operačních systémů na fyzických nebo virtuálních strojích: RHEL, CentOS, Fedora, SLES, Ubuntu, AIX, Windows, VMWare, KVM a PowerVM. Další z jeho předností

[1] stream editor sloužící k proudovému zpracování vstupních dat

jsou pokročilé skriptování (statefull, statelite a stateless OS Image), využití vzdálené správy systémů (LED management, remote console, paralelní shell) a případně rychlá konfigurace služeb pro správu prostředí (DNS, HTTP, DHCP, TFTP, NFS).



Obrázek 4.2: xCAT architektura

Informace o jednotlivých nodech jsou uloženy v PostgreSQL databázi, kde jsou definovány skupiny objektů, kde každý objekt náleží do jedné nebo více skupin pro následnou správu.

Nejvyužívanější příkazy :

- **nodels** - výpis všech definovaných nódů
- **lsdef -t group** - výpis všech definovaných skupin
- **psh** - spuštění paralelních příkazů - `psh <noderange> <příkaz>`
- **pscp** - kopírování souborů na více nodů současně - `pscp <soubor> <noderange>:<cílový soubor nebo adresář>`
- **xcoll** - zpřehlednění výstkup ostatních příkazů - `<libovolný xCAT příkaz> | xcoll`

4.0.15 Postgres databáze

Postgres je objektově-relační databázový systém (ORDBMS) s otevřeným kódem, který je primárně vyvíjen pro unixové systémy, přesto existují i balíčky pro 32bit a 64bit windows platformu. Splňuje seznam požadavků `ACID` na bezpečný transakční systém.

- **Atomic (Atomičnost)** - v rámci transakce se provedou všechny změny nebo žádná.
- **Consistent (Konzistence)** - transakce zajišťují převedení dat z jednoho konzistentního stavu do druhého. Tato podmínka nemusí platit uvnitř transakce.

- **Isolated (Izolace)** - transakce není ovlivněna souběžnými transakcemi.
- **Durable (Trvanlivost)** - pokud je transakce potvrzená, pak jsou změny dat trvalé a to i pokud nastane havárie systému.

Databáze plně podporuje cizí klíče, operace JOIN. Obsahuje většinu nejvyužívanějších datových typů a podporuje i moderní datové typy jako jsou JSON a XML. PostgresSQL lze instalovat z dostupných linuxových depozitářů případně pomocí rpm balíčků. Disponuje vlastnostmi, jako jsou např.:

Funkce

Funkce mají za úkol spouštět jednotlivé bloky kódu na serveru, mimo standardního jazyka pro databáze SQL a mohou být implementovány v několika jazycích.

- zabudovaný jazyk PL/pgSQL
- skriptovací jazyky Perl, Python, Ruby, sh, Scheme, Tcl, LUa
- kompilovací jazyky C, C++, Java, Common Lisp
- statické jazyky R

Jejich návratová hodnota je v podobě řádků, kde výstup funkce obsahuje množinu hodnot, se kterou lze v dalších dotazech pracovat jako s tabulkou. Definice jednotlivých funkcí lze spouštět buď s právy volajícího nebo toho, kdo funkci definoval.

Indexy

PostgreSQL má uživatelské typy indexů, ale také zabudovanou podporu pro B+tree, has, GiST a GiN index. Dále podporuje následující vlastnosti.

- **Indexy nad výrazy** -indexy vytvořené nad výsledkem výrazů a funkcí
- **Částečné indexy** -indexy vytvořené pouze nad částí tabulky (vytváří se přidáním WHERE klauzule na konec příkazu CREATE INDEX)

Triggery

Trigger představuje událost, která je spuštěna akcemi e SQL DML příkazů (INSERT, UPDATE). Příkazy mohou spustit triggery kontrolující hodnoty po zadávání příkazů. Převažuje definice triggerů na tabulkách a na pohledech (View), kde je možné definovat určitá pravidla (Rules). Jestliže je na jedné tabulce definováno několik trigerrů, vykonávají se dle abecedy. Funkce psané přímo v jazyce PL/PgSQL mohou volat funkce psané dalšími jazyky.

Multi-Version Concurrency Control (MVCC)

Představuje systém řešení současného přístupu. Každému uživateli je zpřístupněna pouze „snapshots“ databáze. To umožňuje provádět úpravy, aniž by je ostatní uživatelé viděli. Tento princip eliminuje potřebu zamykání a dodržuje tak ACID principy.

5 Vývoj systému

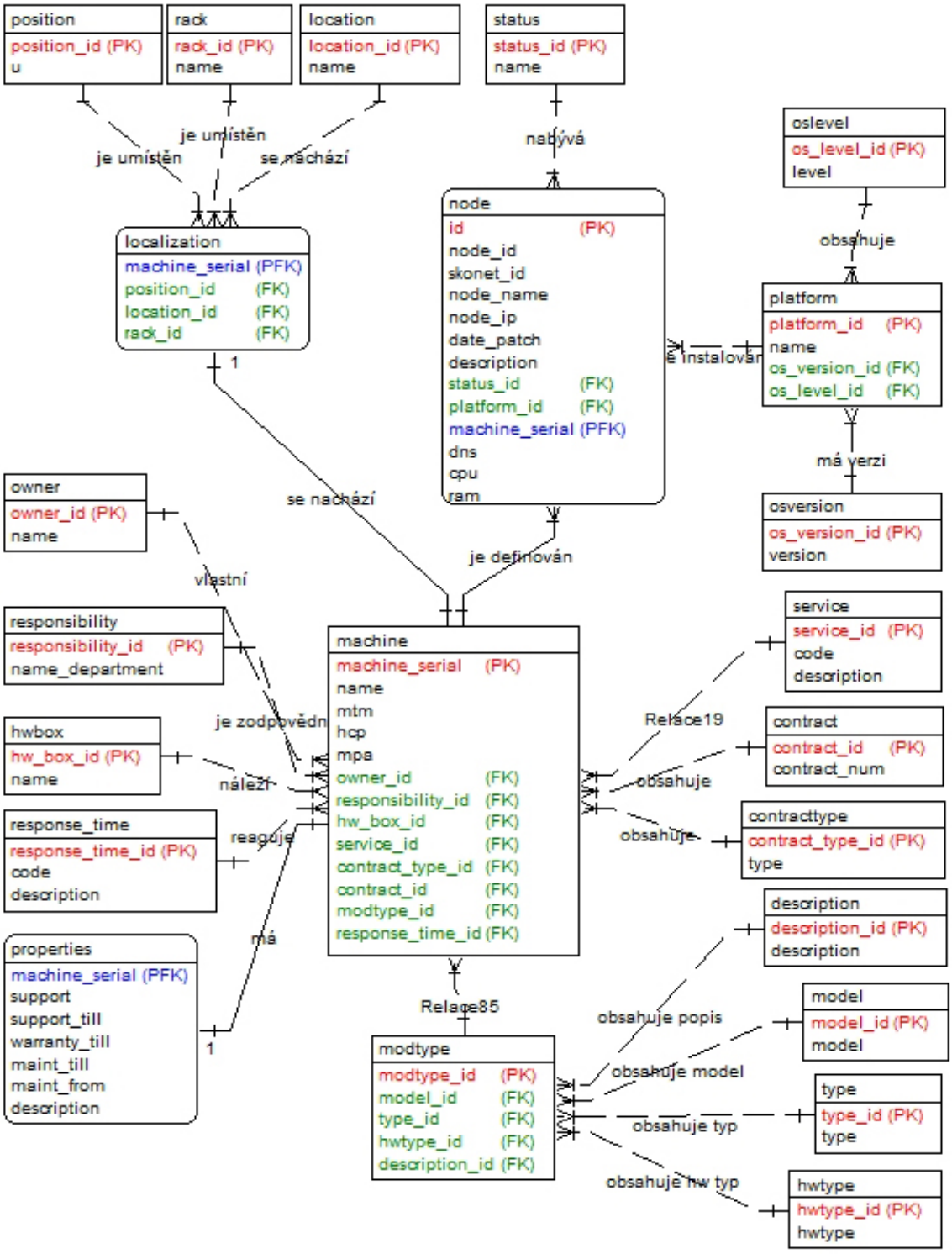
5.1 Návrh databáze

Veškerá zpracovaná data systému jsou uchována v databázi, jejíž struktura je postavena na objektově-relačním databázovém systému PostgresSQL. Tímto je zajištěn souběžný přístup více uživatelů k datům ve stejnou dobu a rychlejší manipulace s těmito daty.

Databáze je rozdělena do několika logických oblastí. Jako hlavní oblast lze považovat tabulky `machine`,`node` a jejich související tabulky propojené cizími klíči.

