Automatizace výkonnostních testů nástrojů Flowmon

Bc. Petr Hromádka

Diplomová práce
2019/2020

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Petr Hromádka**
Osobní číslo:	**A18261**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**Kombinovaná**
Téma práce:	**Automatizace výkonnostních testů nástrojů Flowmon**
Téma práce anglicky:	**The Automatisation of Performance Tests Inside Flowmon Utilities**

Zásady pro vypracování

1. Stanovte konkrétní cíle práce s hledem na analýzu řešení Flowmon a jeho jednotlivých komponent z pohledu výkonnostního (performance) testování.
2. Proveďte analýzu existujících nástrojů a frameworků s ohledem na cíle stanovené v bodu č. 1 a zhodnoťte jejich praktickou použitelnost pro naplnění cílů práce.
3. Navrhněte postup provádění výkonnostních testů jednotlivých komponent řešení Flowmon.
4. Automatizujte tyto testy prostřednictvím vhodného nástroje případně vlastního nástroje (testovací scénáře musí být konfigurovatelné a parametrizovatelné).
5. Otestuje dosažené výsledky v praxi, konzultujte s QA týmem společnosti Flowmon Networks.
6. Výsledky integrujte do systému Jenkins.

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

Seznam doporučené literatury:

1. JORGENSEN, Paul. *Software testing: a craftsman's approach*. Fourth edition. Boca Raton, [Florida]: CRC Press, Taylor & Francis Group, [2014]. ISBN 1466560681.
2. BEIZER, B.:Software Testing Techniques. Dreamtech, 2003.
3. Flowmon Networks: Řešení Flowmon – schéma. [online] 2019, [cit. 14.7.2019]. Dostulné z:https://www.flowmon.com/cs/products/flowmon
4. Flowmon Networks: Řešení Flowmon. [online] 2019, [cit. 14.7.2019]. Dostulné z: https://www.flowmon.com/getattachment/dce90edf-20a2-4366-8c99-e2145f938ac9/Flowmon-Solution-Product-Brief.aspx
5. Flowmon Networks: Flowmon kolektor. [online] 2019, [cit. 14.7.2019]. Dostulné z: https://www.flowmon.com/getattachment/8d285feb-a58a-4d27-a643-43b24c868859/Flowmon-netflow-ipfix-collector-spec.aspx
6. Flowmon Networks: Flowmon ADS. [online] 2019, [cit. 14.7.2019]. Dostulné z: https://www.flowmon.com/getattachment/3498c8ec-8c61-4f2f-96a7-82a611cecf13/Flowmon-ADS.aspx

Vedoucí diplomové práce: **Ing. David Malaník, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **28. listopadu 2019**
Termín odevzdání diplomové práce: **15. května 2020**

doc. Mgr. Milan Adámek, Ph.D.
děkan

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

Ve Zlíně dne 9. prosince 2019

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně 6.8.2020 Petr Hromádka

ABSTRAKT

Tato práce se zabývá automatizací výkonnostního testováni Flowmon nástrojů. Cílem práce je analyzovat způsoby a následně nějaký zprovoznit. Úvod je věnován teorii o testování se zaměřením na výkonnostní testování. V další části je analýza požadavků a dostupných řešení, následována návrhem a na závěr implementace frameworku a jeho integrace do systému Jenkins.

Klíčová slova: Flowmon, Netflow, výkonnostní testování, automatizace, Python

ABSTRACT

This thesis is about automatization of performance tests inside Flowmon utilities. Goal of this thesis is to analyze and find solution. The beginning is about theory of testing focused on performance testing. Next parts are about requirments analysis and available solutions followed by draft. The last part is framework implementation and it´s integration to Jenkins system.

Keywords: Flowmon, Netflow, performance tests, automatization, Python

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat doktoru Malaníkovi za odborné vedení práce a také firmě Flowmon Networks za zadání a možnost na tomto tématu pracovat. Dále děkuji nejlepší tanečnici Adéle za jazykovou korekturu. Další poděkování patří Jindřichovi za konzultace po technické stránce. Je to pravý přítel a rovný chlap. Poděkování také patří spolubydlícím a zvířecí smečce na bytě za vytváření (ne)vhodného prostředí pro psaní diplomové práce. Poslední poděkování je věnováno rodině za neutuchající podporu.

OBSAH

ÚVOD

IT prostředí se neustále a rychle vyvíjí a robustní, spolehlivá a dobře zabezpečená síť je základním kamenem prakticky každé vetší i menší organizace. Správci sítě tak mají v dnešní době složitý úkol. Úspěšný útok, který způsobí výpadek sítě nebo odcizení citlivých údajů, může způsobit poškození dobrého jména firmy nebo dokonce ztrátu zákazníků. Vzhledem ke stále četnějším a sofistikovanějším útokům a dalším hrozbám je potřeba efektivně monitorovat a spravovat počítačovou síť. Naštěstí existují moderní nástroje, které toto síťovým správcům umožňují.

Požadavky na monitorovací a zobrazovací nástroj jsou velké, proto zde možná i více než u jiného softwaru platí, že je potřeba důkladně testovat. Testování může probíhat z pohledu uživatele, zda síťový správce dokáže najít vše, co potřebuje, v přijatelné době, z funkcionálního hlediska, tedy jestli software dělá to, co se očekává, a nebo testování z pohledu výkonu, jestli jsou některé části pomalé, nebo jestli zvládá zpracovat a zobrazit veškerá potřebná data.

Flowmon je řešení pro kompletní a detailní viditelnost do síťového provozu, umožňuje tak optimalizaci výkonu sítě a ochranu před moderními kybernetickými útoky. Celé řešení se skládá z Flowmon sond pro export Netflow/IPFIX statistik, Flowmon kolektorů k uložení, vizualizaci a analýze síťového provozu a Flowmon rozšiřujících modulů. Zásadní výhodou řešení Flowmon je možnost monitorignu velkých sítí s vysokým přenosem dat, nicméně pro monitorování takové sítě je zapotřebí garantovat dostatek výkonu.

Cílem této diplomové práce je navrhnout a vytvořit framework umožňují automatizaci výkonnostních testů nástrojů Flowmon. Obecně o testování, jaké jsou typy, možnosti klasifikace, se zabývám v teoretické části. Stručný popis testovaných nástrojů je ve analytické části spolu s analýzou dostupných řešení. Samotný popis, návrh i případové studie testovacího frameworku je v projektové části.

Motivací je fakt, že výkonnostní testování v současné době probíhá manuálně a je potřeba mnoho času od přípravy, přes testování až po samotné vyhodnocení. Výsledkem by tedy měl být nástroj nebo způsob, jak tento čas ušetřit, či jinak výrazně zefektivnit.

I. TEORETICKÁ ČÁST

V této části bude rozebrán teoretický základ k testování, jaké jsou typy testování a jakým způsobem se provádí.

Testování má dva základní cíle: demonstrovat korektní chování a najít bugy[6]. Zde je důležité zadefinovat, co je to vlastně bug. V knize Software testing od Rona Pattona [7] se bug vyskytuje v softwaru, pokud je splněno alespoň jedno z následujících pravidel:

1. Software nedělá to, co by podle specifikace[1)] měl dělat.
2. Software dělá něco, co by podle specifikace neměl dělat.
3. Software dělá něco, o čem se specifikace nezmiňuje, ale pravděpodobně by to neměl dělat.
4. Software nedělá něco, co ve specifikaci není zmíněno, ale měl by to dělat.
5. Software je nepřehledný, špatně se používá, je pomalý, ...

[1)] Produktová specifikace detailně popisuje jednotlivé části produktu a dává informace o používání a funkcionalitách `https://techwhirl.com/writing-software-requirements-specifications`.

1 Přístupy testování

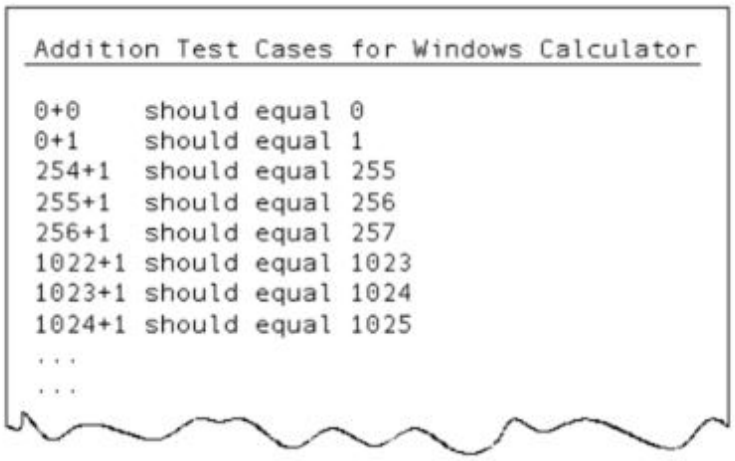
Existují různé přístupy k testování, pro úplné otestování softwaru je obvykle potřeba více různých způsobů. Následující přístupy vycházejí z knih Software testing od R. Pattona [7] a Software Testing Techniques od B. Beizera [2].

1.1 Statické Black-Box testování

Netestuje se produkt jako takový, ale produktová specifikace. Cílem tohoto testování je odhalit bugy dříve, než vůbec vzniknou, najít špatně navržené, špatně pochopitelné nebo nedostatečně popsané funkcionality, případně nadbytečné, či chybějící funkce. V tomto případě tester vidí pouze to, co vidí i zákazník a měl by přemýšlet bez znalosti jak produkt vnitřně funguje, ale jen tak, jak ho vidí zákazník.

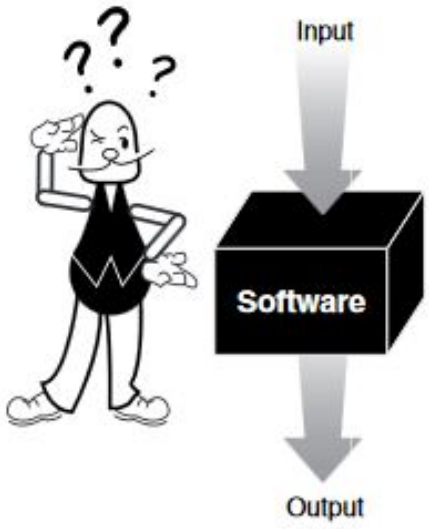
1.2 Dynamické Black-Box testování

Takzvané *testování s klapkami na očích*, pracuje s produktem. Základem je sada vstupů spolu se sadou referenčních výstupů.



Obr. 1.1 Příklad vstupů a k nim referenčních výstupů pro kalkulačku. [7]

Tester nevidí do zdrojového kódu produktu, aby testovací sady mohl přizpůsobit, ale používá produkt stejně, jako by ho používal zákazník. Testovací sady mohou být navržené tak, aby dopadly správně a demonstrovaly korektní chování, nebo aby dopadly špatně a zkontrolovaly tak, jak se produkt chová v případě chyby, jak zobrazuje chybové hlášení, či zda chyba nezpůsobí neočekávaný pád celého produktu.



Obr. 1.2 Znázornění black-box testování.[7]

1.3 Statické White-Box testování

Jedná se o inspekci návrhu a kódu nebo také strukturální analýza. Při tomto přístupu má tester přístup do kódu a snaží se objevit chyby v kódu nebo nevhodnou architekturu bez toho, aby produkt běžel. Vývojáři často využívají tento přístup v rámci tzv. *code review*[1)]. Automatizovanou pomůckou tohoto přístupu mohou být překladače[2)], které dokáží upozornit na špatné přístupy do paměti, nekompatibilní přetypování, či jiné techniky, které se při samotném spuštění mohou projevit pouze ve specifických případech, často nikdy. Nebo specializovaný software, který kontroluje určité standardy při psaní kódu[3)], či různá vývojová prostředí[4)], která výše uvedené chyby kontrolují už během psaní.

1.4 Dynamické White-Box testování

Při dynamickém white-box testování tester vidí do struktury projektu proto tzv. *strukturální testování*. Náhled do projektu mu umožňuje spouštění dle vlastních potřeb. Tím, že má přístup ke zdrojovým kódům, může určit, co testovat. Celkem zahrnuje 4 oblasti:

- Testování nízkoúrovňových funkcí, procedur, nebo knihoven.

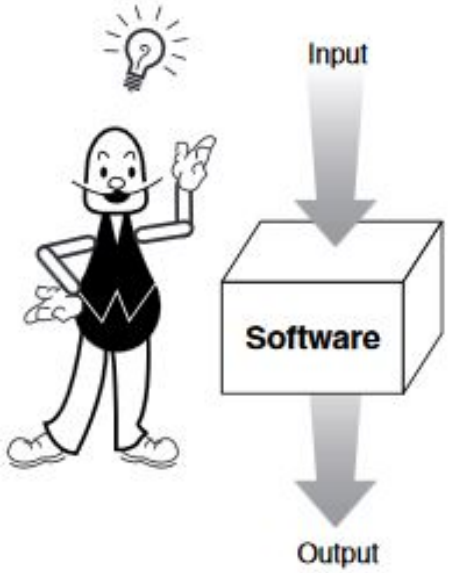
[1)] Mechanismus kdy si vývojáři vzájemně procházejí kód a říkají si co napsat čitelněji nebo efektivněji.

[2)] Např. argumenty -Wall nebo -Wextra u překladače gcc `https://gcc.gnu.org/`.

[3)] Například knihovna pep8 pro python `https://pypi.python.org/pypi/pep8`.

