VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Ondřej Gajdušek

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

VIRTUÁLNÍ SERVERY S OPERAČNÍM SYSTÉMEM FEDORA
VIRTUAL SERVERS WITH FEDORA OPERATING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Ondřej Gajdušek**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. Dan Komosný, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Ondřej Gajdušek ***ID:*** 186505
Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Virtuální servery s operačním systémem Fedora

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s aplikací vyvíjenou na Ústavu telekomunikací pro správu virtuálních serverů s OS Fedora (linuxová distribuce). Tyto servery slouží pro experimentální vývoj a výzkum v oblasti síťové komunikace. Projekt je dostupný na adrese pypi.org/project/plbmng/. Aplikaci rozšiřte o distribuci experimentálního software na vybrané servery. Výběr serverů pro distribuci proveďte automaticky podle účelu experimentu. Zajistěte vzdálené spuštění software ve stanovený čas. Vytvořený zdrojový kód včetně anglického popisu vystavte pod licencí MIT na repositáři pypi.org.

DOPORUČENÁ LITERATURA:

[1] Linux Dokumentační projekt. 4. vyd. Computer Press, 2008. 1336 s. ISBN: 978-80-251-1525-1.

[2] PILGRIM, M. Ponořme se do Python(u) 3. CZ.NIC, 2010. 435 s. ISBN: 978-80-904248-2-1.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: prof. Ing. Dan Komosný, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Aplikace PlanetLab Server Manager, jíž se tato práce zaobírá, slouží pro zjednodušení správy projektů realizovaných v rámci experimentální distribuované sítě PlanetLab. V práci je představena síť PlanetLab a její infrastruktura. Text seznamuje čtenáře s aplikací PlanetLab Server Manager, zkráceně plbmng a představuje mu její aktuální stav a implementovanou funkcionalitu. Práce se zaměřuje především na implementaci funkcionality distribuce software a jeho plánovaného spuštění v zadaný čas. Dále je představen experiment, na němž je demonstrováno reálné použití implementované funkcionality. Výsledný kód aplikace je zveřejněn ve veřejném repozitáři na platformě GitLab pod licencí MIT. Aplikace byla také zveřejněna v repozitáři Python balíčků PyPI.

KLÍČOVÁ SLOVA

Linux, plbmng, PlanetLab Server Manager, Python, naplánované úlohy, PlanetLab, Virtualizace, SSH

ABSTRACT

This thesis deals with the PlanetLab Server Manager application whose aim is to simplify application development within the experimental distributed network PlanetLab. Thesis presents the PlanetLab network and describes its infrastructure. Application PlanetLab Server Manager, shortly abbreviated as plbmng, is described, its current state is evaluated and the existing functionality is presented to the reader. Further work focuses mainly on the implementation of the new functionality that is a software distribution and job scheduling to run the software at the specified time. The last part of the work presents an experiment that demonstrates a real-life usage of the newly added functionality. Work results are published at the public repository on the GitLab platform. The application was published at the package index for Python Packages – PyPI.

KEYWORDS

Linux, plbmng, PlanetLab Server Manager, Python, task scheduling, PlanetLab, Virtualization, SSH

GAJDUŠEK, Ondřej. *Virtuální servery s operačním systémem Fedora*. Brno, 2021, 62 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: prof. Ing. Dan Komosný, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Virtuální servery s operačním systémem Fedora“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu prof. Ing. Danu Komosnému Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Vývoj aplikací a služeb v dnešní době může být náročnou úlohou. Zvláště v situaci, kdy je očekáváno, že aplikace bude distribuována na vícero stanic rozložených napříč kontinenty. V této práci popisovaná síť PlanetLab nabízí prostředí pro testování takovýchto distribuovaných systémů.

Hlavní část práce se věnuje rozšíření aplikace pro správu vzdálených serverů v celosvětové experimentální síti PlanetLab. Rozšiřovaná aplikace s názvem Planetlab server manager (plbmng) je implementována v jazyce Python a slouží k zjednodušení práce se stejnojmennou sítí PlanetLab. Hlavním účelem aplikace je monitorování serverů, zjišťování jejich hardwarové a softwarové konfigurace a přístup k nim a zjednodušit tak práci uživateli při vývoji distribuovaných aplikací.

Cílem této práce je rozšíření existující funkcionality aplikace o distribuci software a spouštění naplánovaných úloh na cílových serverech s možností sběru výsledků. Práce navazuje na existující implementaci aplikace, zpracovanou v rámci závěrečných prací Ivana Andrašova [9] Filipa Šuby [13, 12] a Martina Kačmarčíka [10]. Při implementaci byl kladen důraz na uživatelskou přívětivost a použitelnost celého řešení.

Práce je rozdělena do logických celků, přičemž v první části je popsána síť PlanetLab, představena její terminologie, způsob virtualizace systémů v síti a následně je popsán operační systém Linux, na němž PlanetLab staví.

Další část popisuje stav a funkce aplikace plbmng před provedením rozšíření. V části popisující rozšíření aplikace jsou navrženy dva způsoby řešení, z nichž jedno bylo vybráno. Detailně je popsán způsob plánování a spouštění vzdálených úloh a veškeré implementační kroky, k nimž bylo přistoupeno. Jednotlivé implementované funkce aplikace jsou prezentovány a jejich použití demonstrováno na konkrétních příkladech. Dále jsou popsána vylepšení aplikace, ke kterým v průběhu jejího vývoje došlo, a důvod implementace. Zmíněn je také způsob zveřejnění konečné implementace aplikace ve veřejných repozitářích na platformách GitLab a *Python Package Index* (PyPI). Aplikace je zveřejněna pod licencí *Massachusetts Institute of Technology* (MIT). Práce je zaměřena také na praktické použití aplikace, včetně demonstrace v reálném scénáři. Poslední část práce představuje celkový pohled na aplikaci, logické členění jejích modulů a implementační detaily.

V závěru jsou navržena zlepšení aplikace, která mají za cíl zvýšení přehlednosti a zdokonalení struktury kódu, přidání abstrakce pomocí tříd a také zlepšení uživatelské přívětivosti.

1 Experimentální síť PlanetLab

Síť PlanetLab je celosvětová experimentální síť podporující vývoj a výzkum nových internetových služeb a od svého vzniku v roce 2002 ji použilo stovky akademických i soukromých výzkumných institucí. Následně se zformovalo konsorcium se stejnojmenným názvem PlanetLab, složené z několika univerzit z USA (University of California at Berkeley, Princeton University a University of Washington) a postupem času do něj vstoupily další univerzity a akademické instituce z celého světa, stejně tak jako velké technologické společnosti jako Hewlett-Packard, Google nebo France Telecom [3].

Dle vyjádření Larryho L. Petersona, ředitele konsorcia, byla činnost PlanetLabu ukončena v březnu 2020 [4]. PlanetLab, tedy jeho evropská část, PlanetLab Europe, pokračuje v činnosti pod vedením Timura Friedmana z laboratoře LIP6 na Sorbonne Université ve spolupráci institutem Inria, Univerzitou v Pise a s Hebrejskou univerzitou v Jeruzalémě. PlanetLab Europe úzce spolupracuje s projektem OneLab, jež využívá infrastrukturu PlanetLabu. V čase psaní této práce síť čítá 998 uzlů.

1.1 Terminologie v síti PlanetLab

Pro pochopení fungování sítě PlanetLab je nutno představit terminologii, se kterou síť pracuje [5]. Byly vybrány následující nejčastěji používané termíny:

- **Místo (site)** - Fyzické místo, na kterém jsou umístěny servery sítě PlanetLab.
- **Uzel (node)** - Dedikovaný server, na kterém běží komponenty sítě PlanetLab. Jsou přiřazovány do jednotlivých dílů.
- **Díl (slice)** - Množina alokovaných prostředků distribuovaných napříč sítí PlanetLab. Pro většinu uživatelů tedy představuje přístup k UNIX shellu na uzlech v síti.
- **Silver** - Představuje díl běžící na konkrétním uzlu, jinými slovy, množina alokovaných prostředků v rámci jednoho uzlu.
- **Virtuální server (Virtual server)** - Každý silver je implementován jako virtuální server běžící na uzlu.
- **Uživatel (user)** - Kdokoli, kdo vyvíjí a nasazuje aplikace v rámci sítě PlanetLab
- **Klient (client)** - Klient služby běžící v síti PlanetLab.
- **Služba (service)** - Aplikace běžící v síti PlanetLab.

1.2 Rozhraní sítě PlanetLab pro programování aplikací

PlanetLab disponuje aplikačním rozhraním, které usnadňuje manipulaci s centrální databází. Prostřednictvím tohoto API lze upravovat informace o uživatelích, uzlech, místech aj. PlanetLab disponuje dvěma API: PLCAPI a NMAPI

PLCAPI (PlanetLab Central API) je rozhraní přes které lze přistupovat a spravovat centrální databázi sítě [6]. Toto API lze využít pro různé druhy automatizace práce se sítí PlanetLab. Příkladem je program *plcsh*, který lze volat z shellu a také jej použít pro další skriptování. Aplikace plbmng implementuje PLCAPI, jež poskytuje všechny potřebné funkce pro běh aplikace.

K tomuto API lze přistoupit skrze protokol XML-RPC přes HTTPS. Podporovanými typy autentizace vůči PLCAPI jsou:

- Autentizace pomocí uživatelského jména a hesla.
- Autentizace pomocí GPG klíče.
- Autentizace pomocí ustanovení relace, uživatel musí provést jednu z výše uvedených způsobů autentizace a pomocí metody *GetSession* získá klíč relace.
- Anonymní autentizace.

V každé funkci, kterou API poskytuje, je definováno, jakou funkcionalitu může běžný uživatel využít. Plná funkcionalita některých funkcí je dostupná pouze pro uživatele s oprávněním administrátora.

1.3 Operační systémy GNU/Linux

Operační systém (OS), jež běží na virtuálních serverech v síti PlanetLab je GNU/-Linux. Jedná se o UNIX-like operační systém s modulárním monolitickým jádrem. Vznik OS se datuje do roku 1991, kdy jej tehdejší student Linus Torvalds vytvořil během studia na Univerzitě v Helsinkách. Od svého počátku je jeho kód distribuován pod svobodnou licencí, která podporuje otevřený vývoj. Díky své licenci zanedlouho získal na popularitě, kterou si drží dodnes a na vývoji tohoto OS spolupracují stovky vývojářů. Nejnovější verze jádra je licencována pod licencí GNU General Public License verze 2. Vývoj Linuxu je v dnešní době sponzorován firmami jako je např. Red Hat, Dell, Intel, IBM a Samsung. Torvalds v současnosti vede vývoj jádra, přičemž provádí revize kódu a jeho začleňování do hlavní větve.