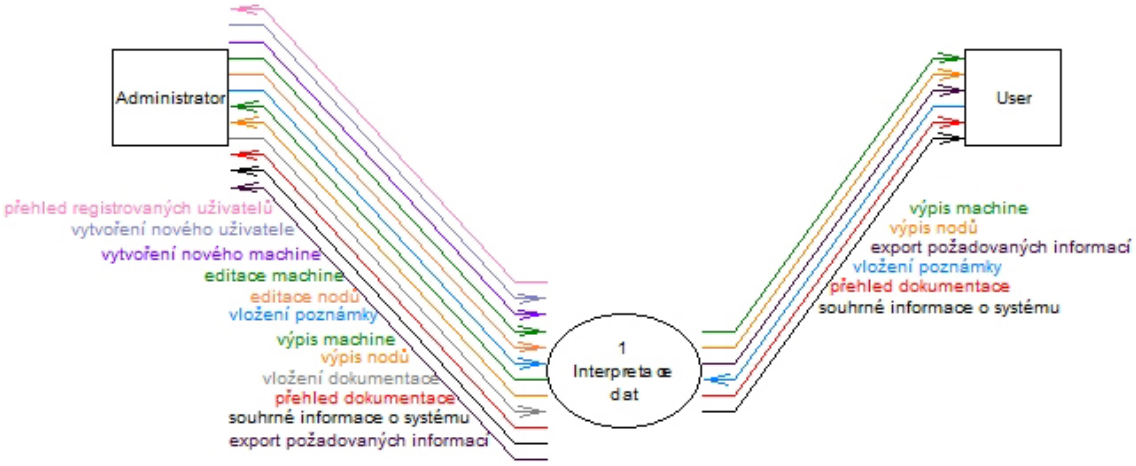
Popis databázových objektů hlavní oblasti :

- **Tabulka machine** uchovává údaje o všech Power serverech, které jsou součástí síťové architektury. Pomocí primárního klíče `machine_serial`, lze získat jeho fyzické umístění a nody, jež jsou na daném systému vytvořeny.
- **Tabulka properties** obsahuje atributy, které blíže specifikují `machine` a to především v oblasti hardwarové podpory
- **Tabulka owner** upřesňuje vlastníka (osobu), který zodpovídá za server
- **Tabulka responsibility** upřesňuje oddělení, které zodpovídá za server
- **Tabulka hwbox** představuje objekt, pod kterým je veden hardware v CMDB
- **Tabulka service** ustanovuje dobu v týdnu, po kterou je vedena podpora
- **Tabulka contract** uchovává číslo smlouvy
- **Tabulka contracttype** definuje typ smlouvy
- **Tabulka node** uchovává základní přehled vlastností a atributů daného nodu. Hlavním identifikátorem v rámci db je primární klíč `id`, ale z pohledu architektury je významným atributem `node_id`.
- **Tabulka status** uchovává stav, jakých může nod nabývat
- **Tabulka platform** uchovává informace o operačním systému, který je instalován na daný nod, a s ní současně verzi (tabulka `oslevel`) a level (tabulka `oslevel`)



Obrázek 5.1: E-R diagram

- **Tabulka localization** je asociativní tabulka a řeší vztah 1:N mezi tabulkou `machine` a tabulkami upřesňující pozici serveru (`location`, `rack`, `position`).
- **Tabulka location** představuje umístění v určité lokalitě
- **Tabulka rack** představuje označení racku
- **Tabulka position** přestavuje konkrétní pozici "Učko" v racku
- **Tabulka modtype** je opět asociativní tabulka se vztahem 1:N, kde každému serveru je přidělen jeho modelový typ s odkazem na tabulky upřesňující modelovou řadu (`model, type, hwtype, description`).



Obrázek 5.2: Kontextový diagram webové aplikace

5.2 Struktura webové aplikace

Struktura webové aplikace je koncipována dle návrhového modelu MVC, kde platí přesná rozdělení prezenční, aplikační a doménové logiky systému. Aplikace je rozdělena na `frontend` a `backend` rozhraní, kde každé z nich využívá vlastní moduly `model`, `view` a `controller`. Některé moduly jsou však využívány oběma rozhraními, a proto jsou definovány pouze jednou. Mezi jednotlivými komponentami jsou vztahy přesně vymezující procesy, které se odehrávají během zpracování uživatelova požadavku a odezvy na tento požadavek.

Samotné rozhraní je řízeno jednotlivými třídami, představující sadu `controllerů`, které řídí danou činnost aplikace. V případě potřeby je načten příslušný `model` umožňující zpracování databázových dat. Pro snazší orientaci, při úpravě jednotlivých tříd, je shodný název třídy s názvem souboru v němž je třída definována.

Při předávání dat z databáze do příslušného pohledu (např. model/Machine.php) nejprve `controller`(controllers/MachineController.php) načte daný model, ve kterém jsou definovány funkce, které popisují vlastnosti atributů a závislosti na

ostatních tabulkách pomocí cizích klíčů. Ty jsou v `modelu` reprezentovány funkcemi s názvem `get<název_modelu>` obsahující návratovou hodnotu v podobě (např.) :

```
return $this->hasMany(Node::className(), ['machine_serial' => 'machine_serial']);
```

Dále jsou zde nastavena pravidla, dle kterých se řídí vyplňování formulářů a předávání hodnot pomocí metody `$_POST`. Tato metoda umožňuje skryté předávání hodnot mezi `view` a `controller`.

Atributy a definice z tabulky machine

```
public function rules()
    {
        return [
    [['machine_serial','name','mtm', 'owner_id',
    'responsibility_id', 'hw_box_id', 'service_id',
    'contract_type_id', 'contract_id'], 'required'],
    [['support'], 'boolean'],
    [['warranty_till', 'maint_till', 'maint_from'], 'safe'],
    [['response_time', 'service_time', 'owner_id',
    'responsibility_id', 'hw_box_id', 'service_id', 'contract_type_id',
     'contract_id'], 'integer'],
    [['description'], 'string'],
    [['machine_serial', 'mtm'], 'string', 'max' => 8],
    [['name'], 'string', 'max' => 30],

        ];
    }
 public function attributeLabels()
    {
        return [
            'machine_serial' => yii::t('common', 'Machine Serial'),
            'name' => yii::t('common', 'Name'),
            'mtm' => 'Mtm',
            'support' => 'Support',
            'warranty_till' => 'Warranty Till',
            'maint_till' => 'Maint Till',
            'maint_from' => 'Maint From',
            'response_time' => 'Response Time',
            'service_time' => 'Service Time',
            'owner_id' => 'Owner ID',
            'responsibility_id' => 'Responsibility ID',
            'hw_box_id' => 'Hw Box ID',
            'service_id' => 'Service ID',
            'contract_type_id' => 'Contract Type ID',
            'contract_id' => 'Contract ID',
            'description' => 'Description',
        ];
    }
```

Definovaný model je posléze načten příslušnému `controlleru` - `use common\models\Machine`, kde jsou z pravidla definovány funkce pro zobrazení dat, vytvoření nového záznamu, úpravu dat a odstranění dat. V následující funkci je znázorněno získání dat z **modelu** node a machine. Následně pak předání pomocí funkce `render()` pohledu **view**, který provede vizualizaci dat řízenou kaskádovými styly (CSS).

```
  public function actionView($id)
    {
        $node = Node::find()->getNode($id)->all();
        return $this->render('view', [
            'model' => $this->findModel($id),
            'node' => $node,
        ]);
    }
```

V `modelu` jsou implementovány i další funkce, které nejsou součástí CRUD [1] operací, ale s tímto modelem souvisejí. Například funkce pro uložení vykonané události:

```
public function insertEvent($name,$machine_serial) {
        $systemEvent = new SystemEvent;
        $systemEvent->application = 'backend';
        $systemEvent->category = 'machine';
        $systemEvent->event = 'afterUpdate';
        $systemEvent->data = ['name' => $name, 'machine_serial' => $machine_serial];
        $systemEvent->created_at = 'NOW()';
        $systemEvent->save();
    }
```