[4)] Např. PyCharm, Netbeans, Clion...

- Testování produktu na nejvyšší úrovni jako celku, podobně jako dynamické black-box testování s tím, že lze testovací sady upravit na základě struktury produktu.
- Přístup ke čtení proměnných a stavových informací, ověřování, že testy opravdu testují to, co mají testovat. Zároveň je tester schopný získávat více informací, které se mohou k testování hodit, což by v případě dynamického black-box testování nebylo možné.
- Měření, jaké množství kódu se provádí při spuštění testů za účelem odstranění nějakých redundantních testů a přidáním chybějících.



Obr. 1.3 White-box testování. [7]

1.5 Výkonnostní testování

Výkonnostní testy se často rozdělují na 3 velice podobné typy, ale každý má jiný účel zkoumání. [4]

1.5.1 Výkonnostní testy

Cílem výkonnostních testů není najít bugy, ale eliminovat úzká hrdla[5)] a nalézt limity systému.

Základem je jasně definovat pravidla, která musí být splněna, tak aby mohly být splněny určité limity, např. webový server zvládne obsloužit 10 000 unikátních transakcí ve stejný okamžik (tedy 10 000 různých uživatelů) při maximálně odezvě do 60 ms.

[5)] tzv. *bottleneck*, tedy místo, kde dochází k největšímu omezení výkonu.

Při hledání úzkých hrdel se pak z pohledu testování jedná o white-box přístup a může se na produkt nahlížet na různých úrovních. V příkladě webového serveru to může být:

- Aplikační vrstva, např. neefektivní vyhledávající algoritmy
- Databázová úroveň, pomalé dotazy
- Operační systém, vytížení CPU, disk I/O, zombie procesy
- Síťová úroveň, zahlcení TCP stacku

1.5.2 Zátěžové testy

Cílem zátěžových testů je zatížit systém na maximální možnou úroveň, tak aby stále zvládal vykonávat svou funkci. Při maximálním vytížení se často objeví i jinak skryté bugy ve formě špatné správy paměti, memory leak[6)] nebo buffer overflow[7)]. Dalším cílem je ověřit, že systém zvládá limit nastavený při výkonnostních testech, a to tak, že zkouší zatížit systém na nejvyšší maximální zátěž a kontroluje, že systém je stále schopen běžet plynule. Cílem není systém nadměrně zahltit a rozbít, ale maximálně ho konstantně zatěžovat.

1.5.3 Stresové testy

Stresové testy se snaží přetížit systém tak, aby zkolaboval. Cílem je ověřit, že je systém schopen znovu začít fungovat, tedy ověřit vlastnost systému znovuobnovitelnost.

Dosahuje toho různými technikami:

- Zvyšováním zátěže, více než naměřily zátěžové testy
- Náhodné vypínání služeb, které řídí systém (vypínání databáze, kontrolních daemonů[8)])
- Znovusestavování RAID pole[9)] při běhu systému
- Spouštění jiných procesů, které zabírají prostředky (paměť, CPU..)

[6)] Situaci, kdy proces alokuje operační paměť a není ji schopen uvolnit poté, co ji již dále ani nepotřebuje.

[7)] Situace, při které proces při zápisu dat do vyrovnávací paměti (tzv. bufferu) překročí vymezené hranice a přepíše sousedící úsek operační paměti.

[8)] Obecně proces, který běží na pozadí a neinteraguje s uživatelem.

[9)] RAID (Redundant Array of Independent Disks) je technologie ukládání dat, která umožňuje spojit více fyzických disků do jednoho logického úložný prostoru.

II. ANALYTICKÁ ČÁST

V této části bude rozebrána analýza projektu, jeho požadavky, analýza dostupných řešení a různých přístupů, jak dosáhnout požadovaného výsledku.

Cílem práce by mělo být najít způsob, jak plně automatizovat výkonností testy nástrojů Flowmon. Pro orchestraci se počítá s jednotkami či nižšími desítkami serverů. Co tedy konkrétně znamená výkonnostní testy nástrojů Flowmon? V současné době se výkon měří na dvou úrovních:

- Objem dat, které je Flowmon kolektor schopen zpracovat a uložit. Objem dat se měří v FPS[10)].
- Čas, za který je Flowmon kolektor schopen zobrazit analýzu nad daty.

Jaké jsou požadavky na měření výkonu? Pro měření objemu dat se se v základní konfiguraci využívají nejmenší možné toky. Síťový tok je v terminologii NetFlow definován jako sekvence paketů se shodnou pěticí údajů: cílová/zdrojová IP adresa, cílový/zdrojový port a číslo protokolu. [1]. Hledá se tedy maximální množství FPS, při kterém nedochází k žádnému zahazování na síti nebo přetížení disku. Vzhledem k tomu, jak kolektor funguje, je potřeba nejprve disk zaplnit tak, aby při testování docházelo k odmazávání starých dat, což je režim, ve kterém v reálném prostředí funguje kolektor nejčastěji.

Pro měření času, za jak dlouho je kolektor schopen zobrazit analýzu, se jako referenční interval využívá poslední hodina dat. Pro testy jsou stanovené hraniční zátěže 10 000 FPS, 50 000 FPS a 100 000 FPS. Tedy je potřeba umět vygenerovat stabilně hodinu definovaného množství provozu a nad ním poté změřit čas, za jak dlouho budou data zobrazena.

Aktuálně jsou výkonnostní testy prováděny manuálně. Jak tedy vypadá celý proces výkonnostního testování a kolik zabere času?

- Instalace operačního systému Flowmon na server. ($\sim$ 1 hodina)
- Konfigurace serveru na kolektor. ($\sim$ 10 minut)
- Aktualizace na nejnovější testovanou vývojovou verzi. ($\sim$ 0,5 - 2 hodiny)
- Výkonnostní testování. ($\sim$ 5 hodin)

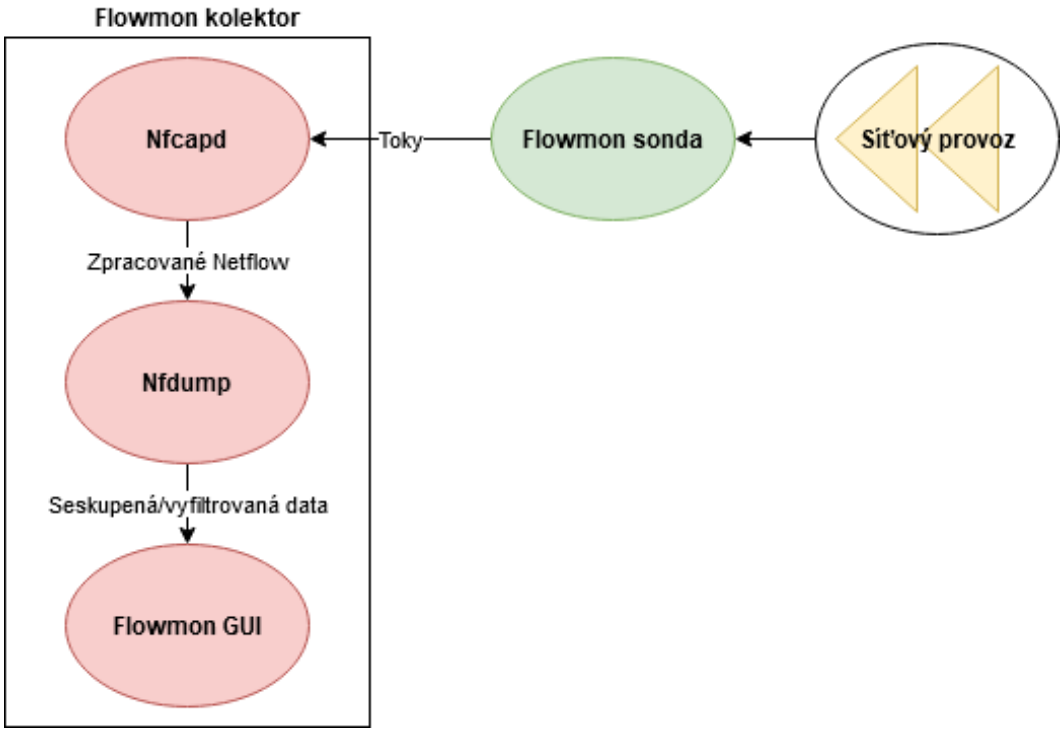
[10)] Flows per second - unikátní toky za sekundu.

2 Flowmon nástroje

Cílem je automatizovat testy pro Flowmon nástroje. Nejprve je potřeba vysvětlit co se pod označením Flowmon skrývá. Flowmon je řešení s množstvím nástrojů založený na CentOS. Existují dvě varianty Flowmon kolektor a Flowmon sonda.

Flowmon kolektor slouží k ukládání a zpracování síťového provozu ve formě Netflow/IPFIX. Jeho hlavní částí je nfcapd, který ukládá data, nfdump, který slouží k zobrazování a analýze nad daty v textové podobě a webové GUI k zobrazování analýz.

Flowmon sonda zpracovává síťový provoz do formy toků a posílá na kolektor ve formátu Netflow/IPFIX.



Obr. 2.1 Flowmon kolektor

V rámci diplomové práce se bude zkoumat výkonnost Flowmon kolektoru a to kolik toků za sekundu je kolektor schopen uložit a poté měření vykonávání analýzy nad daty.

3 Analýza dostupných řešení a nástrojů

Na základě výše provedené analýzy požadavků je potřeba rozdělit i analýzu dostupných řešení, protože každá část je značně odlišná.

3.1 Analýza reinstalace serverů

Serverová řešení obsahují vzdálené rozhraní pro řízení, buď v základní standardizované variantě IPMI 3.1.1, nebo s rozšířenými možnostmi, které pak už jsou závislé na platformě. Příkladem tohoto rozšíření je *racadm*[1)] na serverech od firmy Dell, které umožňuje vzdálenou reinstalaci serverů. Základní varianta IPMI tuto možnost nemá. Existuje tak jediná open source platformě nezávislá varianta: PXE 3.1.2.

3.1.1 IPMI

IPMI (Intelligent Platform Management Interface) definuje abstraktní rozhraní pro správu HW platforem. IPMI je nezávislé na operačním systému (nemusí být dokonce přítomný žádný), standardního firmwaru (např. BIOS, UEFI) i hlavního procesoru. Je potřeba pouze napájení a pro vzdálenou správu i funkční síťové připojení. Jeho základem je jednoúčelový integrovaný obvod nazývaný BMC[2)], který má propojení i na kontrolované moduly počítače i na konzoli pro vzdálenou správu. Mezi základní funkcemi je napojení na teplotní senzory, stav CPU, možnost vzdáleně počítač restartovat i možnost konfigurovat BIOS[8].

3.1.2 PXE

PXE je definovaný široce rozšířený standard pro bootování operačního systému po síti a používá další standartní internetové protokoly jako TCP/IP, DHCP a TFTP. Standardizuje interakce mezi klienty a servery, v tomto kontextu jako servery vystupují DHCP, TFTP a další podobné servery, jako klienti vystupují servery, které bootují po síti. Využívá se pro počítače, které nemají pevný disk, pro diagnostiku a pro automatické instalace operačních systémů [5].

3.1.3 iPXE

iPXE je open source nadstavba na PXE. Poskytuje plnou implementaci PXE i na platformách, které ji nativně neovládají, a rozšiřuje ho o další možnosti jako:

- kontrola procesu zavádění pomocí skriptu

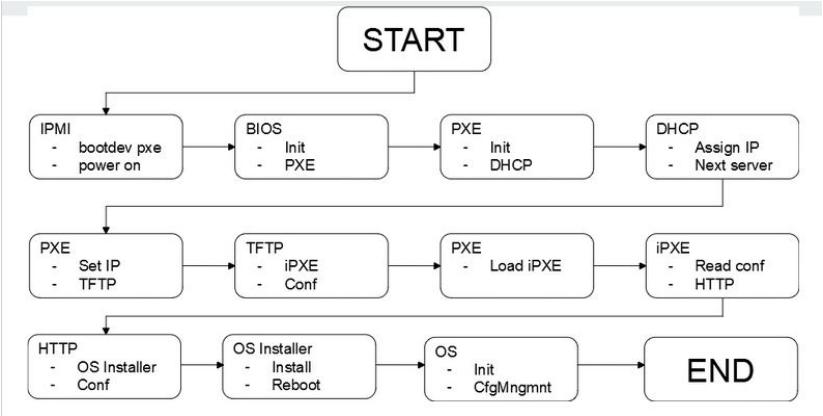
[1)] racadm je uživatelské rozhraní přes příkazovou řádku pro vzdálenou správu Dell serverů.
[2)] Baseboard Management Controller - kontrolér správy základní desky.

- podpora webového serveru přes HTTP
- podpora iSCSI protokolu
- podpora bezdrátových sítí

iPXE může plně nahradit zabudované PXE na síťové kartě (pokud tam je) nebo lze navázat do procesu tak, že PXE spustí iPXE skript a ten pak může využít veškerá iPXE rozšíření.

3.1.4 Shrnutí reinstalačního procesu

Celý reinstalační proces lze znázornit následujícím diagramem.



Obr. 3.1 Diagram znázorňující proces reinstalace.