Jádro systému je psáno v jazyce C. Velkou část operačního systému tvoří programy projektu GNU. Achitektura Intel x86, pro kterou byl systém napsán, však není jedinou podporovanou architekturou. Do dnešní doby byl systém portován na mnoho procesorových architektur. I díky tomu můžeme dnes tento systém spatřit

na mobilních telefonech, chytrých zařízeních, serverech i superpočítačích. Hlavními výhodami Linuxu jsou:

- Linux je multiplatformní, tedy je schopen běhu na více procesorových architekturách,
- Otevřený kód umožňující kolaboraci napříč komunitami, inspekci kódu a následné odstraňování jeho zranitelností.
- Variabilita a přizpůsobitelnost.
- Linux je považován za bezpečný systém. Jeho bezpečnostní model je založen na modelu UNIXovém, jež je komplexní, robustní a ověřený.
- Rychlé dodávání oprav. Díky open source modelu komunita rychle dodává bezpečnostní záplaty.

Linux má však také své nevýhody, zejména:

- Vyžaduje po uživateli více znalostí potřebných pro jeho správu než např. systémy Windows či MacOS.
- Nekompatibilita aplikací určených pro zmíněné operační systémy.
- Velké množství distribucí může být pro uživatele matoucí.
- Absence standardizovaného desktopového prostředí.
- Nejednotný způsob distribuce software.
- Chybějící hardwarové ovladače.

Uživatelům je dnes nabízeno velké množství linuxových distribucí, tedy celku obsahujícího jádro Linux a kolekci aplikací k němu připojených. Složení balíku aplikací a verze jádra se liší v závislosti na distribuci. Nejznámějšími distribucemi jsou Red Hat Enterprise Linux, CentOS, Fedora, Debian, Ubuntu či Arch Linux.

V rámci sítě PlanetLab lze spatřit distribuci Fedora ve verzi 23 až 25 s verzí jádra od verze 4.5.7-202.fc23.x86_64 až po verzi 4.13.16-100.fc25.x86_64. Následující kapitola se věnuje detailnějšímu popisu této distribuce.

1.3.1 Distribuce Fedora

Fedora je distribuce vyvíjená komunitou okolo projektu Fedora Project a sponzorovaná firmou Red Hat. Je založená na balíčkovacím systému RPM. Distribuce klade důraz na použití na desktopech, přesto je také hojně používaná na serverech. Fedora je známá pro svou pokrokovost, kdy v každém novém vydání přichází se zásadními novinkami. Nové verze jsou vydávány přibližně každého půl roku, přičemž vychází v edicích Workstation, Server, IoT a také ve speciálních edicích CoreOS a Silverblue. Mezi její přednosti patří právě aktuálnost softwarových balíčků. Proto je tato distribuce volbou vývojářů, kteří chtějí pracovat s nejnovějšími verzemi software. Všechen software, který Fedora nabízí ve svých repozitářích je licencován pod jednou z mnoha svobodných licencí. Každému uživateli nabízí Fedora Project participaci na vývoji

distribuce mnoha různými způsoby, počínaje pouhým hlášením chyb přes úpravu dokumentace až po samotnou opravu chyb v projektu, či implementaci nových funkcí. K projektu se také pojí početné množství mailing listů, dokumentace ve formě wiki, komunitní fórum aj. Díky velikosti komunity se Fedora stává ideální testovací základnou pro funkce, které se v budoucnu stanou součástí komerční Red Hat distribuce. Partnerství mezi touto firmou a Fedora Projectem přináší komunitě Fedory užitek ve formě participace inženýrů z firmy Red Hat na vývoji Fedory a tedy také implementace nejrůznějších inovací. Taktéž, pro firmu je toto partnerství výhodné, neboť po přidání nových funkcí do Fedory může od uživatelů získat zpětnou vazbu, případné chyby opravit a následně tuto opravu již jako stabilní implementovat do své komerční distribuce a poté ji předat svým zákazníkům. Díky neustálé inovaci, aktuálnosti a pokrokovosti je Fedora ideálním operačním systémem pro experimentální síť PlanetLab.

1.3.2 Využití SSH pro monitorování činnosti operačního systému

Protokol *Secure Shell* (SSH) a programy jej používající jsou dnes neodmyslitelnou součástí nástrojové sady systémového administrátora linuxových systémů. SSH poskytuje zabezpečenou komunikaci v sítích TCP/IP. Umožňuje realizovat bezpečnou komunikaci mezi dvěma počítači při použití klient-server architektury. Server standardně naslouchá na portu TCP/22. Tento port byl protokolu přiřazen organizací IANA. Typicky se využívá pro zprostředkování přístupu k příkazovému řádku, spuštění vzdálených příkazů, kopírování souborů, ale taky tunelování a předávání TCP portů. SSH používá pro autentizaci se vzdáleným počítačem asymetrickou kryptografii. Mezi nejběžnější způsoby použití SSH patří použití automaticky vygenerovaného páru asymetrických klíčů pro šifrování síťového spojení a poté použití hesla pro přihlášení. Druhým nejběžněji používaným způsobem přihlašování je použití manuálně vygenerovaných asymetrických klíčů, soukromého a veřejného. Po umístění veřejného klíče na příslušných počítačích, získává vlastník soukromého klíče z téhož svazku přístup k danému počítači. Architektura protokolu je rozdělena do následujících tří částí [8]:

- Protokol transportní vrstvy - zajišťuje autentizaci serveru, důvěrnost a integritu. Volitelně může poskytovat komprimaci.
- Protokol autentizace uživatele - zajišťuje autentizaci klienta serveru.
- Protokol spojení - Rozděluje šifrovaný tunel do dílčích logických kanálů.

Dnešní standardní implementací je OpenSSH, jež byl původně vyvinut pro operační systém OpenBSD, avšak je již portován do většiny dnes běžně používaných systémů. Fedora není vyjímkou. V repozitářích Fedory se nachází metabalíček *openssh*, který vede balíčky *openssh-clients*, *openssh-askpass* a *openssh-server* jako své závis-

losti. Dohromady tedy poskytují plnou funkcionalitu OpenSSH. Součástí zmíněných balíčků jsou také uživatelské nástroje, spustitelné soubory, které poskytují funkcionalitu popisovaného protokolu. Mezi ty základní patří:

- **ssh** - klientský nástroj sloužící pro připojení k vzdálenému počítači, na němž běží *sshd*.
- **sshd** - serverová část, spuštěn jako služba, naslouchající na příslušném portu.
- **scp** - nástroj pro kopírování souborů, použití podobné příkazu *cp*.
- **sftp** - poskytuje stejnou funkcionalitu jako scp avšak přidává možnost vzdálené správy systému souborů, která umožňuje aplikacím obnovit přerušený přenos souborů, vypsat obsah vzdálených adresářů a odstranit vzdálené soubory.
- **ssh-keygen** - jednoduchý jednoúčelový nástroj určený pro generování nových asymetrických klíčů.
- **ssh-copy-id** - nástroj umožňující přidání veřejného klíče na vzdálený počítač.
- **ssh-add** - nástroj pro přidání identit do OpenSSH autentizačního agenta.

Existuje také mnoho jiných implementací protokolu SSH, mezi něž se řadí také knihovna Paramiko. Tato je implementována v jazyce Python. Své využití tedy nachází v aplikacích psaných v jazyce Python.

1.4 Virtualizační nástroje Linux-VServer a LXC

PlanetLab od svého založení až do roku 2014 využíval virtualizaci Linux-VServer a od roku 2014 do dnes používá *Linux Containers* (LXC) [1, 2]. Oba virtualizační nástroje provádějí virtualizaci na úrovni operačního systému. Tento způsob virtualizace umožňuje běh více izolovaných bezpečných serverů na jednom fyzickém serveru. Prostředí hostovaného OS sdílí s hostitelským systémem jeden OS, tedy sdílejí stejné jádro OS. [11] Aplikace běžící v takovém prostředí vnímají virtualizovaný systém jako samostatný systém. LXC se stará o izolaci a případně omezení systémových prostředků pro jednotlivé *virtuální stroje* (VM) a také zajišťuje izolaci jednotlivých kontejnerů. Mezi nejdůležitější rozdělované prostředky patří souborový systém, přidělená RAM, přístup k procesoru a jiným zařízením. Technologie LXC staví na *cgroups* a je závislá na ostatních technologiích oddělujících jmenné prostory. Tyto jsou implementovány v Linuxovém jádře. Tento způsob virtualizace, oproti plné či nativní virtualizaci, šetří paměť i výkon procesoru, protože samotná režie celého virtualizačního procesu má velice malé požadavky [14]. LXC lze považovat za nástupce technologie Linux-VServer [15] i z důvodu nulového vývoje této technologie ve srovnání s aktivním vývojem u LXC. Ověření, zda systém je virtualizován, případně jaká virtualizace byla použita, lze provést spuštěním nástroje `systemd-detect-virt`, který je schopen detekce až 24 typů virtualizace včetně typu *none*, kdy daný systém není virtualizován [16].

2 Aktuální stav aplikace pro monitorování serverů

Aplikace *PlanetLab Server Mangager* (plbmng) je určena pro správu projektů běžících na jednotlivých uzlech sítě PlanetLab. Jedná se o aplikaci, s níž uživatel primárně interaguje prostřednictvím *Text-based User Interface* (TUI), avšak dostupná je také možnost si aplikaci importovat jako knihovnu v jiné aplikaci. Poslední zveřejněnou verzí je verze `plbmng 0.4.2.post1`. Tato práce na této verzi zakládá nově implementované funkce. Aplikace je implementována v jazyce Python. Obsahuje kompletní dokumentaci v anglickém jazyce. Její zdrojový kód je publikován na platformě GitLab a je dostupná i z repozitáře Python balíčků PyPI.