5.3 Adresářová struktura

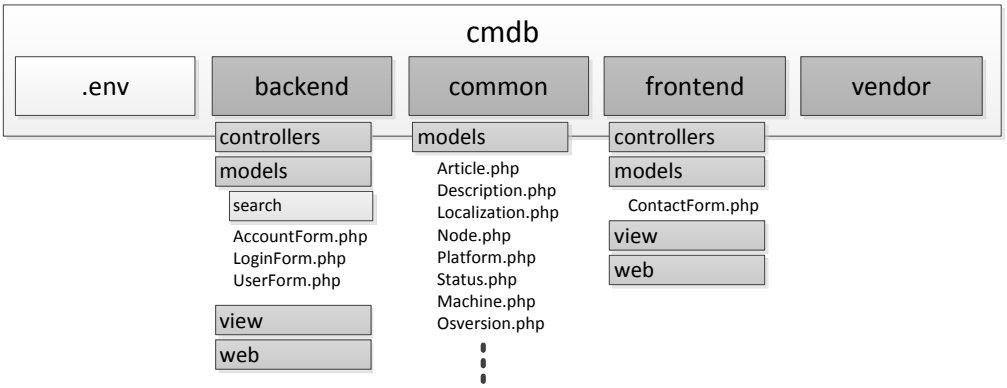
Adresářová struktura webové aplikace je koncipována dle návrhového vzoru MVC. Je rozdělena do třech základních složek `backend, frontend a common`. Každá z těchto složek obsahuje podsložky dle zmíněného návrhového vzoru `model, view a controller`. Mimo již jmenovaných je zde také složka `web`, kde jsou načítány definice příslušných souborů s podrobným nastavením aplikace.

```
<?php
// Composer
require(__DIR__ . '/../../vendor/autoload.php');

// Environment
require(__DIR__ . '/../../common/env.php');

// Yii
require(__DIR__ . '/../../vendor/yiisoft/yii2/Yii.php');

// Bootstrap application
require(__DIR__ . '/../../common/config/bootstrap.php');
require(__DIR__ . '/../config/bootstrap.php');


$config = \yii\helpers\ArrayHelper::merge(
    require(__DIR__ . '/../../common/config/base.php'),
    require(__DIR__ . '/../../common/config/web.php'),
    require(__DIR__ . '/../config/base.php'),
    require(__DIR__ . '/../config/web.php')
);

(new yii\web\Application($config))->run();
```

Použitá struktura odděluje složky jádra `frontend a backend` od složek, které s aplikací komunikují (knihovny - `vendor`). Pro maximální využití a efektivitu

[1] souhrn čtyř základních operací (create, read, update, delete)

zpracování dat je využívána pro `forntend a backend` sdílená složka `common`. Zde jsou uloženy soubory, které jsou sdíleny mezi oběma aplikacemi. Např.: každá aplikace muže přistupovat k informacím uložených v databázi prostřednictvím třídy `ActiveRecord`, a tak načítat data z databáze s využitím modelu dané třídy (třída `Machine` pro načtení informací o serveru).



Obrázek 5.3: Adresářová struktura

Výpis významných složek :

- `backend` - aplikace systému určená především pro administrační správu
- `common` - obsahuje sdílené prostředky, které využívá, jak backend, tak frontend aplikace
- `frontend` - aplikace systému, která poskytuje uživatelské rozhraní koncovému uživateli
- `vendor` - definice modulů, widgetů a podpůrných prostředků pro základní aplikační funkce
- soubor `.env` obsahuje hodnoty pro definici přístupu do databáze a pro přesměrování, dle druhu aplikace - frontend/backend

5.4 Uživatelské role

Uživatelé systému můžeme rozdělit do dvou skupin. První skupinou jsou uživatelé role **`user`**, kteří mají přístup pouze k veřejnému obsahu stránek představující `frontend` aplikaci. Zde mohou nalézt informace o jednotlivých systémech a modifikovat povolená informační pole.

Druhá skupina zahrnuje uživatele, kteří mají povolen přístup do `backendu` aplikace, tito uživatelé jsou vedeni pod rolí **`administrator`**. Mají kontrolu nad celým systémem, který zahrnuje funkce jak `frontend`, tak `backend` aplikace. Funkce, jež má `administrátor` k dispozici jsou znázorněny v kontextovém diagramu celého systému (obrázek 5.2).

5.5 Funkce systému

5.5.1 Načtení a zpracování dat z AIX systémů

Pro získání dat z rozsáhlého `AIX` prostředí je využíván nástroj xCAT, který načte příslušné atributy u jednotlivých systémů, které jsou dále zpracovávány sadou skriptů. Sada skriptů načítá, zpracovává a ukládá data do databáze PostgreSQL.

Skripty jsou spouštěny ve dvou fázích. První část je spouštěna na centrálním nodu, kde je instalován nástroj xCAT master. xCAT master má za úkol za určitý časový úsek kontrolovat aktuální stav nodů a udržovat jej v xCAT databázi. Tyto nody, jsou dle předem stanovených pravidel, rozděleny do xCAT skupin s rozdílnými atributy, které jsou načítány příslušnými skripty.

Definice pojmů :

- **machine** - fyzický hardware
- **node** - LPAR - virtuální systém, kde je instalován operační systém
- **blade** - fyzický hardware, který je součástí Blade chassis, může obsahovat virtuální systémy

Pro serverou část jsou určeny skripty :

- **getMachine.sh**
- **getLpar.sh**
- **getInfo.sh**

getMachine.sh

Skript `getMachine.sh`, pomocí xCATu a jeho rozdělení do skupin, získá základní informace typu: `jméno, sériové číslo, typ, jméno managementu, druh managementu (HMC, BLADE)`. Tyto informace jsou pomocí AWK programu upraveny do formátu.

```
jméno,jméno managementu,druh managementu,typ,sériové číslo,
```

getLpar.sh

Skript `getLpar.sh` také využívá xCAT databázi, tentokrát pro xCAT skupinu `lpar`, ze které získá informace o jméně, jméně managementu, identifikačním čísle a rodiči. Tyto informace korespondují s již zjištěnými informacemi o "`machine`". Jelikož se zjišťují detailní informace přímo z konkrétních nodů, postačí k dalšímu zpracování uložit pouze informace v následujícím tvaru:

```
jméno,skupiny,jméno managementu,id nodu,rodič,
```

getInfo.sh

Skript `getInfo.sh` se již připojuje prostřednictvím SSH (Security Shell), případně PSH (Parallel Remote Shell) protokolu k jednotlivým nodům. A to tak, že prochází seznam všech nodů získaných skriptem `getLpar.sh`, kde načte jméno lparu, které slouží jako hlavní identifikátor v xCAT prostředí. Pomocí něho zjistí příslušnou IP adresu a DNS záznam.

```
dnsip=$(nslookup $name | grep -e 'Name' -e 'Address' | sed '/#/d'|awk '{print $2}'
| awk '{printf "%s%s",$0,NR%2?":":"\n" ; }')
```

IP adresa je důležitým, ne však jediným, faktorem pro kontrolu aktuálního stavu systému. V případě aktivního systému, jsou získávány další informace charakterizující systém.

```
#---------GET oslevel---------
        oslevel=$(psh $name 'oslevel -s'|awk '{print $2}')
#---------GET os platform-----
        platform=$(psh $name 'uname'|awk '{print $2}')
#---------GET date patch-----
        psh $name 'lslpp -ha bos.adt.base | grep APPLY' > datePatch_tmp
        date_patch=$(tail -1 datePatch_tmp | awk '{print $2"," $5}')
        rm datePatch_tmp
#---------GET CPU------------
        cpu=$(psh $name 'lscfg -v | grep proc | wc -l'|awk '{print $2}')
#---------GET MEM----------
        mem=$(psh $name  'vmstat | head -2 | tail -1'|awk '{print $5}'|sed -e 's/mem=//' -e 's/MB//')
```

Všechny doplňující informace jsou opět uloženy do textové podoby do souboru ve tvaru:

```
jméno nodu,id nodu,jméno machine,dns,ip adresa,level operačního systému,platforma,
verze operačního systému,datum instalace posledního patche,počet CPU,veliksot RAM
```

Serverové skripty jsou uchovávány v adresáři `./server/output`, odkud jsou kopírovány, případně synchronizovány na server, kde je instalována databáze a webová aplikace. Zde bude také probíhat jejich další zpracování druhou skupinou skriptů. Ta obsahuje skripty určené pro plnění a úpravu databázových záznamů.

Skripty pro modifikaci databáze :

- **01-importIBM.sh**
- **02-machineToDB.sh**
- **03-oldToDB.sh**
- **04-lparToDB.sh**
- **05-lparInfoToDB.sh**
- **06-skonetToDB.sh**

01-importIBM.sh

Skript `importIBM.sh` načítá aktuální informace o dostupném IBM hardwaru ze souboru dodaném touto společností. Jedná se o název modelové řady, typ, typ hardwaru a popis daného zařízení. Práce skriptu spočívá v procházení záznamů, kontrole duplicit a porovnávání s aktuálními daty v databázi. V případě, že se jedná o nový typ, hardwarový typ nebo popis, je přidán nový záznam do databáze a aktualizována tabulka s modelovými řadami produktů.