1. Vzdáleně se pomocí IPMI zadá požadavek, aby další bootování proběhlo přes PXE a zapne server

   ```
   $ ipmitool –H <IP> –U <uzivatel> –P <heslo> chassis bootdev pxe
   $ ipmitool –H <IP> –U <uzivatel> –P <heslo> chassis power on
   ```

2. Server při inicializaci předá řízení PXE
3. PXE pro svou funkci potřebuje ip adresu, při inicializaci požádá DHCP server o svojí adresu
4. DHCP server nabídne ip adresu, zároveň pošle položku "další server", což je adresa TFTP serveru s dodatečnými daty

   ```
   next-server <IP>;
   filename "undionly.kpxe";
   ```

5. PXE přijme adresu a připojí se na TFTP server, ke kterému dostal IP adresu.

6. TFTP server ukazuje na iPXE skript
7. PXE stáhne iPXE skript a předá mu řízení
8. iPXE obsahuje odkaz na HTTP server
9. z HTTP serveru si stáhne instalační soubor operačního systému
10. provede se instalace operačního systému
11. po nastartování operačního systému může být následná další konfigurace (nastavení IP adresy, nainstalování dodatečných nástrojů, nahrání SSL klíčů, atd.)

3.2 Analýza konfigurace a aktualizace

Tyto dva kroky lze spojit dohromady, protože dělají podobné věci jen na trochu jiné úrovni nebo z jiného pohledu. Hlavním požadavkem na konfigurace je konzistence, aby se stejným vstupním konfiguračním souborem došlo vždy ke stejnému nakonfigurování. Další jsou jednoduchost používání a jednoduchost instalace.

3.2.1 Puppet

Puppet[3)] je rozsáhlý orchestrační nástroj, který umožňuje správu nad vzdálenými zařízeními kdekoli i v cloudu. Architektura tohoto nástroje je *master-slave*[4)], tedy na každém obsluhovaném serveru musí být nainstalována klientská aplikace. Vstupní konfigurace pak vypadá jako sada Ruby skriptů, které postupně řídí *master* a vzdáleně jsou vykonávány na *slave* jednotkách.

3.2.2 Ansible

Ansible[5)] je nástroj určený k orchestraci serverů. Hlavním cílem Ansible je minimalismus, není tedy potřeba žádná klientská aplikace na orchestrovaných serverech. Veškeré řízení probíhá přes protokol SSH[6)]. Konfigurace je pak zapsaná ve formátu YAML[7)].

[3)] viz. `https://puppet.com`

[4)] Existuje hlavní server, který řídí ostatní. `https://en.wikipedia.org/wiki/Master/slave_(technology)`.

[5)] viz. `https://www.ansible.com/`

[6)] Protokol pro zabezpečenou vzdálenou komunikaci.

[7)] Formát pro serializaci strukturovaných dat.

3.2.3 Python

Další možností je vytvořit vlastní nástroj. Nabízí se Python[8)] díky knihovně Fabric[9)]. Fabric zajišťuje kompletní rozhraní pro vzdálenou správu přes SSH, či FTP[10)]. Další významnou knihovnou je pytest[11)], který se dá využít v systému Jenkins, tedy pro výslednou integraci není zapotřebí žádných dalších zásahů.

3.2.4 Vyhodnocení

Puppet je komplexní nástroj, který by šel použít, ale jeho hlavní nevýhoda je *master-slave* architektura. Z toho důvodu by se muselo zajistit instalace klientské aplikace na orchestrované servery.

Ansible se zdá být jako zlatá střední cesta mezi komplexitou a jednoduchostí, nepotřebuje instalaci žádné další aplikace ani žádná specifika prostředí.

Python s výše zmíněnými knihovnami zvládne poměrně jednoduše zajistit to stejné jako Puppet nebo Ansible. Integrace do systému Jenkins je důležitý závěrečný požadavek, což v tomto případě implicitně umožňuje knihovna pytest.

3.3 Analýza nástrojů pro výkonnostní testy

Z pohledu výkonnostních testů je potřeba určit, jaké veličiny a jak měřit. V tomto typu měření se objevují dvě, jak bylo popsáno v analýze požadavků. V obou typech měření je potřeba generovat určité FPS. Ideálně tak, aby to co nejvíce připomínalo reálný provoz.

3.3.1 Packetgen a Flowmonexp

Packetgen je interní nástroj firmy Flowmon Networks, který odesílá čisté TCP-SYN[12)] pakety, u kterých může měnit zdrojové a cílové adresy i porty.

Flowmonexp je nástroj firmy Flowmon Networks, který zpracovává pakety a převádí je na toky. Umí zpracovávat pakety za sítě, ale i zachycené v souboru, není tedy potřeba dedikovaného serveru pro každý nástroj zvlášť.

IP tok je definován zdrojovými a cílovými adresou a portem a protokolem. Tím, že packetgen generuje pakety s různými zdrojovými a cílovými adresami a porty je Flowmonexp zpracuje jako rozdílné toky. Tedy co paket, to tok. Kombinace těchto

[8)] Vysokoúrovňový skriptovací programovací jazyk.
[9)] viz. http://www.fabfile.org/
[10)] Protokol pro přenos souborů po síti.
[11)] Framework pro snadné psaní malých, ale i komplexních testů. Viz. https://docs.pytest.org/en/stable/
[12)] Příznak SYN se odesílá v první paketu, značí, že klient chce navázat komunikaci.

dvou nástrojů umožňuje posílání toků na testovaný kolektor.

3.3.2 Nfreplay_ng

Nfreplay_ng je interní nástroj firmy Flowmon Networks. Umožňuje posílat toky, které jsou definované vstupní nebo výchozí sadou uložených toků ve formátu JSON[13)]. Tímto způsobem se dá vygenerovat provoz, který odpovídá více reálnému provozu (není to jen jeden stejný paket, který se opakuje pouze s jinou adresou), pouze dochází k duplikaci. To ale pro výkonnostní test kolektoru nehraje žádnou roli, protože kolektor v defaultním režimu neprovádí žádnou behaviorální analýzu, ale pouze ukládá a zobrazuje síťový provoz.

[13)] Způsob zápisu dat nezávislý na platformě, určený pro přenos dat.

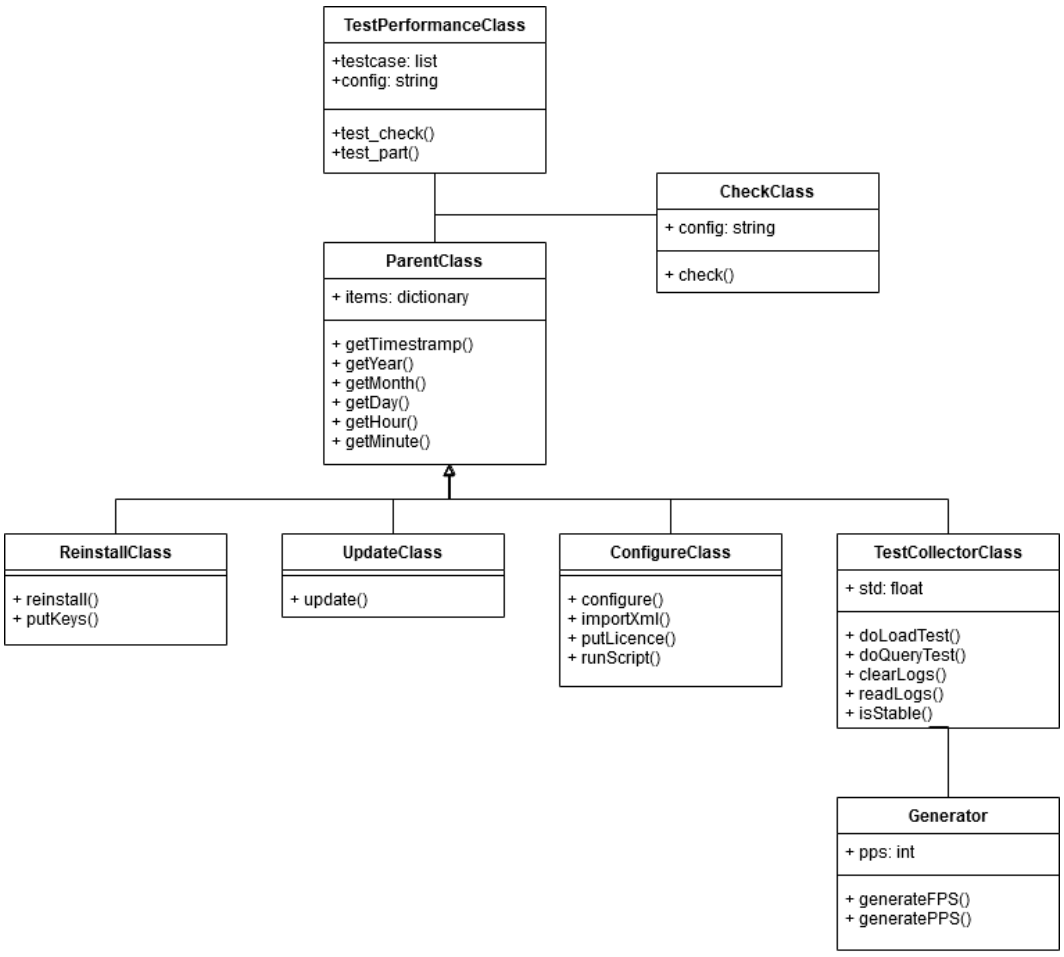
4 Návrh testovacího frameworku

S ohledem na předchozí analýzu framework bude napsaný v Python3.7, bude využívat pytest, díky kterému bude jednoduchá integrace do systému Jenkins. Pytest navíc umožňuje přehledně zobrazovat výsledky. Pytest bude fungovat jako *main*, který bude volat jednotlivé moduly - třídy, které budou vzájemně nezávislé, tedy bude zajištěna rozšiřitelnost.

V první fázi budou existovat hlavní třídy:

- Reinstall
- Update
- Configure
- TestCollector

Jednotlivé třídy a jejich vztahy jsou uvedeny v následujícím diagramu tříd:



Obr. 4.1 Diagram tříd

4.1 Gramatika konfiguračního souboru

Konfigurační soubor určuje běh celého frameworku. Obsah konfiguračního souboru je před spuštěním kontrolován, aby odpovídal určité gramatice. Zde zapsáno v ABNF normě. [3]

```
configFile =     1*(reinstall/update/configure/
                 test_collector)
reinstall_str = "reinstall" *"_" STRING/NUMBER
update_str =     "update" *"_" STRING/NUMBER
configure_str = "configure" *"_" STRING/NUMBER
test_col_str =  "test_collector" *"_" STRING/NUMBER
reinstall =      "[reinstall" *(NUMBER/STRING) "]" CRLF 1*server
server =         "server" STRING/NUMBER "=" IP_ADDRESS CRLF
path =           "path=" PATH_TO_FILE CRLF
update =         "[update" *(NUMBER/STRING) "]" CRLF 1*server path
                 method
method =         "method=" "url"/"file" CRLF
configure =      "[configure" *(NUMBER/STRING) "]" CRLF 1*server
                 1*(1*script/config/license)
script =         "script=" PATH_TO_SCRIPT CRLF
config =         "config=" PATH_TO_FILE CRLF
license =        "license=" PATH_TO_FILE CRLF
test_collector = "[test_collector" *(NUMBER/STRING) "]" CRLF
                 collector generator *(flowsource)
collector =      "collector=" IP_ADDRESS CRLF
generator =      "generator=" IP_ADDRESS CRLF
flowsource =     "flowsource"  *"_"/STRING/NUMBER
                 "=" IP_ADDRESS CRLF
```

III. PROJEKTOVÁ ČÁST

V této části se budu zabývat konkrétními naimplementovanými částmi projektu. Zaměřil jsem se na vysvětlení důvodů pro vybrání konkrétních postupů, co se objevilo za problémy a jak je to připravené pro použití a další rozšiřování.

Dle návrhu je projekt napsán v jazyce Python3.7 a s využitím knihoven pytest a fabric. Důraz byl kladen na rozšiřitelnost a modulárnost. To, co vzniklo v rámci této diplomové práce, je základ, který plně umožňuje automatizovat výkonnostní testy Flowmon kolektoru, nicméně do budoucna je možné a žádané rozšířit o vše, co se týká hardware serverů od výkonnostních testů Flowmon sond po aktualizaci firmware.

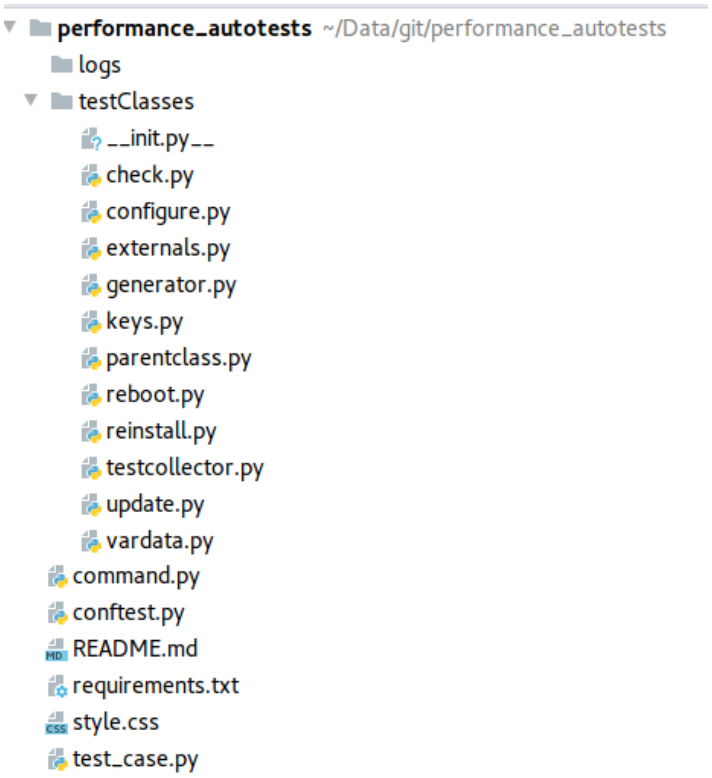
Nedílnou součástí implementace bylo i nastavení prostředí, což bohužel nelze plně automatizovat. V rámci přípravy došlo alespoň k prozkoumání a dokumentaci toho, co je potřeba.