Aplikace pro vykreslení TUI využívá knihovnu `pythondialog`, která na pozadí využívá aplikace `Dialog` pro zobrazení samotného TUI. Tato aplikace na pozadí využívá knihovny `curses` a `ncurses`, které poskytují rozhraní pro tvorbu aplikací v textovém režimu. Načítání geodat v aplikaci plbmng zajišťuje použitá knihovna `geocoder`. Pro vykreslování uzlů sítě PlanetLab na mapových podkladech aplikace využívá knihovnu `folium`, která využívá mapových podkladů z projektu OpenStreetMap, či jiných projektů s koncepcí otevřeného software.

2.1 Popis stávající funkcionality aplikace

Po prvním spuštění je uživateli nabídnut dialog, který uživatele vyzývá k zadání potřebných proměnných v konfiguračním souboru, které jsou potřebné pro plnou funkcionalitu aplikace. Uživatel je vyzván k zadání názvu dílu v rámci sítě PlanetLab, cesty k SSH klíči a uživatelského jména a hesla pro přihlášení k PLCAPI. Tento konfigurační soubor je možné kdykoli změnit, a to buď přímo z aplikace vybráním možnosti *Set credentials* v hlavním menu, či přímou editací souboru pomocí libovolného textového editoru.

V hlavním menu, zobrazeném na obrázku 2.1, je dostupných pět voleb, přičemž pod každou je skryto další pod-menu. Funkcionalita, která je umístěna do těchto voleb je popsána dále.

První volba *Access servers* uživateli nabízí přistoupení k jednotlivým uzlům v síti PlanetLab. Nabízí různé možnosti filtrování a to na základě DNS (hostname serveru), IP adresy, lokace. Dále je dostupné filtrování na základě dostupného software a hardware jednotlivých uzlů. Zde aplikace nabízí filtrování podle verze kompilátoru GCC, verze Python interpretu, verze linuxového jádra a celkové dostupné paměti na konkrétním silveru. Před využitím této možnosti filtrování je nutné předem servery oskenovat a zjistit jednotlivé atributy, na základě kterých aplikace následně filtruje.

Obr. 2.1: Hlavní menu aplikace plbmng.

Poslední možnost filtrování se nachází v menu *Filtering options*. Nabízí uživateli filtrování na základě dostupnosti, tedy na základě toho, zda cílový uzel odpovídá na ICMP Echo request dotazy, či zda je na něm otevřen port TCP/22 (běží na něm SSH démon). Tato funkcionalita umožňuje uživateli výběr uzlů na základě jím zadaných kritérií, čímž usnadňuje výběr užší množiny serverů na základě zvolených parametrů.

Druhou položkou v hlavním menu je *Monitor servers*, která po otevření rozbalí dvoupoložkové menu, kde se nachází dvě akce, které může uživatel provést. První možnost *Update server list* dovoluje uživateli získat z PLCAPI informace o všech uzlech v rámci sítě PlanetLab a uložit je do lokální databáze všech uzlů. Pro zjištění těchto informací používá plbmng modul `planetlab_list_creator.py`. Ten ukládá získané informace do souboru `default.node` ve formátu CSV. Provedení této operace trvá v rozmezí 30-60 minut. Její provedení však není pro funkci aplikace kritické, jelikož aplikace při instalaci distribuuje již předpřipravený soubor `default.node` obsahující databázi uzlů sítě PlanetLab. Druhá operace, kterou menu *Monitor servers* umožňuje, je *Update server status*. Ta slouží k zjištění, zda jednotlivé uzly odpovídají na zprávu ICMP Echo request, zda má otevřený SSH port a zjištění informací o dostupném software a hardware silverů (viz předchozí odstavec), a ty následně rovněž uloží do lokální databáze. Tato operace obvykle trvá v rozmezí 5-15 minut v závislosti na celkovém počtu uzlů v databázi `default.node`.

Další, třetí, možnost v hlavním menu *Plot servers on map* poskytuje funkcionalitu vykreslení uzlů na mapě. Pro toto vykreslení lze aplikovat dva filtry: uzly odpovídající na zprávu ICMP Echo request, uzly s běžícím SSH démonem. Další možností je vykreslení všech uzlů uložených v lokální databázi. Po zvolení jedné ze tří možností se ve výchozím internetovém prohlížeči otevře mapa s vyznačenými

Obr. 2.2: Mapa zobrazující všechny servery s běžícím SSH démonem.

body reprezentující polohu uzlů. Po kliknutí na uzel se rozbalí informační bublina s detailními informacemi o uzlu. Na obrázku 2.2 lze vidět tuto mapu s vykreslenými uzly, na kterých běží SSH démon. Uzly jsou vykresleny na mapových podkladech vytvořených projektem OpenStreetMap.

Předposlední možnost v menu, *Set credentials*, umožňuje editaci konfiguračního souboru, do kterého uživatel zadává název dílu v síti PlanetLab, cestu k souboru obsahující veřejný SSH klíč, který bude použit pro připojování k jednotlivým silverům prostřednictvím SSH. Posledními dvěma atributy jsou uživatelské jméno a heslo, sloužící pro přihlášení k PLCAPI. Soubor je v počátečním stavu naplněn komentářem k atributům a atributy samotnými, kterým však není přiřazena hodnota. Zde je na uživateli, aby tyto hodnoty doplnil. Bez této konfigurace je funkcionalita aplikace omezena na minimum. Proto, jak již bylo zmíněno výše, je uživatel při prvním spuštění aplikace vyzván k editaci tohoto souboru.

Poslední položka hlavního menu *Extras* obsahuje dodatečnou funkcionalitu, kterou aplikace plbmng poskytuje. Najdeme zde možnost přidání vlastního serveru do lokální databáze uzlů. Pokud uživatel chce přidat server, je vyzván k zadání IP adresy serveru, případně dalších atributů, které lokální databáze uzlů vyžaduje. Druhou možností je kopírování souborů ze stanice, kde běží aplikace plbmng, na vzdálené servery vedené v databázi. Uživateli je prezentován dialog pro výběr souborů, které chce zkopírovat a následně je užiavteli umožněno zvolit cílovou množinu serverů na základě filtrovacího menu popsaného výše. V posledním kroku uživatel volí umístění na cílovém serveru, kde mají být soubory uloženy. Další možností menu *Extras* je zobrazení statistiky dostupnosti uzlů v síti na základě záznamů v lokální databázi uzlů. Uživateli je prezentován počet uzlů v databázi, na kolika z nich běží

SSH démon a kolik z nich odpovídá na ICMP Echo request. Poslední možnost zobrazuje informace o aplikaci, jejích autorech a licenci, pod kterou je aplikace plbmng distribuována.

2.2 Dokumentace aplikace a způsob zveřejnění

K aplikaci je přiložena dokumentace, která má konfiguraci uloženu v adresáři `source`. Dokumentace je psána ve značkovacím jazyce *ReStructuredText* (RST). Tyto soubory jsou při sestavování dokumentace předány nástroji Sphinx. V konfiguraci je uvedena reference na jednotlivé moduly aplikace plbmng, které obsahují dokumentační textové řetězce pro každou funkci a třídu v nich obsaženou. Výsledná dokumentace je převážně generována z těchto dokumentačních textových řetězců. Každá dokumentovaná funkce obsahuje popis jejího účelu, vstupní parametry, popis a datový typ návratové hodnoty. Výstupními formáty dokumentace jsou PDF, EPub, Texinfo a man. Samotný dokumentační nástroj Sphinx je napsán v jazyce Python, stejně tak jako plbmng. Používají jej projekty, jež jsou v komunitě vývojářů používající Python velmi dobře známé. Mezi ně se řadí např. Django, SciPy a SQLAlchemy. Vývojáři linuxového jádra taktéž používají Sphinx pro jeho dokumentaci.

Dokumentace se průběžně sestavuje za využití GitLab CI. Operace sestavení dokumentace se automaticky spouští pokaždé, když je publikována změna ve větvi *master* v repozitáři *plbmng* publikovaném na serveru GitLab. Výsledek sestavení je publikován na `https://utko-planetlab.gitlab.io/plbmng/`. Tato publikovaná dokumentace tedy vždy odpovídá aktuálnímu stavu větve *master*.

3 Rozšíření aplikace

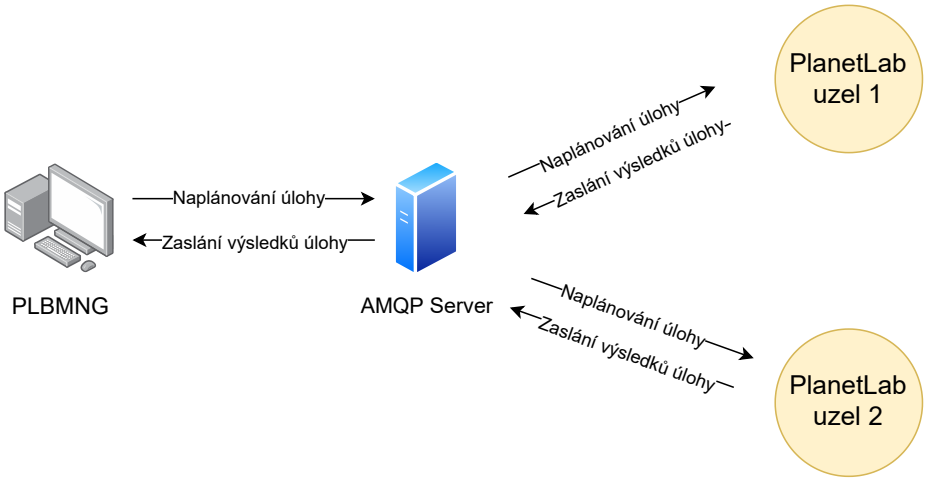
Nové rozšíření aplikace je z principu procesu vývoje software v síti PlanetLab dalším logickým doplněním její funkcionality. Standardní proces vývoje síťových aplikací zahrnuje vývoj a následně její testování. Nová funkcionalita aplikace plbmng značně přispěje k zjednodušení procesu testování vyvíjených aplikací. Těmito rozšířeními jsou:

- Distribuce vývojového software na zvolený silver.
- Spuštění distribuovaného software v uživatelem zadaný čas na zvolené silvery.
- Sběr výsledků z proběhlých úloh.