Kontrola nového typu :

```
typ=$(echo $line | awk 'BEGIN { FS = "," } ;{print $3}')

chType=$($sdbconnect "select type_id from type where type='$typ'"| sed 's/ //g')
 if [ "$chType" = "" ];then
  $dbconnect "insert into type(type) values('$typ')"
  echo 'pridan typ:' $type
 else
  echo "Tento typ jiz existuje ###### " "'$typ'"
 fi
```

02-machineToDB.sh

Skript `02-machineToDB.sh` provádí načtení hardwaru ze souboru `machine` vytvořeného serverových skriptem `getMachine.sh`. Při zpracování je hardware rozdělen do skupin, které rozlišují, zda se jedná o management či systém jím řízen. Pro managementy jsou definovány skupiny `HMC a BLADE CHASSE`. Do těchto skupin jsou rozděleny jednotlivé systémy dle jejich managementu. Při importu do databáze je použito sériové číslo jako jednoznačný identifikátor, jestliže u hardwaru není definováno, je uložen záznam do `/log` a musí se přidat ručně.

Vložení systému typu blade :

```
if [ "$hcp" = "blade" ];then
 mtm=$(echo $line | awk 'BEGIN { FS = ","} ;{print $4}')
 machine_serial=$(echo $line | awk 'BEGIN { FS = ","} ;{print $5}')
 db_blade=$($sdbconnect "select machine_serial from machine where
  machine_serial='$machine_serial'"| sed 's/ //g')

 if [ "$db_blade" = '' ];then
 # echo $hcp ' ' $mtm ' ' $machine_serial
  $dbconnect "INSERT INTO machine
  (machine_serial,name,mtm,hcp,mpa,owner_id,responsibility_id,
  response_time_id,hw_box_id,service_id,contract_type_id,contract_id,
  modtype_id)
  VALUES ('$machine_serial','$name','$mtm','$hcp','undef','0',
  '0','0','0','0','0','0','0')"

  $dbconnect "insert into
  localization(machine_serial,position_id,location_id,rack_id)
  values('$machine_serial','0','0','0')"

  $dbconnect "insert into properties(machine_serial)
  values('$machine_serial')"

 fi
fi
```

03-oldToDB.sh

V případě importu doposud využívaných dat, které nekorespondují s aktuálním stavem architektury, je využit skript `03-oldToDB.sh`, pomocí něhož jsou porovnávána stará data s aktuálním stavem, který je již uložen v `CMDB` databázi. Skript nejprve načte do databáze hodnoty atributů, jestliže doposud nebyly definovány a to: `service, response_time, contract_type, owner, contract`. V dalším kroku je načteno sériové číslo, které je porovnáno s údaji v databázi. Jestliže jej databáze již obsahuje, jsou u něj aktualizovány hodnoty atributů.

04-lparToDB.sh

Skript `04-lparToDB.sh` načítá ze souboru `lpar` jméno systému `node`, jeho `id` a jméno systému, který slouží k jeho konfiguraci `node_parent`. Na základě toho je vyhledáno `id` managementovacího systému a uloženo společně se jménem do databáze.

05-lparInfoToDB.sh

Poslední fází procesu přidání systémů do databáze je spuštění skriptu `05-lparInfoToDB.sh`, který provede update záznamů získaných z jednotlivých systémů. Skript načítá získané záznamy ze zdrojového souboru. V případě záznamu s datem instalace posledního patche je nutné přetransformovat datum, aby odpovídalo formátu používaného v databázi.

06-skonetToDB.sh

V důsledku předávání informací mezi centrální databází a systémem CMDB je nutné načítat případné úpravy. K tomuto účelu slouží `06-skonetToDB.sh` skript, který prochází jednotlivá pole a kontroluje jejich shodu s aktuální databází.

Při importu z datových souborů je nutné zajistit korektnost jazykového kódování `UTF-8` a způsobu uložení csv formátů (`MS-DOS`), které zaručí korektní rozpoznání řádků. Z důvodu obavy ze ztráty dat, je před použitím skriptů spuštěn zálohovací proces databáze, to umožňuje případnou obnovu.

5.5.2 Správa uživatelů

Správa uživatelů je řízena pomocí centralizovaného přístupu RBAC (Role-Based-Acces), který je rozdělen do dvou částí. První z nich je nastavení dat pro autorizaci a druhá část využívá data o registraci pro řízení přístupu ke konkrétním stránkám.

Jelikož webová aplikace pracuje s citlivými daty, není možné, aby se uživatelé registrovali samovolně. Nejprve si musejí o registraci zažádat. Administrátor tak má absolutní přehled o všech uživatelích systému. K jejich správě je vytvořeno v `backend` aplikaci sekce „`Users`“.

Každému přihlášenému uživateli je k dispozici jeho profil, kde má možnost:

- upravit jméno
- upravit příjmení
- zvolit jazykovou mutaci stránek
- nahrát profilový obrázek
- zaslat žádost o změnu hesla

5.5.3 Správa serverů (machine)

Správa serverů je hlavní článek celého systému. Všichni uživatelé mají možnost získat výpis všech serverů, které jsou součástí infrastruktury. Záznamy jsou získávány z databáze prostřednictvím `modelu Machine`. Pomocí tohoto modelu jsou načítány i detailní informace pro jednotlivé servery s atributy pro:

- umístění
- definice vlastníka
- modelový typ
- definice pro centrální databázi
- definice smluv

Tyto atributy plynou z dlouhodobých zkušeností a potřeb při řešení nečekaných událostí.

5.5.4 Správa nodu (LPARů)

Správa virtuálních serverů je realizována `modelem Node`, ve kterém jsou definovány atributy. Každý LPAR může být umístěn právě na jednom fyzickém serveru a jeho jméno musí být unikátní. Stejně jako u fyzických systémů jsou zde načítány jednotlivé informace získané z databáze, jednak o příslušném LPARu, tak o i fyzickém serveru, kde je LPAR definován.

5.5.5 Export dat

Jednou z hlavních funkcí systému je možnost exportu dat, který je prioritně využíván pro sestavení podkladů při aktualizaci centrální databáze. Kdy v sekci `export` je možné vytvořit sestavu požadovaných informací, které se mají vyexportovat do formátu csv, případně xlsx.

Export dat lze provádět i v sekcích `machine a nodes`, kdy je možné vyexportovat celý seznam nebo pouze jeho vyfiltrovanou část.

5.5.6 Vyhledávání

Integrovaná funkce `search` umožňuje vyhledávat současně mezi systémy a nody, a to dle následujících pravidel.

Systémy se vyhledávají podle :

- jména systému
- sériového čísla
- MTM (Machine Types and Models)

Nody se vyhledávají podle :

- jména nodu
- ip adresy

Při zadání výrazu v pohledu (view) je prostřednictvím metody `\texttt{$_POST}` hodnota předána `SearchControlleru`. Ten ji, pokud existuje, zpracuje voláním modelu `search`. Návratové hodnoty se rozlišují dle toho, jestli se jedná o nalezený systém či nod.

SearchController

```
if ($model->load(Yii::$app->request->post()) && $model->validate()) {
                $findSearch = new Search;
                $machine = $findSearch->searchMachine($model->record);
                $node = $findSearch->searchNode($model->record);

                if ($machine != null){
                return $this->render('index', [
                    'model' => $model,
                    'machine' => $machine, ]);
                }else{
                    $msgn = $model->record . ' not found';
                }

                if ($node != null){
                return $this->render('index', [
                    'model' => $model,
                    'node' => $node, ]);
                }else{
                   $msgn = $model->record . ' not found';
                }
```

5.5.7 Jazyková mutace webové aplikace

Jelikož bude systém nasazen v mezinárodní společnosti, bylo nutné vytvořit jazykovou mutaci webových stránek. Pro tuto funkci byl využit plugin `i18n`, který po implementování umožňuje zobrazení textových řetězců v předem určeném jazyce. Překlad bylo nutné nadefinovat již v samotných `modelech`, kde je volána funkce `Yii:t()`, která kontroluje, jaký jazyk je aktuálně zvolen a dle toho nastaví definici překladu daného jazyka.

```
public function attributeLabels()
   {
       return [
           'username' => Yii::t('common', 'Username'),
           'role' => Yii::t('common', 'Role'),
           'email' => Yii::t('common', 'E-mail'),
           'status' => Yii::t('common', 'Status'),
           'created_at' => Yii::t('common', 'Created at')
       ];
   }
```

Definice překladů představuje seznamy polí řetězců, jejichž klíčem je výchozí jazyk textu. Pro systém je jako výchozí jazyk nastavena angličtina, proto jsou u ostatních jazyků překládány právě anglické výrazy. Zde je příklad pro český překlad, kde byl použit kód jazyka "cs". Funkce `Yii:t()` vyhledává konkrétní výraz pro daný klíč, který pak zobrazí.

```
return [
    'Administrator' => 'Administrátor',
    'Category' => 'Kategorie',
    'Comment' => 'Komentář',
    'Firstname' => 'Jméno',
    'Lastname' => 'Příjmení',
    'Title' => 'Titulek',
    'Username' => 'Uživatelské jméno',
    'Value' => 'Hodnota',
];
```

5.5.8 Ostatní funkce systému

Dokumentace projektů

Webová aplikace disponuje možností vytvářet články, které obsahují informace o infrastruktuře. Články je možné uspořádat do kategorií dle jednotlivých projektů. Pro vkládání nového článku je v administrátorském prostředí určena kategorie `Documentation`, kde je integrovaný textový editor, pomocí něhož lze snadno vytvářet nové články.

