ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Nástroj pro správu a monitorování systému souborů ZettaByte
Student:	Tomáš Šimáček
Vedoucí:	Ing. Zdeněk Muzikář, CSc.
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Seznamte se s funkcí, principy a vnitřními strukturami systému souborů ZettaByte (ZFS).

Nad základními příkazy operačního systému (Linux/Solaris) a systému ZFS vytvořte přes webové rozhraní graficky orientovaný administrátorský nástroj. Součástí tohoto nástroje budou i monitorovací funkce zobrazující statické i dynamické chování ZFS systému souborů postavené nad systémovými nástroji sar, iostat, apod.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 25. listopadu 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Nástroj pro správu a monitorování systému souborů ZettaByte

Tomáš Šimáček

Vedoucí práce: Ing. Zdeněk Muzikář, CSc.

12. května 2016

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Zdeňku Muzikářovi, CSc. za cenné rady, připomínky a nápady v průběhu vzniku této bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 12. května 2016 .....

Odkaz na tuto práci

Šimáček, Tomáš. *Nástroj pro správu a monitorování systému souborů ZettaByte*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce se zabývá problematikou administrace souborového systému Zettabyte na operačním systému Solaris. Její součástí je popis základních struktur, vnitřních principů a administračních technik ZFS. Praktická část práce je věnována návrhu a popisu implementace administrátorského nástroje, který slouží pro monitorování a správu souborového systému ZFS. Návrh klade důraz především na bezpečnost, budoucí rozšiřitelnost a integraci výsledného nástroje do operačního systému Solaris.

Klíčová slova Souborový systém Zettabyte, Solaris, webové rozhraní, administrace, monitorování

Abstract

This thesis deals with issues of administrating Zettabyte file system on operating system Solaris. It describes basic structure, internal principles and administrating techniques of ZFS. The practical part deals with design and implementation of administrating tool that will serve for monitoring and managing ZFS. The design is focused mainly on security, extensibility and integration to operation system Solaris.

Keywords Zettabyte file system, Solaris, web interface, administration, monitoring

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Tato práce se věnuje administraci souborového sytému Zettabyte. Popisuje návrh a implementaci grafického nástroje pro jeho monitorování a správu.

Souborový systém Zettabyte (ZFS) je jeden z mnoha souborových systémů, se kterými se dnes můžeme setkat. Byl vyvinut společností Sun Microsystems a integrován do operačního systému Solaris. Tento souborový systém dokáže spravovat velké množství dat díky své 128-bitové architektuře a mimo jiné nabízí funkce pro ověřování integrity dat, správu fyzických úložišť, vlastní softwarový RAID a v neposlední řadě také šifrování dat. Jedním z nedostatků tohoto souborového systému je absence grafického rozhraní pro jeho správu. ZFS nabízí administrátorovi pouze rozhraní na příkazové řádce.

Hlavním cílem této bakalářské práce je navrhnout a implementovat graficky orientovanou nadstavbu souborového systému ZFS, která bude umožňovat jeho monitorování a správu. Tento nástroj bude koncentrovat funkcionalitu ZFS do jednoho místa, aby usnadnil uživateli správu tohoto souborového sytému.

Struktura této bakalářské práce se skládá ze šesti hlavních kapitol. V první kapitole je objasněn důvod, proč by měl být tento administrátorský nástroj vytvořen a jaký přínos by jeho implementace měla pro různé skupiny uživatelů souborového systému ZFS. Druhá kapitola popisuje účel souborových systémů a představuje jejich nejznámější zástupce. V závěrečné části této kapitoly je uvedeno krátké porovnání základních parametrů vybraných souborových systémů. V úvodu třetí kapitoly je stručně představen souborový systém ZFS, jeho historie a vnitřní struktury. Hlavním obsahem této kapitoly je představení základních administračních funkcí a zajímavých principů, které může administrátor ZFS využívat. Čtvrtá kapitola rozebírá několik vybraných nástrojů, které slouží pro monitorování nebo administraci ZFS. Hlavním účelem analýzy těchto nástrojů je objevení jejich kladných a záporných stránek. Implementace výsledného nástroje by se mohla opírat o tyto poznatky a naopak by se mohla vyvarovat případným chybám a problémům, které vybrané nástroje obsahují. Úvod páté kapitoly definuje požadavky, které bude implementovaný nástroj

splňovat. Hlavní obsah této kapitoly se týká návrhu architektury, volby uživatelského rozhraní a analýzy bezpečnostních rizik administrátorského nástroje. Téma poslední šesté kapitoly je samotná implementace navrhnutého nástroje. V této kapitole jsou představeny a stručně popsány jednotlivé komponenty aplikace. Pozornost je také věnována integraci nástroje do operačního systému Solaris a implementaci bezpečnostních opatření, které budou sloužit pro bezpečný chod aplikace. Po hlavních kapitolách následuje stručný závěr, kde je shrnuto, zda bylo dosaženo stanovených cílů a také jsou zde uvedeny některé možnosti budoucího rozšíření implementovaného nástroje.

Nezbytnou součástí této práce jsou zdrojové kódy aplikace a uživatelská příručka, která popisuje instalaci, konfiguraci a spuštění nástroje. Aplikace i příručka jsou dostupné na přiloženém médiu.

Kapitola 1

Cíle práce

V této kapitole je přiblížena problematika, kterou se tato bakalářská práce zabývá. V závěru jsou zmíněny výhody, které by případné vyřešení problematiky mohlo přinést.

1.1 Problematika

Následující problém si můžeme zobecnit na většinu nástrojů či aplikací, které slouží k administraci nějakého systému. U rozsáhlých systémů platí, že s rostoucím počtem funkcí, které systém nabízí, roste i složitost jeho administrace. Aby administrátor mohl rychle a efektivně spravovat tyto systémy, musí důkladně znát všechny jejich funkce a principy. V případě ZFS se jedná o desítky příkazů, které slouží pro ovládání souborového systému. I proto společnost Oracle vydává mnoho online návodů, které společně s manuálovými stránkami jednotlivých příkazů vytvářejí dobrý zdroj informací pro administrátory ZFS.

Pro zkušené administrátory, kteří se v prostředí tohoto souborového systému pohybují jíž delší dobu, je příkazová řádka pravděpodobně nejrychlejší a nejefektivnější volbou. Potřebný příkaz stačí napsat do příkazové řádky a sytém ho provede. Administrátor může tyto příkazy použít ve skriptech, které může využít pro automatizaci některých činností. Na druhou stranu existují uživatelé, kteří se teprve seznamují se souborovým systémem ZFS a nemají potřebné znalosti. Jelikož v ZFS neexistuje grafické rozhraní, kde by byla funkcionalita sdružena na jednom místě, musí začínající uživatelé často využívat návodů nebo manuálových stránek.

1.2 Řešení

Z výše uvedených důvodů jsem se rozhodl navrhnout a implementovat graficky orientovaný administrátorský nástroj pro ZFS, který by sdružoval funkciona-

litu ZFS do jednoho místa. Z tohoto místa by administrátor mohl pohodlně ovládat funkce systému. Tento nástroj by měl být určený především pro seznámení uživatelů se souborovým systémem ZFS a jeho základními funkcemi.

Kapitola 2

Souborové systémy

V úvodu této kapitoly je uvedena definice souborového systému, která je následována výčtem nejznámějších zástupců souborových systémů. V závěru této kapitoly porovnáme několik vlastností vybraných souborových systémů.

2.1 Souborový systém

Souborový systém označuje způsob, kterým se organizují data v datových úložištích. Jak už název napovídá, data jsou v úložišti organizována do souborů. Soubory umožňují ukládání informací na disku a adresáře tyto informace umožňují hierarchicky uspořádat [1]. Tento systém si o každém souboru udržuje informace, aby bylo později možné zjistit, kde se soubor nachází nebo jak je velký. Dále tento systém umožňuje tyto data číst, měnit nebo vytvářet.

V dnešní době existuje mnoho způsobů organizace dat, a proto máme k dispozici i mnoho různých souborových systémů. Ty se liší nejenom ve způsobu organizace dat, ale také ve velikosti datového úložiště, které dokáží adresovat. Dalším rozlišovacím prvkem může být závislost na operačním systému. Jelikož nejrozšířenější operační systémy jsou systémy typu UNIX a Windows, uvedeme několik typických zástupců souborových systémů pro tyto platformy.

Mezi nejznámější zástupce UNIXových souborových systémů patří následující systémy:

- UFS
- ext2, ext3, ext4

Mezi nejznámější zástupce souborových systémů pro platformu Windows se řadí následující systémy:

- FAT12, FAT16, FAT32
- NTFS

Tabulka 2.1: Porovnání souborových systémů

Název	Velikost adresy	Velikost bloku	Max. velikost svazku
FAT-32	28b	4KB-32KB	2TB
NTFS	64b	512B-64KB	16EB
ZFS	128b	512B-128KB	256ZB

2.2 Porovnání

V tabulce 2.1 [1] můžeme vidět porovnání vybraných souborových systémů z hlediska velikosti diskové adresy, velikosti datových bloků a maximální velikosti svazku, který dokáží spravovat.

Kapitola 3

ZFS

Úvod následující kapitoly stručně popisuje vznik a historii souborového sytému ZFS. Zbytek této kapitoly představuje strukturu ZFS, jeho základní stavební kameny a některé zajímavé principy, které tento souborový systém využívá. Pro názornost jsou v této kapitole uvedeny praktické ukázky příkazů sloužící k administraci ZFS.

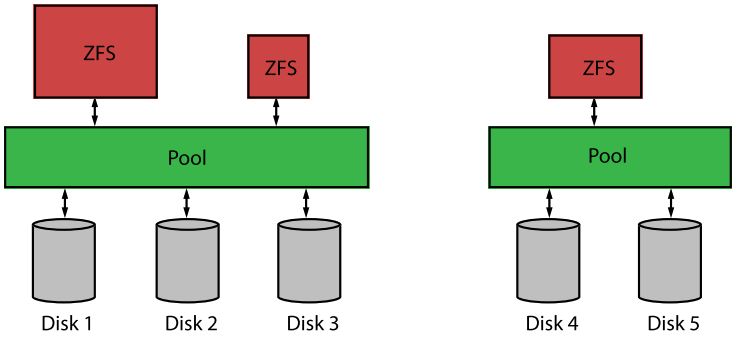
3.1 Úvod

Souborový systém Zettabyte byl původně vyvinut společností Sun Microsystems a následně integrován do operačního systémů Solaris od stejnojmenné společnosti. Jak už to tak v dnešním světě bývá v roce 2010 společnost Oracle akvizicí společnosti Sun Microsystems získala operační systém Solaris a tím i vlastnictví ZFS [2]. V dnešní době tedy veškerý vývoj a podpora pro tyto systémy pochází právě od firmy Oracle. Oficiální dokumenty a návody od této firmy byly hlavním zdrojem informací pro tuto bakalářskou práci [3].

Souborový systém ZFS byl původně navrhnut pro použití čistě v operačním systému Solaris, s čímž se pojily i následující požadavky pro jeho provoz [4]:

- Architektura procesoru SPARC nebo x86
- Operační systém Solaris 10 6/06 nebo novější
- Minimální místo na disku 128 MB
- Minimální místo pro vytvoření poolu 64 MB
- Pro optimální výkon ZFS alespoň 1 GB operační paměti

Díky zveřejnění zdrojových kódů ZFS, došlo k adaptaci souborového systému i na jiné operační systémy než je Solaris. Hlavním příkladem jsou systémy s Linuxovým jádrem jako je například Debian, Fedora nebo CentOS.



Obrázek 3.1: Agregace zařízení do poolu

3.2 Struktura

Většina tradičních souborových systémů se váže na jedno konkrétní zařízení. Aby bylo možné spojit se souborovým systémem více zařízení, byl zaveden tzv. volume manager, který souborovému systému poskytoval iluzi, že se jedná o samostatné zařízení [5]. Ve skutečnosti se pod touto vrstvou může skrývat více disků, které se tváří jako jeden. ZFS se v tomto směru vydalo svojí vlastní cestou a zařízení agreguje do takzvaných poolů, které slouží jako datový základ pro jednotlivé souborové systémy. Tento proces je naznačen na obrázku 3.1.

3.3 Pool

Souborový systém ZFS se skládá ze dvou hlavních stavebních kamenů. Prvním z nich je tzv. pool. V terminologii ZFS je to logické sdružení virtuálních zařízení, které popisuje rozložení a fyzické vlastnosti těchto zařízení [6]. Pool je tedy spojení fyzických nebo virtuálních zařízení, které poskytuje základ pro souborové systémy ZFS.

Souborové systémy v ZFS se už neváží přímo na konkrétní fyzické zařízení, jak tomu bylo dříve, ale na pool jako celek. Historické spojení souborového systému s konkrétním fyzickým zařízením přinášelo mnohá omezení. Například pokud jsme souborový systém chtěli rozšířit, museli jsme ho celý znovu vytvořit. Tato operace mohla být časově náročná, ale v ZFS tomu tak není.

3.3.1 Vlastnosti poolu

ZFS si o každém vytvořeném poolu udržuje informace, které používá pro správu. Každý pool má seznam vlastností, které ho popisují a jednoznačně určují. Hlavním identifikátorem poolu je jeho jméno. Toto jméno musí být

v kontextu celého ZFS unikátní, protože se používá v příkazech manipulujících s daným poolem.

Vlastnosti můžeme rozdělit na dvě skupiny. Na jedné straně jsou statické vlastnosti, které popisují stav poolu a nedají se změnit přímo. Jsou tzv. read-only. Pod tímto druhem vlastností si můžeme představit informace týkající se úložného prostoru jako například velikost poolu, obsazená kapacita nebo volné místo. Velikost poolu nemůžeme změnit přímo nastavením odpovídající vlastnosti na jinou hodnotu, ale můžeme přidat do poolu další zdroj. ZFS tento krok rozpozná a související vlastnosti upraví.

Na druhé straně jsou vlastnosti, které můžeme měnit dynamicky. Některé z těchto vlastností se dají nastavovat pouze v okamžiku, kdy daný pool vytváříme nebo importujeme. Nastavit můžeme například vlastnost *read-only*. Tato vlastnost zajistí, že od doby, kdy byla nastavena, lze z daného poolu data pouze číst a nikoliv data zapisovat. Další zajímavou vlastností je vlastnost *bootfs*, která odkazuje na souborový systém uvnitř poolu, který slouží pro načítání operačního systému. Manuální nastavovaní této vlastnosti se nedoporučuje, protože špatné nastavení může vést k nenačtení operačního systému.