5 Struktura projektu

Projekt se dá rozdělit do několika částí tak, jak bylo popsáno v analýze. V rámci tohoto rozdělení byl kladen důraz na modulárnost a znovupoužitelnost napříč celým projektem i s ohledem na rozšíření do budoucna. Jednotlivé soubory jsou popsány v další podkapitole.

5.1 Adresářová struktura

Pro každou část vznikla samostatná třída, několik tříd se sdílenými funkcemi a soubory obsahující sdílené a konfigurovatelné proměnné.



Obr. 5.1 Adresářová struktura

- adresář *logs* - Do této složky se ukládají logy, které vzniknou při běhu testu.
- adresář *testClasses* - Tato složka sdružuje jednotlivé třídy.
  - *init.py* - Soubor, který používá Python interpreter jako návěští. Díky tomu lze jednotlivé třídy zavádět jako balíčky.
  - *check.py* - Třída, která zajišťuje základní kontrolu vstupního konfiguračního souboru.

  - *configure.py* - Třída, kde je implementována část *Konfigurace*. Jedna z hlavních tříd, jejíž detailnější popis je uveden v jedné z následujících podkapitol.
  - *externals.py* - Soubor, který obsahuje sdílené proměnné. Konkrétně adresu repozitáře, kam se ukládají výsledky a adresu pomocného pracovního serveru
  - *generator.py* - Třída, která zajišťuje generování provozu pro test kolektoru
  - *keys.py* - Třída, která obstarává nahrání ssh klíčů na cílový server
  - *parentclass.py* - Hlavní třída, ze které dědí ostatní třídy, aby došlo k synchronizaci časových známek a dědění atributů
  - *reboot.py* - Třída zajišťující bezpečný reboot serveru
  - *reinstall.py* - Další z hlavních tříd, která implementuje část *Reinstalace*
  - *testcollector.py* - Hlavní třída pro část *Test kolektoru*
  - *update.py* - Hlavní třída pro část *Aktualizace*
  - *vardata.py* - Soubor obsahující proměnné, které jsou sdílené napříč celým projektem, například počáteční hodnota pro testy, povolené procento zahozených paketů nebo časové limity.

- *command.py* - Pomocná třída, která kompletně implementuje vzdálené spouštění příkazů. Využívá framework *fabric*. Jsou zde implementovány metody pro spuštění příkazů s administrátorskými právy, spuštění příkazů na pozadí, vzdálené nahrávání a stahování souborů.

- *conftest.py* - Konfigurační soubor pro framework *pytest*. Jsou zde implementovány metody pro vstupní parametry a způsob, jakým se bude vytvářet výsledný HTML report.

- *README.md* - Stručná dokumentace

- *requirments.txt* - Konfigurační soubor obsahující verze použitých knihoven. Je důležitý pro běh testů ve stále stejném prostředí.

- *style.css* - Jednoduché stylování výsledného HTML výstupu.

- *test_ case.py* - Hlavní třída, přes kterou framework *pytest* spouští jednotlivé kroky testu.

5.2 Nastavení prostředí

Pro spuštění samotného frameworku postačí linuxový operační systém, kde bude nainstalovaný Python3.7. Potřebné knihovny jsou v souboru *requirments.txt*, které lze doinstalovat:

```
pip install -r requirments.txt
```

Nicméně pro to, aby vše fungovalo, jsou další požadavky na celou infrastrukturu.

Pro část *Reinstall* je potřeba mít připravené prostředí pro PXE boot. To zahrnuje několik kroků:

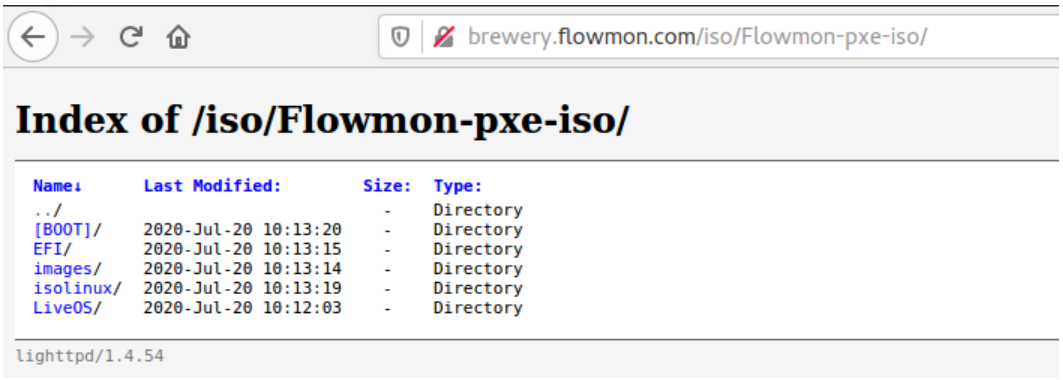
- Framework využívá pomocný server tzv. *worker*, odkud se provádějí IPMI příkazy. Adresa tohoto serveru je definována v souboru *externals.py* a používá se ze dvou důvodů:
  - Není potřeba instalovat nástroje pro IPMI na server odkud se framework spouští.
  - Lepší škálovatelnost.
- Přeinstalovávaný server musí podporovat bootování po síti. To standardně podporují všichni výrobci, ale je potřeba zajistit, přes které rozhraní se tak bude dít. Pokud server má více síťových karet, a obzvlášť od různých výrobců, stává se, že systém není schopen určit, přes které rozhraní zahájit bootování. Proto se doporučuje povolit tuto možnost v BIOS pouze pro rozhraní, které je připojené do sítě.
- Nastavit DHCP server. DHCP server musí podporovat PXE boot, a to tak, že při dotazu od PXE vrátí adresu TFTP serveru a cestu k PXE skriptu

  ```
  next-server X.X.X.X;
  filename "undionly.kpxe";
  ```

- Nastavit TFTP server. TFTP server obsahuje PXE skript "undionly.kpxe", jehož adresu posílá DHCP server. Tento skript mimo jiné ukazuje na iPXE skript, který řeší výsledné bootování. V iPXE skriptu je definováno, jaké jádro se má zavést, a cesta, odkud se stáhne instalační disk přes HTTP. iPXE skript vypadá následovně:

```
#!ipxe

menu Flowmon's iPXE menu
item flowmon-setup Flowmon OS
item
item ipxeshell iPXE Shell

choose os && goto ${os}

:ipxeshell
shell

:flowmon-setup
set base http://brewery.flowmon.com/iso/Flowmon-pxe-iso/
kernel ${base}/images/pxeboot/vmlinuz initrd=initrd.img
repo=${base}
initrd ${base}/images/pxeboot/initrd.img
boot || read void
```

- Poslední částí je HTTP server, kde je uložen instalační disk.



Obr. 5.2 HTTP server

Důležitou součástí bootování po síti je připravené instalační médium, které podporuje připojování disku přes síťové rozhraní. Jádro Flowmon OS toto implicitně nezahrnuje. Z důvodu, aby nemuselo dojít k znovusestavování jádra Flowmon OS se do instalačního média přidalo jádro z minimální instalace CentOS. Proces tak vypadá, že PXE stáhne nejprve CentOS jádro, které je připravené pro bootování po síti, to stáhne instalační médium Flowmon OS, kde už je obsaženo Flowmon jádro s tím se nainstaluje Flowmon OS.

Části *Update* a *Configure* nemají žádné požadavky na prostředí, pouze to, aby na cílovém serveru byly náhrány přihlašovací klíče, což se stane v části *Reinstall*.

Část *TestCollector*, kromě přihlašovacích klíčů potřebuje, aby generátor a testovaný kolektor měly mezi sebou dostupné spojení, ideálně aby byly v jedné síti, protože mezi nimi bude probíhat velký provoz, a aby byl dostupný repozitář definovaný v *externals.py*

5.3 Popis hlavních tříd

Všechny hlavní třídy dědí z třídy *ParentClass*. Tato třída zajišťuje zpracování vstupního konfiguračního souboru a pro ostatní třídy jsou už vstupy připravené.

- Proměnné:
  - items - soubor zpracovaných vstupních parametrů pro konkrétní část z konfiguračního souboru
  - path_to_ini - cesta ke vstupnímu souboru
- Metody:
  - Všechny následující metody slouží k získávání časových známek unifikovaným způsobem: *timestamp, getYear, getMonth, getDay, getHour, getMinute*

5.3.1 Třída Reinstall

Třída *Reinstall* zajišťuje reinstalaci pomocí PXE. Využívá k tomu vzdálený server tzv. *worker* definovaný v *externals.py* kde je nainstalovaný nástroj ipmitool. Po úspěšném přeinstalování cílového serveru se nahrají přihlašovací klíče.

- Proměnné:
  - pouze zděděné z *ParentClass*
- Metody:
  - doReinstall() - tato metoda zajišťuje reinstalaci.Připojí se vzdáleně na *worker* a nastaví PXE boot pomocí ipmi protokolu.

```
cmd = command.CommandClass(Externals.worker["worker_ip"])
# nastavit PXE boot
command = "ipmitool -I lanplus -H {} -U {} -P {}"
          "chassis bootdev pxe".format(
          self.items["server"],
          config['conf']['user'],
          config['conf']['pw'])
        \item getFlousources() - zpracuje zdroje z konfigura[U+FFFD][U+
```

  - putKeys() - nahraje přihlašovací klíče na vzdálený server. Využívá k tomu systémový nástroj ssh-copy-id.

5.3.2 Třída Configure

Třída *Configure* slouží ke konfiguraci serveru. Dochází zde ke vkládání Flowmon licence, bez které neběží kolektor, importování Flowmon konfigurace a poslední možností je spustit skript.

- Proměnné:
  - pouze zděděné z *ParentClass*
- Metody:
  - configure() - hlavní metoda, která na základě vstupní konfigurace volá příslušné metody
  - putLicense() - tato metoda na všechny servery z konfiguračního souboru vloží zadanou licenci
  - runScript() - metoda spustí na všech serverech přiložený skript. Skript může být v jakémkoli interpretovatelném jazyce, protože před spuštěním se přidává spustitelný příznak.

```
# pridat spustitelny priznak
_cmd = "/usr/bin/chmod +x /home/flowmon/{}".format(
        self.items["script"])
success, out = cmd.execute_admin_command(_cmd)
if not success:
    return False, out

# spustit skript
_cmd = "/home/flowmon/%s".format(self.items["script"])
success, out = cmd.execute_admin_command(_cmd)
if not success:
    return False, out
```

  - importXML() - na všechny specifikované servery nahraje přiloženou Flowmon konfiguraci ve formátu XML.

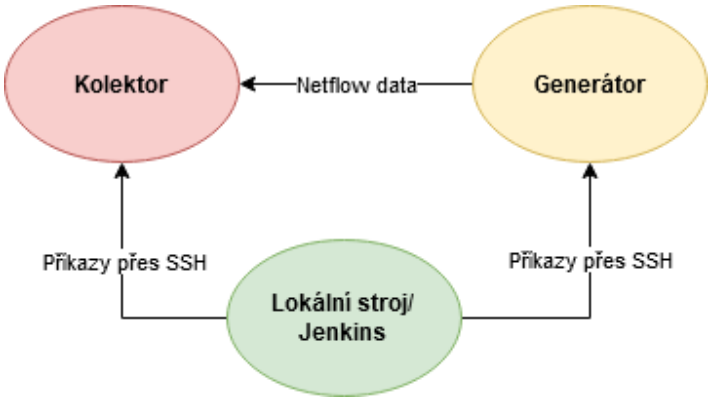
5.3.3 Třída Update

Třída *Update* řídí aktualizaci Flowmon OS. Balíčky pro novější verze lze přikládat ke konfiguračnímu souboru nebo definovat jejich adresu na nějaký HTTP/HTTPS server, odkud si je skript stáhne pomocí *curl*.

- Proměnné:
  - pouze zděděné z *ParentClass*
- Metody:
  - update() - tato metoda řídí aktualizaci Flowmon OS. Nainstaluje zadanou verzi na všechny servery definované v konfiguračním souboru.