Souhrnné informace o systémech

Jestliže je uživatel přihlášen, je mu zobrazena jako domovská stránka `Timeline`, která obsahuje souhrn informací o celém systému (např.: celkový počet fyzických systémů, celkový počet nodů, součet pamětí a cpu). Dále je zde umístěna rychlá navigace dle využití systémů.

5.6 Testování systému

Protože systém bude instalován a využíván v datovém centru automobilové společnosti, bylo nutné před samotným nasazením podrobit aplikaci testování. To probíhalo v několika fázích.

První fáze testování probíhala při samotném vývoji aplikace. A to na lokálním serveru operačního systému Linux prostřednictvím balíku programů XAMPP. Tento

balík obsahuje základní balíčky pro vývoj webových aplikací (`PHP, Apache HTTPD, FileZilla FTP Server, phpMyAdmin, MySQL`). `phpMyAdmin a MySQL` byly nahrazeny, dle nefunkčních požadavků, balíčky phpPgAdmin a PostgreSQL databází. Při testování a během vývoje nebyly zaznamenány problémy, které by souvisely s kompatibilitou balíčků.

Tabulka 5.1: Konfigurace vývojového operačního systému

Operační systém	Linux 3.16.7-21-desktop
Procesor	Intel(R) Core(TM) i7-3520M CPU @ 2.90GHz
Paměť RAM	8 GB
HTTPD	verze Apache/2.4.7 (Unix)
PHP	verze5.5.9
Postgres	verze 9.3.5

Testování při vývoji bylo zaměřeno především na korektní provedení jednotlivých funkcí, přesné zobrazení výsledků a autorizaci dle uživatelských rolí do `backend` nebo `forntend` prostředí. Jelikož samotné PHP nenabízí možnost samotného ladění, bylo využito rozšíření frameworku Yii, které disponuje přehledem chybových hlášení. Ale i přes to byly využívány vlastní funkce s výpisy, které pomáhaly odladit složitější procesy. Tyto funkce jsou však pro běžného uživatele skryty. Jestliže nastane při běhu aplikace neočekávaná chyba, uživateli je v pohledu (`view`) zobrazena stránka s informací o chybě. V pohledu mohou být zobrazovány také chyby serveru: odmítnutí přístupu (403), neexistence stránky (404) nebo vnitřní chyba serveru (500).

Tabulka 5.2: Parametry virtuálního systému

Operační systém	AIX 7.1 7100-03-01-1341
Virtuální procesory	minimální: 1,0
	přiřazeno: 2,0
	maximální: 4,0
Virtuální paměť	minimální: 512 MB
	přiřazeno: 4 GB
	maximální: 8 GB
Virtuální adaptéry	SCSI
	Ethernet adaptér

Druhá fáze testování obnášela testování na operačním systému AIX. Pro jeho využití bylo nejprve nutné daný systém připravit. Pro tento účel byl v datovém centru společnosti SÍŤ, spol. s r. o. vyhrazen jeden LPAR, který byl nakonfigurován jako virtuální server na Power6 Bladu. Tento Blade je umístěn v BladeCenter H Chassis a je spravován pomocí Hardware Managemant Console (HMC). Přehled parametrů virtuálního systému jsou znázorněny v následující tabulce.

Na systému byla instalována PostgreSQL databáze, HTTP server a PHP balíčky pro webovou aplikaci. Zde již byly zaznamenány určité problémy. Především se závislostmi AIX balíčků určených k instalaci PHP. Ty musely být přeloženy pro operační systém AIX, aby bylo možné databázi spravovat i z webového prostředí. Při samotném testování se však využívalo spíše příkazového řádku, který nabízí rychlejší práci s databází.

Základní balíčky pro podporu aplikace :

```
#rpm -qa | grep -e 'php' -e 'postgres'
mod_php_ap22-5.4.33-1
php-cli-5.5.17-1
php-common-5.5.17-1
php-devel-5.5.17-1
php-fpm-5.5.17-1
postgresql-contrib-9.3.5-1
postgresql-devel-9.3.5-1
postgresql-docs-9.3.5-1
postgresql-libs-9.3.5-1
postgresql-server-9.3.5-1
postgresql-test-9.3.5-1
postgresql-9.3.5-1
```

Jelikož se jedná o webovou aplikaci, která bude využita pouze v interní síti, odpadají testy na SEO optimalizaci. I přesto jsou dodržována základní pravidla kaskádových stylů a uživatelsky přívětivých url adres.

5.7 Možná rozšíření systému

Jelikož je IT infrastruktura v automobilové společnosti velice rozsáhlá, naskýtá se tu prostor pro další rozvoj. Systém by mohl být rozšířen o získávání informací z dalšího neméně důležitého hardwaru připojeného v síti, např. :

Informace ze storwize :

- stav využité paměti (fyzické i virtuální)
- k jakému VIO serveru je virtuální disk namapován
- informace o mirroringu

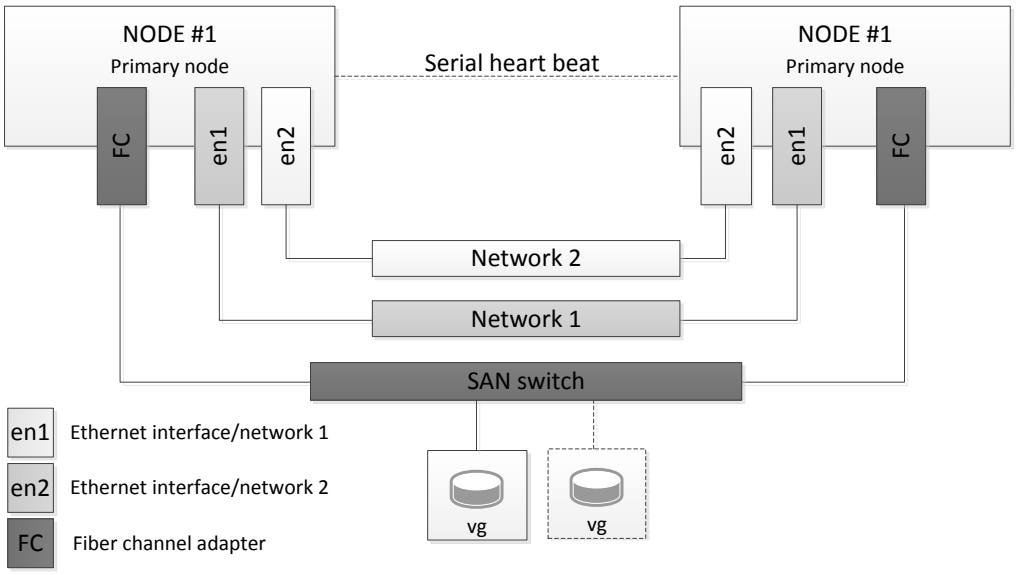
Informace z ethernet a san switche :

- stav portů
- co je na daném portu připojeno

HMC :

- celkový počet a využití RAM
- celkový počet a využití CPU
- stav LPARu

Mimo přidání hardwaru by se také systém mohl rozšířit o další informace získané z jednotlivých nodů. A to především o záznam, který by určoval, zdali se nod nachází v clusteru či nikoli. V tomto případě by se jednalo o cluster s vysokou dostupností (obrázek 5.4), který poskytuje dané služby i při výpadku jednoho serveru definovaného v clusteru.



Obrázek 5.4: Cluster s vysokou dostupností

Pro zjištění, zda je nod součástí clusteru, by bylo nejprve nutné zjistit, zda je na příslušném serveru instalováno rozšíření HACMP, případně jaká verze.