Pro splnění prvního požadavku – distribuce softwaru, je zapotřebí patřičný software nainstalovat na cílové servery - silvery. Touto funkcionalitou již aplikace plbmng disponuje. Pro jednoduché způsoby distribuce uvažujme kopírování software na cílové servery. Tento způsob distribuce je postačující např. pro distribuci shellových skriptů nebo skriptů pro Python interpret, jejichž závislosti jsou již na cílových systémech přítomny. Tento způsob distribuce je však možné použít také pro distribuci sofistikovanějších aplikací, a to tak, že jsou na cílový server nakopírovány dvě položky – Archiv obsahující aplikaci samotnou a instalační skript ležící mimo tento archiv. Pomocí instalačního skriptu lze rozbalit archiv s aplikací a tu následně nainstalovat na systém.

Pro naplánování spuštění úlohy, tedy spuštění zvoleného software s patřičnými atributy, je zapotřebí zvážit implementaci, kterou bude plbmng vhodně rozšířena. Nabízí se více možností implementace, které následující text rozebírá.

Jako první způsob řešení problému plánování úloh se nabízí implementace služby, která bude spuštěna na cílovém serveru, která bude přijímat úlohy od aplikace plbmng, které tato služba bude spouštět a výsledky odesílat zpět do plbmng. V tomto případě by bylo zapotřebí upravit aplikaci plbmng tak, aby byla schopna odesílat instrukce klientské službě běžící na jednotlivých silverech a zároveň aktivně přijímat zprávy o změnách stavu úloh od klientských služeb. To by obnášelo změnit princip fungování samostatně stojící aplikace plbmng, která by se změnila na službu, komunikující s klientskými službami. Komunikace by mezi plbmng a klienty probíhala po centralizované sběrnici, např. za využití technologie *Advanced Message Queuing Protocol* (AMQP), kdy by centrálním bodem komunikace byla plbmng samotná. Na obrázku 3.1 je tato architektura zobrazena. Plánování úloh a sběr jejich výsledků by probíhal metodou *publish/subscribe*. Plbmng by publikovalo na AMPQ server zprávu na téma, na kterém poslouchají uzly v síti PlanetLab a příslušné uzly by spustily úlohy. Po dokončení úloh by uzly zaslaly výsledky úloh zprávou s tématem, na kterém poslouchá aplikace plbmng. Tato architektura vyžaduje neustále běžící AMPQ server a k němu připojené uzly sítě PlanetLab, stejně tak aplikace plbmng



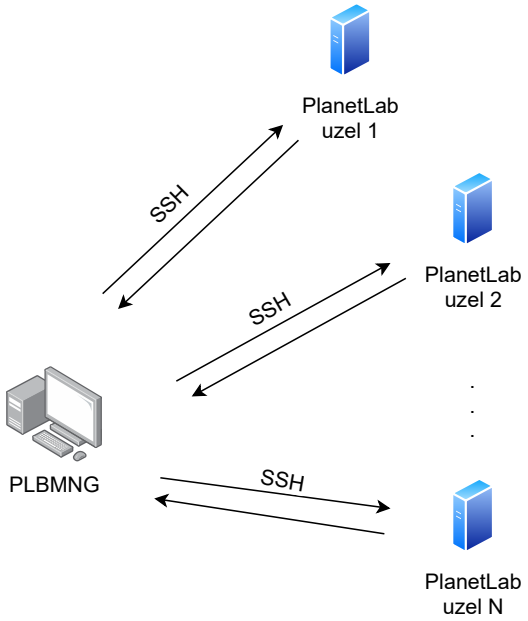
Obr. 3.1: Komunikace plbmng se silvery při použití AMPQ.

by musela být neustále přihlášena k AMPQ serveru. AMPQ klient by tedy musel být nainstalován ideálně na všech uzlech v síti před samotným naplánováním úlohy.

Druhý možný způsob implementace by zachoval jednoduchost aplikace, přičemž veškerá nová funkcionalita by nenarušila současný design aplikace. Při použití tohoto způsobu by byl vytvořen spouštěcí skript, zajišťující spuštění úloh v daný čas a obstarávající manipulaci s výsledky, které jím spuštěný software vygeneruje. Tento skript by bylo nutno nejprve distribuovat na předem definované místo na všech serverech, na kterých je požadováno spuštění software. Dalším krokem by bylo spuštění vzdáleného příkazu za využití SSH, kdy zavolaným příkazem by byl samotný spouštěcí skript, kterému by byly předány parametry určující, kdy se má spustit zadaný software a jaké parametry mu mají být při spuštění předány. Toto spuštění vzdáleného příkazu by bylo provedeno aplikací plbmng. Následně by spouštěcí skript na cílovém serveru čekal do uživatelem definovaného času spuštění a spustil by zadaný příkaz i s patřičnými parametry. Jakmile by běh software skončil, spouštěcí skript by uložil artefakty do lokální databáze na každém serveru zvlášť. Tato lokální databáze by obsahovala aktuální stav jednotlivých úloh, čas plánovaného spuštění, reálný čas spuštění a čas dokončení příkazu, či jiné informace asociované s konkrétním spuštěním příkazu. Aplikace plbmng by tedy kdykoli mohla tento soubor z množiny serverů prostřednictvím SSH získat, přečíst a jeho obsah prezentovat uživateli. Taktéž by bylo možno získat artefakty, které na cílovém serveru spuštění daného příkazu zanechalo.

První zmíněný způsob rozšíření by byl jak implementačně náročný, tak by navíc šel proti myšlence fungování aplikace plbmng. Ta je koncipována tak, aby byla schopna běžet i na uživatelské stanici, která se nachází za NAT, tedy na stanici, která nemá vlastní přiřazenou veřejnou IP adresu, skrze kterou by byla stanice

adresovatelná v síti internet. V případě takového použití se vytrácí jednoduchost aplikace, jelikož by bylo nutno konfigurovat nejen aplikaci samotnou, ale také síťovou infrastrukturu, případně samostatnou stanici adresovatelnou z internetu. Také instalace plbmng by byla obsáhlejší, zahrnovala by nejen instalaci aplikace samotné, ale také spuštění aplikace jako služby a konfiguraci firewallu systému, na němž běží. Toto řešení bylo z důvodu zachování jednoduchosti aplikace zavrhnuto a řešení této práce volí druhý popisovaný způsob implementace. Při zachování jednoduchosti tedy lze realizovat všechna nová, požadovaná rozšíření. Výhodou tohoto řešení je přítomnost SSH serveru na každém uzlu v síti PlanetLab. Lze tedy, bez nutnosti instalování jakékoli další služby na vzdálené servery, realizovat cíle práce za použití komunikace výzva–odpověď, jak je znázorněno na obrázku 3.2.



Obr. 3.2: Komunikace plbmng se silvery při plánování úloh a sběru výsledků.

3.1 Plánování a spouštění úloh

Pro splnění hlavního cíle práce je zapotřebí aplikaci obohatit o možnost plánování úloh, tedy procesu pro výběr okamžiku, ve který bude spuštěn jeden či více příkazů na vzdálených serverech. V rámci tohoto procesu je tedy nutno tuto úlohu zadat, následně monitorovat status této úlohy a po jejím skončení získat detailní informace o jejím běhu.

Při řešení problému spouštění úloh v daný čas je nutno přihlédnout k problému časových zón. Každý počítač může mít nastavenou rozdílnou časovou zónu. Aplikace pracující s entitami nacházejícími se ve více než jednom časovém pásmu, musí při

konverzi času z jednoho časového pásma do druhého, mít přehled o časových zónách, ve kterých se tyto entity nacházejí. Z tohoto důvodu byl při implementaci použit Unixový čas, který je na každém počítači stejný, nehledě na časovou zónu. Tento čas je shodný s časem v časové zóně UTC+0.

3.1.1 Implementace spouštěcího skriptu

Díky zvolenému, výše popsanému způsobu implementace, tedy distribuce úloh na vzdálené servery pomocí SSH, je zapotřebí nejen úlohu naplánovat a spustit, ale také mít přehled o jejím stavu v čase. Proto bylo přistoupeno k implementaci spouštěcího skriptu, jehož úlohou je správa životního cyklu úloh. Tento skript, pojmenován jako `executor.py`, byl implementován jako modul do knihovny aplikace plbmng. Je možno přímo spouštět a také z něj importovat funkce a třídy.

Na vzdáleném serveru, na kterém je naplánována úloha, je očekávána přítomnost interpretu jazyka Python ve verzi 3.5 či vyšší. Tento požadavek splňuje většina dnešních distribucí Linuxu, ty totiž již ve svých standardních i minimálních distribucích Python interpret obsahují. Pro zachování minimálních závislostí spouštěcí skript implementuje pouze knihovny ze základní distribuce Pythonu. Nejsou tedy pro jeho běh nutné žádné externí knihovny a pro běh skriptu postačuje standardní, základní distribuce Pythonu.

Skript implementuje třídu *PlbmngJob* popisující objekt úlohy, třídu *PlbmngJobsFile* reprezentující databázi úloh uloženou na vzdáleném serveru, držící informace o úlohách na něm spuštěných. Tato třída je implementována jako kontextový manažer (Context Manager) a umožňuje jednoduchou manipulaci s databází úloh. Na výpisu 1 je ukázka kódu funkce *create_job*, která používá tohoto manažera, přičemž do databáze zapíše novou úlohu.

```
def create_job(job_id: str, cmd_argv: str, scheduled_at: int):
    with PlbmngJobsFile(JOBS_FILE) as jf:
        jf.add_job(
            job_id,
            cmd_argv,
            scheduled_at=time_from_timestamp(scheduled_at)
        )
    create_job_dir(job_id)
```

Výpis 1: Ukázka použití manažera kontextu ve funkci *create_job*.

Výhoda použití kontextového manažera je, že odpadá nutnost otevírat a po provedení všech žádaných operací zavírat soubor s databází. Manažer soubor s databází otevírá na začátku `with` bloku a zavírá po jeho skončení. *PlbmngJobsFile* poskytuje metody pro vytváření, čtení, aktualizaci a mazání úloh.