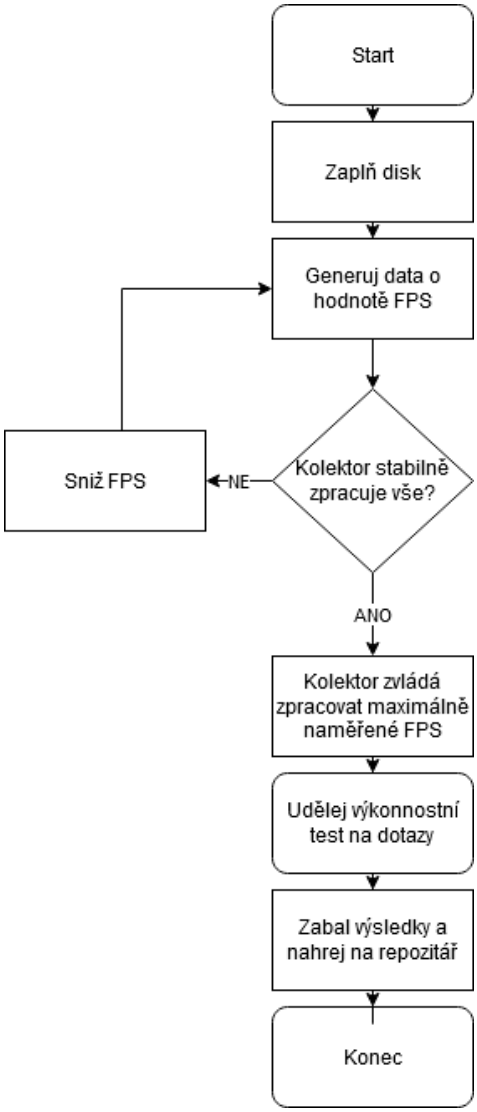
5.3.4 Třída TestCollector

Třída *TestCollector* realizuje výkonnostní testování Flowmon kolektoru. Pro svůj výkon potřebuje kolektor a generátor, které jsou definované ve vstupním konfiguračním souboru. S těmito vzdálenými servery komunikuje přes protokol SSH realizovaný metodami definovanými v souboru *command.py*, který je implementován pomocí knihovny *fabric*. Generátor pak posílá Netflow data na kolektor, kde je zpracuje, a pak uloží nfcapd daemon. Data, která si zpracuje, uloží, a která nestihne zpracovat nfcapd ukládá do logovacích souborů. Z těchto souborů je pak vyhodnocován výsledek.



Obr. 5.3 Dataflow diagram

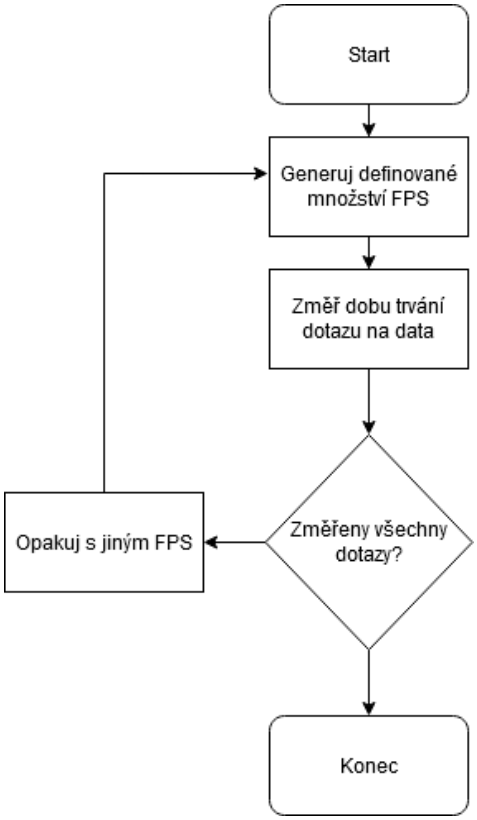
Pro výkonnostní testy se provádějí dva scénáře: maximální stabilně zpracovávané FPS a měření času zpracování analýzy nad poslední hodinou dat. Celý pracovní postup je znázorněn na následujících diagramech.



Obr. 5.4 Diagram vizualizující výkonnostní testování Flowmon kolektoru.

Na začátku celého testu se zaplní disk testovacími daty. Je to z důvodu, že kolektor musí zvládat pracovat dlouhodobě a na výkonu se nesmí projevit odmazávání historických dat.

Po zaplnění začíná cyklus, na jehož začátku se generuje určité množství FPS, po určitém intervalu se vyhodnotí, zda kolektor zvládá zpracovávat, a pokud ne, sníží se generované množství FPS. Vyhodnocování se uskutečňuje ve dvou intervalech, první je 10 minut, kdy se test poměrně rychle dostane k přibližnému limitu, a druhý interval je 30 minut, kdy test ověřuje, zda nedochází k výkyvům mezi jednotlivými 5 minutovými intervaly, po kterých nfcapd ukládá data. Tyto výkyvy obvykle znamenají, že kolektor nestíhá zpracovávat a data uloží až v následujícím intervalu, což se ale v průměru nemusí projevit. Detaily, jak dochází k vyhodnocování, je v popisu metody *isStable()*.



Obr. 5.5 Diagram měření času zpracování analýzy nad daty

Po nalezení maximální hodnoty, kterou je kolektor schopen zpracovat, se provádí měření času zpracování analýzy nad daty. Ve výchozím nastavení se tato analýza provádí nad 10 000, 50 000 a 100 000 FPS. Toto nastavení je v souboru *vardata.py*. Analýza probíhá tak, že se hodinu generuje určité množství FPS a nad těmito daty se provede analýza. Měří se čas, který trvá systému na zpracování příkazu pomocí systémového nástroje *time*.

Po provedení analýzy se sesbírají logovací soubory a další informační soubory, které by mohly být zajímavé pro další analýzu (detail hardware, seznam nainstalovaných balíčků atd, seznam generovaných toků, použitá konfigurace, zatížení CPU, RAM, disků a logovací soubory kolektoru) a spolu s výsledky se archivují na repozitáři (definovaný v *externals.py*).

Výsledky se nakonec předají knihovně *pytest*, která z nich zpracuje HTML report. Vzhled reportu je v souboru *style.css*, a data, která se v něm uloží, je definováno v souboru *conftest.py*. Tento report je možné zobrazit v systému Jenkins a následně odeslat e-mailem.

- Proměnné:
  - fps - aktuální generované množství FPS. Počáteční hodnota je získána ze souboru *vardata.py*
  - gen - IP adresa generátoru. Získaná z items
  - col - IP adresa testovaného kolektoru. Získaná z items
  - description - popis testu, používá se k identifikaci logovacích souborů. Získaná z items
  - std - povolené množství ztrát ve formě směrodatné odchylky. Získané ze souboru *vardata.py*
  - std_parc - povolené množství ztrát jednotlivých intervalů od dlouhodobého průměru. Získané ze souboru *vardata.py*
  - cpu - množství jader CPU, kvůli lepšímu přepočítání zátěže. Průměrné zatížení CPU nemusí být rozhodující, pokud jednotlivá jádra nestíhají. Získává se na začátku běhu testu přímo z dat kolektoru.
  - flowsources - pole zdrojů. Umožňuje simulovat posílání dat z více Flowmon sond, což pro některé testovací scénáře poskytuje reálnější data
- Metody:
  - doTest() - hlavní metoda v níž je definovaný celý testovací cyklus znázorněný v diagramech výše. V přílohách je podrobný diagram k celému cyklu této metody 1.
  - cleanLogs(ip) - metoda, která před spuštěním testů odrotuje logovací soubory. Pomocí vstupního parametru, ip adresy, lze čistit logovací soubory na kolektoru i generátoru.
  - readLogs(ip) - metoda čte logovací soubory. Pomocí vstupního parametru, ip adresy, lze čistit logovací soubory na kolektoru i generátoru. To jaké, logovací soubory se čistí, je definováno v *vardata.py*
  - gatherInfo(ip, data, filename) - metoda posbírá systémová data, která mohou být užitečná k další analýze. To, která data se sbírají, se předává vstupním parametrem *data* a parametrem *filename* se definuje název výstupního souboru. V průběhu testu se sbírají různá data a různé soubory. Jejich definice je v souboru *vardata.py*
  - getFlousources() - zpracuje zdroje z konfiguračního souboru a poskládá je do pole. Toto pole se pak dává na vstup generátoru, který podle toho generuje data.

– isStable(fps_part) - principem této metody je ověřit, zda kolektor zvládá zpracovávat testované množství dat. Ověřuje se, jestli průměrně zvládá zpracovat a poté zda nedochází k odchylkám. Vstupní parametr fps_part je pole 5minutových intervalů (tj. kolik FPS bylo zpracováno každých 5 minut).

```
average = int(sum(fps_part) / len(fps_part))
# prumer musi byt v itervalu +- odchylka
if average in range(int(fps*(1-std)),
                    int(fps*(1+std))):
    for f in fps_part:
        # kazda cast musi byt v intervalu +- odchylka
        if int(f) not in range(fps * (1-std_parc)),
                               fps * (1+std_parc))):
            self.fps = int(f)
            return False
    return True
else:
    # pokud neni stabilni -> sniz FPS
    self.fps = int(average)
    return False
```

– getLoad(ip) - metoda získá zatížení stroje definovaného vstupním parametrem ip. Využívá k tomu systémový nástroj top.

– doQueryTest() - metoda pro provedení testu na měření času analýzy. Pro všechny limity definované v *vardata.py* spouští metodu doQuery().

– doQuery() - metoda, která provede jedno měření času analýzy. Probíhá tak, že spustí generování definovaného množství FPS a po hodině provede měření.

– roundToFive(x) - pomocná metoda používaná pro zaokrohlování na nejbližší 5minutový interval.

– decreaseHour(x) - pomocná metoda, která vytvoří časový interval pro metodu doQuery(), aby bylo možné vytvořit analýzu pro poslední hodinu.

– checkConnection(ip1, ip2) - zkontroluje konektivitu mezi dvěmi servery (kolektor a generátor). Využívá k tomu systémový nástroj ping.

– gatherUsedProperties(ip, ip2) - tato metoda zabalí informace použité k testování (typ generovaných toků z generátoru, konfigurace kolektoru...).

– startCPULoad(ip) - metoda na zapnutí monitorování CPU.

– gatherCPULoad(ip) - metoda, která posbírá data z monitorování CPU.

– packAfterTest() - metoda pro zabalení výsledků a nahrání na repozitář.

5.4 Integrace do systému Jenkins

S integrací do systému Jenkins se počítalo od návrhu, proto byl celý projekt naimplementován tak, aby nebylo zapotřebí žádných zásahů, ale pouze vytvořit úkol[1)] v systému Jenkins.

Obr. 5.6 Nastavení Python prostředí

Při vytvoření úkolu je potřeba nastavit interpretaci projektu. Projekt byl napsán v jazyce Python3.7, proto je i v úkolu nastaveno virtuální prostředí Python3.7. Do tohoto prostředí se instalují potřebné knihovny, které jsou definované v souboru *requirments.txt*.

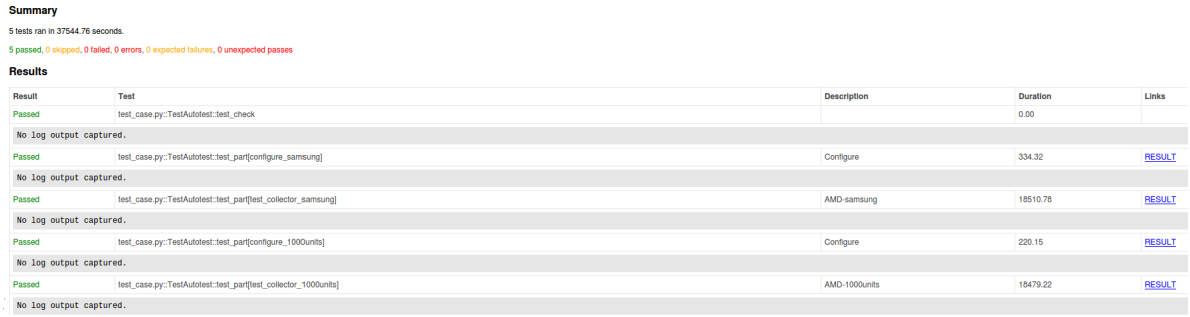
Obr. 5.7 Vstupní parametry pro test

[1)] Tzv. job. Jedná se o nějakou definovou parametrizovatelnou posloupnost příkazů.

Dále je potřeba zajistit předávání vstupních parametrů. Projekt bere na vstup konfigurační soubor. Pro tento účel je v úkolu připravené textové pole s připraveným příkladem, jak takový konfigurační soubor může vypadat. Ke konfiguračnímu souboru mohou přináležet další soubory důležité pro konfiguraci. Tyto soubory (pokud budou použity) se automaticky ukládají do pracovního adresáře úkolu.

Obr. 5.8 Nastavení HTML reportu

Posledním krokem integrace je nastavení vytváření HTML výstupu s výsledkem. HTML výstup pak může vypadat následovným způsobem. Nejjednodušší formou je pouhé oznámení výsledku pro každou část běhu s odkazem na detail výsledku.



Obr. 5.9 HTML report

6 Spuštění

Framework je napsán v jazyce Python3.7, pro spuštění je tedy potřeba mít prostředí s interpreterem pro Python3.7, nainstalované všechny knihovny z *requirments.txt* a vytvořený vstupní konfigurační soubor. Příkaz ke spuštění vypadá následovně:

```
pytest --config=konfiguracni_soubor.ini -s -v test_case.py
```

- *pytest* je framework a zároveň Python interpreter
- parametrem –config se předává vstupní konfigurační soubor
- parametr -s zapíná ukládání příkazů z příkazové řádky
- parametr -v zvyšuje množství informací, které projekt vypisuje při běhu
- soubor *test_ case.py* funguje jako *main* pro *pytest*

6.1 Požadavky

Spuštění lze provést jedním příkazem, nicméně pro správnou funčknost celého projektu je potřeba zajistit několik věcí:

- Python3.7 - osvědčený postup pro spouštění každého projektu napsaném v jazyce Python je vytvářet virtuální prostředí přes nástroj, který vytváří virtuální prostředí, virtualenvwrapper[1)]

  ```
  $ pip install virtualenvwrapper
  $ mkvirtualenv project_folder
  $ workon project_folder
  $ pip install -r requirments.txt
  ```

  Tímto způsobem se zajistí, že pro každý projekt jsou nainstalovány pravě ty knihovny, které jsou potřeba a nemůže dojít k chybám kvůli vazbám na jiné knihovny u jiných projektů.