Následující příkaz umožňuje administrátorovi vypsat všechny vlastnosti daného poolu. Pro stručnost jsou vybrány pouze základní vlastnosti.

```
# zpool get all rpool
NAME   PROPERTY       VALUE                SOURCE
rpool  allocated      11.9G                -
rpool  bootfs         rpool/ROOT/solaris   local
rpool  capacity       38%                  -
rpool  dedupratio     1.19x                -
rpool  free           18.9G                -
rpool  health         ONLINE               -
rpool  readonly       off                  -
rpool  size           30.8G                -
```

3.3.2 Vytváření poolu

Pool můžeme v systému ZFS kdykoli dynamicky vytvořit bez potřeby zásahu do operačního systému. Stačí mít k dispozici potřebné zdroje (disk, partition, soubor) k vytvoření námi požadovaného poolu. Pokud máme požadované zdroje, vybereme unikátní jméno pro pool v kontextu ZFS a provedeme například následující příkaz.

```
# zpool create tank c1t2d0 c2t1d0
```

V případě dostupnosti disků c1t2d0 a c2t1d0, tento příkaz vytvoří pool jménem `tank`, do kterého tyto disky přiřadí. Výsledná velikost bude součtem velikostí těchto disků. Takto vytvořený pool neobsahuje žádnou redundanci a

v případě selhání disku může dojít ke ztrátě dat. Druhým typem, které ZFS podporuje, jsou redundantní pooly.

3.3.3 Zrcadlení

Prvním druhem redundantních poolů jsou ty, které obsahují virtuální zařízení *mirror* nebo-li zrcadlení. Tento typ virtuálního zařízení můžeme pomocí ZFS vytvořit nad dvěma a více zařízeními. Uvnitř zařízení dochází k replikaci dat na všechny disky v zařízení. Výsledná kapacita virtuálního zařízení je tedy rovna kapacitě jednoho disku. Pokud zrcadlení vytvoříme nad třemi disky, nakonec budou data na všech třech discích stejná a v případě výpadku dvou z nich o data nepřijdeme. Jiná situace je samozřejmě v případě výpadku všech tří disků najednou. Zrcadlení nám v tomto případě nepomůže a všechny data ztratíme. Nicméně pravděpodobnost, že dojde k výpadku všech tří disků najednou je malá.

Následující příkaz demonstruje vytvoření poolu s redundantním zařízením *mirror* o třech discích.

```
# zpool create Pool mirror c1t1d0 c1t2d0 c2t1d0
```

3.3.4 Raidz

Druhým typem redundantních poolů, které ZFS umožňuje vytvářet, jsou pooly obsahující virtuální zařízení *raidz*. Toto virtuální zařízení využívá jednoho nebo více paritních disků, které nám v případě výpadku nějakého disku umožní dopočítat ztracená data. Podle počtu paritních disků nám ZFS umožňuje vytvářet následující typy zařízení:

- *raidz1* - stripování s jedním paritním diskem
- *raidz2* - stripování se dvěma paritními disky
- *raidz3* - stripování se třemi paritními disky

Zápis dat na disk se provádí v pevně stanovených jednotkách tzv. stripech. Jeden stripe je rovnoměrně rozprostřen po všech datových discích virtuálního zařízení *raidz*. Pokud bychom tedy zapisovali na disky soubor, který by měl přesně velikost stripu, soubor bude rovnoměrně rozprostřen na všech discích. Pokud by došlo k výpadku jednoho disku z virtuálního zařízení, dojde tím i k poškození určité části souboru a nebude možné tyto data přečíst. Pro tento účel se využívá paritních disků. Při zápisu stripu na disky se ze zapisovaných dat vypočte jedna hodnota, která se následně uloží na paritní disk. V případě výpadku jednoho disku z virtuálního zařízení, jsme schopni dopočítat chybějící data z hodnoty parity. Ztracená data jsme schopni dopočítat pouze v případě, že v jednom okamžiku vypadne maximálně takový počet datových disků, jako máme k dispozici paritních disků.

Jelikož jsou data rovnoměrně rozmístěna na všechny datové disky ve virtuálním zařízení, můžeme při čtení dat využívat více disků najednou. Tím dokážeme docílit vyšší rychlosti čtení.

Následující příkaz ukazuje, jak jednoduše se dá v souborovém systému ZFS vytvořit redundantní pool typu *raidz1*.

```
# zpool create Pool raidz1 c1t1d0 c1t2d0 c2t1d0 c2t2d0
```

3.3.5 Rozšiřování poolu

Výhodou architektury poolů je možnost dynamického přidávání fyzických nebo virtuálních zařízení do poolu. Tím získáváme možnost dynamického rozšiřování kapacity celého poolu, která u tradičních souborových systému byla komplikovaná (nutnost použití VM) nebo nebyla dostupná vůbec. Souborové systémy v ZFS, které jsou vytvořené nad stejným poolem, sdílejí jeho prostředky, a proto při rozšiřování kapacity tohoto poolu budou mít k dispozici více místa pro ukládání dat.

V případě neredundantních poolů, které neobsahují virtuální zařízení typu *mirror* nebo *raidz*, můžeme pro rozšíření použít soubor, disk nebo část disku (partition). ZFS nám pro tento účel nabízí příkaz `zfs add`, kterým můžeme dynamicky pool rozšiřovat.

Pokud pool obsahuje redundantní zařízení *mirror* nebo *raidz*, je doporučené pool rozšiřovat pouze o stejný typ zařízení. Pokud se do poolu typu *mirror* pokusíme přidat virtuální zařízení *raidz*, ZFS nám vypíše varovnou hlášku.

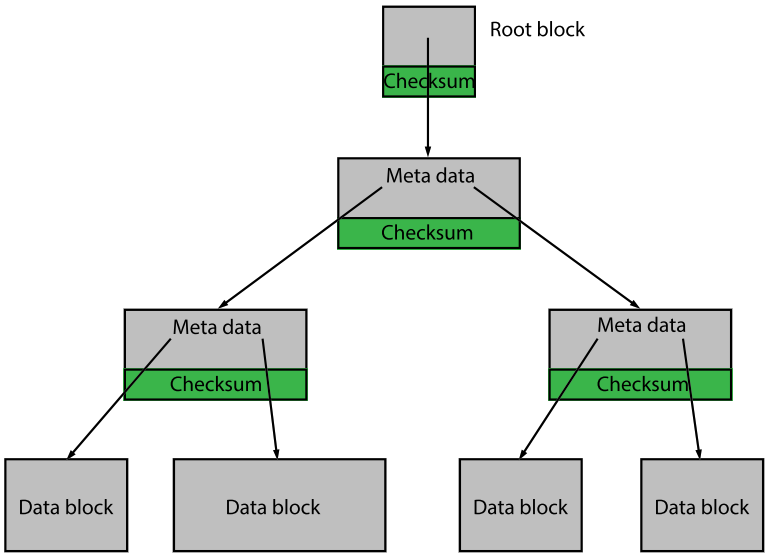
Využití souboru jako zdroje pro ZFS pool se v praxi téměř nevyužívá. Znamená to přidání další vrstvy mezi ZFS a samotný zdroj dat. Hlavní nevýhodou tohoto způsobu je značné zpomalení procesu čtení a zápisu dat, ale na druhou stranu se tato volba výborně hodí pro testování funkcí souborového systému ZFS.

3.3.6 Zrušení poolu

Stejně jako můžeme dynamicky pool vytvářet a rozšiřovat, můžeme pool i kdykoli zničit a uvolnit tak prostředky, které využíval. Tyto prostředky jsou ihned k dispozici a můžeme z nich vytvořit nový pool. Jelikož všechny souborové systémy vytvořené nad poolem sdílejí jeho prostředky, jsou tyto systémy zničeny spolu s tímto poolem. Ke zničení poolu, který jsme dříve vytvořili, můžeme použít následující příkaz.

```
# zpool -r destroy tank
```

Pokud v systému existuje pool jménem `tank`, příkaz ho zničí bez ohledu na to, kolik souborových systému nad tímto poolem bylo vytvořeno.



Obrázek 3.2: Struktura souborového systému ZFS

3.4 Souborový systém

Druhým ze stavebních kamenů ZFS jsou samotné souborové systémy. Jak bylo zmíněno v kapitole 2.1, souborový systém je způsob organizace dat do souborů v datových úložištích. Tradiční souborové systémy využívají k organizaci různých datových struktur. Některé systémy využívají tzv. inodů, což je datová struktura držící atributy souboru a ukazatele na jednotlivé datové bloky [1]. Souborový systém FAT k organizaci dat využívá tzv. file allocation table [1]. ZFS k tomuto účelu využívá stromové datové struktury, která je znázorněna na obrázku 3.2. Jedná se v podstatě o systém odkazů, který zároveň udržuje informace o souborech a adresářích. Bloky s daty se nacházejí v listech stromu.

Tato stromová struktura přináší zajímavé výhody, které budou podrobněji popsány v dalších kapitolách.

3.4.1 Hierarchie souborových systémů

Jednou z vlastností ZFS je možnost hierarchického vnořování souborových systémů do sebe. Pool se sám chová jako souborový systém, a proto už při vytváření souborového systému uvnitř poolu dochází k hierarchickému vnořování. Přesvědčit se o tom můžeme pomocí následujícího příkazu, kterému místo jména souborového systému předáme jméno poolu. Pokud příkaz proběhne dobře, vypíše základní informace o souborovém systému.

```
# zfs list rpool
NAME    USED  AVAIL  REFER  MOUNTPOINT
rpool  12.1G  18.2G  4.85M  /rpool
```

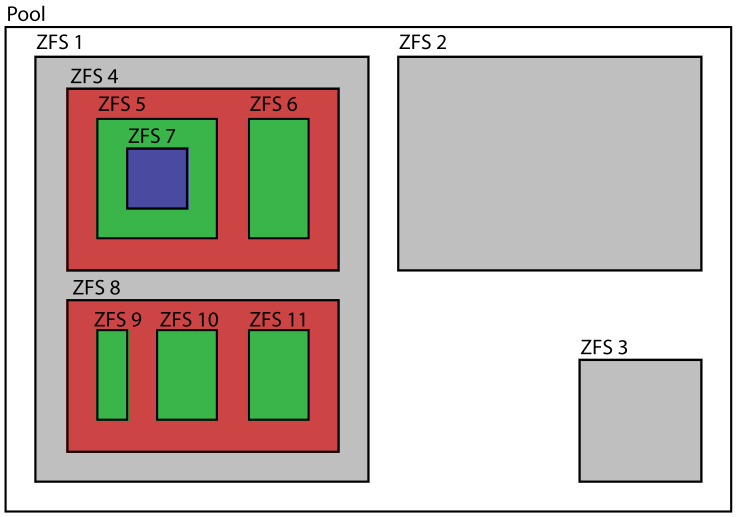
Pokud by zmíněný souborový systém neexistoval, příkaz by vrátil chybovou hlášku.

Pool tedy tvoří hlavní uzel v této hierarchické posloupnosti a všechny souborové systémy vytvořené nad tímto poolem jsou jeho potomkem. V jednom poolu můžeme vytvořit více souborových systémů a také je do sebe můžeme libovolně vnořovat. Nejlépe si můžeme tuto hierarchii souborových systémů ukázat na obrázku 3.3, kde můžeme vidět, jak se do sebe jednotlivé souborové systémy vnořují.

3.4.2 Vlastnosti souborového systému

Stejně jako si ZFS udržuje u každého poolu jeho vlastnosti, tak si je také udržuje u každého souborového systému. Nastavováním těchto vlastností je administrátor schopen určit specifické chování konkrétního souborového systému. V kontextu ZFS má každý souborový systému jednoznačný identifikátor, který je odvozen z hierarchie souborových systémů. Skládá se z názvu systému a názvů jeho rodičů. Identifikátor by mohl vypadat následovně *rpool/ROOT/solaris*.

Vlastnosti souborových systémů můžeme rozdělit, stejně jako u vlastností poolu, na dvě kategorie. První kategorií jsou vlastnosti, které se dají pouze číst. Velkou část této skupiny tvoří vlastnosti popisující využití místa v souborovém



Obrázek 3.3: Hierarchie souborových systémů ZFS

systému. Příkladem může být místo spotřebované vnořenými souborovými systémy nebo souborovým systémem samotným. V této skupině vlastností můžeme najít také datum vytvoření nebo poměr deduplikace a komprese dat.

V druhé kategorii jsou vlastnosti, které může administrátor změnit za běhu nebo při vytváření souborových systémů. Stejně jako na úrovni poolu, tak i na úrovni souborového systému, můžeme nastavit vlastnost *read-only*, která znemožní uživatelům zapisovat nebo měnit data v souborovém systému. Za povšimnutí dále stojí vlastnost *exec*, která dovoluje v souborovém systému spouštět procesy.

Administrátor může jednoduše zapínat, vypínat či měnit vlastnosti souborového systému pomocí příkazu `zfs set property=value rpool`, popřípadě si zobrazit seznam všech dostupných vlastností o souborovém systému pomocí příkazu `zfs get all rpool`.

3.4.3 Dědičnost

Díky hierarchické struktuře se dá jednoduše zjistit, kdo je předkem daného souborového systému a kdo je jeho potomkem. Všechny nastavitelné vlastnosti, kromě kvót a rezervací, se dědí od rodičovského souborového systému. Pokud žádný rodič nemá vlastnost nastavenou explicitně, je použita standardní hodnota [7].

Pokud chceme explicitně říci, aby některá vlastnost byla převzata od rodiče, můžeme použít příkaz `zfs inherit property rpool`. Tento příkaz odnastaví danou vlastnost ze souborového systému a pokusí se převzít hodnotu z rodiče. Pokud v žádném z rodičů není hodnota nastavena explicitně, je použita standardní hodnota [7].

3.4.4 Vytváření souborového systému

Vytváření souborových systémů v ZFS je velice jednoduché. Jediné co potřebujeme k vytvoření souborového systému je jednoznačný identifikátor. ZFS umožňuje vytvoření těchto systémů pomocí příkazu `zfs create pool`, kde *pool* je identifikátor. Pokud neexistují všichni předci, je potřeba použít přepínač `-p`, jinak příkaz selže.

Při vytváření je možné specifikovat, které vlastnosti daný souborový systém bude mít. Mimo jiné existují vlastnosti, které se dají nastavit pouze při vytváření. Po vytvoření souborového systému jsou tyto vlastnosti vedeny jako read-only a nedají se změnit. Příkladem může být vlastnost *encryption*, která umožňuje šifrování souborového systému pomocí různých kryptografických funkcí. Pro využití této vlastnosti můžeme při vytváření poolu použít následující příkaz.

```
# zfs create -o enryption=on tank/pool/pool
```

3.4.5 Zrušení souborového systému

Rušení souborových systému je v ZFS stejně snadné jako jejich vytváření. Je důležité mít na paměti, že zrušením souborového systému, který má nějaké potomky, zrušíme i všechny vnořené souborové systémy. Pro zrušení souborového systému již stačí použít příkaz `zfs destroy` s příslušným identifikátorem popřípadě přepínačem `-r` pro zrušení všech potomků.