Spouštěcí skript dále obsahuje metody pro manipulaci s časovými hodnotami, povětšinou funkcemi zajišťující konverzi času mezi různými formáty. Dále implementuje hlavní metodu, která se spouští v případě, kdy je modul přímo spuštěn. Skript zajišťuje přítomnost složkové struktury v domovském adresáři uživatele, který úlohu spouští. Vytvoří zde složku `.plbmng`, která obsahuje složku `jobs`, do které skript ukládá artefakty z jednotlivých úloh. Ve stejné složce je umístěn také soubor `jobs.json`, který slouží jako lokální databáze úloh. Tato složková struktura je zobrazena na výpisu 2.

```
~/.plbmng
├── jobs.json
└── jobs
    └── b23c4354-9e06-48de-b0a7-996a7e61717d
        └── artefacts
            └── stdout
```

Výpis 2: Struktura složky ~/.plbmng s jednou vytvořenou úlohou.

Hlavní metoda dále spouští plánovač úloh a přidává do něj uživatelem zadanou úlohu. Ta je v zadaný čas provedena a artefakty, tedy výstup ze standardního výstupu a standardního chybového výstupu, jsou uloženy do složky *artefacts* pro příslušnou úlohu.

Skript také implementuje funkce, výjimky, enumerátory, kodéry a dekodéry, jejichž popis je nad rámec tohoto textu. Tyto jsou však popsány v dokumentaci aplikace v anglickém jazyce.

Všechna skriptem požadovaná vstupní data jsou při spuštění poskytnuta jako argumenty. Pro analýzu a zpracování argumentů je použita třída *ArgumentParser* z modulu `argparse`, který je součástí standardní distribuce Pythonu. Použití této třídy poskytuje jednoduché rozhraní pro předávání vstupních dat mezi uživatelem skriptu a skriptem samotným. Třemi povinnými argumenty skriptu `executor.py` jsou:

- −−run-at,
- −−run-cmd,
- −−job-id.

Pro uživatele je zpřístupněn také parametr −−help, který vypíše nápovědu ke všem dostupným argumentům tohoto skriptu.

Argument *run-at* očekává celé číslo – epochu, která reprezentuje čas, ve který má být daný příkaz spuštěn. Argument *run-cmd* očekává textový řetězec, představující příkaz, který má být spuštěn. A poslední argument *job-id* očekává textový řetězec definující unikátní identifikátor ve formátu specifikovaném RFC 4122. Kterýkoli jiný formát je také akceptován, avšak není doporučeno používat identifikátory, které nejsou v souladu se zmíněnou specifikací.

Příklad spuštění spouštěcího skriptu je zobrazen ve výpisu 3. Spouštěcí skript je primárně určen pro použití s aplikací plbmng, avšak lze ho použít i samostatně

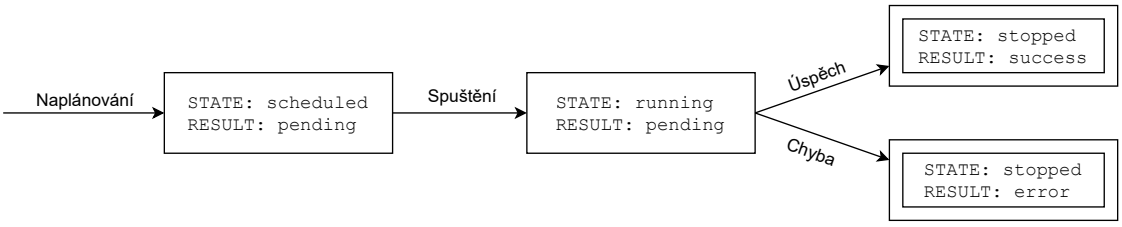
```
python3 executor.py --run-at 1606254787 --run-cmd "date -d now"
↪  --job-id b23c4354-9e06-48de-b0a7-996a7e61717d
```

Výpis 3: Příklad spuštění spouštěcího skriptu.

3.1.2 Databáze úloh a sběr artefaktů

Pro ukládání stavů jednotlivých úloh používá plbmng soubor `jobs.json`, který je přítomný na každém vzdáleném serveru. Obsahem tohoto souboru je serializovaný objekt obsahující list úloh formátovaný jako *JavaScript Object Notation* (JSON). Tento soubor vždy reflektuje aktuální stav všech úloh, které jsou či byly spuštěny na daném serveru. Aplikace plbmng tento soubor může ze vzdáleného serveru kdykoli stáhnout a získat přehled o aktuálním stavu úloh.

Úloha může nabývat určitých stavů, které popisují její aktuální stav. Tedy zda úloha běží nebo již skončila a zda skončila úspěšně. Obrázek 3.3 zobrazuje všechny možné stavy a výsledky, které může úloha nabývat.



Obr. 3.3: Stavy naplánované úlohy.

Každá úloha při naplánování nabývá stavu *naplánováno* (*scheduled*) a její výsledek je nastaven na *čekání* (*pending*). Tento stav indikuje naplánovanou úlohu čekající na spuštění, její výsledek zatím není znám. Z tohoto stavu se stav úlohy po jejím spuštění mění na *běžící* (*running*), což indikuje, že daná úloha byla spuštěna

a zatím neskončila, proto ani její výsledek není znám. Po skončení úlohy se výsledek úlohy může v případě úspěšného provedení příkazu změnit na *úspěch* (*success*), nebo na výsledek *chyba* (*error*) v případě neúspěšného provedení příkazu. Stav úlohy se po jejím skončení vždy přepne na *zastaveno* (*stopped*).

Výpis 4 zobrazuje jednu úlohu uloženou v souboru `jobs.json`, která již byla ukončena a skončila chybou. Databáze ukládá o každé úloze následující atributy:

- Stav,
- výsledek,
- čas, kdy má být úloha spuštěna,
- jméno uživatele, jež úlohu spustil,
- identifikátor uživatele (UID),
- *Fully Qualified Domain Name* (FQDN) - plně specifikované doménové jméno vzdáleného serveru,
- časovou zónu, ve které se server nachází,
- *Universally unique identifier* (UUID) úlohy,
- příkaz,
- reálný čas spuštění úlohy,
- čas dokončení úlohy,
- čas, po který úloha běžela.

3.2 Implementace knihoven Paramiko a Parallel-SSH

Do aplikace byla implementována knihovna Paramiko, která implementuje SSH protokol, jak jeho serverovou část, tak část klientskou. Do knihovny funkcí `plbmng/lib` přibyl nový soubor `ssh.py`, který vytváří abstrakční vrstvu nad knihovnou Paramiko a jejími funkcemi od sestavení SSH spojení přes spouštění příkazů, nahrávání a stahování souborů až po manipulaci s SSH klíči. Tato abstrakční vrstva umožňuje jednoduchou manipulaci s funkcemi SSH ve funkcích provádějících konkrétní akce, tedy např. nahrávání souboru na vzdálený server. Z tohoto nově přidaného modulu jsou klíčovými metodami `command`, `upload_file` a `download_file`.

První metoda `command` přijímá nepovinný argument *background*, který nabývá hodnoty *True* nebo *False* na základě toho, zda se jedná o příkaz, který má běžet na pozadí či na popředí. V prvním případě se příkaz spustí a metoda nemá dále přehled o jeho stavu, kdežto v případě druhém, kdy je *background* nastaven na hodnotu *False*, metoda `command` čeká na výsledek spuštěného příkazu, a to maximálně do doby vypršení časového limitu, který je specifikován v konfiguračním souboru aplikace. Obě tyto varianty jsou v plbmng využity. Tato metoda se spouští s argumentem *background=True* pouze v případě, kdy se spouští spouštěcí skript na vzdáleném serveru a není žádoucí, aby další průběh aplikace plbmng byl blokován čekáním na

```
[
  {
    "state": "stopped",
    "result": "error",
    "scheduled_at": "2020-11-25T14:01:40.000",
    "user": "ogajduse",
    "user_id": 1000,
    "hostname": "ogajduse-alfa.usersys.ogajduse.com",
    "timezone": "Europe/Prague",
    "job_id": "b23c4354-9e06-48de-b0a7-996a7e61717d",
    "cmd_argv": "rpm -q foo",
    "started_at": "2020-12-02T19:43:52.175",
    "ended_at": "2020-12-02T19:43:52.197",
    "execution_time": 625332.197607,
    "real_time": 0.022607
  }
]
```

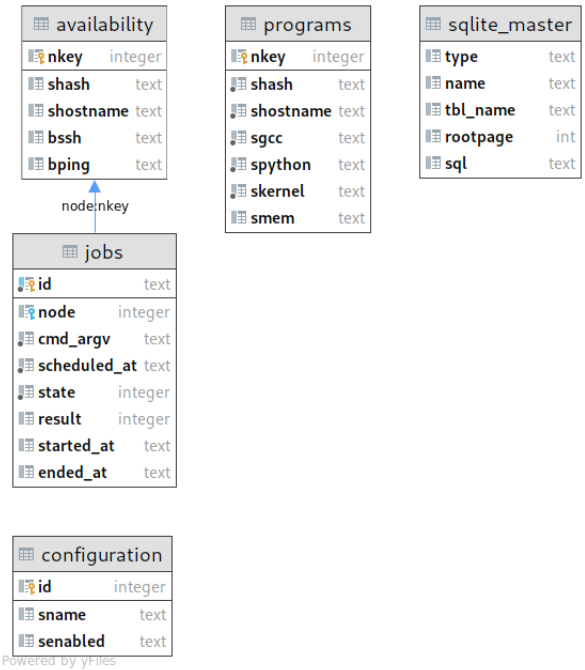

Výpis 4: Příklad záznamu v souboru jobs.json.

vykonání příkazu na vzdáleném serveru. V některých případech se tak může dít i v řádu hodin nebo dní, v závislosti na nastaveném čase spuštění a aktuálním čase. V ostatních případech použití metody je parametr `background` nastaven na *True* a výsledek příkazu je uživateli znám ihned po jeho dokončení.

Druhou implementovanou knihovnou pro operace prostřednictvím protokolu SSH je Parallel-SSH. Tato knihovna poskytuje API pro spouštění SSH příkazů napříč množinou serverů. Těchto serverů mohou být stovky či tisíce, jak je znázorněno na obrázku 3.2. Díky designu knihovny jsou příkazy spouštěny asynchronně, což značně snižuje časovou náročnost ve srovnání se sériovým zpracováním příkazů. Plbmng implementuje dvě funkce z této knihovny, a to `copy_remote_file` a `copy_file` z modulu `BaseParallelSSHClient`. Tyto jsou použity při kopírování souborů na vzdálené servery či na stanici, kde je spuštěna aplikace plbmng.