Ověření existence HACMP :

```
# lslpp -l "cluster.*"
  Fileset                      Level  State      Description
  ----------------------------------------------------------------------------
    .
    .
    .
  cluster.es.server.rte     6.1.0.10  APPLIED    ES Base Server Runtime
  cluster.es.server.testtool
                             6.1.0.1  APPLIED    ES Cluster Test Tool
  cluster.es.server.utils   6.1.0.10  APPLIED    ES Server Utilities
  cluster.es.worksheets      6.1.0.3  APPLIED    Online Planning Worksheets
  cluster.license            6.1.0.0  COMMITTED  HACMP Electronic License
    .
    .
    .
```

V případě, že je rozšíření instalováno, musí být v systému definována skutečnost, zda jsou nody opravdu součástí clusteru. Pro tuto definici by byl do systému implementován nový `model`, kde by administrátor či odpovědná osoba měla možnost nadefinovat požadovaný stav.

Dalším krokem ověření korektního nastavení clusteru jsou HACMP nástroje pro monitoring `clinfo a clstat`. `Clinfo` (cluster information) představuje program v podobě démona, který musí běžet na lokálním uzlu a dotazuje se na informace o aktuálním stavu clusteru z cluster Manageru. Cluster Manager aktualizuje data používající interní, dynamicky přidělené datové struktury, které jsou přístupné Clinfo klientům aplikace používající funkce Clinfo API.

Pro sběr informací by bylo vhodnější využít nástroj `clstat`, který vypisuje užitkové zprávy o jednotlivých částí clusteru.

Informační výstup obsahuje :

- název clusteru
- stav jednotlivých nodů (up/down)
- počet nodů v clusteru
- IP adresy a stav jednotlivých rozhraní u příslušného nodu
- definované Resource groupy a jejich stav

Pro tyto informace by bylo nutné modifikovat databázi a upravit importovací skripty.

6 Závěr

Cílem této diplomové práce byl návrh systému, který bude shromažďovat a interpretovat metriky v rozsáhlém IBM Power prostředí. Součástí bylo zvolit vhodný způsob pro uchovávání dat a možnost jejich přenosu.

Nejprve byly uvažovány možnosti využití některého z open source řešení, které již sběr informací z jednotlivých systémů umožňuje. Systémy se však díky specifickým požadavkům ukázaly jako nevyhovující. Z tohoto důvodu byl navržen systém, který se skládá ze dvou částí. Jedna část řeší sběr dat a druhá jejich interpretaci.

Pro načítání dat byla vytvořena sada skriptů, která získává požadované informace z jednotlivých zařízení a ukládá je do databáze postgreSQL. Tyto skripty jsou rozděleny do dvou částí. První část je spouštěna na serveru (centrální management), kde je instalován nástroj xCAT, který umožní získat potřebné data o daných zařízení. Následně jsou data kopírována, případně synchronizována se serverem, na kterém je instalována databáze. Zde je využita druhá sada skriptů, která načte získané informace nejen ze synchronizovaných souborů, ale také ze starších záznamů. Importovány jsou i informace o aktuálních modelech hardwaru společnosti IBM (případně dalších výrobců)

Interpretace dat je reprezentována webovou aplikací, které je navržena pomocí frameworku založeného na architektuře MVC. Návrh probíhal za využití programovacích metod HTML, PHP a CSS. Tato aplikace nabízí dvě oddělená rozhraní, administrátorské a uživatelské. Uživatelské rozhraní umožňuje získávat informace o spravovaných systémech, v administrátorském rozhraní je mimo jiné možné nastavovat systém a manuálně modifikovat záznamy o jednotlivých zařízeních.

Navržený systém splňuje nejen základní požadavky v podobě přesné lokalizace jednotlivých hardwarových zařízení, ale také specifické požadavky, např. informace a upozornění související s hardwarovou podporou.

Při návrhu systému jsem se setkal s chybami, které způsobovaly nefunkčnost skriptů i webové aplikace. V případě testování byly chyby z větší části způsobovány nekompatibilitou příkazů mezi jednotlivými platformami či verzemi software (operační systém, VIO servery, apod). Tyto chyby se podařilo odstranit za pomoci spolupracovníků, diskusních fór a oficiální dokumentace IBM.

V současné době je systém využíván v testovacím provozu, kde jsou testována další možná rozšíření, především pak modul pro získávání informací o clusterech a logických vazbách mezi jednotlivými operačními systémy. Během celozávodní dovolené v roce 2015 je plánován rollout systému na produkční systémy.

Literatura

[1] KROSING Hannu, ROYBAL Kirk, MLODGENSKI Jim. *PostgreSQL Server Programming* Packt Publishing, 2013. ISBN 978-1-84951-698-3

[2] RANDAL K. Michael. *AIX 5L Administration* Mcgraw-hill, 2002. ISBN 078-3254039063

[3] MOMJIAN Bruce. *PostgreSQL: Praktický průvodce* Computer Press, 2003. ISBN 80-7226-954-2

[4] ROBBINS Arnold. *sed and awk Pocket Reference* O'Reilly Media, 2002. ISBN 9780596003524

[5] Postgresql Global Development Group. *PostgreSQL 9.0 (Volume I)* Fultus, 2011. ISBN 9781596822467

[6] SAFRONOV Mark, VINESETT Jeffrey. *Web Application Development with Yii 2 and PHP* Packt Publishing, 2014. ISBN 9781783981885

[7] NEWHAM Cameron, ROSENBLATT Bill. *Learning the bash Shell*, 2. vydání O'Reilly Media, 1998. ISBN 978-1-56592-347-8

[8] XUE Qiang, ZHUO W. Xiang. *The Definitive Guide to Yii 1.1.* 2008 - 2010. Dostupné z WWW: `http://phpqqgroup.googlecode.com/svn/trunk/Php/yii-guide-1.1.2.pdf`.

[9] IBM RedBooks [Online] 2015. [cit.2015-01-10] Dostupné z WWW: `http://www.redbooks.ibm.com/`.

[10] Perzl.org [Online] 2015. [cit.2014-10-18] Dostupné z WWW: `http://www.perzl.org/aix/`.

[11] IBM Knowledge Center [Online] 2015. [cit.2014-11-10] Dostupné z WWW: `http://www-01.ibm.com/support/knowledgecenter/`.

[12] AIX/UNIX Online Documentation [Online] 2015. [cit.2014-12-18] Dostupné z WWW: `http://bio.gsi.de/DOCS/AIX/docs.html`.

[13] AIX for System Administrators [Online] 2015. [cit.2015-02-25] Dostupné z WWW: `http://aix4admins.blogspot.cz/`.

[14] AIX for System Administrators [Online] 2015. [cit.2015-02-25]
Dostupné z WWW: http://aix4admins.blogspot.cz/.

[15] YiiBooster [Online] 2015. [cit.2015-03-14]
Dostupné z WWW: http://yiibooster.clevertech.biz/.

[16] Posts containing - Stack Overflow [Online] 2015. [cit.2015-04-09]
Dostupné z WWW: http://stackoverflow.com/search?q=yii2.

[17] oop-php | Interval.cz [Online] 2015. [cit.2015-04-09]
Dostupné z WWW: https://www.interval.cz/stitek/oop-php/.

[18] SÍŤ, spol. s r.o. [Online] 2015. [cit.2015-04-20]
Dostupné z WWW: http://www.sit.cz/.

A Kompletní seznam databázových relací

```
 Schéma |        Jméno          |   Typ   | Vlastník
--------+-----------------------+---------+----------
 public | article               | tabulka | hop
 public | article_attachment    | tabulka | hop
 public | article_category      | tabulka | hop
 public | contract              | tabulka | hop
 public | contracttype          | tabulka | hop
 public | description           | tabulka | hop
 public | file_storage_item     | tabulka | hop
 public | hwbox                 | tabulka | hop
 public | hwtype                | tabulka | hop
 public | i18n_message          | tabulka | hop
 public | i18n_source_message   | tabulka | hop
 public | key_storage_item      | tabulka | hop
 public | localization          | tabulka | hop
 public | location              | tabulka | hop
 public | machine               | tabulka | hop
 public | model                 | tabulka | hop
 public | modtype               | tabulka | hop
 public | node                  | tabulka | hop
 public | oslevel               | tabulka | hop
 public | osversion             | tabulka | hop
 public | owner                 | tabulka | hop
 public | page                  | tabulka | hop
 public | platform              | tabulka | hop
 public | position              | tabulka | hop
 public | properties            | tabulka | hop
 public | rack                  | tabulka | hop
 public | rbac_auth_assignment  | tabulka | hop
 public | rbac_auth_item        | tabulka | hop
 public | rbac_auth_item_child  | tabulka | hop
 public | rbac_auth_rule        | tabulka | hop
 public | response              | tabulka | hop
 public | responsibility        | tabulka | hop
 public | service               | tabulka | hop
 public | status                | tabulka | hop
 public | system_event          | tabulka | hop
 public | system_log            | tabulka | hop
 public | system_migration      | tabulka | hop
 public | type                  | tabulka | hop
 public | users                 | tabulka | hop
 public | users_profile         | tabulka | hop
 public | widget_carousel       | tabulka | hop
 public | widget_carousel_item  | tabulka | hop
 public | widget_menu           | tabulka | hop
 public | widget_text           | tabulka | hop
```