3.5 Kvóty

Kvóty v ZFS umožňují administrátorovi spravovat limity souborových systémů týkajících se využitého místa. Tyto limity omezují možnosti využití dostupného místa v souborovém systému. Kvóty můžeme rozdělit do následujících kategorií podle toho, koho omezují:

- **Obecné** - týkající se souborového systému
- **Uživatelské** - omezují jednotlivé uživatele
- **Skupinové** - omezují celé skupiny uživatelů

Obecné kvóty omezují místo, které může souborový systém využít jako celek. K tomuto účelu v ZFS slouží vlastnost *quota*, která stanovuje limit místa, které může daný souborový systém využít. V tomto případě se do využitého místa započítává i místo spotřebované potomky [8]. Pokud bychom chtěli omezit souborový systém bez ohledu na to kolik místa využijí jeho potomci, musíme použít vlastnost *refquota*.

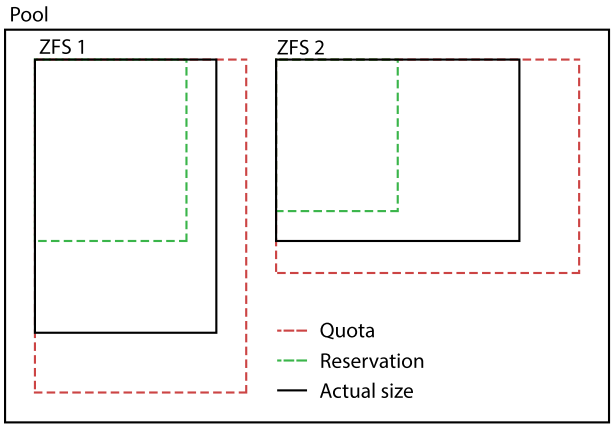
Vzhledem k faktu, že souborové systémy v jednom poolu sdílí diskové místo, ZFS přichází s vlastností *reservation*. Tato vlastnost oproti kvótám garantuje souborovému systému diskové místo dané velikosti [8]. Stejně jako v případě vlastnosti *quota* je toto místo vyhrazeno pro souborový systém a všechny jeho potomky. Pokud bychom chtěli garantovat místo přímo souborovému systému, bez ohledu na to kolik místa využívají jeho potomci, musíme využít vlastnost *refreservation*.

Jelikož vlastnosti *quota* resp. *refquota* a vlastnosti *reservation* resp. *refreservation* jsou vlastnostmi souborového systému, můžeme je nastavit stejně jako kteroukoli jinou vlastnost pomocí následujících příkazů:

```
# zfs set quota=10G rpool/export/home
# zfs set reservation=10G rpool/export/home
```

Vztahy mezi vlastnostmi *quota* a *reservation* jsou vyznačeny na obrázku 3.4.

Dále můžeme omezit využití diskového místa pomocí **uživatelských** resp. **skupinových kvót**. Tyto kvóty limitují uživatele resp. skupinu v tom, kolik místa můžou v souborovém systému využít. Každý souborový systém má



Obrázek 3.4: Vztahy mezi vlastnostmi *quota* a *reservation*

svůj vlastní *userspace* resp. *groupspace*, kde si udržuje informace o tom, jaké mají limity jednotlivé skupiny resp. uživatelé. Administrátor si může hodnoty uživatelských resp. skupinových kvót vypsat pomocí `zfs userspace` resp. `zfs groupspace`. Následující příkaz demonstruje, jak vypadá výpis uživatelských kvót pro domovský adresář.

```
# zfs userspace rpool/export/home/simactom
TYPE        NAME       USED  QUOTA  SOURCE
POSIX User  root      1.50K   none  default
POSIX User  simactom   641M   none  default
```

Pokud se uživatel resp. skupina v seznamu nenachází nebo je u něj uvedena hodnota *none*, uživatelské resp. skupinové kvóty se na něj nevztahují. Je-li v seznamu uvedena jiná hodnota, pak uživatel v daném souborovém systému nesmí překročit tuto hodnotu. Pokud by tuto hodnotu překročil, systém ho varuje a požadovanou akci neprovede. Přiřazení kvóty uživateli resp. skupině provedeme pomocí následujících příkazů:

```
# zfs userquota@username=value filesystem
# zfs groupquota@username=value filesystem
```

ZFS nabízí ještě vlastnost *defaultuserquota* resp. *defaultgroupquota*, kterou lze nastavit na určitou hodnotu. Tento limit se použije v případě, že hodnotu uživatelské resp. skupinové kvóty nastavíme na hodnotu *default*.

3.6 Deduplikace

Jednou z další vlastností, které ZFS nabízí, je deduplikace. Jak již z názvu vyplývá, jedná se o proces, který zabraňuje duplikaci stejných dat na disku a tím šetří i místo. Deduplikace se obecně dá rozdělit na následující úrovně:

- **Úroveň souborů**
- **Úroveň datových bloků**
- **Úroveň bajtů**

K identifikaci stejných dat se v ZFS využívá hašovací funkce. Z porovnávaných dat jsou vytvořeny haše, které jsou následně porovnány. Pokud se haše rovnají a používáme kvalitní hašovací funkci, můžeme s velkou pravděpodobností říci, že jsou porovnávaná data opravdu totožná [9]. Pokud je administrátor z nějakého důvodu stále nedůvěřivý, může u vlastnosti deduplikace nastavit hodnotu *verify*, která zajistí celkové porovnání datových částí bajt po bajtu.

Deduplikace **souborů** využívá nejméně systémových prostředků, protože se data hašují a porovnávají po velkých částech. Na druhou stranu sebemenší změna v souboru vyžaduje přepočítání haše, který se již bude lišit od původního [9]. Souborový systém pak rozpozná, že se soubory liší a uspořené místo bude zaplněno změněným souborem.

Deduplikace **datových bloků** vyžaduje oproti deduplikaci souborů vyšší využití systémových prostředků, ale na druhou stranu přináší v jistých situacích lepší úsporu místa [9].

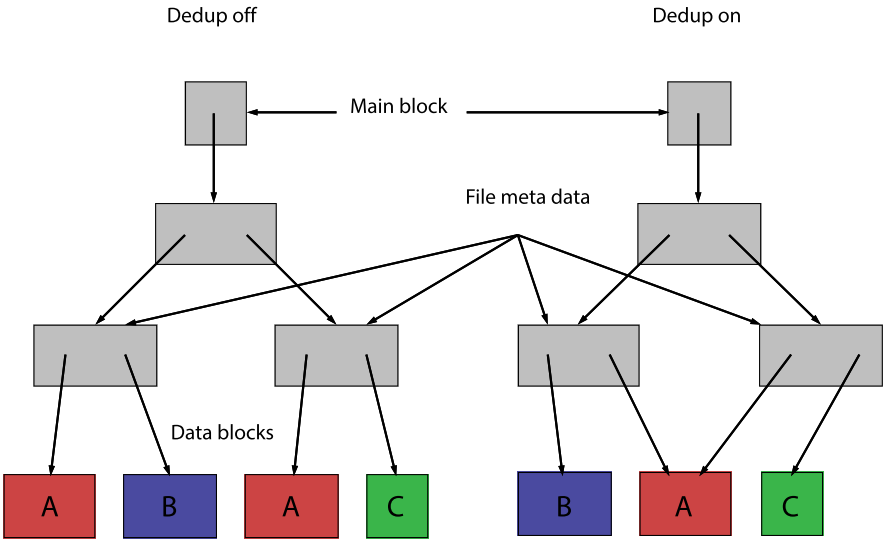
V poslední řadě existuje deduplikace na úrovni **bajtů**, která je ze všech uvedených úrovní nejnáročnější na využití výpočetních prostředků. Tato možnost se používá jen ve speciálních případech.

ZFS z výše uvedených možností nabízí právě deduplikaci na úrovni datových bloků, protože tato jednotka deduplikace přináší nejvíce výhod v obecných případech použití [9]. Na obrázku 3.5 můžeme vidět porovnání souborového systému při zapnuté a vypnuté vlastnosti deduplikace.

3.7 Konzistence

Starší souborové systémy zapisují data přímo do bloku, který mění. To může v některých případech, jako je selhání systému nebo výpadek proudu, vést k nekonzistenci souborového systému. V takovém případě je nutné celý souborový systém zkontrolovat například pomocí příkazu `fsck`. Bohužel ani tento nástroj v některých případech nedokáže některé nekonzistence opravit [10].

Některé souborové systémy využívají tzv. žurnálů k pomoci s udržování konzistence dat. Žurnál je speciální záznam, kam se ukládají informace o tom, co a kde se bude měnit. Po provedení popisované akce se do žurnálu zapíše,



Obrázek 3.5: Ukázka deduplikace na úrovni datových bloků

že daná operace byla provedena. Když systém zkolabuje v jakémkoli kroku tohoto postupu, je možné ho při startu sytému dostat do konzistentního stavu. K tomuto účelu se opět využívá nástroj `fsck`.

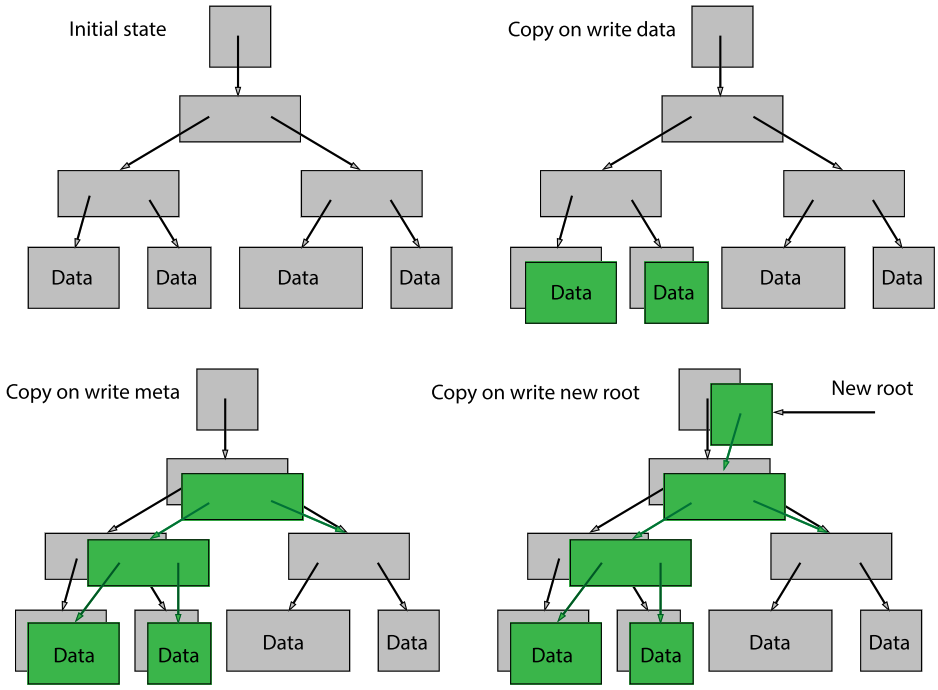
ZFS přichází s transakčním systémem a technikou copy on write. Tato technika zajišťuje neustálou konzistenci souborového systému i v okamžiku výpadku, a proto není třeba žurnálu. Data na disku nejsou nikdy přímo přepisována [10].

Transakce může vypadat následovně. Nejprve se vytvoří kopie bloků, které mají být změněny. V těchto kopiích dojde k vlastní změně dat, zatímco původní data jsou stále na disku a nejsou nijak změněna. Následně se změní potřebná metadata. V poslední řadě se provede atomická operace, která připojí větev s novými resp. změněnými bloky do stromu souborového systémů. Stará data jsou smazána.

Průběh transakce je naznačen na obrázku 3.6. Celá transakce buď proběhne v pořádku nebo nebude přijata vůbec. Pokud v průběhu transakce dojde k výpadku, celý souborový systém zůstává konzistentní, protože původní data jsou stále na disku a nemohla se provést atomická operace, která by připojila nová data.

3.8 Integrita dat

Od souborového systému očekáváme, že při čtení dat z určitého datového bloku z disku, dostaneme data, která do tohoto bloku byla dříve zapsána.



Obrázek 3.6: Demonstrace copy on write

Pokud tomu tak není, měl by to souborový systém detekovat a vrátit chybu [11].

Vlastnost neustálé konzistence souborového systému ZFS zmíněná výše v kapitole 3.7 zajišťuje, že chybná data se v souborovém systému mohou vyskytnout jen chybou hardwaru [12].

ZFS dokáže tyto chyby detekovat pomocí tzv. kontrolních součtů. Tyto součty vznikají a jsou přepočítávány při každé změně bloku v souborovém sytému. Jak jsme mohli vidět na obrázku 3.2, součty nejsou uloženy v bloku, kterého se týkají, ale v bloku jemu nadřazeném. Při každém požadavku na čtení dojde k vypočítání kontrolního součtu na základě dat, která se v souborovém systému momentálně nachází a poté dojde k porovnání vypočteného součtu se součtem uloženým v nadřazeném bloku [11]. Tím ZFS dokáže rozhodnout, zda je blok poškozený či nikoliv.

Díky těmto kontrolním součtům se souborový systém dokáže v určitých situacích sám opravovat. Tato situace nastává v případě, že blok, při jehož čtení došlo k chybě, je součástí redundantního virtuálního zařízení *raidz* nebo *mirror*. V tomto případě ZFS přečte datový blok i z replikovaného disku nebo sestaví blok pomocí parity a následně opraví chybný blok, při jehož čtení došlo k chybě. Aplikace dostane správná data i přes to, že původně načtený datový blok byl poškozený.

Ačkoliv dochází k ověřování integrity dat automaticky při každém čtení z disku, je možné tento proces manuálně vyvolat a ověřit pomocí následující sekvence příkazů:

```
# zpool scrub pool
# zpool status pool
pool: pool
state: ONLINE
scan: scrub repaired 0 in 6s with 0 errors
```

3.9 Mountpoint

Mountpoint je jednou ze základních vlastností všech souborových systémů. Je to vlastně cesta k bodu v adresářovém stromu operačního systému, kde má být daný souborový systém připojen.

Jelikož se ZFS snaží administrátorům maximálně ulehčit práci, stará se o automatické připojování souborových systémů sám. K účelům připojování souborových systémů v ZFS slouží jejich vlastnost *mountpoint*, kde administrátor může specifikovat, kam se má daný souborový systém připojit. Pokud administrátor tuto vlastnost explicitně neurčí, je odvozena od vlastnosti rodičovského souborového systému.

Pro připojování souborových systémů ZFS se dají použít i tradiční nástroje *mount* a *umount*. V tradičních souborových systémech se pro automatické připojování používal soubor */etc/vsfstab*, kde administrátor specifikoval co a kam se má připojit. Operační systém po svém startu tento soubor zpracoval a vyznačené souborové systémy automaticky připojil.