3.3 Úprava schématu databáze

V samotné aplikaci plbmng bylo nutno aktualizovat schéma lokální databáze tak, aby byla schopna ukládat data o úlohách a tuto informaci držela i po ukončení aplikace a znovu načetla při jejím spuštění. Pro splnění tohoto požadavku byla do schématu existující SQLite databáze přidána nová tabulka `jobs`, která je určena pro ukládání všech úloh naplánovaných aplikací plbmng. Takto upravené databázové schéma je možno vidět na obrázku 3.4.



Obr. 3.4: Upravené schéma aplikace plbmng.

Tato nová tabulka o úlohách ukládá ID úlohy, server, na kterém byla úloha spuštěna, příkaz, naplánovaný čas spuštění, stav a výsledek úlohy, čas spuštění a čas skončení úlohy. Klíč *node* v tabulce *jobs* je cizím klíčem, kdy rodičovskou tabulkou v této vazbě je existující tabulka *availability*, která ukládá informace o uzlech v síti PlanetLab.

Celkové schéma databáze je inicializováno při prvním spuštění aplikace plbmng, kdy jsou zároveň do tabulky *availability* přidána data, která jsou obsažena v souboru *default.node*, který byl popsán v kapitole 2.1. Tabulka *jobs* je po prvním spuštění aplikace prázdná a první záznam do ní přibývá ve chvíli, kdy je naplánována první úloha. Pokud má uživatel nainstalovanou aplikaci plbmng verze 0.5.1 či nižší, je vyžadována její opětovná instalace. Aplikace má implementovanou kontrolu existence databáze a v případě, že neexistuje, je vytvořena.

Úpravy v modulu pro manipulaci s databází

Ruku v ruce se změnou schématu jdou také změny do modulu `plbmng.lib.database`, do kterého bylo přidáno 7 nových funkcí, které zajišťují správnou manipulaci s entitami v databázi. Jedná se o funkce pro získání úloh z databáze:

- *add_job* pro přidání nové úlohy,
- *update_job* pro aktualizaci informací o úloze,
- *get_non_stopped_jobs* pro získání běžících úloh,
- *get_stopped_jobs* pro získání zastavených úloh,
- *get_all_jobs* pro získání všech úloh z databáze,
- *delete_job* pro smazání úlohy z databáze,
- *init_db_schema* pro vytvoření schématu databáze a uložení databáze na disk.

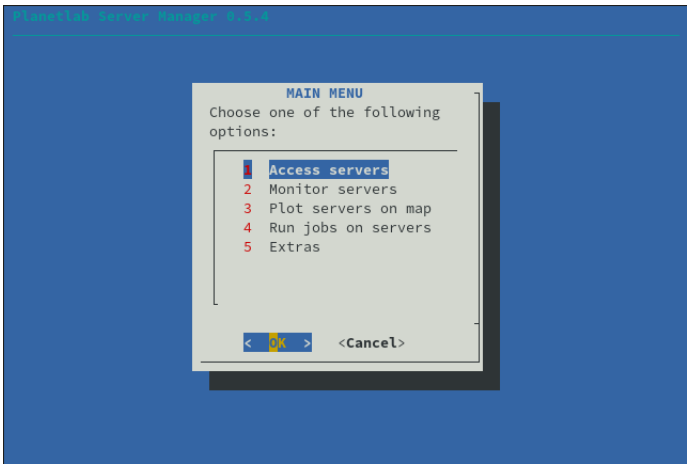
Poslední zmíněná metoda inicializuje schéma pouze v případě, že soubor s databází neexistuje. Pokud však tento soubor neexistuje, vytvoří se a je do něj zavedeno schéma a do tabulky *availability* jsou vložena data ze souboru *default.node*.

3.4 Plánování úloh v aplikaci

Do modulů `plbmng.engine` a `plbmng.lib.library` aplikace plbmng bylo implementováno 30 nových funkcí zajišťujících správnou funkci veškerých požadavků uživatele pro práci s úlohami. Modul *engine* obsahuje převážně funkce zajišťující prezentaci aplikace uživateli prostřednictvím dialogů, zobrazování dat v nich a sběr dat od uživatele. Přidané funkce do modulu *library* jsou pak funkce, které jsou znovupoužitelné a mohou býti implementovány na jednom či více místech v *engine*. Případně si je uživatel může importovat do své vlastní aplikace a použít plbmng jako knihovnu. Po spuštění plbmng je uživateli prezentována hlavní nabídka, ve které se nachází nová položka pod číslem 4 – „Spouštění úloh na serverech“. Toto hlavní menu se nachází na obrázku 3.5.

Veškerou funkcionalitu pro manipulaci se vzdálenými úlohami poskytuje právě menu „Spouštění úloh na serverech“. Jeho obsah je zobrazen na obrázku 3.6.

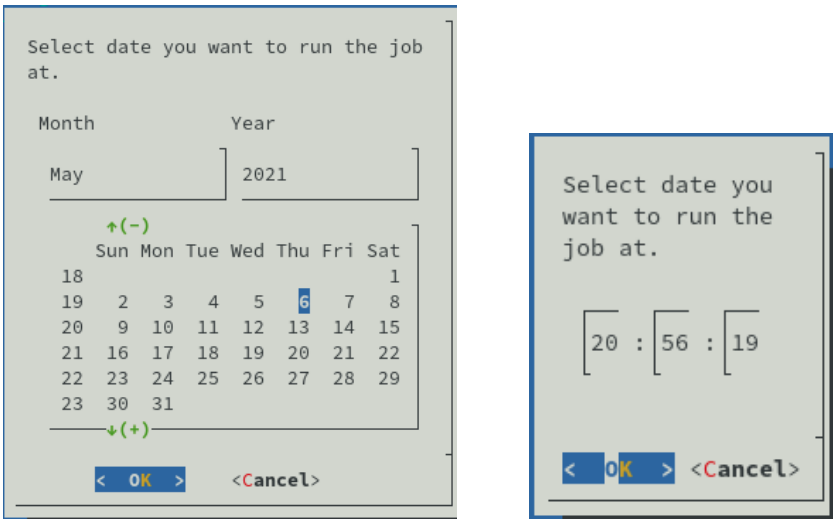
První položka v tomto menu „Kopírování souborů na server(y)“ zde byla přesunuta z menu „Extras“, jelikož jeho funkcionalita logicky zapadá mezi ostatní funkce, které toto menu poskytuje. Nově implementovanou položkou v tomto menu je „Spuštění jednorázového příkazu“. Tato volba uživateli poskytuje možnost okamžitého spuštění příkazu na jednom či více serverech. Nejprve je uživatel vyzván k zadání příkazu, který má být spuštěn a následně vybere množinu serverů, na které chce příkaz spustit. Účelem je poskytnout uživateli možnost spouštět příkazy, které jsou provedeny okamžitě bez nutnosti procesu plánování spuštění.



Obr. 3.5: Hlavní nabídka aplikace plbmng.

Obr. 3.6: Obsah nabídky „Spouštění úloh na serverech“.

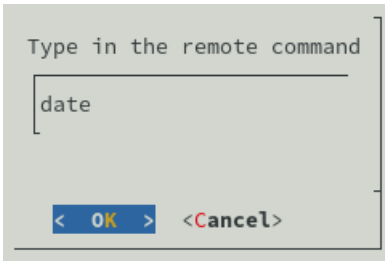
Možnost č. 3 – „Plánování vzdálených úloh“ uživateli poskytuje možnost plánování spuštění vzdálených úloh na množině serverů v zadaný čas. Po otevření této nabídky je uživatel vyzván k zadání data a následně času, kdy má být úloha spuštěna. Při výběru data lze nastavit rok, měsíc a den. Uživateli je přehledně prezentováno rozhraní podobné standardnímu kalendáři s měsíčním rozložením. Pro pohyb v tomto dialogu uživatel použije šipky, klávesy *Tab* a *Enter* pro potvrzení výběru. V dialogu pro výběr času uživatel vybírá konkrétní hodinu, minutu a sekundu, přičemž pro navigaci v tomto dialogu lze použít stejných kláves jako v dialogu předchozím. Oba tyto dialogy jsou zobrazeny na obrázcích 3.7.



(a) Zadání data. (b) Zadání času.

Obr. 3.7: Dialogy pro zadání data a času, ve který má být úloha spuštěna.

Po zadání data a času je od uživatele očekáváno zadání příkazu, který má být spuštěn v rámci vzdálené úlohy. Pro jeho zadání dialog poskytuje jednoduchý kontejner se zprávou a buňkou pro vložení textového vstupu. V příkladu na obrázku 3.8 je zadán příkaz `date`, který na standardní výstup vypíše aktuální datum a čas v době spuštění příkazu.



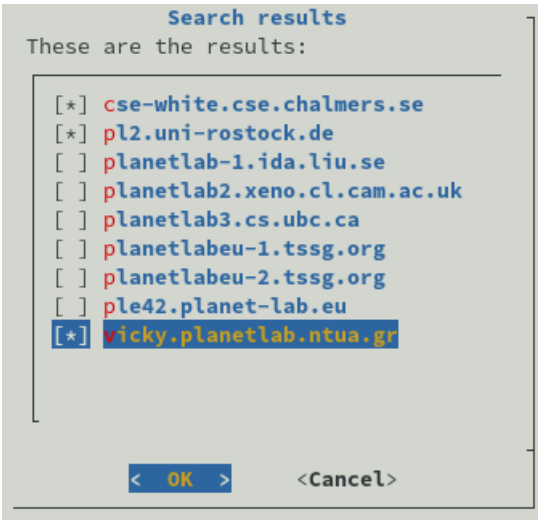
Obr. 3.8: Dialog s výzvou pro zadání příkazu pro spuštění.

Po potvrzení tohoto dialogu je uživatel vyzván k výběru serverů, na kterých má být daná úloha naplánována. K tomuto bylo využito stávajícího menu „Přistoupit k serverům“, které je zobrazeno na obrázku 3.9.