  Tento požadavek je vyřešen v systému Jenkins, který si virtuální prostředí vytváří automaticky.

- Připravené prostředí pro PXE boot popsané v kapitole Nastavení prostředí 5.2 a instalační médium pro Flowmon OS.
- Zpřístupněný repozitář k ukládání výsledku, definovaný v *externals.py*
- Dostupné fyzické servery pro test kolektoru

[1)] Rozšíření pro virtualenv, nástroj pro správu virtuálních prostředí pro Python interpreter. `https://virtualenvwrapper.readthedocs.io/en/latest/`

7 Případová studie

V této kapitole bude dopodrobna rozebrán jeden běh projektu. Vstupní konfigurační soubor je v příloze 2 a zkrácený výstupní log celého běhu v příloze 3. Konfigurační soubor říká, že se bude provádět reinstalace serveru s IP adresou iDrac rozhraní 192.168.6.24, tento server následně dostane adresu 192.168.6.124 od DHCP serveru a výchozí uživatel operačního systému je flowmon (standartně na Flowmon OS).

Po reinstalaci následuje konfigurace serveru následovaná výkonnostním testem kolektoru, u kterého je definována adresa generátoru a přeinstalovaného kolektoru, popisek a verze testu. Toto se opakuje pro 3 scénáře, které se liší konfigurací kolektoru.

Spuštěno bylo pomocí systému Jenkins, který provedl příkaz:

```
py.test --config=config.ini --html test_result/result.html
--css style.css --junit-xml test_result/result.xml -s -v test_case.py
```

- parametr config určitě konfigurační soubor
- parametr html vytváří výsledný HTML report
- parametr css zajišťuje, ze kterého souboru se má vzít definice pro styly HTML dokumentu
- parametr junit-xml interně ukládá statistiky z běhů pro Jenkins
- ostatní parametry byly rozebrány v kapitole Spuštění 6

Prvním krokem je implicitně vždy kontrola vstupního konfiguračního souboru:

```
test_case.py::TestAutotest::test_check PASSED
```

Poté začne provádění podle konfiguračního souboru. Začíná se tedy reinstalací:

```
test_case.py::TestAutotest::test_part[reinstall]
[worker] run: ipmitool -I lanplus -H 192.168.6.24 -U usr
 -P pw chassis bootdev pxe
[worker] run: ipmitool -I lanplus -H 192.168.6.24 -U usr
 -P pw chassis power reset
[worker] run: ipmitool -I lanplus -H 192.168.6.24 -U usr
 -P pw chassis power reset
PASSED
```

Nastaví se PXE boot a restartuje server. Po restartu automaticky začíná instalace. Na konci instalace se objevuje hlášení "Instalace je dokončena, restartujte počítač".

Standartně se vždy čeká na podobné hlášení u každého operačního systému, proto aby nemuselo dojít k výraznější úpravě instalačního média, a tím i k nestandartnímu chování, bylo pouze experimentálně stanoveno, že po 20 minutách dojde k rebootu. Když server nastartuje, získá IP adresu od DHCP serveru a nahrají se na něj přihlašovací klíče. To se do logu neukládá z bezpečnostních důvodů.

Dalším krokem je konfigurace:

```
test_case.py::TestAutotest::test_part[configure_default]
Putting license license.key
[collector] Executing task 'put_file'

Putting XML configuration.xml
[collector] Executing task 'put_file'
[collector] Run import XML

Putting script script.sh
[collector] Executing task 'put_file'
[collector] run: /usr/bin/chmod +x /home/flowmon/script.sh
[collector] run: /home/flowmon/script.sh
PASSED
```

Následuje testování kolektoru. Na jehož začátku je ověření dostupnosti spojení mezi kolektorem a generátorem pomocí systémového nástroje ping.

```
test_case.py::TestAutotest::test_part[test_collector] [192.168.6.124]
[collector] run: ping -c 4 192.168.6.121

[generator] run: nfreplay_ng  -g -l -s 800000
                 -o ipfix,192.168.6.124,3000

[collector] restore_factory_settings.sh
[generator] run: nfreplay_ng  -g -l -s 800000
                 -o ipfix,192.168.6.124,3000
[collector] run: cat nfcapd.log |
grep "INFO  stats - stream_architecture_report_stats"

[collector] run: restore_factory_settings.sh
[generator] run: nfreplay_ng  -g -l -s 464805
                 -o ipfix,192.168.6.124,3000
```

```
[collector] run: cat nfcapd.log |
grep "INFO  stats - stream_architecture_report_stats"

[collector] run: /home/flowmon/restore_factory_settings.sh
[generator] run: nfreplay_ng  -g -l -s 464805
                 -o ipfix,192.168.6.124,3000
[collector] run: cat nfcapd.log |
grep "INFO  stats - stream_architecture_report_stats"
```

Z této části výstupu je vidět, že se generátor spustil celkem 4x.

- Poprvé při zaplňování disku
- Podruhé s výchozí hodnotou FPS, což kolektor nestíhal zpracovávat a snížilo se FPS na 464805
- Potřetí se sníženou FPS na první kratší časový interval
- Naposledy znovu se sníženou FPS na delší interval, aby se ověřilo, že kolektor zvládá zpracovávat i delší dobu bez výkyvů mezi jednotlivými intervaly.

Před každým spouštěním generátoru se pomocí skriptu restore_factory_settings.sh promazávají logovací soubory, aby nemohlo dojít k pomíchání výsledných dat.

Když se vyhodnotí kolik je výsledný limit posbírají se statistiky, systémové informace a statistiky zatížení jednotlivých jader procesoru.

```
[collector] run: cpuload.sh
[collector] run: echo "top -n 1 -b" &>> systemInfo.txt

[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-usedConfig.xml
<- /home/flowmon/usedConfig.xml

[generator] Executing task 'get_file'
[generator] download: logs/R5-2880 SSD-default-usedFlows.txt
<- /home/flowmon/flows.txt
[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-cpu0-load.log
<- /home/flowmon/cpu0-load.log
[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-cpu1-load.log
```

```
<- /home/flowmon/cpu1-load.log
[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-cpu2-load.log
<- /home/flowmon/cpu2-load.log
[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-cpu3-load.log
<- /home/flowmon/cpu3-load.log
[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-cpu4-load.log
<- /home/flowmon/cpu4-load.log
[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-cpu5-load.log
<- /home/flowmon/cpu5-load.log
```

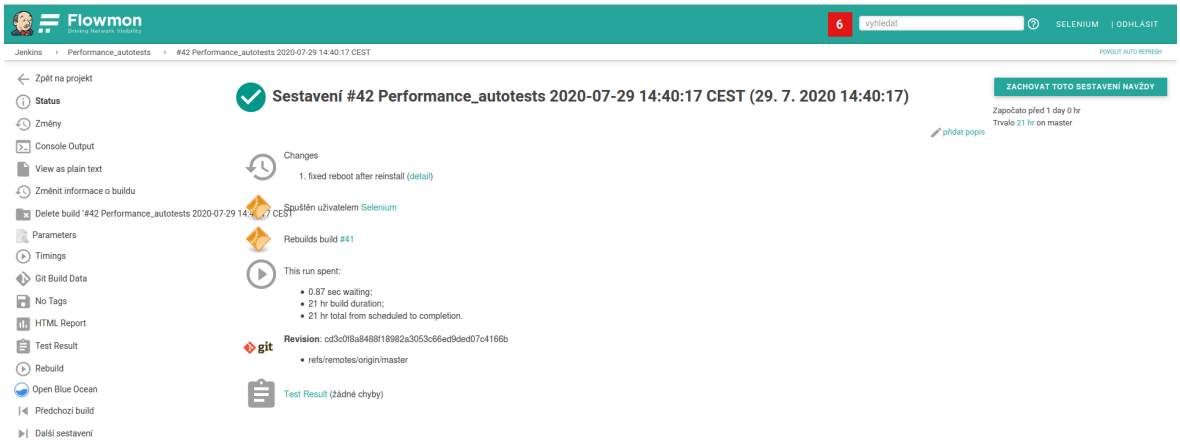
Po nalezení limitu probíhá měření času analýzy nad konkrétním množstvím dat. Zde znovu dochází ke spouštění generátoru a poté spuštění analýzy nad poslední hodinou dat a vyhodnocení pomocí systémového nástroje time.

```
[generator] run: nfreplay_ng -g -l -s 10000
                -o ipfix,192.168.6.124,3000
[collector] run:
time nfdump -M profiles-data/'live'/'192-168-6-121'_p3000
-R 2020/07/29/nfcapd.202007291705:2020/07/29/nfcapd.202007291805
-n '10' -S 'ip'/'bytes' -6 -o 'csv'

[generator] run: nfreplay_ng -g -l -s 50000
            -o ipfix,192.168.6.124,3000
[collector] run:
time nfdump -M profiles-data/'live'/'192-168-6-121'_p3000
-R 2020/07/29/nfcapd.202007291810:2020/07/29/nfcapd.202007291910
-n '10' -S 'ip'/'bytes' -6 -o 'csv'

[generator] run: nfreplay_ng -g -l -s 100000
            -o ipfix,192.168.6.124,3000
[collector] run:
timenfdump -M profiles-data/'live'/'192-168-6-121'_p3000
-R 2020/07/29/nfcapd.202007291915:2020/07/29/nfcapd.202007292015
-n '10' -S 'ip'/'bytes' -6 -o 'csv'
```

Posledním krokem je zabalení výsledků do archivu a nahrání na repozitář.

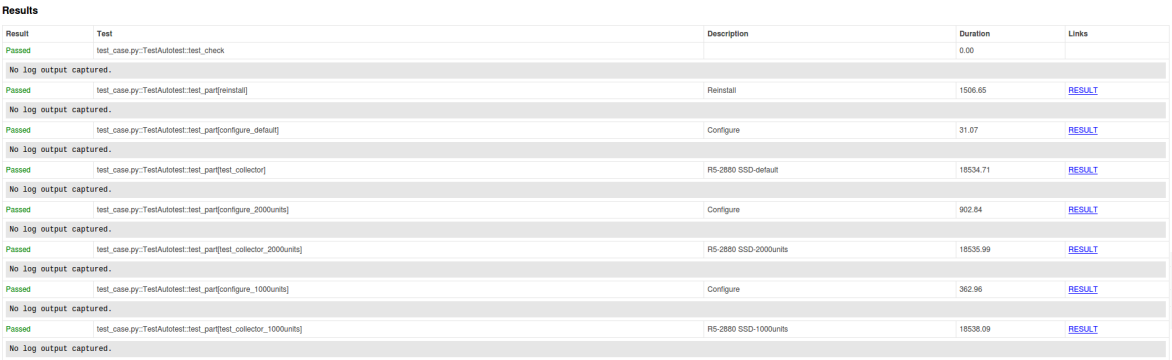
```
[repo] run: mkdir /home/flowmon/repo/performance/11.0.4
[repo] Executing task 'put_file'
[repo] put: R5-2880 SSD-default.zip
-> /home/flowmon/repo/performance/11.0.4/R5-2880 SSD-default.zip
PASSED
```

V systému Jenkins je výsledek graficky zobrazen, kromě dalšího je zde vidět, že celý test běžel 21 hodin, což plně dokazuje obrovskou časovou úsporu. Ruční provádění by zabralo téměř 3 pracovní dny, takto stačí spustit a čekat na výsledek.



Obr. 7.1 Sestavení výsledku v systému Jenkins

Po výsledku je možné prokliknout se na HTML report, který vypadá následovně:



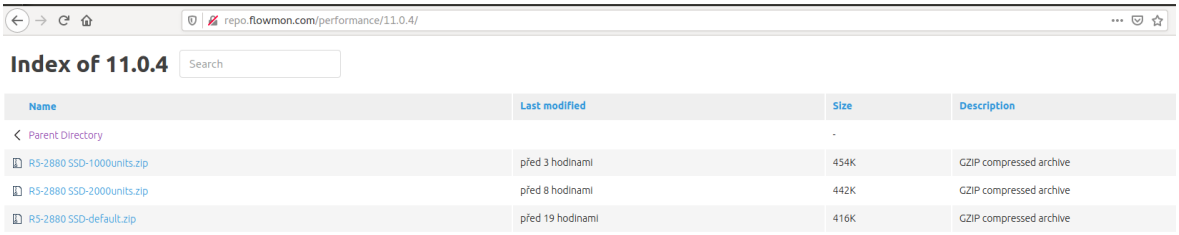
Obr. 7.2 HTML report v systému Jenkins

Jednotlivé výsledky pak obsahují detail meření společně s odkazem na repozitář, který obsahuje veškeré detaily, informace o systému a logovací soubory:

```
Description:R5-2880 SSD-default
Measured maximum stable FPS: 464805
1-hour data query over following FPS:
10000:
```

```
time nfdump -M profiles-data/'live'/'192-168-6-121'_p3000
-R 2020/07/29/nfcapd.202007291705:2020/07/29/nfcapd.202007291805
-n '10' -S 'ip'/'bytes' -6 -o 'csv'
...

Summary
flows,bytes,packets,avg_bps,avg_pps,avg_bpp
32914384,1844337262423343,3366123340501 4605,4471126055724,
10200386123199,0

real    0m5.806s
user    0m5.669s
sys     0m0.050s
...

Details:
http://repo.flowmon.com/performance/11.0.4/R5-2880 SSD-default.zip
```

Na repozitář se podle verze ukládají všechny výsledky testů.