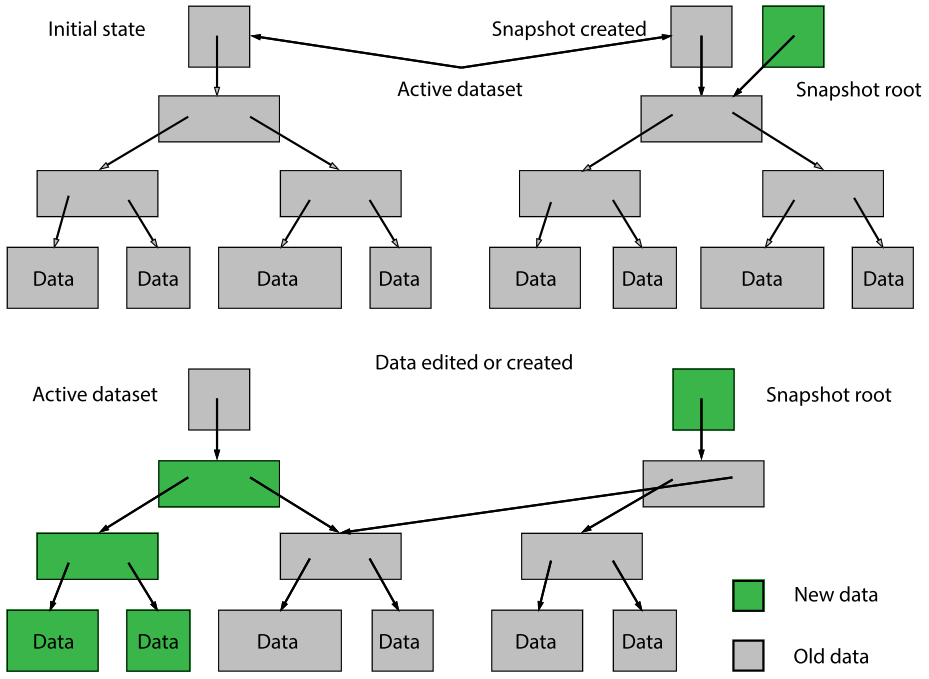
Pokud by administrátor chtěl využívat tradičního připojování pomocí souboru */etc/vsfstab*, je možné nastavit vlastnost *mountpoint* souborového systému na hodnotu *legacy* [13]. Tím administrátor řekne, že se o souborový systém bude starat sám a ZFS ho již automaticky připojovat nebude.

3.10 Snapshot

ZFS snapshot je stav daného souborového systému v čase, kdy byl snapshot pořízen. Jedná se o read-only instanci daného souborového systému [14].

Díky stromové architektuře souborového systému ZFS je pořizování snapshotů velice elegantní a nenáročné. Pro vytvoření můžeme použít příkaz `zfs snapshot filesystem@snapshot`, kde *filesystem* je jméno souborového systémů, jehož stav chceme zachytit a *snapshot* je pojmenování konkrétního stavu tohoto systému.

Po provedení příkazu si ZFS zapamatuje odkaz na vrchol souborového systému, jehož snapshot tvoříme. Tento odkaz se stane hlavním kořenem snapshotu jakožto nového souborového systému. Jelikož snapshot je souborový



Obrázek 3.7: Vytváření snapshotu

systém určený pouze ke čtení, nemůžeme pomocí něj změnit data v aktivním souborovém systému. Po vytvoření snapshotu je jeho velikost nulová, protože se provedlo pouze zapamatování odkazu a nebyly vytvořeny žádné kopie. Pokud se ovšem data v aktivním souborovém systému změní, je nutné, aby snapshot stále odkazoval na stará data. Toho docílíme tak, že stará data zkopírujeme a zvětšíme tak velikost snapshotu [14]. Velikost snapshotu tedy závisí na množství změněných dat od chvíle, kdy byl vytvořen. Proces vytváření snapshotu a rozšiřování jeho velikosti je názorně ukázán na obrázku 3.7.

Rekurzivní tvorba snapshotů se provádí rychle pomocí jedné atomické operace. Snapshoty jsou vytvořeny naráz všechny dohromady nebo nejsou vytvořeny vůbec. Výhodou atomické operace je fakt, že data ve všech snapshotech jsou konzistentní vzhledem k jednomu časovému okamžiku [14].

Snapshot pro souborový systém a všechny jeho potomky můžeme vytvořit a ověřit pomocí následujících příkazů:

```
# zfs snapshot -r rpool/export/home@snap
# zfs list -t snapshot -r rpool/export/home
NAME                                USED  AVAIL  REFER  MOUNTPOINT
rpool/export/home@snap                 0      -    33K  -
rpool/export/home/simactom@snap        0      -   641M  -
```

Druhý příkaz v pořadí ukazuje skutečnost, že snapshoty byly opravdu vytvořeny i pro všechny potomky daného souborového systému a že jejich velikost je opravdu nulová.

Snapshot se dá použít k navrácení souborového systému do stavu, v jakém se nacházel v okamžiku vytvoření snapshotu. Veškeré změny, které proběhly v souborovém systému od okamžiku vytvoření snapshotu budou smazány. K vyvolání tohoto procesu stačí použít příkaz `zfs rollback snapshot`, kde *snapshot* je celé jméno snapshotu. Jelikož se každý snapshot váže přímo s určitým souborovým systémem, není třeba zadávat jméno souborového systému, který chceme navrátit do původního stavu.

Kapitola 4

Analýza implementovaných řešení

V následující kapitole je rozebráno několik nástrojů sloužících pro monitorování a správu souborového systému ZFS. Hlavním předmětem analýzy jsou výhody a nevýhody implementovaných řešení. Výhody těchto nástrojů by mohly pomoci ke správnému návrhu vlastního řešení administračního nástroje pro ZFS.

4.1 zfsmon

V analýze implementovaných řešení je kladen důraz především na nástroj *zfsmon*. Tento nástroj slouží pro monitorování souborových systémů ZFS na více oddělených souborových serverech. Aplikace se dá rozdělit na dvě hlavní části, které spolu spolupracují.

První částí je skript, který běží na počítači se souborovým systémem ZFS. Tento skript je spouštěn pomocí *crontabu* a zajišťuje aktualizaci informací v databázi, která je součástí webové aplikace. V UNIXových operačních systémech *crontab* zajišťuje automatické spouštění programů v určitých intervalech. Doporučený interval pro spouštění skriptu nástroje *zfsmon* je 15 minut [15].

Druhou částí aplikace *zfsmon* je samotná webová aplikace, která se stará o zobrazování dat z databáze. V databázi se nacházejí informace o stavu jednotlivých souborových systémů z různých serverů. Aplikace zobrazuje tyto informace klientovi v podobě HTML stránek.

Velkou výhodou této aplikace je, že dokáže zobrazovat informace z více oddělených úložišť najednou. Na druhou stranu způsob zpracování dat přináší značnou nevýhodu, která spočívá v intervalu aktualizace dat v databázi. V průběhu tohoto intervalu aplikace znemožňuje uživateli vidět aktuální data ze souborového serveru. Uživatel je nucen počkat na konec tohoto intervalu,

kdy jsou opět nahrána aktuální data do databáze. Další nevýhodou tohoto nástroje je absence veškeré funkcionality, která by se týkala administrace souborového systému ZFS.

Tento nástroj nám tedy umožňuje kvalitně monitorovat informace z více zdrojů najednou, ale neposkytuje nám možnost administrace monitorovaných souborových systémů.

4.2 zfswatcher

Druhým nástrojem, který slouží pro monitorování souborového systému ZFS, je nástroj jménem *zfswatcher*.

Tento nástroj slouží jako webové rozhraní pro sledování souborového systému ZFS. Stejně jako předchozí nástroj zpracovává periodicky data a zobrazuje je pomocí HTML stránek, s čímž souvisí stejný problém, který se vyskytuje u předchozího nástroje. Oproti nástroji *zfsmon* umí zobrazovat informace pouze z jednoho souborového serveru. Umí však poslat upozornění v podobě e-mailu nebo zalogování do logu vždy, když dojde ke změně stavu ZFS [16].

4.3 smcwebserver

Na závěr této kapitoly zmíníme ještě nástroj *smcwebserver*. Tento nástroj je součástí operačního systému Solaris 10 6/06 a umožňuje administrátorovi spravovat některé funkce tohoto operačního systému. Mimo jiné se část tohoto nástroje věnuje právě správě souborového systému ZFS [17]. Bohužel v novější verzi Solarisu 11 se tento nástroj již nevyskytuje. Jako jediný ze zmíněných nástrojů právě *smcwebserver* nabízí administrátorovi funkce pro správu souborového systému ZFS.

Kapitola 5

Návrh řešení

Následující kapitola popisuje návrh nástroje pro administraci souborového systému ZFS. Zaměřuje se především na architekturu a bezpečnostní opatření navrhované aplikace.

5.1 Požadavky

Na začátku bychom měli stanovit funkcionalitu, která bude od aplikace vyžadována. Ještě jednou si připomeneme, že hlavním cílem této práce je vytvořit graficky orientovaný administrační nástroj pro správu souborového systému ZFS.

Souborový systém ZFS byl původně integrován do operačního systému Solaris. I přes to, že dnes již existují operační systémy, na které bylo ZFS přeneseno, hlavní platformou aplikace bude právě operační systém **Solaris**.

Jelikož má nástroj být grafického charakteru, můžeme brát jako první požadavek na aplikaci přítomnost **uživatelského rozhraní**.

Jak bylo zmíněno v úvodu, účelem toho nástroje má být ulehčení a zpřehlednění správy souborového systému ZFS a to i pro začínajícího administrátora. Bylo by tedy vhodné, aby zvolené uživatelské rozhraní bylo **přehledné a jednoduché** na používání.

ZFS je souborový systém určený především pro servery. Tyto počítače mohou být v mnohých situacích pro administrátora fyzicky nedostupné. Abychom umožnili administrátorovi správu i v těchto situacích, bude náš nástroj poskytovat možnost **vzdáleného přístupu** ke správě ZFS.

Na konci předchozí kapitoly jsme se dozvěděli, že většina nástrojů využívá takový způsob sběru dat, který uživateli v některých situacích znemožňuje vidět přímo aktuální data. Tohoto problému bychom se při návrhu měli vyvarovat a zvolit takový způsob, který by uživateli zobrazil **aktuální data** při každém jeho požadavku.

Další nedostatek objevený při analýze již vytvořených nástrojů pro správu ZFS, je ten, že většina těchto nástrojů vůbec neposkytuje funkce pro adminis-

traci. Dovolují nám tedy daný souborový systém sledovat, ale ne ho spravovat. Jelikož je ZFS rozsáhlý souborový systém, obsahuje i velké množství funkcí a příkazů pro jeho správu. Bylo by tedy dobré některé **základní funkce** a příkazy pro administraci ZFS do tohoto nástroje zahrnout.

Posledním bodem požadavků by měla být **bezpečnost**. Hlavním důvodem proč se starat o bezpečnost aplikace je fakt, že pomocí ZFS lze jednoduchým příkazem zničit vše, co se na disku nachází. Jelikož se z disku načítá při startu i operační systém, mohly by následky být fatální. Z tohoto důvodu bude přístupná pouze specifikovaným uživatelům.

Pro shrnutí můžeme požadavky na administrační nástroj shrnout do následujících bodů:

- **Solaris**
- **Uživatelské rozhraní**
- **Jednoduchost a přehlednost**
- **Vzdálený přístup**
- **Aktuálnost dat**
- **Základní funkce pro administraci**
- **Bezpečnost**

Pokud splníme všechny stanovené požadavky, můžeme uživateli zajistit bezpečnou správu souborového systému ZFS, která mu bude poskytovat aktuální informace.

5.2 Možnosti uživatelských rozhraní

Pro návrh uživatelského rozhraní aplikace přicházejí v úvahu následující možnosti:

- Příkazová řádka
- Grafické rozhraní
- Webové rozhraní

5.2.1 Příkazová řádka

Každý uživatel, který se někdy nějakým způsobem setkal s administrací UNIXových operačních systémů, by měl mít alespoň nějaké povědomí o tom, co je to příkazová řádka. Příkazová řádka je v UNIXu nástroj, který umožňuje využívat různé příkazy. Je rychlý, nenáročný a jediné co potřebujete vědět je, jaký příkaz napsat a co daný příkaz udělá. Vše se obejde bez myši a klikání.

Na druhou stranu může být pro začínajícího uživatele složité si všechny příkazy pamatovat a nějakou dobu trvá, než získá potřebné zkušenosti s tímto nástrojem. Pokud uživatel příkazy příliš neovládá, je to pro něj spíše zpomalení práce než zrychlení.

Protože jsme si v požadavcích stanovili, že náš nástroj má být jednoduchý, přehledný a uživatelsky příjemný i pro začínající uživatele, je tato volba nevhodná.

5.2.2 Grafické rozhraní

Jelikož jsme zavrhli příkazovou řádku, je třeba přijít s něčím, co bude pro uživatele přívětivé. Každý uživatel počítače se již jistě setkal s pojmem grafické rozhraní. Pokud ne, pod pojmem grafické rozhraní si můžeme představit například některé programy z operačního systému Windows. Prakticky všechno v operačním systému Windows má svoje grafické rozhraní, aby měl uživatel přehled o tom, co může v daný okamžik využívat za funkce.

Grafické rozhraní nám pomůže vyřešit problém s uživatelským komfortem. Dále nám podle požadavků zbývá zvážit možnost vzdáleného přístupu. Ano i v tomto případě bychom byli schopni tento požadavek splnit. Jde o to, jak by to pro nás bylo pohodlné. Aplikace by musela být nainstalovaná na každém počítači, ze kterého bychom chtěli ZFS administrovat. Tento problém není nepřekonatelný, ale existuje ještě jedno komfortnější a dnes stále populárnější řešení.

5.2.3 Webové rozhraní

Nejkomfortnějším řešením je použití webového rozhraní. Hlavní výhodou tohoto řešení je fakt, že pro přístup k aplikaci potřebujeme jen webový prohlížeč. Ten je dnes přítomný v drtivé většině operačních systémů. Tato skutečnost nám přináší možnost přistupovat k aplikaci z webového prohlížeče v libovolném operačním systému, přestože aplikace poběží na operačním systému Solaris.

Webové stránky jsou navíc známé pro většinu uživatelů, a tak by při správném návrhu stránek neměl být sebemenší problém s jednoduchostí a přehledností tohoto nástroje.

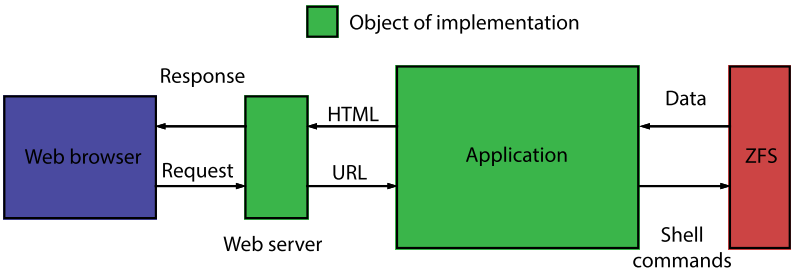
Zbytek požadavků jako je bezpečnost jednoduše splníme pomocí správné implementace tohoto řešení.