Toto menu obsahuje různé možnosti filtrování jako je filtrování serverů podle jejich FQDN, IP adresy, polohy a jejich dostupného hardware a software. Nachází se zde také nabídka *Možnosti filtrování*, která dle volby uživatele omezuje zobrazenou množinu serverů na základě toho, zda na cílovém serveru běží SSH server a zda daný server odpovídá na ICMP Echo. Volba „Přistoupit k poslednímu serveru“ vybere pro naplánování úlohy poslední server, ke kterému bylo přistoupeno. Pro demonstraci je na obrázku 3.10 ukázán dialog pro výběr serverů, jež byly vyfiltrovány na základě

Obr. 3.9: Dialog „Přistoupit k serverům“.

nejnovější dostupné verze Python interpretru. Uživatel se v tomto dialogu pohybuje šipkami nahoru a dolů, výběr serverů provádí mezerníkem a potvrdí jej tlačítkem „OK“.



Obr. 3.10: Zaškrtávací seznam serverů.

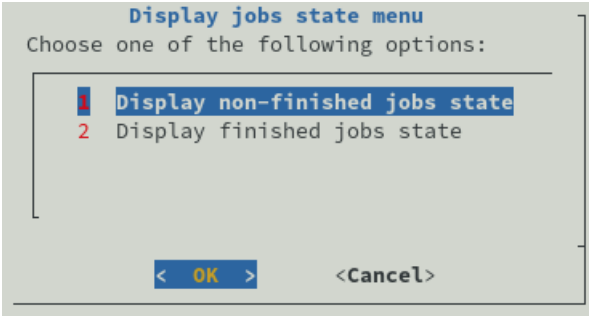
Po potvrzení dialogu s výběrem serverů aplikace na pozadí vygeneruje pro každou úlohu jedinečný identifikátor UUID, dále jen jako ID úlohy. Pokud uživatel naplánoval spuštění úlohy na 4 serverech, vzniknou 4 úlohy, kdy každá má unikátní ID. Aplikace dále zkontroluje, zda na vzdálených serverech existuje spouštěcí skript a v případě, že neexistuje, skript na ně zkopíruje. Následně je na vzdáleném serveru spuštěn tento spouštěcí skript s parametry odpovídajícími uživatelskému vstupu po-

dobně, jak je zobrazeno na výpisu 3. Ten po spuštění čeká do zadaného času spuštění a spustí daný příkaz. Všechny popsané operace se vzdáleným serverem probíhají za pomocí funkcí z modulu `plbmng.lib.ssh`, který pro ně poskytuje patřičné funkce.

3.4.1 Aktualizace stavu úlohy a její zobrazení

Popis výše uvádí způsob naplánování úlohy, avšak od chvíle, kdy ji uživatel naplánuje, ztrácí přehled o jejím aktuálním stavu. Tento problém řeší položky „Zobrazení stavu úloh“ a „Aktualizace stavu úloh“ v menu „Spouštění úloh na serverech“ zobrazeného na obrázku 3.6.

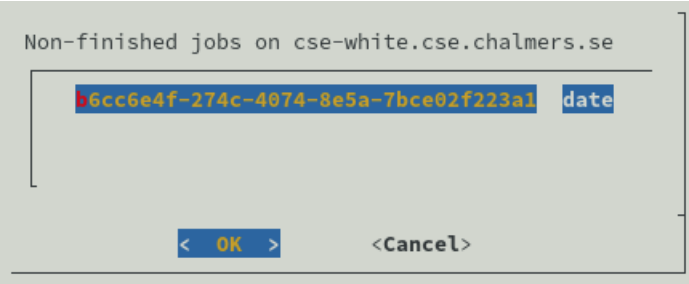
Předpokládejme, že máme naplánovány 3 úlohy a zatím neznáme jejich výsledek. Pro zobrazení aktuálního stavu úloh slouží dvě možnosti „Zobrazit nedokončené úlohy“ a „Zobrazit dokončené úlohy“, obě zobrazené na obrázku 3.11.



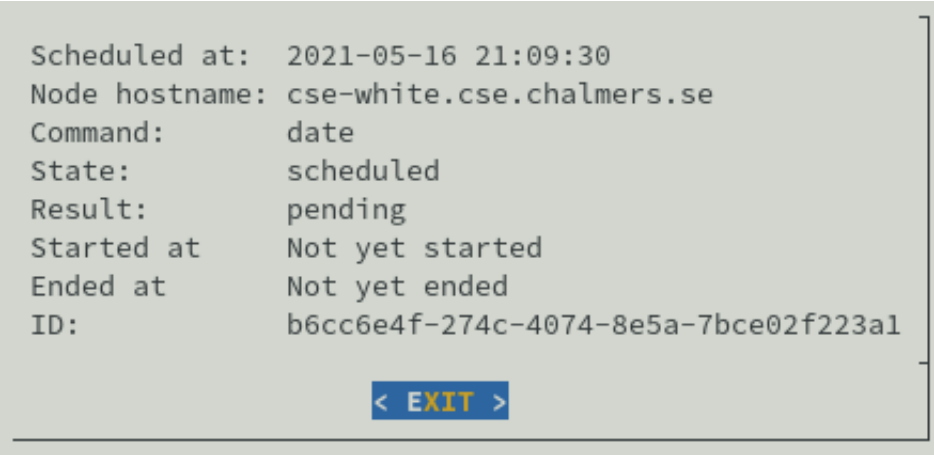
Obr. 3.11: Obsah menu „Zobrazení stavu úloh“.

Nedokončené úlohy jsou takové, které mají podle obrázku 3.3 stav jiný než „zastaveno“, nehledě na výsledek. Dokončené úlohy jsou všechny ty, které jsou ve stavu „zastaveno“, opět nehledě na výsledek. Pod-dialogy tohoto menu se větví stejně, tedy způsob pohybu v nich je pro obě možnosti totožný. Po potvrzení kterékoli z těchto možností je uživatel vyzván k výběru serveru, na němž běží/běžely úlohy. Po potvrzení výběru serveru je uživateli prezentován dialog s výčtem úloh běžících/proběhlých na vybraném serveru, jak je prezentováno na obrázku 3.12.

Pro každou z úlohu zobrazenou v tomto dialogu je uvedeno ID úlohy a spuštěný příkaz. Pokud se v dialogu nachází více než je jedna stránka schopna pojmout, je možno výpis úloh posouvat pomocí kláves šipek. Po otevření kterékoli úlohy z tohoto seznamu jsou uživateli prezentovány informace o úloze, které jsou aplikaci plbmng známy. Tyto jsou prezentovány v prostém dialogovém okně s jedinou možností „Zavřít“, která okno s informacemi zavře. Okno obsahuje informace o plánovaném času spuštění úlohy, FQDN vzdáleného serveru, příkaz, stav, výsledek, čas spuštění, čas skončení a ID úlohy. Toto dialogové okno je zobrazeno na obrázku 3.13.



Obr. 3.12: Zobrazení neukončených úloh na konkrétním serveru.



Obr. 3.13: Dialogové okno s informacemi o vybrané úloze.

Chce-li uživatel aktualizovat tyto informace, pak v menu „Spouštění úloh na serverech“ zvolí možnost „Aktualizace stavu úloh“. Ta na pozadí spustí funkci *refresh_jobs_status*, jež je zobrazena ve výpisu 5.

Tato funkce nejprve z databáze získá seznam nedokončených/běžících úloh a zkontroluje zda takové úlohy existují. Pokud se v lokální databázi aplikace plbmng nenachází žádná neukončená úloha, pak funkce končí a uživateli je prostřednictvím dialogu tato skutečnost ohlášena. Pokud se v databázi nachází pouze skončené úlohy, není důvod opětovně zjišťovat jejich stav. Následně je z tohoto listu úloh extrahován seznam serverů, na nichž tyto běží. Tento seznam serverů je předán paralelnímu SSH klientovi *ParallelSSHClient*, který ze všech těchto serverů stáhne soubor `jobs.json` obsahující aktuální informace o stavu úloh na něm běžících, či proběhlých. Aplikace plbmng všechny tyto stažené soubory ukládá do složky `~/.plbmng/remote-jobs`. Každý z těchto souborů má název `jobs.json_<název serveru>`, kdy název serveru je právě ten server, odkud byl soubor stažen. Skrze tento design může funkce *get_remote_jobs* zobrazená na výpisu 6 z tohoto souboru načíst úlohy a vrátit je jako list objektů typu *PlbmngJob*.

Funkce *refresh_jobs_status* ze všech těchto úloh, které z daných serverů stáhla,

```
def refresh_jobs_status(self) -> None:
        ns_jobs = get_non_stopped_jobs(self.db)
        if len(ns_jobs) < 1:
            self.d.msgbox("There are no non-stopped jobs to
            ↪ update.")
            return None
        hosts = list(dict(groupby(ns_jobs, lambda job:
        ↪ job.hostname)).keys())
        ssh_key = settings.remote_execution.ssh_key
        user = settings.planetlab.slice
        client = ParallelSSHClient(hosts, user=user,
        ↪ pkey=ssh_key)
        cmds = client.copy_remote_file(
            f"/home/{user}/.plbmng/jobs.json",
            f"{get_remote_jobs_path()}/jobs.json"
        )
        joinall(cmds, raise_error=True)

        fetched_jobs = []
        for host in hosts:
            # get jobs for the current host
            fetched_jobs.extend(get_remote_jobs(host))
        jobs_intersection =
        ↪ set(fetched_jobs).intersection(set(ns_jobs))

        # update database
        for job in jobs_intersection:
            job = next((fjob for fjob in fetched_jobs if fjob ==
            ↪ job), None)
            self.db.update_job(job)
        self.d.msgbox("Jobs updated successfully.")
```

Výpis 5: Funkce provádějící aktualizaci informací o úlohách.

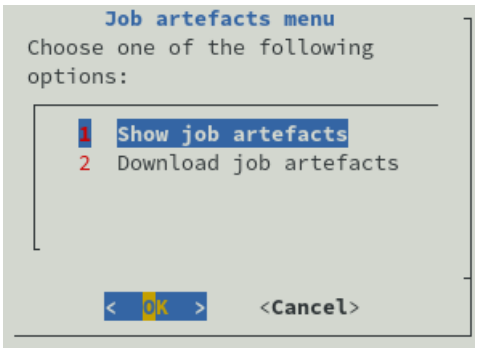
```
1 def get_remote_jobs(host: str) -> List[executor.PlbmngJob]:
2     with
      ↪ PlbmngJobsFile(f"{get_rmt_jobs_path()}/jobs.json_{host}")
      ↪ as jobs:
3         return jobs.jobs
```

Výpis 6: Funkce načítající úlohy proběhlé na vzdáleném serveru.

vytvoří množinu a následně provede průnik této množiny s množinou nedokončených úloh, které získala v prvním kroku. Informace o každé úloze z tohoto průniku jsou v posledním kroku této funkce aktualizovány a uloženy do lokální databáze aplikace plbmng. O úspěšné aktualizaci všech úloh je uživatel informován dialogem. Výsledek této aktualizace může uživatel ověřit v nabídce „Zobrazení stavu úloh“.