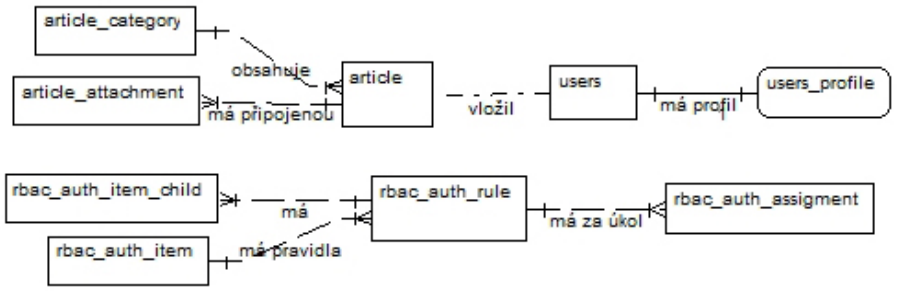
A.1 Detail relací pro správu uživatelů

rbac_auth_rule :

```
  Sloupec   |          Typ          | Modifikátory
------------+-----------------------+--------------
 name       | character varying(64) | not null
 data       | text                  |
 created_at | integer               |
 updated_at | integer               |
Indexy:
    "rbac_auth_rule_pkey" PRIMARY KEY, btree (name)
Odkazovaný:
    TABLE "rbac_auth_item" CONSTRAINT "rbac_auth_item_rule_name_fkey"
    FOREIGN KEY (rule_name) REFERENCES rbac_auth_rule(name) ON UPDATE CASCADE ON DELETE SET NULL
```

rbac_auth_assignment :

```
Sloupec   |          Typ          | Modifikátory
------------+-----------------------+--------------
 item_name  | character varying(64) | not null
 user_id    | character varying(64) | not null
 created_at | integer               |
Indexy:
    "rbac_auth_assignment_pkey" PRIMARY KEY, btree (item_name, user_id)
Podmínky cizího klíče:
    "rbac_auth_assignment_item_name_fkey" FOREIGN KEY (item_name)
    REFERENCES rbac_auth_item(name) ON UPDATE CASCADE ON DELETE CASCADE
```



Obrázek A.1: Doplňující ER-diagram pro správu uživatelů

rbac_auth_item :

```
   Sloupec   |          Typ          | Modifikátory
-------------+-----------------------+--------------
 name        | character varying(64) | not null
 type        | integer               | not null
 description | text                  |
 rule_name   | character varying(64) |
 data        | text                  |
 created_at  | integer               |
 updated_at  | integer               |
Indexy:
    "rbac_auth_item_pkey" PRIMARY KEY, btree (name)
    "idx-auth_item-type" btree (type)
Podmínky cizího klíče:
    "rbac_auth_item_rule_name_fkey" FOREIGN KEY (rule_name) REFERENCES rbac_auth_rule(name)
```

```
    ON UPDATE CASCADE ON DELETE SET NULL
Odkazovaný:
    TABLE "rbac_auth_assignment" CONSTRAINT "rbac_auth_assignment_item_name_fkey"
    FOREIGN KEY (item_name) REFERENCES rbac_auth_item(name) ON UPDATE CASCADE ON DELETE CASCADE
    TABLE "rbac_auth_item_child" CONSTRAINT "rbac_auth_item_child_child_fkey"
    FOREIGN KEY (child) REFERENCES rbac_auth_item(name) ON UPDATE CASCADE ON DELETE CASCADE
    TABLE "rbac_auth_item_child" CONSTRAINT "rbac_auth_item_child_parent_fkey"
    FOREIGN KEY (parent) REFERENCES rbac_auth_item(name) ON UPDATE CASCADE ON DELETE CASCADE
```

rbac_auth_item_child :

```
 Sloupec |          Typ          | Modifikátory
---------+-----------------------+--------------
 parent  | character varying(64) | not null
 child   | character varying(64) | not null
Indexy:
    "rbac_auth_item_child_pkey" PRIMARY KEY, btree (parent, child)
Podmínky cizího klíče:
    "rbac_auth_item_child_child_fkey" FOREIGN KEY (child)
    REFERENCES rbac_auth_item(name) ON UPDATE CASCADE ON DELETE CASCADE
    "rbac_auth_item_child_parent_fkey" FOREIGN KEY (parent)
    REFERENCES rbac_auth_item(name) ON UPDATE CASCADE ON DELETE CASCADE
```

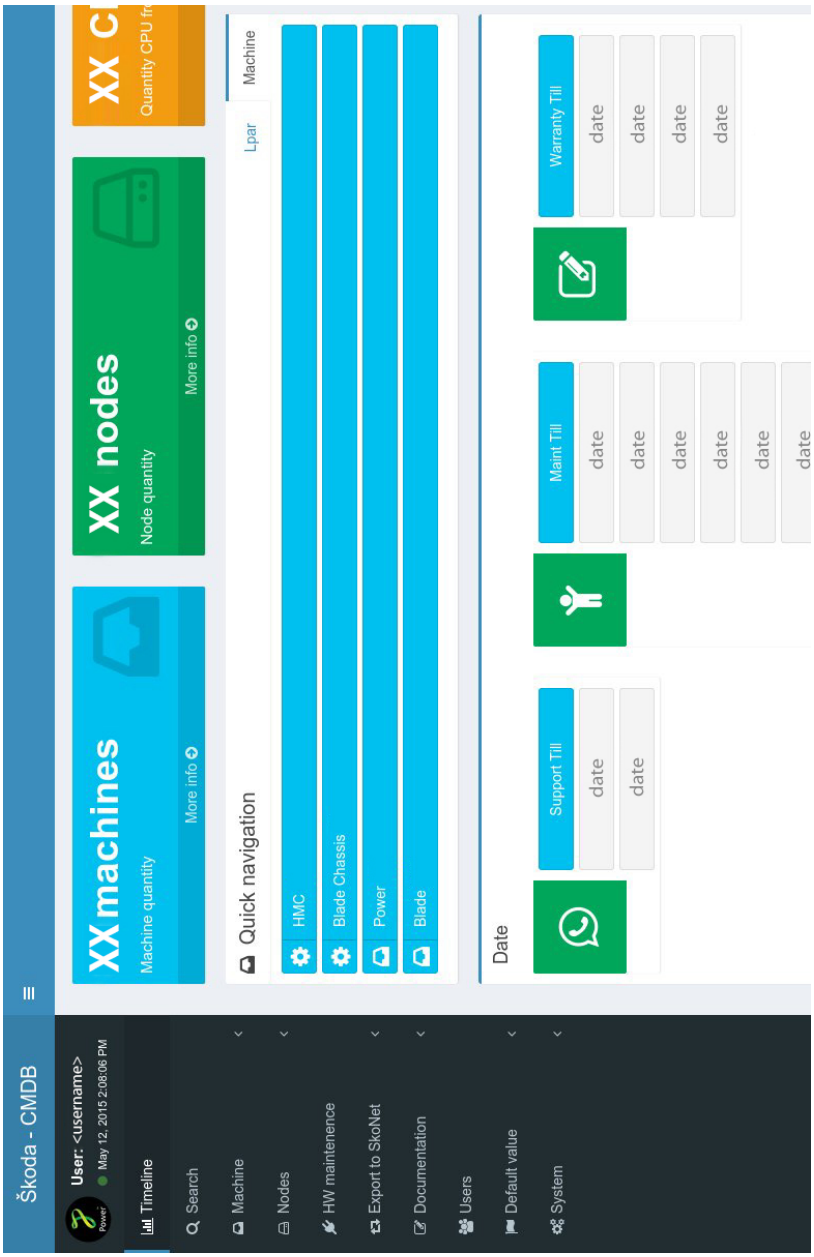
users :

```
      Sloupec        |          Typ           |      Modifikátory
---------------------+------------------------+-----------------------------------------
 id                  | integer                | not null implicitně nextval
                                                ('users_id_seq'::regclass)
 username            | character varying(32)  |
 auth_key            | character varying(32)  | not null
 password_hash       | character varying(255) | not null
 password_reset_token | character varying(255) |
 oauth_client        | character varying(255) |
 oauth_client_user_id | character varying(255) |
 email               | character varying(255) | not null
 role                | smallint               | not null implicitně 1
 status              | smallint               | not null implicitně 1
 created_at          | integer                | not null
 updated_at          | integer                | not null
Indexy:
    "users_pkey" PRIMARY KEY, btree (id)
```