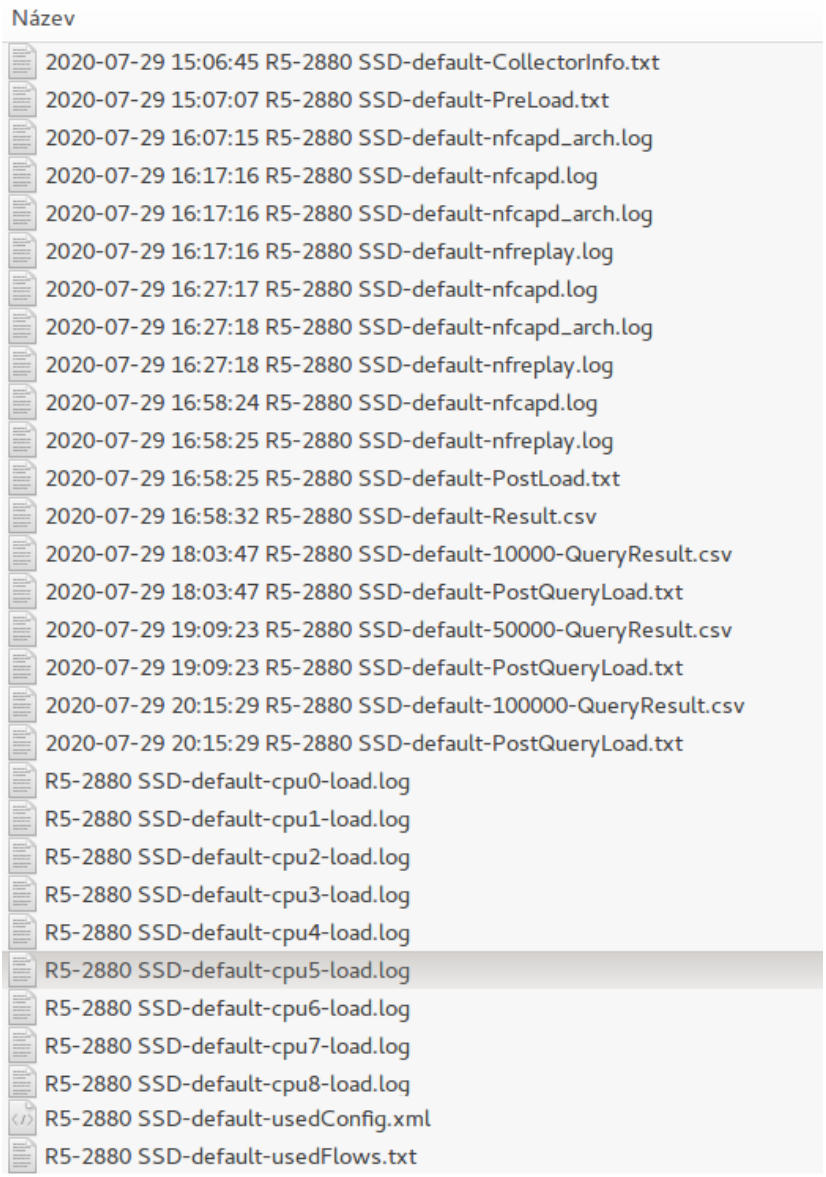


Obr. 7.3 HTML report

Jednotlivé archivy obsahují následující soubory:

- CollectorInfo.txt - Informace o testovaném kolektoru
- PreLoad.txt - systémové zatížení před spuštěním testu
- nfcapd_arch.log - záloha logovacího souboru pro nfcapd - proces kolektoru
- nfcapd.log - logovací soubor pro nfcapd pro poslední časové období (mezi čištěním logovacích souborů)
- nfreplay.log - logovací soubor pro generátor
- PostLoad.txt - systémové zatížení po spuštění testu (zaznamenává celý průběh testu)

- Result.csv - strojově zpracovatelný výsledek
- X-QueryResult.csv - výsledek pro měření analýzy
- cpuX-load.log - logovací soubor pro vytížení jednotlivých jáder procesoru v průběhu testu
- usedConfig - použitá konfigurace kolektoru pro test
- usedFlows - použité toky, které použil generátor pro test



Obr. 7.4 HTML report

ZÁVĚR

Podařilo se vytvořit framework pro automatizaci výkonnostních testů nástrojů Flowmon. V rámci diplomové práce bylo naimplementováno automatizace několika částí: reinstalace, konfigurace, aktualizace a test kolektoru. Framework je zaintegrován do systému Jenkins.

V závěru se ukázalo jako nejproblematičtější část celého frameworku nastavení prostředí pro PXE boot. Problém spočíval v tom, že Flowmon OS není úplně standartní CentOS a tedy jeho jádro nepodporuje bootování po síti.S touto situací bylo zapotřebí se vypořádat, což se nakonec podařilo přidáním druhého jádra k instalačnímu médiu.

Jak bylo rozebráno v případové studii manuálně otestování jednoho typu kolektoru trvá řádově 2-3 dny, pomocí frameworku je to i s přípravou konfiguračního souboru 10 minut práčě a následně je výsledek k dispozici za 20 hodin bez jakékoli další manipulace. Ve výsledku manuálně trvalo otestovat všechny typy kolektorů přibližně dva pracovní týdny, nyní stačí spustit test mimo pracovní dny bez potřeby dalších manuálních zásahů. Tato automatizace umožňuje rozšiřování testovacích scénářů, čímž se lze dostat k relevantnějším výsledkům. Doposud bylo totiž velice pracné nasimulovat data a konfiguraci, která by odpovídala reálnému provozu.

Framework tak jak byl navrhnut a implementován má velice komplexní využití. Kromě kompletního testu kolektoru mohou framework využít testeři i vývojáři na reinstalaci HW serverů bez nutnosti fyzické přítomnosti v serverovně. Aktualizace OS nebo konfigurace serveru lze také provést jednoduchým konfiguračním souborem a jedním kliknutím v systému Jenkins.

Do budoucna je framework připravený k rozšiřování. Jednou z logických možností je rozšíření o výkonnostní testy Flowmon sondy, kde bude potřeba vyřešit automatizované přepojování síťových rozhraní. K tomu by bylo potřeba pořídit nějaký programovatelný switch, který bude automaticky umožňovat propojování různých síťových rozhraní na základě konkrétní konfigurace.

Další možností do budoucna je hlubší integrace do systému Jenkins, např. vytvořením samostatných úkolů pro jednotlivé části projektu tak, aby uživatel nemusel vůbec znát syntaxi konfiguračního souboru. Tak může vzniknout úkol pro reinstalaci, kde uživatel napíše pouze IP adresu, jehož stroj chce přeinstalovat a úkol konfiguraci pro framework vygeneruje sám.

SEZNAM POUŽITÉ LITERATURY

[1] B. Trammell B. Claise and P. Aitken. Specification of the ip flow information export (ipfix) protocol for the exchange of flow information. STD 77, RFC Editor, September 2013. `http://www.rfc-editor.org/rfc/rfc7011.txt`.

[2] B. BEIZER. *Software Testing Techniques*. Dreamtech, 2003.

[3] P. OVERELL D. CROCKER. Augmented bnf for syntax specifications: Abnf. STD 68, RFC 5234, January 2008.

[4] Grig Gheorghiu. Performance vs. load vs. stress testing, 2005. Dostupné: http://agiletesting.blogspot.com/2005/02/performance-vs-load-vs-stress-testing.html. Navštíveno: 8.3.2020.

[5] Intel Corporation. Preboot execution environment (pxe) specification, 1999. Dostupné: https://web.archive.org/web/20110524083740/http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf. Navštíveno: 7.7.2020.

[6] PAUL C. JORGENSEN. *Software testing - A craftsman s Approach*. CRC Press, 2014.

[7] R. PATTON. *Software testing*. Indianapolis : Sams Publishing, 2006.

[8] Slaight T. Advances in intelligent platform management: Introducing the new ipmi v2.0 spec, 004. Dostupné: https://www.intel.com/content/www/us/en/servers/ipmi/new-ipmi-specifications.html. Navštíveno: 11.7.2020.

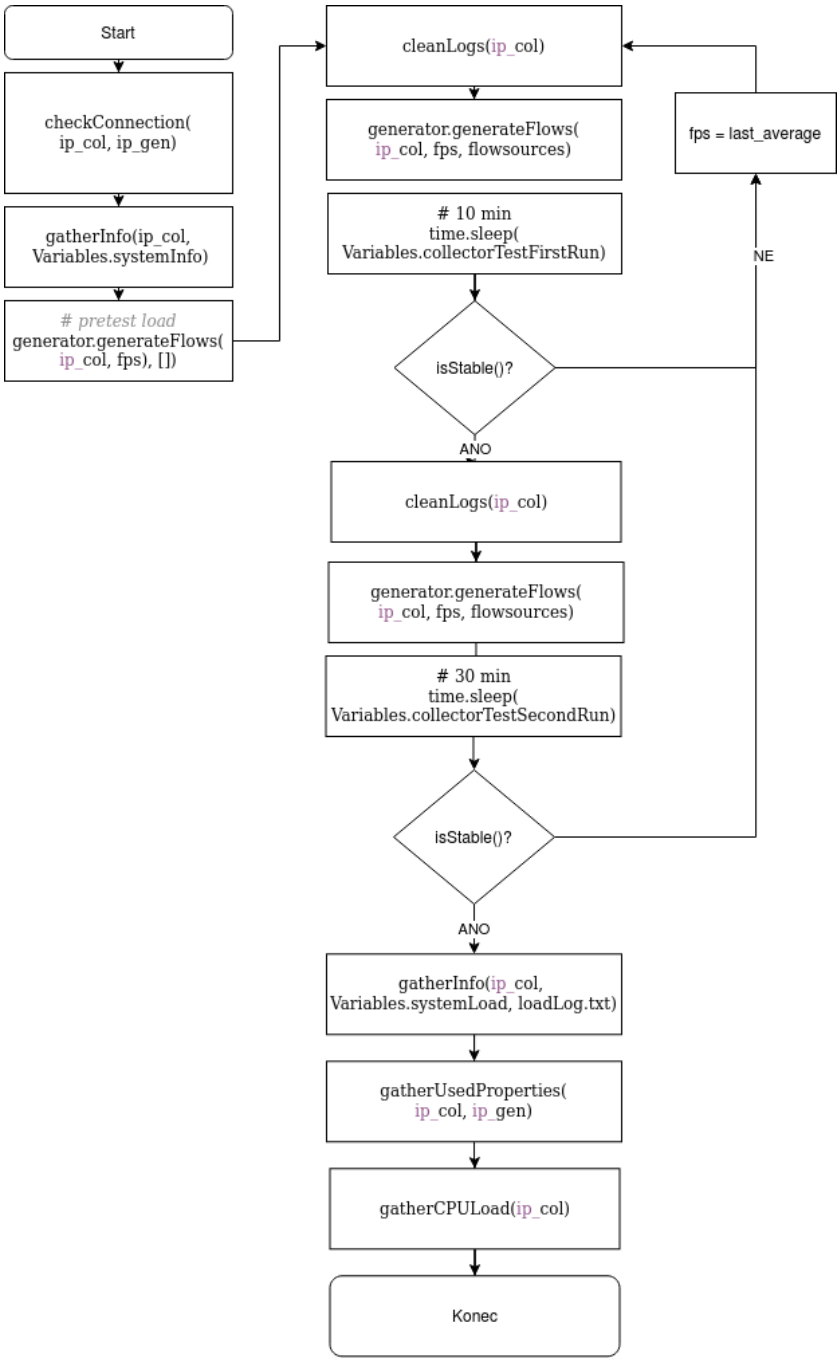
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ABNF	Metajazyk používaný pro definici komunikačních protokolů
BMC	Kontrolér správy základní desky
CPU	Procesor
DHCP	Protokol pro automatickou konfiguraci zařízení připojených do sítě
FPS	Toky za sekundu
FTP	Protokol pro přenos souborů mezi zařízeními v síti
GUI	Grafické uživatelské rozhraní
HTML	Značkovací jazyk používaný pro tvorbu webových stránek
HTTP	Protokol pro komunikaci s webovými servery.
iPXE	Otevřená rozšířená implementace PXE
iSCSI	Standard pro sdílení diskových uložišť přes síť.
IPFIX	Rozšiřující protokol pro Netflow standard.
IPMI	Abstraktní rozhraní pro správu HW zařízení.
JSON	Formát pro serializaci strukturovaných dat
PXE	Technologie pro bootování zařízení ze sítě
RAID	Technologie ukládání dat
RAM	Operační paměť
SSH	SZabezpečený komunikační protokol pro komunikaci v síti.
TCP	Transportní protokol
TFTP	Zjednodušená forma FTP
XML	Obecný značkovací jazyk pro serializaci dat
YAML	Formát pro serializaci strukturovaných dat

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I. WORKFLOW DIAGRAM - DOTEST()



Obr. 1.1 Workflow diagram - doTest()