5.3 Architektura aplikace

V předchozí kapitole jsme pro náš nástroj zvolili možnost zobrazování dat pomocí webového rozhraní. Toto rozhraní se bude skládat z webových stránek, které budeme chtít uživateli zobrazovat. Pro tento účel si budeme muset zvolit

nebo implementovat nějaký webový server, který bude zprostředkovávat komunikaci mezi naší aplikací a klientským prohlížečem. Jakmile uživatel požádá pomocí svého prohlížeče o nějakou stránku, server tento požadavek obdrží a předá ho naší aplikaci. Aplikace ho vyhodnotí a prostřednictvím webového serveru pošle uživateli odpovídající odpověď.

Náš nástroj má sloužit pro správu souborového systému ZFS a tak bude nutné vymyslet, jak bude naše aplikace s tímto systémem komunikovat. Administrátor může ZFS spravovat pomocí příkazů, které jednoduše zadá do příkazové řádky. Bude tedy nutné vymyslet vrstvu aplikace, která bude tyto příkazy sestavovat a následně provádět. Na obrázku 5.1 můžeme vidět graficky znázorněnou komunikaci mezi hlavními komponenty aplikace.



Obrázek 5.1: Struktura aplikace

5.3.1 Architektura MVC

Nyní navrhneme řešení, jak bude aplikace fungovat uvnitř. Jak již bylo zmíněno, náš nástroj bude umožňovat správu souborového systému ZFS pomocí jeho základních funkcí. Tudíž ne všechny funkce budou ve výsledné implementaci zahrnuty. Z tohoto důvodu je aplikaci nutno navrhnout tak, aby se dala jednoduše rozšiřovat o potřebnou funkcionalitu.

K tomuto účelu použijeme objektový návrh společně s architekturou MVC. Model-view-controller (MVC) je softwarová architektura, která rozděluje aplikaci do tří vrstev tak, že jsou na sobě minimálně závislé [18]. Pokud bychom chtěli naši aplikaci rozšířit o novou funkcionalitu, jednoduše stačí rozšířit požadovanou vrstvu.

Komponenty jednotlivých vrstev aplikace rozdělujeme na `model`, `view` a `controller`.

5.3.1.1 Datová vrstva

`Model` je označení pro třídu, která je součástí datové vrstvy aplikace. Úkol této třídy, stejně jako i celé datové vrstvy, je manipulace a práce s daty. Všechny

datové struktury si ZFS spravuje sám a my k nim nemáme přímý přístup. Naše datová vrstva a její třídy nám tedy budou zajišťovat vykonávání ZFS příkazů, které budou sestaveny na základě dat předaných z logické vrstvy.

5.3.1.2 Prezentační vrstva

Další vrstvou aplikace je tzv. prezentační vrstva. Tato vrstva se skládá ze tříd nesoucích obecný název `view`. Hlavním úkolem této vrstvy je vykreslování dat uživateli. V našem případě aplikace poběží na počítači se souborovým systémem ZFS a uživatel se k ní bude připojovat pomocí webového prohlížeče. Naše prezentační vrstva tedy nebude přímo vykreslovat uživatelské rozhraní, protože neví kam. Pro zachování nezávislosti jednotlivých vrstev, bude prezentační vrstva pouze generovat HTML kód stránky z dat, které jí budou předány. Následně takto vygenerovanou stránku předá zpět do logické vrstvy, která zařídí odeslání stránky uživateli.

5.3.1.3 Logická vrstva

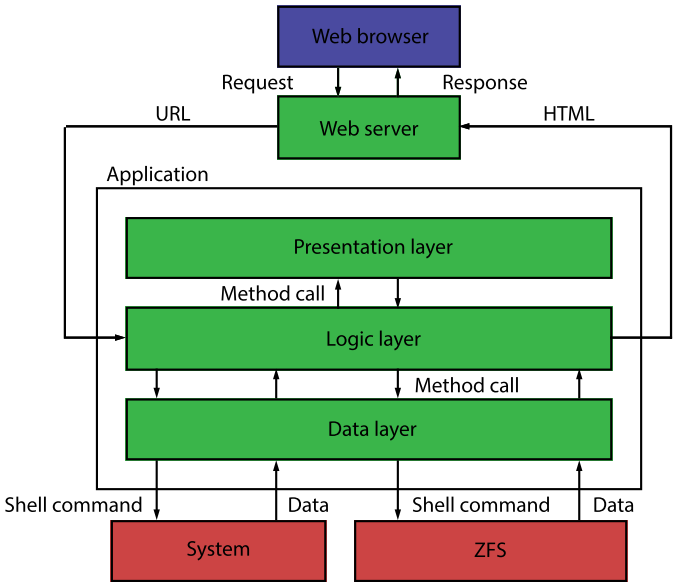
Poslední vrstvou aplikace je logická vrstva. Hlavním úkolem této vrstvy je řízení toku dat mezi datovou a prezentační vrstvou. V okamžiku, kdy webový server obdrží nějaký požadavek od uživatele, aplikace vybere z logické vrstvy třídu, která dokáže tento požadavek vyřídit. Tato třída, kterou obecně nazýváme `controller`, může požádat datovou vrstvu, aby provedla potřebné operace se souborovým systémem nebo od souborového systému získala potřebná data. Následně data převede do potřebného formátu a požádá prezentační vrstvu o jejich vykreslení. V našem případě vykreslená data opět převezme a odešle klientovi.

Komunikace mezi jednotlivými vrstvami aplikace je znázorněna na obrázku 5.2. Na obrázku je také vidět způsob předávání dat mezi webovým serverem a aplikací.

5.4 Interakce s aplikací

K aplikaci bude administrátor moci přistupovat pomocí webového prohlížeče. Tato výhoda pramení ze způsobu přenosu webových stránek pomocí protokolu HTTP, který je pro tento účel určený. Jeho definici můžeme najít v RFC dokumentu [19]. Tento protokol je postavený na architektuře klient-server, kdy klient posílá požadavky na webový server a ten klientovi odpovídá v podobě HTML dokumentu. Jak již bylo zmíněno, výhodou tohoto způsobu je oddělenost serveru a klienta.

Uživatel bude pomocí webového prohlížeče vznášet požadavky na webový server v podobě URL. Uniform resource locator (URL) se používá pro lokalizování zdrojů na internetu poskytnutím jedinečného řetězce znaků [20]. HTTP



Obrázek 5.2: Architektura aplikace

požadavek, který uživatel bude pomocí webového prohlížeče odesílat, obsahuje právě tento URL řetězec. Tím administrátor specifikuje jakou stránku chce zobrazit popřípadě jakou akci chce povést.

Naši aplikaci tedy postavíme tak, aby byla schopná přijmout URL řetězec a na základě jeho struktury provedla určitou akci.

5.5 Bezpečnost

Další důležitou kapitolou v procesu návrhu administrátorského nástroje je bezpečnost.

Některé příkazy poskytované souborovým systém ZFS jsou dostupné každému uživateli v systému. Jedná se hlavně o příkazy, které vypisují nejrůznější informace o stavu souborových systémů a poolů. Prováděním těchto příkazů se v nastavení ZFS nic nezmění, a proto nemůže dojít k poškození nebo ztrátě dat. Na druhou stranu ZFS nabízí příkazy, které běžní uživatelé nemohou využívat. K provádění těchto příkazů je potřeba oprávnění *root* uživatele. Jedná se zejména o příkazy, které vytvářejí, mění nebo ničí souborové systémy či pooly. Toto chování je naprosto pochopitelné, protože nesprávným nebo neuváženým použitím těchto příkazů může dojít ke zničení cenných dat nebo celého operačního sytému.

Našim cílem je vytvořit nástroj pro správu souborového systému ZFS. Je tedy zřejmé, že budeme potřebovat, aby naše aplikace dokázala využívat i pří-

kazy, které nejsou běžným uživatelům dostupné. V operačním systému Solaris, kde naše aplikace poběží, dostává každý spuštěný proces tzv. identitu. Tato identita spojuje běžící proces s uživatelem, pod kterým byl proces spuštěn. Díky tomu je operační systém schopen určit, jaká práva proces má a co vše může v systému využívat. Pokud bychom spustili naší aplikaci pod uživatelem, který nemá právo využívat některé příkazy souborového systému ZFS, pak by je nemohl využívat ani náš nástroj. Tento fakt nás tedy vede k tomu, abychom naší aplikaci spouštěli pod uživatelem, který má právo vykonávat všechny ZFS příkazy.

Máme tedy aplikaci, která může v systému provádět potřebné příkazy a je dostupná na nějaké URL adrese. Problém přístupu neoprávněných uživatelů k nebezpečným příkazům byl vyřešen na úrovni systému a jednoduše nemohli nebezpečné příkazy spouštět. Nyní jsme aplikaci spustili pod uživatelem, který tato práva vlastní a navíc má tato aplikace veřejné webové rozhraní, které je dostupné na stanové URL adrese. Každý kdo tuto adresu zná, může v tuto chvíli pomocí webového prohlížeče k aplikaci přistoupit a provádět jejím prostřednictvím nebezpečné příkazy. To není úplně ideální případ. My bychom chtěli, aby k aplikaci mohli přistupovat jenom oprávnění uživatelé pomocí přihlašovacího jména a hesla. K tomuto použijeme vlastnost HTTP protokolu Basic Authentication, která nám zajistí přístup jen pro autentizované uživatele.

Aplikace bude přístupná jen pro oprávněné uživatele, kteří se autentizují pomocí metody Basic Authentication. To s sebou přináší další starosti, protože uživatelské jméno a heslo musíme ověřit. Tyto citlivé informace pomocí HTTP protokolu putují přes počítačovou síť až ke koncovému webovému serveru. Podle specifikace [19] HTTP protokol neumožňuje šifrování. To znamená, že potencionální útočník může na síti odposlechnout požadavek, který obsahuje uživatelské jméno a heslo a použít tyto údaje pro vstup do aplikace. Z tohoto důvodu budeme přenos dat šifrovat pomocí HTTPS protokolu.

5.5.1 Systémový uživatel

Z důvodů uvedených výše budeme potřebovat vhodného uživatele, pod kterým budeme tuto aplikaci spouštět. V úvahu přicházejí následující možnosti:

- **uživatel *root***
- **role (RBAC)**

První možností je zvolit **uživatele *root***. Tento uživatel má právo na všechny operace v systému. Tato volba by sice náš problém vyřešila, ale přinesla by další bezpečnostní rizika. Pokud bychom spustili aplikaci s identitou uživatele *root*, pak by tato aplikace mohla v systému provádět všechny příkazy. To by bylo pro naši aplikaci zbytečné. Nám bude stačit, pokud aplikace bude moci využívat přesně ty nástroje, které ke své správné funkčnosti potřebuje.

Dalším rizikem je fakt, že by potenciální útočník mohl tato práva nějakým způsobem zneužít.

Druhá možnost, která přichází v úvahu, je použití tzv. **RBAC**. Jedná se o rozšíření bezpečnostního modelu, které nám umožňuje kontrolovat přístup uživatelů k úkonům přístupným pouze pro superuživatele *root*. V nerozšířeném bezpečnostním modelu jste buď uživatelem s omezenými možnosti a nebo jste superuživatel a můžete všechno [21]. Ve zkratce RBAC umožňuje vytváření tzv. rolí, které se chovají téměř jako normální uživatel. Rozdíl je v tom, že na roli se nedá přihlásit přímo. Do systému se nejprve musí přihlásit klasický uživatel, který si poté smí přiřadit určité role. Rolím jsou přiřazovány bezpečnostní profily, které obsahují jednotlivá privilegia.

Pro bezpečnost našeho nástroje si tedy vytvoříme roli, které přiřadíme potřebná oprávnění pro vykonávání následujících příkazů:

- *zfs*
- *zpool*
- *format*
- *fdisk*

Aplikaci budeme spouštět pod touto rolí. Ve výsledku aplikace bude mít veškerá oprávnění pro svoji správnou činnost a bezpečnostní rizika budou minimální.

5.5.2 HTTP Basic Authentication

Basic Authentication je způsob autentizace uživatelů pomocí protokolu HTTP. Pokud webový server vyžaduje tuto metodu ověření, uživatel je před přístupem k obsahu vyžádán o autentizaci pomocí jména a hesla. Jméno a heslo je pak spolu s požadavkem odesláno webovému server k ověření [22].

Pro naší aplikaci to znamená, že webový server bude muset umět ověřit uživatele oproti lokální databázi. Pokud se uživatelské jméno a heslo odeslané webovým prohlížečem bude shodovat s uživatelskými údaji v lokální databázi, bude uživateli umožněn vstup do aplikace. Do všech částí aplikace bude umožněn vstup pouze autentizovaným uživatelům.

5.5.3 HTTPS

HTTP protokol nám neumožňuje nijak šifrovat data, která jsou přenášena. Jelikož jsou po síti přenášena citlivá data jako je uživatelské jméno a heslo, musíme zajistit šifrování tohoto přenosu. HTTPS je protokol, který nám to umožní. Tento protokol ve svém základu využívá ke komunikaci HTTP protokol, ale rozdíl je v tom, že přenášená data jsou šifrována pomocí SSL nebo TSL [23].

Použití protokolu HTTPS zajistí naší aplikaci, že bude pro útočníka téměř nemožné získat uživatelské údaje, které byli použity pro vstup do aplikace.

5.5.4 Shrnutí

Pro shrnutí zopakujeme, jaké bezpečnostní opatření budou provedena pro zajištění aplikace. Vytvoříme systémovou roli pomocí RBAC, která bude mít privilegia k vykonávání potřebných operací. Výsledná aplikace bude spouštěna pod touto rolí. K autentizaci uživatelů, kteří budou přistupovat k aplikaci, použijeme metodu Basic Authentication protokolu HTTP. Uživatele budeme ověřovat proti lokální databázi uživatelů. A konečně celý přenos dat mezi webovým serverem a klientem zašifrujeme pomocí protokolu HTTPS.

5.6 Integrace do systému

Pro ulehčení správy celé aplikace ji zaregistrujeme v operačním systému Solaris jako službu. Docílíme tím toho, že se operační systém bude sám starat o zapínání, vypínání a restartování celé aplikace. Samozřejmě, že budeme stále schopni explicitně říci, jestli se má daná aplikace spouštět.

Service Management Facility (SMF) je v operačním systému Solaris nástroj pro spravování služeb. Nahrazuje tím tradiční způsob spravování služeb pomocí *init* skriptů, který byl běžný v dřívějších verzí operačního systému Solaris a ostatních UNIXových operačních systémech. Velkou výhodou SMF je možnost definování závislostí dané služby na ostatní službách. Tím dokážeme stanovit, že se daná služba nespustí, dokud nebudou spuštěny všechny služby stanovené v závislostech. SMF také uchovává veškeré informace o startu, běhu i ukončování služby v logu. Administrátor má tedy kdykoli možnost tyto logy prozkoumat a získat požadované informace. V pozadí každé služby zaregistrované v SMF je tzv. manifest [24].