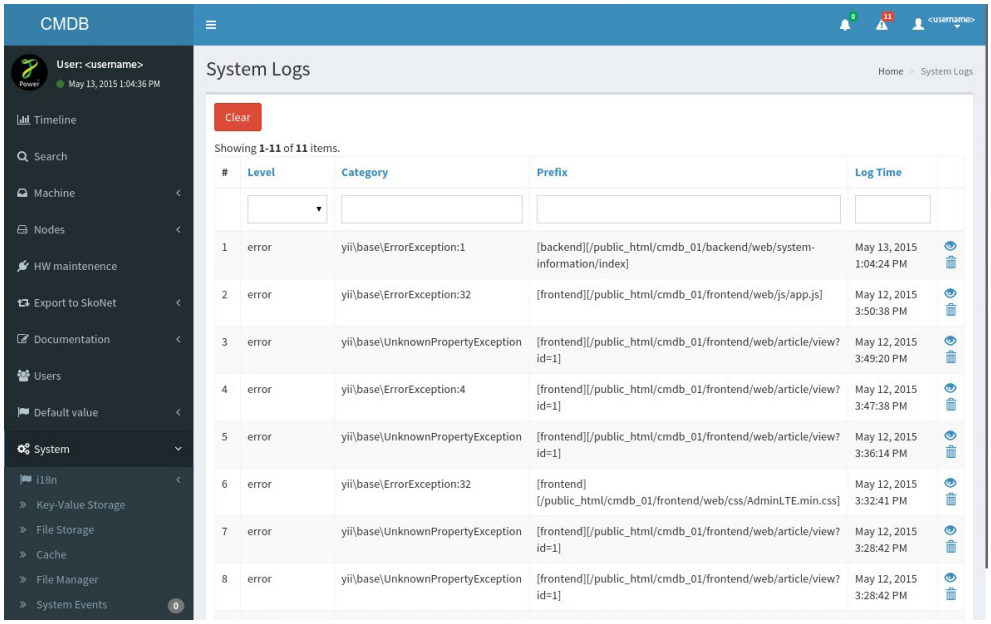
users_profile :

```
 Sloupec     |          Typ           |       Modifikátory
-------------+------------------------+-------------------------------------------
 user_id     | integer                | not null implicitně nextval
                                        ('users_profile_user_id_seq'::regclass)
 firstname   | character varying(255) |
 middlename  | character varying(255) |
 lastname    | character varying(255) |
 avatar_path | character varying(255) |
 avatar_base_url | character varying(255) |
 locale      | character varying(32)  | not null
 gender      | integer                |
Indexy:
    "users_profile_pkey" PRIMARY KEY, btree (user_id)
```

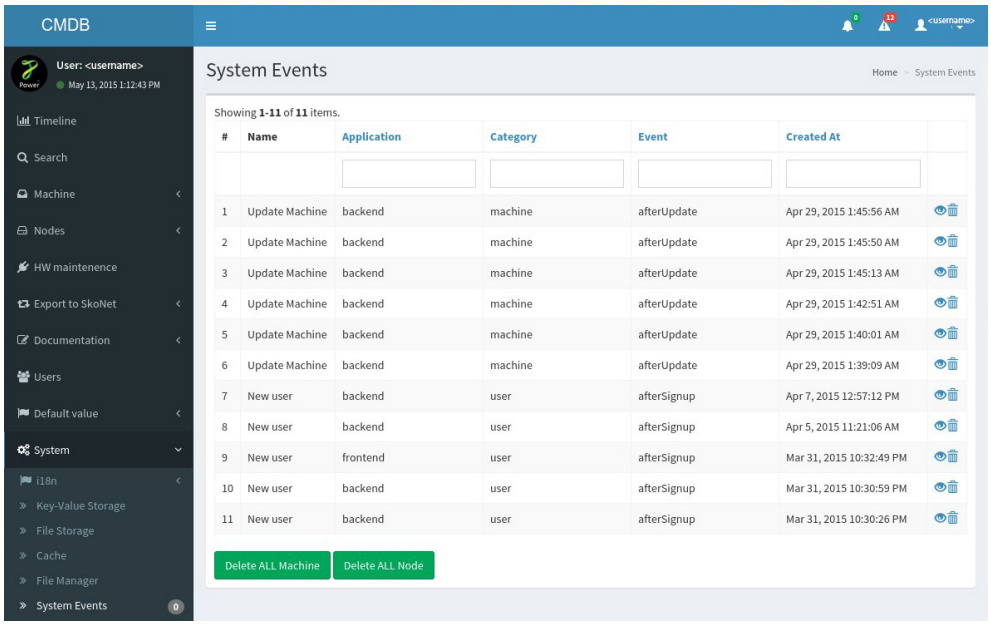
B Administrační prostředí



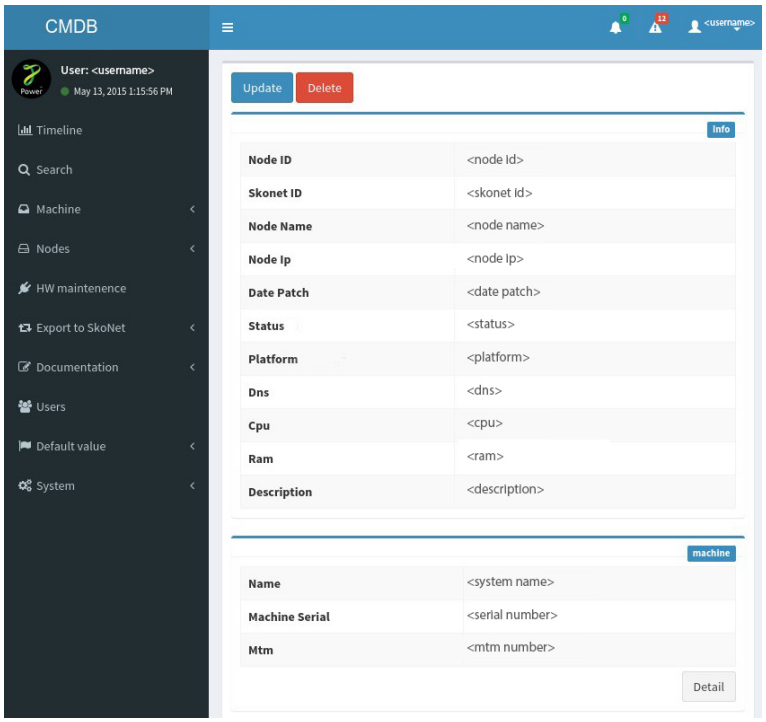
Obrázek B.1: Hlavní stránka administračního rozhraní



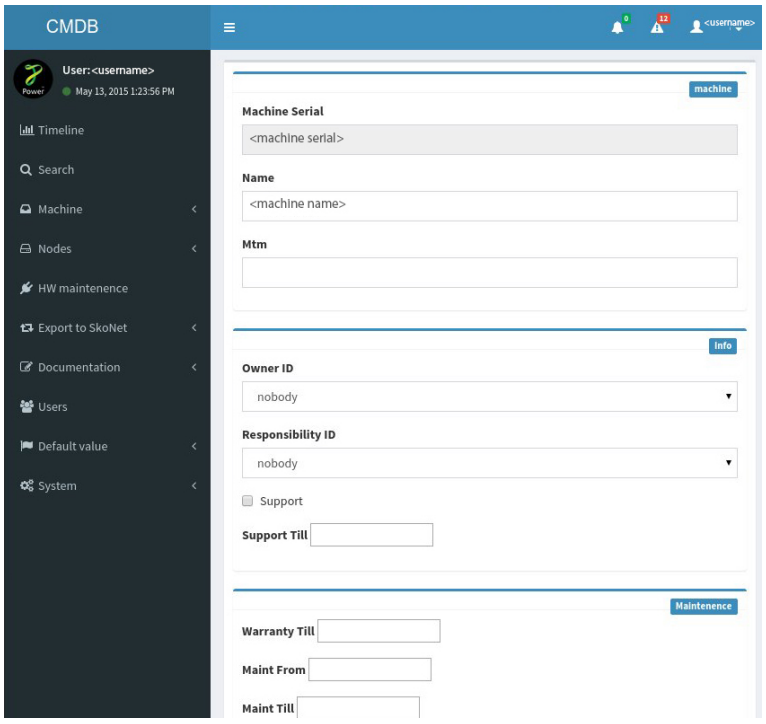
Obrázek B.2: Error logová stránka



Obrázek B.3: Stránka se systémovými logy



Obrázek B.4: Detail stránky s informacemi o LPARu



Obrázek B.5: Stránka pro update machine