PŘÍLOHA P II. KONFIGURAČNÍ SOUBOR PRO PŘÍPADOVOU STUDII

```
[reinstall]
idrac = 192.168.6.24
server = 192.168.6.124
default_user = flowmon

[configure_default]
Server = 192.168.6.124
Config = configuration.xml
script = script.sh
license = license.key

[test_collector]
Collector = 192.168.6.124
Generator = 192.168.6.121
description = R5-2880 SSD-default
version = 11.0.4

[configure_2000units]
Server = 192.168.6.124
Config = configuration3.xml

[test_collector_2000units]
Collector = 192.168.6.124
Generator = 192.168.6.121
description = R5-2880 SSD-2000units
version = 11.0.4

[configure_1000units]
Server = 192.168.6.124
Config = configuration4.xml

[test_collector_1000units]
Collector = 192.168.6.124
Generator = 192.168.6.121
description = R5-2880 SSD-1000units
version = 11.0.4
```

PŘÍLOHA P III. ZKRACENÝ VÝSTUP Z PŘÍPADOVÉ STUDIE

```
New run name is '#42 Performance_autotests 2020-07-29 14:40:17 CEST'
Copying file to configuration.xml
Copying file to script.sh
Copying file to license.key
Copying file to configuration2.xml
Copying file to configuration3.xml

[workspace] $ python3.7 jobs/87c6ee60/virtualenv.py
jobs/87c6ee60/virtualenvs/d41d8cd9
Using base prefix '/usr/local'
New python executable in
jobs/87c6ee60/virtualenvs/d41d8cd9/bin/python3.7
Also creating executable in
jobs/87c6ee60/virtualenvs/d41d8cd9/bin/python
Installing setuptools, pip, wheel...done.

+ pip3 install --upgrade pip
Requirement already up-to-date: pip in
jobs/87c6ee60/virtualenvs/d41d8cd9/lib/python3.7/site-packages (20.2)
+ pip3 install -r requirements.txt

+ py.test --config=config.ini --html test_result/result.html
--css style.css --junit-xml test_result/result.xml -s -v
test_case.py
============================= test session starts ==============================
platform linux -- Python 3.7.2, pytest-6.0.0, py-1.8.1,
pluggy-0.13.1 -- jobs/87c6ee60/virtualenvs/d41d8cd9/bin/python3.7
cachedir: .pytest_cache
metadata: {'Python': '3.7.2', 'Platform': 'Linux-with-centos-Core',
'Packages': {'pytest': '6.0.0', 'py': '1.8.1', 'pluggy': '0.13.1'},
'Plugins': {'metadata': '1.10.0', 'html': '2.1.1'},
'BUILD_NUMBER': '42', 'BUILD_ID': '42',
'BUILD_URL': 'jenkins/job/Performance_autotests/42/',
'NODE_NAME': 'master', 'JOB_NAME': 'Performance_autotests',
'BUILD_TAG': 'jenkins-Performance_autotests-42',
'WORKSPACE': 'jobs/Performance_autotests/workspace',
'GIT_URL': 'git@git:performance_autotests.git'}
```

```
collecting ... collected 8 items

test_case.py::TestAutotest::test_check
PASSED
test_case.py::TestAutotest::test_part[reinstall]
[worker] run: ipmitool -I lanplus -H 192.168.6.24 -U usr
-P pw chassis bootdev pxe
[worker] run: ipmitool -I lanplus -H 192.168.6.24 -U usr
-P pw chassis power reset
[worker] run: ipmitool -I lanplus -H 192.168.6.24 -U usr
-P pw chassis power reset
PASSED

test_case.py::TestAutotest::test_part[configure_default]
Putting license license.key
[collector] Executing task 'put_file'
[collector] run: /usr/bin/chmod 644 /etc/flowmon/license

Putting XML configuration.xml
[collector] Executing task 'put_file'
[collector] Run import XML

Putting script script.sh
[collector] Executing task 'put_file'
[collector] run: /usr/bin/chmod +x /home/flowmon/script.sh
[collector] run: /home/flowmon/script.sh
PASSED

test_case.py::TestAutotest::test_part[test_collector] [192.168.6.124]
[collector] run: ping -c 4 192.168.6.121

[generator] run: nfreplay_ng  -g -l -s 800000
                -o ipfix,192.168.6.124,3000

[collector] restore_factory_settings.sh
[generator] run: nfreplay_ng  -g -l -s 800000
                -o ipfix,192.168.6.124,3000
```

```
[collector] run: cat nfcapd.log |
grep "INFO  stats - stream_architecture_report_stats"

[collector] run: restore_factory_settings.sh
[generator] run: nfreplay_ng  -g -l -s 464805
                 -o ipfix,192.168.6.124,3000

[collector] run: cat nfcapd.log |
grep "INFO  stats - stream_architecture_report_stats"

[collector] run: /home/flowmon/restore_factory_settings.sh
[generator] run: nfreplay_ng  -g -l -s 464805
                 -o ipfix,192.168.6.124,3000
[collector] run: cat nfcapd.log |
grep "INFO  stats - stream_architecture_report_stats"

[collector] run: cpuload.sh
[collector] run: echo "top -n 1 -b" &>> systemInfo.txt

[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-usedConfig.xml
<- /home/flowmon/usedConfig.xml

[generator] Executing task 'get_file'
[generator] download: logs/R5-2880 SSD-default-usedFlows.txt
<- /home/flowmon/flows.txt
[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-cpu0-load.log
<- /home/flowmon/cpu0-load.log
[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-cpu1-load.log
<- /home/flowmon/cpu1-load.log
[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-cpu2-load.log
<- /home/flowmon/cpu2-load.log
[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-cpu3-load.log
<- /home/flowmon/cpu3-load.log
```

```
[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-cpu4-load.log
<- /home/flowmon/cpu4-load.log
[collector] Executing task 'get_file'
[collector] download: logs/R5-2880 SSD-default-cpu5-load.log
<- /home/flowmon/cpu5-load.log

[generator] run: nfreplay_ng  -g -l -s 10000
                 -o ipfix,192.168.6.124,3000
[collector] run:
time nfdump -M profiles-data/'live'/'192-168-6-121'_p3000
-R 2020/07/29/nfcapd.202007291705:2020/07/29/nfcapd.202007291805
-n '10' -S 'ip'/'bytes' -6 -o 'csv'

[generator] run: nfreplay_ng  -g -l -s 50000
            -o ipfix,192.168.6.124,3000
[collector] run:
time nfdump -M profiles-data/'live'/'192-168-6-121'_p3000
-R 2020/07/29/nfcapd.202007291810:2020/07/29/nfcapd.202007291910
-n '10' -S 'ip'/'bytes' -6 -o 'csv'

[generator] run: nfreplay_ng  -g -l -s 100000
            -o ipfix,192.168.6.124,3000
[collector] run:
timenfdump -M profiles-data/'live'/'192-168-6-121'_p3000
-R 2020/07/29/nfcapd.202007291915:2020/07/29/nfcapd.202007292015
-n '10' -S 'ip'/'bytes' -6 -o 'csv'

[repo] run: mkdir /home/flowmon/repo/performance/11.0.4
[repo] Executing task 'put_file'
[repo] put: R5-2880 SSD-default.zip
-> /home/flowmon/repo/performance/11.0.4/R5-2880 SSD-default.zip
PASSED
...
```

```
========== 8 passed in 77642.77s (21:34:02) ==========

[htmlpublisher] Archiving HTML reports...
[htmlpublisher] Archiving at BUILD level
jobs/Performance_autotests/workspace/test_result to
jobs/Performance_autotests/builds/42/htmlreports/HTML_20Report
Recording test results
Finished: SUCCESS
```

PŘÍLOHA P IV. VÝSLEDEK TESTU Z PŘÍPADOVÉ STUDIE

```
Description:R5-2880 SSD-default
Measured maximum stable FPS: 464805
1-hour data query over following FPS:
10000:
time nfdump -M profiles-data/'live'/'192-168-6-121'_p3000
-R 2020/07/29/nfcapd.202007291705:2020/07/29/nfcapd.202007291805
-n '10' -S 'ip'/'bytes' -6 -o 'csv'
...

Summary
flows,bytes,packets,avg_bps,avg_pps,avg_bpp
32914384,1844337262423343,3366123340501 4605,4471126055724,
10200386123199,0

real    0m5.806s
user    0m5.669s
sys     0m0.050s
50000:
time nfdump -M profiles-data/'live'/'192-168-6-121'_p3000
-R 2020/07/29/nfcapd.202007291810:2020/07/29/nfcapd.202007291910
-n '10' -S 'ip'/'bytes' -6 -o 'csv'
...

Summary
flows,bytes,packets,avg_bps,avg_pps,avg_bpp
164569152,9221529882099082,168303383662801140,22355339120472,
51001288086272,0

real    0m28.715s
user    0m28.435s
sys     0m0.139s

100000:
time nfdump -M profiles-data/'live'/'192-168-6-121'_p3000
-R 2020/07/29/nfcapd.202007291915:2020/07/29/nfcapd.202007292015
-n '10' -S 'ip'/'bytes' -6 -o 'csv'
...
```

Summary
flows,bytes,packets,avg_bps,avg_pps,avg_bpp
329140418,18443176587402583,336608927862547286,44710622731587,
102002458134206,0

real 0m58.172s
user 0m57.710s
sys 0m0.250s

Details: repo.flowmon.com/performance/11.0.4/R5-2880 SSD-default.zip

PŘÍLOHA P V. MANUAL (PRO POTŘEBY ZADAVATELE V AJ)

Performance Autotests Framework for Performance - a tool for configuring, running and evaluating performance tests.

Table of Contents

How It Works

Framework automatize all parts needed for performance tests.

- Server reinstall
- Servers interconnection (for probe testing) - TODO
- Update
- Configure
- Perform collector test
- Perform probe test - TODO

All steps are completely independent, so any combinations can be used in configuration file and there is no mandatory sequence. But it's logical to upload license with "configure" and then try to "update" package etc. Tests are configured by ini file in following format.

```
[reinstallX]
idrac = iDrac ip
server = server ip
default_user = user

[configureX]
Server1 = ip
license = license.key
```

```
[updateX]
method = url/file
Server = ip
package = path_on_repo/path_to_file

[configureY]
Server1 = ip
Config = config.xml
script = script.sh

[test_collectorX]
Collector = ip
Generator = ip
description = e.g. "R5-3000 Pro"
version = version
```

Notice: X can be anything, but it has to be unique for every step, and it's optional.

Detailed steps description

Every step is fully independent. X in section can be anything, but it has to be unique for every step, and it's optional. Steps are run in sequence from top to bottom.

[ReinstallX]

Reinstall defined server via PXE boot (iso is fixed to newest flowmon-os). Can contain multiple servers.

```
idrac = iDrac ip
server = server ip
default_user = user
```

[ConfigureX]

Configures defined server/s. On every server it puts license.key, configuration.xml and runs script.sh. Path to files is relative to configuration file. Name of used file has to be same as name used in configuration file.

```
serverX = ip
config = configuration.xml
script = script.sh
license = license.key
```

[UpdateX]

Updates defined server/s. One method is by url, then script automatically downloads

it. Second methond is to put path to local file, then script upload it to server via SCP.

```
method = url/file
serverX = ip
package = url_on_repo/path_to_file
```

[test_collectorX]

This step fully controls collector performance. Generator uses nfreplay_ng. After test it flushes used flow-set to result archive. Limits like "allowedDrops", "startingFPS", etc. are defined in vardata.py. Firstly it tries to find the maximum stable FPS collector can handle. Then generator generates exact FPS (defined by "queryLoad"in vardata.py) and runs query for last hour.

The result is send to http://repository/performance/"version"/"description".zip. Repository is defined in externals.py.

```
Collector = ip
Generator = ip
description = e.g. "R5-3000 Pro"
version = version
```

Repo example: http://repo.flowmon.com/performance/11.0.0/R5-3000

How to run

Tests can be easily runned by Jenkins. If you don't know credentials just ask your QA best buddy. In this job you just run Build with Parameters. Most important is parameter config where you put your configuration file. Other parameters are files which can be used in configuration file. These files are automatically renamed to configuration.xml, script.sh and license.key.

Examples

```
# Basic Performance collector test.

[configure]
Server1 = 192.168.6.119
Server4 = 192.168.6.115
Server5 = 192.168.6.124
Config = collectorPerformance_config.xml
script = install_hwinfo.sh

[test_collector]
Collector = 192.168.6.119
Generator = 192.168.6.121
```

```
description = R5-3000 Pro
version = 11.0.0

[test_collector2]
Collector = 192.168.6.115
Generator = 192.168.6.121
description = R6-48000 Pro
version = 11.0.0

[test_collector3]
Collector = 192.168.6.124
Generator = 192.168.6.121
description = R5-2880 SSD
version = 11.0.0

# Update

[configure]
Server1 = 192.168.6.245
license = IFC.key

[update1]
method = curl
Server = 192.168.6.245
package = http://repo.flowmon.com/flowmon_updates/
Flowmon_released/flowmon-v10.02.08.tar.gz

[update2]
method = curl
Server = 192.168.6.245
package = http://repo.flowmon.com/flowmon_updates/
Flowmon_released/flowmon-v10.03.06.tar.gz

[update3]
method = curl
Server = 192.168.6.245
package = http://repo.flowmon.com/flowmon_updates/
dev_flowmon-v11.00.00.tar.gz
```