5.6.1 Manifest

Manifest je XML dokument, který danou službu popisuje. Právě zde administrátor stanovuje, na kterých službách bude tato služba závislá nebo jak se daná služba jmenuje. V našem případě to znamená, že si budeme muset tento manifest vytvořit a zaregistrovat službu v SMF.

5.6.2 Metody

Důležitou součástí manifestu jsou metody, které SMF říkají, jak má danou službu zapnout nebo vypnout. V těchto metodách se může spouštět více procesů, které jsou svázány dohromady jako součást dané služby. Administrátor se pak stará jenom o službu jako takovou a nemusí vůbec vědět, jaké procesy byly v rámci služby spuštěny [24]. Mimo jiné můžeme v manifestu určit tzv.

kontext metod. Tato vlastnost nám dovoluje určit pod jakým uživatelem mají být jednotlivé metody vykonávány. V kontextu naší aplikace to znamená, že metody služby budou spouštěny pod námi vytvořenou rolí, která bude mít dostatečná práva.

5.6.3 Shrnutí

Zaregistrování aplikace jako služby v operačním systému Solaris nám přinese následující výhody pro správu této aplikace. Administrátor bude mít jasný přehled nad tím, zda je daná služba spuštěna. Dále bude moci jednoduše danou službu spustit popřípadě zastavit. Pokud dojde k nějakému problému při spuštění dané služby, SMF administrátorovi poskytne informace o zdroji tohoto problému. V poslední řadě SMF také umožní automatické spuštění služby při startu systému.

Jako pracovní název aplikace zvolíme *wzfsadm*, který vznikl spojením slov web, ZFS a administrace. Z názvu je jasně patrné, že nástroj bude sloužit pro webovou administraci souborového systému ZFS.

Kapitola 6

Implementace

Obsahem následující kapitoly je popis vlastní implementace nástroje pro správu ZFS. Pozornost je věnována především spolupráci jednotlivých vrstev aplikace, stručnému popisu hlavních tříd a integraci aplikace do operačního systému Solaris.

6.1 Python

Než začneme s implementací, je nutné si ujasnit v jakém jazyce budeme aplikaci psát.

V operační systému Solaris interpretuje příkazovou řádku shell. ZFS je součástí Solarisu a shell nám dovoluje využívat jeho rozhraní, které je dostupné právě z příkazové řádky. Z počátku se tedy možnost skriptování v shellu zdála jako dobrá volba. Nicméně shell nám nedovoluje využívat výhody objektového programování, protože nepodporuje třídy. To je v rozporu s naším návrhem, který využívá objektové architektury MVC.

Proto sáhneme po skriptovacím jazyku Python, který nám umožní implementaci objektově orientované architektury MVC.

6.2 HTTP Server

Při návrhu aplikace jsme zvolili webové rozhraní. Pro implementaci tohoto řešení budeme muset zvolit nějaký webový server, který nám bude zprostředkovávat komunikaci mezi naší aplikací a webovým klientem (prohlížečem). Od našeho webového serveru budeme požadovat podporu HTTPS protokolu a možnost autentizace uživatelů pomocí metody Basic protokolu HTTP.

6.2.1 Implementované řešení

Mezi nejznámější a nejrozšířenější zástupce webových serverů patří například *Apache*. Tato komplexní implementace webového serveru by jistě dokázala

splnit všechny požadavky, které požadujeme. Nevýhodou této volby je však zbytečná obsáhlost a nutnost složitější konfigurace. Z tohoto důvodu zvolíme vlastní řešení webového serveru, které bude přesně odpovídat požadavkům aplikace.

6.2.2 Vlastní řešení webového serveru

Implementace vlastního řešení webového serveru bude využívat dvou standardních knihoven jazyka Python a bude se skládat z následujících tříd:

- `WzfsadmServer`
- `WzfsadmRequestHandler`
- `Authenticator`

Součástí těchto tříd bude i implementace navržených bezpečnostních opatření.

6.2.2.1 Třída WzfsadmServer

Třída `WzfsadmServer` reprezentující vlastní webový server, bude potomkem tříd `TCPServer` a `ThreadingMixIn`, které jsou součástí standardní knihovny `SocketServer` [25] jazyka Python. To znamená, že zdědí metody obou rodičovských tříd a bude tyto metody moci využívat. Dále tato třída bude implementovat šifrování komunikačního kanálu pomocí knihovny `ssl`.

Hlavním úkolem této třídy bude poslouchat příchozí spojení na předem stanoveném síťovém rozhraní a portu. V okamžiku připojení klienta k serveru dojde k vytvoření instance třídy `WzfsadmRequestHandler`, která bude implementovat hlavní komunikaci pomocí HTTP protokolu. Na této úrovni jsme schopni filtrovat IP adresy, které se k serveru budou moci připojovat. V hlavním konfiguračním souboru serveru budeme schopni stanovit seznam IP adres, které se serverem budou moci komunikovat.

Aby bylo možné využít šifrování pomocí knihovny `ssl`, budeme muset vytvořit privátní klíč pro šifrování a certifikát, kterým se server bude identifikovat. Tuto dvojici můžeme vytvořit například pomocí nástroje `openssl`. Cestu k souborům s klíčem a certifikátem můžeme explicitně určit v hlavním konfiguračním aplikace.

Ke spuštění serveru stačí vytvořit instanci třídy `WzfsadmServer`, které předáme port a adresu síťového rozhraní a následně na této instanci zavolat metodu `serve_forever()`. Od této chvíle bude server naslouchat na stanovené adrese a portu dokud nebude ukončen.

6.2.2.2 Třída WzfsadmRequestHandler

Hlavní součástí webového serveru je zmíněná třída `WzfsadmRequestHandler`. Tato třída je potomkem třídy `BaseHTTPRequestHandler`, která je součástí standardní knihovny `BaseHTTPServer` a zajišťuje komunikaci pomocí HTTP protokolu. Součástí této třídy bude i implementace autentizační metody Basic.

Při ověřování požadavku ověříme, jestli klient poslal autentizační údaje. Na všechny požadavky, které nebudou obsahovat hlavičku *Authorization* se správnými uživatelskými údaji, odpovíme HTTP kódem 401(Unauthorize). K ověření validity uživatelských údajů budeme využívat třídu `Authenticator`.

V případě, že požadavek úspěšně projde procesem autentizace, dojde ke zpracování požadované HTTP metody. Jelikož pro účely naší aplikace budou stačit prostředky, které nabízí metoda GET, ostatní metody nebude náš webový server podporovat. V případě, že klient pošle požadavek na jinou HTTP metodu, server odpoví kódem 501 (Not Implemented).

Uvnitř metody, která reprezentuje HTTP metodu GET, server ověří, zda požadovaná URL odpovídá souboru. Pokud URL odpovídá nějakému souboru, je jeho obsah odeslán klientovi a volání metody ukončeno. V opačném případě server vytvoří instanci třídy `App` reprezentující naší aplikaci, které předá URL požadovanou klientem. Aplikace požadavek zpracuje a server v konečné fázi odešle výsledek zpět uživateli.

6.2.2.3 Třída Authenticator

Poslední třídou, která přímo souvisí s funkcionalitou webového serveru, je třída `Authenticator`. Tato třída má za úkol zkontrolovat validitu uživatelského jména a hesla, které přišli spolu s požadavkem. Tyto uživatelské údaje ověří oproti lokální databázi uložené v předem určeném souboru s danou strukturou.

Každá řádka tohoto souboru bude reprezentovat uživatele, který se k aplikaci bude moci připojit. Formát dat uložených v souboru bude následující. Řádka se bude skládat ze tří sloupců oddělených dvojtečkou. V prvním sloupci bude uloženo uživatelské jméno. Ve druhém sloupci bude následovat jméno hašovací funkce použité při tvorbě haše uživatelského hesla a konečně v posledním sloupci pak bude hexadecimální reprezentace tohoto haše. Soubor bude postupně procházen a uživatelské údaje porovnány. Pokud nebude nalezena shoda, autentizace uživatele bude vyhodnocena jako chybná.

6.3 Směrování

Druhou částí aplikace bude logická část, kde bude docházet ke zpracovávání jednotlivých požadavků. Tato část bude implementována pomocí MVC architektury, která nám pomůže oddělit logiku aplikace od vrstvy, která bude interagovat se souborovým systémem ZFS. Po zpracování požadavku dojde

k vygenerování HTML stránky, která bude prostřednictvím webového serveru odeslána uživateli.

Aby mohlo dojít k vygenerování požadované stránky, budeme muset z URL rozpoznat, která akce se má vykonat. Vytvoříme směrovač, který bude mapovat URL adresy na metody tříd logické vrstvy. Pro tento účel stanovíme následující pravidla, podle kterých budeme adresu interpretovat.

Z hlavní části adresy určíme jméno třídy a metody, kterou budeme volat. Parametry určené v URL adrese budou předány volané metodě. Pro ukázku požadovaná adresa, kterou nám předá web server, může vypadat následovně `/zpool/detail?pool=rpool`. Při požadavku na tuto adresu dojde k vytvoření instance třídy `zpool`, na které se pokusíme zavolat metodu `detail()` s dostupnými parametry. Tato metoda bude zobrazovat detailní přehled informací o poolu jménem *rpool*.

6.3.1 Třída App

Princip směrování požadavků k jednotlivým třídám v naší aplikaci zajišťuje třída `App`. Třída zpracuje přijatou URL způsobem popsaným v kapitole 6.3, což nám zajistí jméno třídy logické vrstvy aplikace a jméno metody, kterou máme na této třídě zavolat. Pokud požadovaná třída a metoda neexistují, je klientovi vrácena odpověď s kódem 404 (Not found). V opačném případě je třída dynamicky načtena z předem stanoveného adresáře a následně vytvořena její instance.

Hlavním účelem třídy `App` je tedy zpracovat URL adresu a následně vytvořit třídu, která se postará o zpracování požadavku.

6.4 Datová vrstva

Datová vrstva je v terminologii MVC architektury vrstva, která pracuje s daty. Naše aplikace si přímo žádná data držet nebude, protože ZFS si všechny datové struktury spravuje sám. Datová vrstva naší aplikace bude tedy zprostředkovávat komunikaci a tok dat mezi ZFS a logickou vrstvou aplikace. Třídy této vrstvy budou využívat rozhraní ZFS. Metody těchto tříd na základě dat obdržených z logické vrstvy sestaví potřebný ZFS příkaz a pomocí shellu ho vykonají na příkazové řádce. Výsledek operace a požadovaná data pak předají zpět do logické vrstvy, kde se zpracují.

Jelikož nástroj nebude implementovat všechny funkce souborového systému ZFS, bylo by vhodné navrhnout tuto vrstvu způsobem, který by umožňoval jednoduché rozšíření její funkcionality. Z tohoto důvodu rozdělíme datovou vrstvu do modulů, které se budou specializovat na určitou oblast administrace ZFS. V každém z modulů bude hlavní třída, která bude poskytovat logické vrstvě rozhraní pro komunikaci se souborovým systémem. V okamžiku, kdy logická vrstva bude chtít komunikovat se souborovým systémem, bude požadovaný modul (jeho hlavní třída) dynamicky načten do aplikace.

```
app ..... Adresář v kořenovém adresáři aplikace
└── model ..... Adresář datové vrstvy aplikace
    └── modules ..... Adresář pro moduly
        └── SystemModule ..... Adresář modulu
            ├── __init__.py ..... Soubor pro inicializaci modulu
            └── SystemSource.py ..... Hlavní třída modulu
```

Obrázek 6.1: Struktura modulu

6.4.1 Třída ModuleInterface

Abychom zajistili třídám logické vrstvy jednotný přístup k metodám datové vrstvy, vytvoříme třídu `ModuleInterface`, která bude zajišťovat dynamické načítání tříd z datové vrstvy. Tato třída zkontroluje zda požadovaný modul existuje a splňuje požadavky stanovené v kapitole 6.4.2. V případě úspěchu daný modul načte.

Velkou výhodou toho přístupu je fakt, že po dobu zpracovávání požadavku můžeme načítat pouze moduly, které nutně potřebujeme k jeho zpracování. Dynamické načítání modulů nám tedy poskytne jistou úsporu ve využití operační paměti systému.

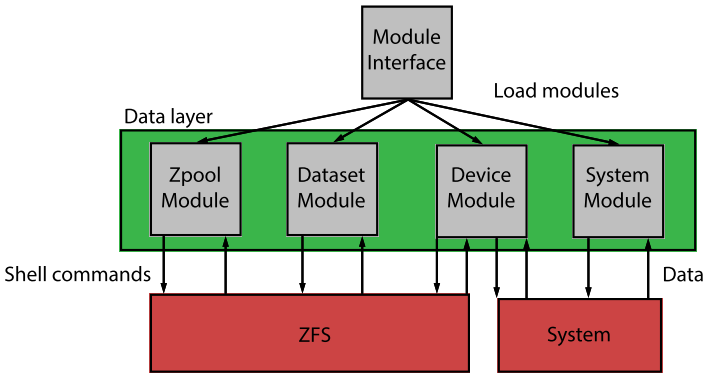
6.4.2 Struktura modulu

Všechny moduly a jejich součásti budou součástí adresářového stromu aplikace. Budou se nacházet v adresáři *app/model/modules* a jejich struktura bude odpovídat struktuře na obrázku 6.1.

Pro popis struktury adresáře modulu na obrázku 6.1 budeme používat modul *System*. Každý modul bude uložen ve svém adresáři, jehož jméno se bude skládat ze jména modulu a klíčového slova *Module*. V první řadě adresář modulu musí obsahovat soubor `__init__.py`. Tento soubor říká interpretu jazyka Python, že se jedná o modul a umožní nám ho jednoduše načítat.

Hlavní součástí modulu bude soubor se zdrojovými kódy. Název souboru bude opět odpovídat názvu modulu, ale tentokrát bude následovat klíčové slovo *Source*. V tomto souboru bude uložena definice hlavní třídy modulu, která bude logické vrstvě nabízet požadované funkce. Název této třídy se musí shodovat s názvem modulu. Obsahem adresáře modulu mohou být i soubory s definicemi vedlejších tříd nebo jiné pomocné skripty. V tomto případě si musí modul samostatně zajistit jejich načtení, protože třída `ModuleInterface` umožňuje načítat pouze hlavní třídy modulů.

Konfigurační soubory modulů budou standardně dostupné v adresáři */etc/wzfsadm*, kde se bude nacházet i hlavní konfigurační soubor aplikace. Každý modul, který bude využívat konfiguračních souborů, je sám odpovědný za jejich načtení a zpracování. Aplikace se stará pouze o načítání hlavního konfiguračního souboru.



Obrázek 6.2: Datová vrstva aplikace

6.4.3 Základní moduly

V základní verzi výsledné aplikace jsou implementované následující moduly:

- `ZpoolModule`
- `DeviceModule`
- `DatasetModule`
- `SystemModule`

Každý z těchto modulů implementuje metody, které umožňují práci s operačním systémem Solaris nebo specifickou částí souborového systému ZettaByte. Modul `PoolModule` obsahuje metody, které se týkají správy ZFS poolů. Umožňuje především shromažďovat informace o požadovaných poolech, vytvářet nové pooly nebo vytvořené pooly zničit. Hlavní součástí tohoto modulu je třída `Pool`, která shromažďuje a drží všechny informace o konkrétním ZFS poolu.

Druhý modul `DeviceModule` umožňuje manipulovat se zařízením uvnitř poolu. Doplňuje především funkcionalitu modulu `PoolModule` o možnosti přidávat zařízení do poolů, změnit stav konkrétního zařízení nebo provádět funkce *attach* a *detach*.

Nejobsáhlejším základním modulem je `DatasetModule`, který poskytuje základní metody pro správu jednotlivých souborových systémů. Tento modul umožňuje administrátorovi vytvářet souborové systémy uvnitř poolu a libovolně je vnořovat. Dále nabízí možnost tyto souborové systémy ničit, upravovat a vytvářet jejich snapshoty. Informace o souborových systémech shromažďuje a udržuje třída `Dataset`, kterou tento modul využívá.

Posledním implementovaným modulem je `SystemModule`. Tento modul slouží ke shromažďování informací o systémových prostředcích nebo pro získání informací o operačním systému.

Celková struktura datové vrstvy je představena na obrázku 6.2, kde můžeme vidět jednotlivé moduly. Na obrázku je také vidět jak jsou jednotlivé moduly načítány pomocí třídy `ModuleInterface`.

6.5 Logická vrstva

Hlavní vrstvou aplikace bude logická vrstva. Tato část bude mít za úkol provádět požadované akce na souborovém systému ZFS za pomoci ostatních vrstev aplikace. Data obdržená od datové vrstvy zpracuje a požádá prezentační vrstvu o vygenerování HTML stránky. Výsledek je prostřednictvím webového serveru odeslán uživateli, který si ho pomocí webového prohlížeče může zobrazit.

Třídy logické vrstvy se obecně nazývají kontroléry. Kontroléry budou obsahovat metody, které budou představovat jednotlivé akce nabízené uživateli. Na základě směrování popsaného v 6.3 dojde k dynamickému načtení kontroléru a vyvolání požadované metody. Volaná metoda přesně definuje, které moduly datové vrstvy budou načteny a jak obdržená data zpracuje.

6.5.1 Kontrolér

V adresářové struktuře aplikace opět stanovíme adresář, kde najdeme jednotlivé definice kontrolérů. V tomto případě veškeré zdrojové kódy tříd logické vrstvy najdeme v adresáři *app/controllers*.

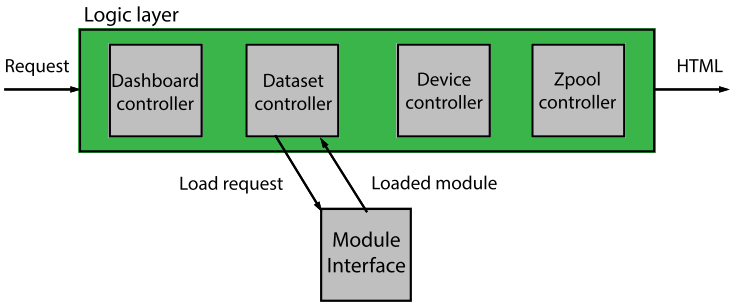
Právě odtud bude třída *App*, která se stará o směrování, dynamicky načítat požadované kontroléry. Stejně jako v případě datové vrstvy nám dynamické načítání umožní načítat pouze ten kontrolér, který potřebujeme ke zpracování daného požadavku a opět ušetříme operační paměť.

6.5.2 Základní kontroléry

Do výsledné aplikace budou zařazeny následující kontroléry:

- `DashboardConroller`
- `DatasetConroller`
- `DeviceConroller`
- `ZpoolConroller`

Třída `DashboardConroller` bude zajišťovat zobrazování úvodní stránky aplikace. Obsahem této stránky bude přehled základních informací týkajících



Obrázek 6.3: Logická vrstva aplikace

se souborového systému obecně, vytvořených poolů nebo například všech dostupných souborových systémů.

Stránky, které se týkají administrace souborových systémů v ZFS, bude řídit třída `DatasetConroller`. Tento kontrolér bude umožňovat zobrazit detailní informace o jednotlivých souborových systémech a bude také nabízet funkce pro jejich administraci. Nebude tedy chybět možnost souborové systémy ničit, vytvářet, nastavovat nebo vytvářet jejich snapshoty.

V případě kontrolérů `DeviceConroller` a `ZpoolConroller` se bude jednat o podobné funkce týkající se správy zařízení a ZFS poolů.

Výsledná aplikace bude generovat HTML stránky, které v sobě ponesou odkazy na metody výše zmíněných kontrolérů. Uživatel tedy vůbec nemusí znát strukturu aplikace a jednotlivé metody kontrolérů, protože mu budou nabídnuty prostřednictvím těchto stránek. Stránky mezi sebou budou logicky provázané tak, aby se uživatel mohl po aplikaci libovolně a pohodlně pohybovat.

Struktura logické vrstvy aplikace je představena na obrázku 6.3, kde můžeme vidět jednotlivé kontroléry. Na obrázku je také znázorněno načítání modulů datové vrstvy pomocí třídy `ModuleInterface`.

6.6 Prezentační vrstva

Poslední vrstvou aplikace je tzv. prezentační vrstva. Jediným úkolem této vrstvy je na základě obdržených dat vygenerovat požadovanou HTML stránku. Tato vrstva bude v naší aplikaci zastoupena jedinou třídou `BaseView`. Vzhledem k dynamické povaze dat, které budeme v aplikaci zobrazovat, využijeme šablonovací systém *Jinja2* [26], který je pro Python dostupný. Tento systém nám umožní vytvoření univerzálních šablon, které budou použity k zobrazování více různých stránek. Stránka zobrazující detail souborového systému se bude vždy skládat ze stejných komponentů a bude mít stejné rozložení bez ohledu na to, jaký souborový systém právě zobrazujeme. Šabloně předáme

data specifická pro konkrétní souborový systém a následně vygenerujeme výslednou HTML stránku.

6.6.1 Jinja2

Generování šablon ve třídě `BaseView` je zajištěno pomocí modulu *Jinja2*. Při vytváření instance této třídy dojde k inicializaci tohoto modulu a stanovení adresáře, ze kterého se budou šablony načítat. Tento adresář se v našem případě bude jmenovat *template* a bude se nacházet v kořenovém adresáři aplikace. Modul při každém požadavku na vykreslení šablony z tohoto adresáře dynamicky načte potřebnou šablonu. Ta je následně vyplněna daty a vrácena jako textový řetězec.

6.6.2 Google charts

Pro demonstraci dynamického zobrazování dat v HTML dokumentu využijeme jazyka JavaScript a technologie AJAX. Tato technologie nám umožní posílat HTTP požadavky z již načtené stránky v klientském webovém prohlížeči. V požadavku klasicky specifikujeme požadovanou URL adresu a odešleme ho webovému serveru. Server zpracuje požadavek klasickým způsobem a odpoví klientovi. V klientském prohlížeči data zpracujeme a dynamicky změníme část načtené stránky. Pro lepší práci s JavaScriptem budeme využívat knihovnu jQuery [27] a dynamická data budeme zobrazovat pomocí Google charts [28].

Součástí implementované verze aplikace je funkce pro dynamické zobrazování procentuálního využití procesoru. Hlavním zdrojem dynamických informací o systému je nástroj *sar*.

6.6.3 Bootstrap

Designu webových stránek bylo dosaženo s pomocí webového frameworku Bootstrap. Tento framework umožňuje rychlou tvorbu responzivních webových stránek. Hlavní důvod volby tohoto frameworku je možnost jednoduchého rozložení elementů stránky do řádků a sloupců [29].

6.7 Rozšiřitelnost

Výsledná aplikace je díky MVC architektuře a třídám, které zajišťují dynamické načítání komponentů, snadno rozšiřitelná.

Dynamické načítání logické vrstvy nám umožňuje přidat nově vytvořené třídy do adresářové struktury aplikace, bez nutnosti celou aplikaci restartovat. Pokud se požadovaná třída v době požadavku v aplikaci nenachází, jednoduše uživateli sdělíme, že stránka neexistuje. Pro rozšíření funkcionality stačí vytvořit novou třídu s požadovanými funkcemi, která bude splňovat stanovené požadavky, a následně ji vložit do správného adresáře.

Jednotlivé moduly datové vrstvy jsou také načítány dynamicky, a proto můžeme datovou vrstvu rozšiřovat stejně lehce jako logickou. Vytvoříme modul podle struktury stanovené v kapitole 6.4.2 a vložíme ho do adresářové struktury aplikace. O načtení nově vytvořeného modulu se aplikace postará sama.

6.8 Startup

Celá aplikace bude zaregistrována v SMF, abychom docílili jednoduché manipulace s aplikací. Pro registraci aplikace jako služby v operačním systému si budeme muset vytvořit XML dokument, který jí bude popisovat. Dále budeme muset vytvořit tzv. start skript, který bude naši aplikaci ovládat. Nakonec pro úspěšný chod aplikace vytvoříme roli v operačním systému Solaris, která bude mít práva provádět potřebné příkazy.

6.8.1 Role

Role se v operačním systému Solaris vytvářejí pomocí příkazu `roleadd`. Naše role se bude jmenovat *wzfsadm* a nebude mít žádný domovský adresář. Nebudeme ji přiřazovat ani heslo. Tím dosáhneme toho, že se na ní bude moci přepnout pouze uživatel *root* a nikdo jiný. Další výhodou je, že se na roli nedá přihlásit přímo při přihlašování do operačního systému, což omezuje některá bezpečnostní rizika.

Pomocí RBAC přiřadíme roli bezpečnostní profil, který bude sdružovat práva na vykonávání potřebných příkazů. V systémovém souboru */etc/security/prof_attr.d/core-os*, který obsahuje definice bezpečnostních profilů, vytvoříme nový profil *wzfsadm*. Na tento profil se budeme odkazovat při vytváření práv na vykonávání příkazů v souboru */etc/security/exec_attr.d/core-os*. Na konec tohoto souboru vložíme seznam příkazů, které bude majitel tohoto profilu moci vykonávat s identitou uživatele *root*. Pro ukázku je zde uveden jeden řádek, který budeme přidávat na konec souboru s právy.

```
wzfsadm:solaris:cmd:RO::/usr/sbin/zfs:euid=0
```

V prvním sloupci je název bezpečnostního profilu, ke kterému se právo vztahuje a v předposledním sloupci je samotný příkaz. Poslední sloupec udává identitu, pod kterou bude příkaz spuštěn. Takto vytvořený profil s právy přiřadíme roli pomocí příkazu `rolemod`.

6.8.2 Start skript

Start skript bude umět aplikaci spustit, zastavit a restartovat. Jelikož jsme použili RBAC práva, bude tento skript napsaný v tzv. profile shellu, který umí tyto práva interpretovat. Skript bude umět zpracovat parametry *start*, *stop* a *restart*, které budou určovat prováděnou akci.

Při startu aplikace skript spustí soubor *run.py*, který spustí webový server a celou aplikaci. Standardní výstup přesměruje do souboru *default_log* a chybový výstup do souboru *error_log*. Oba tyto soubory se budou nacházet v adresáři */var/log/wzfsadm*, který bude sloužit pro uchovávání logů aplikace. V závěru je uloženo PID spuštěného procesu do souboru, aby bylo později možné tento proces ukončit.

Zastavení aplikace znamená ukončení procesu webového serveru, jehož PID máme uložené v souboru. Pro ukončení procesu je zavolán příkaz `kill`.

6.8.3 Manifest

Manifest můžeme vytvořit například pomocí příkazu `svcbundle`. Tento příkaz vygeneruje XML dokument, který bude naši službu popisovat. Jeho obsahem jsou především metody, které se budou provádět při zapínání a vypínání služby. Pro tento účel jsme vytvářeli výše zmíněný start skript, který použijeme jako metodu pro ovládání služby. Další součástí manifestu je tzv. kontext metody, který nám umožní specifikovat uživatele, pod kterým se bude celá služba spouštět. Zde uvedeme námi vytvořenou roli *wzfsadm*. Posledním součástí je jméno služby, pod kterým budeme moci aplikaci spravovat. Součástí jména jsou i nadřazené kategorie, kterých se služba týká. V našem případě bude jméno služby *system/filesystem/wzfsadm*.

Následně zaregistrujeme službu pomocí příkazu `svccfg import`, který přesune vytvořený manifest do adresáře */lib/svc/manifest* a restartuje službu *manifest-import*, která se stará o načítání jednotlivých manifestů.

Výhoda registrace služby v systému spočívá v jednoduchém ovládání pomocí příkazu `svcadm`. SMF se také stará o zaznamenávání chyb při manipulaci se službou. Administrátor se tedy může kdykoli podívat k jaké chybě došlo.

6.9 Balíčkovací systém

Zdrojové kódy aplikace zabalíme pomocí nástroje `pkgmk` do balíčku *wzfsadm-i386.pkg*, který uživateli značně usnadní instalaci celé aplikace. Součástí balíčku budou i instalační skripty, které zajistí vytvoření role *wzfsadm* s potřebnými právy a také registrují službu *wzfsadm* v systému. Při odstranění balíčku dojde k odstranění vytvořené role a služby spolu se všemi zdrojovými kódy aplikace.

6.10 Testy

Veškerý vývoj a testování probíhalo na operačním systému Solaris 11 verze 5.11, který byl spouštěn pomocí aplikace *Virtualbox*. Hlavní výhodou aplikace *Virtualbox* byla možnost přidávání disků do systému, bez nutnosti fyzické disky vlastnit. Stačí vytvořit virtuální disk v hostitelském operačním systému a

následně ho připojit k virtuálnímu systému. Po startu systému je disk ihned k dispozici.

Závěr

Souborový systém ZettaByte je jeden z nejpokročilejších souborových systémů, se kterými se dnes můžeme setkat. Ke správě dat na disku využívá moderních principů, které mimo jiné zajišťují úsporu místa, neustálou konzistenci dat nebo elegantní tvorbu snapshotů. Nevýhodou tohoto systému je absence přehledného grafického rozhraní, které by umožňovalo jeho správu. Z tohoto důvodu bylo hlavním cílem této bakalářské práce vytvořit graficky orientovaný administrátorský nástroj sloužící pro správu souborového systému Zettabyte. Tento cíl byl úspěšně naplněn.

Vytvořený nástroj umožňuje administrátorovi jednoduše a komfortně spravovat některé části souborového systému ZFS. Poskytuje základní funkce pro administraci ZFS poolů a souborových systémů. Dále nástroj umožňuje přehledné zobrazování aktuálního stavu ZFS nebo dynamické zobrazení procentuálního využití procesoru. V neposlední řadě nástroj umožňuje sledovat historii jednotlivých příkazů, které byly v ZFS provedeny. Funkcionalita aplikace je postavena především nad ZFS příkazy *zfs* a *zpool* a nad systémovými příkazy *fdisk*, *format* a *sar*.

Nástroj je implementován pomocí webového rozhraní, které využívá architektury klient-server. Server a klient spolu komunikují pomocí HTTPS protokolu. Hlavní výhody této implementace spočívají v možnosti vzdáleného přístupu a nezávislosti klienta a serveru. Z tohoto důvodu může být aplikace spouštěna pod operačním systémem Solaris, zatímco klient může odesílat požadavky z jiného operačního systému. Hlavním předmětem komunikace jsou HTML stránky, po kterých se uživatel může libovolně pohybovat bez nutnosti hlubší znalosti architektury aplikace.

Při vývoji aplikace byl kladen důraz především na bezpečnost, jednoduchost používání a budoucí rozšiřitelnost. Bezpečnost přenosu dat mezi klientem a webovým serverem je zajišťována pomocí HTTPS protokolu, který data šifruje. Do aplikace je umožněn vstup pouze autentizovaným uživatelům, jelikož nástroj umožňuje provádět nebezpečné příkazy. Autentizace uživatelů se provádí oproti lokální databázi uložené v souboru. Nástroj má dva režimy

bezpečnosti, které povolují resp. zakazují práci se systémovým poolem.

Celá aplikace byla integrována do operačního systému Solaris, pro který byl souborový systém ZFS původně vyvíjen. Integrace do systému umožňuje administrátorovi aplikaci jednoduše ovládat a v případě nějaké chyby mu poskytne informace potřebné k jejímu odstranění. Jednotlivé vlastnosti a funkce aplikace se dají nastavit pomocí konfiguračních souborů. Tato možnost administrátorovi umožňuje aplikaci přizpůsobit jeho požadavkům. Dále byla v operačním systému Solaris vytvořena speciální role, pod kterou se výsledný nástroj spouští. Tento krok aplikaci zajišťuje práva k vykonávání potřebných příkazů a minimalizuje bezpečnostní rizika.

Z časových důvodů nebylo možné do výsledné aplikace zahrnout všechny funkce správy souborového systému ZFS, a proto byla její struktura navržena tak, aby se dala v budoucnu jednoduše rozšířit o nové funkce. K tomuto účelu bylo využito objektové architektury MVC, která odděluje jednotlivé vrstvy aplikace tak, že jsou na sobě minimálně závislé. Pro rozšíření funkcionality nástroje stačí rozšířit nebo modifikovat jen některé z těchto vrstev. Vrstva zajišťující komunikaci se souborovým systémem ZFS byla dále rozdělena do modulů. Jednotlivé moduly se specializují na určité oblasti správy ZFS a zajišťují tak aplikaci potřebnou funkcionalitu. Aplikaci je možné o tyto moduly kdykoli rozšířit.

Nejbližší rozšíření aplikace by mělo spočívat v implementaci modulů, které doplní funkcionalitu aplikace o chybějící funkce. Vzhledem ke struktuře aplikace by to neměl být větší problém. Výsledná implementace aplikace umožňuje správu jednoho souborového serveru. Další vylepšení aplikace by mohlo spočívat ve správě více těchto serverů. Bylo by nutné centralizovat sběr dat a na jednotlivé souborové servery umístit sondy, které by zajišťovaly sběr dat.

Pro usnadnění instalace byl vytvořen balíček *wzfsadm-i386.pkg*, který obsahuje všechny zdrojové kódy aplikace a zajišťuje celkovou instalaci nástroje se všemi potřebnými komponenty.

Všechny cíle bakalářské práce a požadavky stanovené na aplikaci se podařilo splnit. Výsledná implementace administrátorského nástroje sloužícího pro správu souborového systému ZFS je dostupná na přiloženém médiu.

Literatura

[1] Trdlička, J.: *Operační systémy, přednáška 11. Systémy souborů* [online]. Praha, České vysoké učení technické v Praze, Fakulta informačních technologií [cit. 2016-05-10]. Dostupné z: `https://edux.fit.cvut.cz/courses/BI-OSY/lectures/11/start`

[2] Oracle: *Oracle and Sun Microsystems* [online]. [cit. 2016-05-10]. Dostupné z: `https://www.oracle.com/sun/index.html`

[3] Oracle: *Oracle Help Center* [online]. [cit. 2016-05-10]. Dostupné z: `http://docs.oracle.com/en/`

[4] Oracle: *ZFS Hardware and Software Requirements and Recommendations* [online]. [cit. 2016-05-10]. Dostupné z: `http://docs.oracle.com/cd/E19253-01/819-5461/setup-1/index.html`

[5] Oracle: *ZFS and Traditional File System Differences* [online]. [cit. 2016-05-10]. Dostupné z: `https://docs.oracle.com/cd/E19120-01/open.solaris/817-2271/gbcik/index.html`

[6] Oracle: *ZFS Terminology* [online]. [cit. 2016-05-10]. Dostupné z: `http://docs.oracle.com/cd/E19253-01/819-5461/ftyue/index.html`

[7] Oracle: *Inheriting ZFS Properties* [online]. [cit. 2016-05-10]. Dostupné z: `http://docs.oracle.com/cd/E19253-01/819-5461/gazup/index.html`

[8] Oracle: *Setting ZFS Quotas and Reservations* [online]. [cit. 2016-05-10]. Dostupné z: `http://docs.oracle.com/cd/E19253-01/819-5461/gazvb/index.html`

[9] Bonwick, J.: *ZFS Deduplication* [online]. Publikováno 1.11.2009, [cit. 2016-05-10]. Dostupné z: `https://blogs.oracle.com/bonwick/entry/zfs_dedup`

[10] Oracle: *Transactional Semantics* [online]. [cit. 2016-05-10]. Dostupné z: http://docs.oracle.com/cd/E19253-01/819-5461/zfsover-2/

[11] Oracle: *Checking ZFS File System Integrity* [online]. [cit. 2016-05-10]. Dostupné z: http://docs.oracle.com/cd/E23823_01/html/819-5461/gbbwa.html

[12] Bonwick, J.: *ZFS End-to-End Data Integrity* [online]. Publikováno 8.12.2005, [cit. 2016-05-10]. Dostupné z: https://blogs.oracle.com/bonwick/entry/zfs_end_to_end_data

[13] Oracle: *Managing ZFS Mount Points* [online]. [cit. 2016-05-10]. Dostupné z: http://docs.oracle.com/cd/E19253-01/819-5461/6n7ht6r3o/index.html

[14] Oracle: *Overview of ZFS Snapshots* [online]. [cit. 2016-05-10]. Dostupné z: https://docs.oracle.com/cd/E23824_01/html/821-1448/gbciq.html

[15] LaFave, J.; Buckley, J. B.; Negado, E.: *zfsmon* [online]. GitHub [cit. 5.5.2016]. Dostupné z: https://github.com/CRBS/zfsmon

[16] Snabb, J.; Tötterman, P.: *zfswatcher* [online]. GitHub [cit. 2016-05-10]. Dostupné z: https://github.com/damicon/zfswatcher

[17] Oracle: *ZFS Web-Based Management* [online]. [cit. 2016-05-10]. Dostupné z: http://docs.oracle.com/cd/E19253-01/819-5461/gbsbp/index.html

[18] *Design Patterns - MVC Pattern* [online]. [cit. 2016-05-10]. Dostupné z: http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

[19] Fielding, R. T.; Gettys, J.; Mogul, J. C.; aj.: *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616, RFC Editor, June 1999, [cit. 2016-05-10]. Dostupné z: http://www.rfc-editor.org/rfc/rfc2616.txt

[20] Berners-Lee, T.; Fielding, R. T.; Masinter, L.: *Uniform Resource Identifier (URI): Generic Syntax*. STD 66, RFC Editor, January 2005, [cit. 2016-05-10]. Dostupné z: http://www.rfc-editor.org/rfc/rfc3986.txt

[21] Oracle: *Role-Based Access Control* [online]. [cit. 2016-05-10]. Dostupné z: https://docs.oracle.com/cd/E23824_01/html/821-1456/rbac-1.html

[22] Franks, J.; Hallam-Baker, P. M.; Hostetler, J. L.; aj.: *HTTP Authentication: Basic and Digest Access Authentication*. RFC 2617, RFC Editor, June 1999, [cit. 2016-05-10]. Dostupné z: http://www.rfc-editor.org/rfc/rfc2617.txt

[23] Rescorla, E.: *HTTP Over TLS*. RFC 2818, RFC Editor, May 2000, [cit. 2016-05-10]. Dostupné z: `http://www.rfc-editor.org/rfc/rfc2818.txt`

[24] Foster, G.: *Introducing the Basics of Service Management Facility (SMF) on Oracle Solaris 11* [online]. [cit. 2016-05-10]. Dostupné z: `http://www.oracle.com/technetwork/articles/servers-storage-admin/intro-smf-basics-s11-1729181.html`

[25] *SocketServer — A framework for network servers* [online]. Python Software Foundation, [cit. 2016-05-10]. Dostupné z: `https://docs.python.org/2/library/socketserver.html`

[26] Ronacher, A.: *Jinja2* [online]. [cit. 2016-05-10]. Dostupné z: `http://jinja.pocoo.org/`

[27] *jQuery* [online]. [cit. 2016-05-10]. Dostupné z: `https://jquery.com/`

[28] *Google charts* [online]. [cit. 2016-05-10]. Dostupné z: `https://developers.google.com/chart/`

[29] *Bootstrap* [online]. [cit. 2016-05-10]. Dostupné z: `http://getbootstrap.com/`

Příloha A

Seznam použitých zkratek

AJAX Asynchronous Javascript And XML

FAT File Alocation Table

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

HTTPS HyperText Transfer Protocol Secure

MVC Model View Controller

PID Process ID

RAID Redundant Array of Independent Disks

RBAC Role-Based Access Control

RFC Request For Comments

SMF Solaris Management Facility

SSL Secure Socket Layer

URL Uniform Resource Locator

VM Volume Manager

XML Extensible Markup Language

ZFS Zettabyte File System

Příloha B

Uživatelská příručka

B.1 Požadavky

Instalaci aplikace je nutné provádět na operačním systému Solaris, který využívá souborový systém ZFS. To znamená, že v systému budou dostupné příkazy `zfs` a `zpool`. Aplikace je napsána v jazyku Python 2.7, a proto je nutné mít nainstalovaný jeho interpret. Pro vytvoření balíčku *wzfsadm-i386.pkg* ze zdrojových kódů je potřeba program `make`.

B.2 Instalace

Vytvoření balíčku ze zdrojových kódů aplikace se provádí pomocí `make pkg` v adresáři se zdrojovými kódy balíčku. Po provedení předchozího příkazu se vytvoří balíček *wzfsadm-i386.pkg*, který nainstalujeme příkazem `pkgadd -d`. Instalaci aplikace je nutné provádět pod uživatelem *root*, protože součástí instalace jsou i skripty, které připraví potřebnou roli, zaregistrují aplikaci v SMF a nainstalují šablonovací modul *Jinja2*.

B.3 Spuštění

V případě bezproblémové instalace by se aplikace měla automaticky spustit. Přesvědčit se o tom můžeme pomocí následujícího příkazu, který nám ukazuje, jestli je daná služba spuštěná či nikoliv.

```
# svcs -a | grep wzfsadm
online  9:25:22 svc:/system/filesystem/wzfsadm:default
```

Aplikaci můžeme ovládat pomocí příkazu `svcadm enable/disable wzfsadm` stejně jako jiné služby. Informace o běhu aplikace, chybách a varováních jsou uloženy v souborech *default_log* a *error_log*, které se nacházejí v adresáři */var/log/wzfsadm*.

Podobně můžeme aplikaci spouštět přímo pomocí skriptu *run.py*. Tento skript je součástí kořenového adresáře aplikace */usr/wzfsadm*. Zde si musíme dát pozor na práva uživatele, pod kterým aplikaci spouštíme.

B.4 Konfigurace

Všechny konfigurační soubory aplikace budou uloženy v adresáři */etc/wzfsadm*. Aplikace se řídí hlavním konfiguračním souborem *main.conf*, který v základu vypadá následovně.

```
[server]
address = 0.0.0.0
port = 4443
address_filter = 127.0.0.1

[ssl]
ssl = yes
key_file = /etc/wzfsadm/ssl/key.pem
cert_file = /etc/wzfsadm/ssl/cert.pem

[auth]
user_database = /etc/wzfsadm/.auth_file

[app]
safe_mode = yes
```

Pod direktivou `server` lze nastavovat síťové rozhraní a port, na kterém bude aplikace přijímat spojení. Položka `address_filter` slouží ke stanovení IP adres, které budou moci k aplikaci přistupovat. Direktiva `ssl` stanovuje, zda má být využito šifrování komunikačního kanálu a určuje cestu k souborům s certifikátem a privátním klíčem. Položka `user_database` představuje soubor s databází uživatelských jmen a hesel, které slouží k ověřování identity uživatelů. Poslední položka `safe_mode` určuje, zda bude aplikace zobrazovat informace o systémovém poolu. V případě nedostupnosti hlavního konfiguračního souboru *main.conf* jsou použity předefinované hodnoty.

Druhý konfigurační soubor `poolmodule.conf` se týká implementovaného modulu `PoolModule`. Obsahuje jedinou položku `file_sources`, která určuje adresář pro soubory sloužící jako zdroje pro ZFS.

B.5 Přístup do aplikace

Do spuštěné aplikace se připojíme pomocí webového prohlížeče, do kterého zadáme adresu, na které je aplikace dostupná. Standardně je to `https://`

`localhost:4443`. Při vstupu do aplikace bude uživatel požádán o zadání uživatelského jména a hesla. Počáteční uživatelské jméno je *admin* a heslo také *admin*. Administrátor může tyto údaje modifikovat v souboru s uživatelskou databází, která se standardně nachází v souboru */etc/wzfsadm/.auth_file*. Při modifikaci tohoto souboru je nutné dodržet jeho strukturu.

B.6 Odstranění aplikace

Aplikaci lze odstranit pomocí příkazu `pkgrm wzfsadm`, který ze systému odstraní vytvořenou roli společně se všemi zdrojovými kódy aplikace.

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `package` ..... zdrojové kódy instalačního balíčku
  - `wzfsamd` ..... zdrojové kódy aplikace wzfsadm
  - `wzfsamd-i386.pkg` ..... instalační balíček
- `text`
  - `BP_Simacek_Tomas_2016.pdf` ..... text práce ve formátu PDF
  - `src` ..... zdrojové soubory práce ve formátu LaTeX