3.4.2 Zobrazení výstupu úlohy

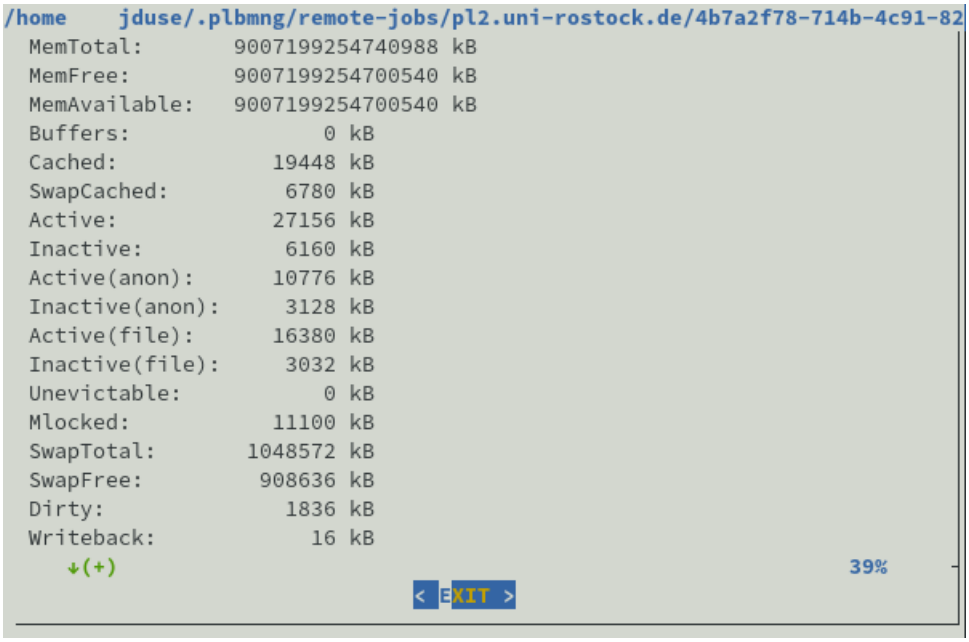
Aplikace plbmng také nabízí uživateli zobrazení výstupu jednotlivých úloh přímo uvnitř aplikace bez nutnosti jejího opuštění. Tuto funkcionalitu poskytuje položka s číslem 6 – „Výstupy úloh“ v menu „Spouštění úloh na serverech“. Obsahem tohoto menu, zobrazeného na obrázku 3.14, jsou dvě položky.



Obr. 3.14: Obsah menu pro manipulaci s výpisy úloh.

První položka – „Zobrazení výpisů“ otevře dialog, kde se nachází seznam serverů, na nichž běžely úlohy, jejichž výpisy byly staženy. Po potvrzení výběrů serveru je uživateli zobrazen seznam úloh asociovaných s vybraným serverem, pro něž byly výpisy již staženy. Potvrzením výběru úlohy uživatel dostává na výběr z dostupných výpisů pro tuto úlohu. V současné chvíli spouštěcí skript aplikace plmbng podporuje

vytváření dvou výpisů, a to standardního výstupu příkazu a standardního chybového výstupu příkazu. Po potvrzení výběru v tomto menu je uživateli ve čtecím dialogovém okně zobrazen obsah vybraného výpisu.



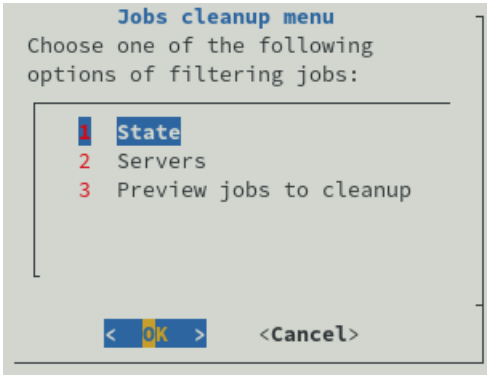
Obr. 3.15: Zobrazení výstupu úlohy v dialogovém okně.

Ukázka na obrázku 3.15 zobrazuje výpis příkazu `cat /proc/meminfo`, který zobrazuje informace o systémové paměti. Tento výpis se však kvůli své délce nevejde do dialogového okna. V tomto případě se v textu může uživatel pohybovat pomocí kláves šipek všemi směry, tedy i doprava a doleva v případě, že výpis obsahuje dlouhé řádky.

Menu „Výstupy úloh“ obsahuje také druhou možnost nazvanou „Stažení výpisů“, pomocí které může uživatel stáhnout výpisy ze serverů. Pokud neexistují úlohy ve stavu *zastaveno*, které nemají stažené výpisy, potom je uživateli zobrazena hláška „Žádné výpisy ke stažení“. Stažení výpisů je nutné spustit před jejich zobrazením.

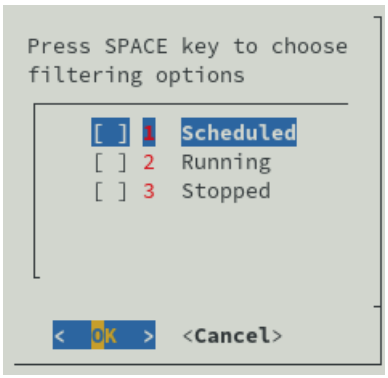
3.4.3 Smazání úloh

V průběhu práce s aplikací plbmng a např. také při opětovném plánování úloh může uživateli práci ztěžovat velké množství starých, již dávno proběhlých úloh, které uživatel již nemá zájem držet v databázi. Pro tento účel je v menu „Spouštění úloh na serverech“ přidána možnost „Vymazání úloh“, pomocí níž je uživatel schopen mazat úlohy na základě definovaných parametrů. Toto menu je zobrazeno na obrázku 3.16.



Obr. 3.16: Menu „Vymazání úloh“.

Menu nabízí tři možnosti, z nichž dvě jsou určeny pro nastavení filtrování a třetí slouží pro náhled na seznam úloh k vymazání. První možnost „Stav“ slouží k filtrování úloh určených k vymazání na základě jejich stavu uloženém v lokální databázi aplikace plbmng. Nachází se zde tři stavy, na základě kterých lze filtrovat viz obrázek 3.3. Jejich počet je určen enumerátorem *PlmbngJobState* z modulu *plbmng.executor*. Tento dialog je zobrazen na obrázku 3.17.



Obr. 3.17: Dialog s výběrem stavů úloh pro filtrování.

Druhá možnost „Servery“ nabízí filtrování podle serveru. Tato nabídka je téměř totožná s nabídkou zobrazenou na obrázku 3.10. Uživatel zde vybere servery, jejichž úlohy chce vymazat.

Třetí, poslední, možností je „Náhled úloh k vymazání“, která uživateli nabízí náhled na úlohy, které byly na základě jím zadaných filtrů vybrány k vymazání. Tyto úlohy jsou seskupeny podle serverů, na kterých běžely pro jednodušší orientaci v dialogu. Pokud uživatel s výběrem není spokojen, může se vrátit o krok zpět a patřičně filtry upravit. Pokud uživatel souhlasí s vymazáním uvedených úloh, potrvrdí výběr tlačítkem „Vymazat úlohy“. Úlohy jsou následně vymazány a výsledek je oznámen uživateli v dialogovém okně.

Operace vymazání úloh provádí nejprve smazání úloh ze vzdálených serverů. To je realizováno tak, že ze všech vybraných serverů je stažen soubor *jobs.json* pomocí paralelního SSH klienta. Následně jsou z těchto souborů vymazány záznamy o úlohách, jež byly určeny ke smazání. Takto modifikované soubory jsou následně distribuovány zpět na servery. Tyto operace zajišťuje funkce *delete_remote_host_jobs*. Následně jsou v rámci operace vymazání vymazány také výpisy úloh uložené lokálně a nakonec je také vymazán záznam úlohy z lokální databáze aplikace plbmng.

3.5 Refaktorizace aplikace a její vylepšení

Aplikace během vývoje v rámci této práce prošla výraznou refaktorizací, tedy zpřehledněním zdrojového kódu a také implementací jiných technologií zvyšujících komfort užívání a vývoje aplikace. Předchozí verze aplikace plbmng ukládaly konfiguraci aplikace a všechny její databáze mezi ostatní soubory do adresáře, kde se nachází zdrojový kód aplikace. Při takovém způsobu ukládání by mohlo při aktualizaci aplikace dojít např. k přepsání databází či konfigurací obsahujících cenná data. Je nepřípustné, aby kterákoli aplikace při její sebemenší aktualizaci přepsala databáze či konfigurační soubory. Proto bylo přistoupeno k přesunu těchto souborů do domovského adresáře uživatele, který aplikaci spouští – `~/.plbmng`. Výpis 7 zobrazuje strukturu tohoto adresáře po prvním spuštění aplikace.

```
~/.plbmng
├── settings.yaml
├── database
│   ├── user_servers.node
│   ├── internal.db
│   ├── last_server.node
│   └── default.node
└── geolocation
    ├── vega.json
    └── plbmng_server_map.html
```

Výpis 7: Struktura složky ~/.plbmng po instalaci aplikace.

Nachází se zde adresář `database`, který je určen pro ukládání lokální SQLite databáze – `internal.db`, souboru `user_servers.node` pro konfiguraci aplikací spravovaných uzlů mimo síť PlanetLab, dále soubor `default.node` obsahující seznam uzlů v síti PlanetLab k datu vydání právě používané verze aplikace plbmng. Posledním souborem v tomto adresáři je `last_server`, který se zde však nenachází ihned po prvním spuštěním aplikace, nýbrž po přistoupení na kterýkoli server pomocí aplikace.

Soubor `default.node` je distribuován spolu s aplikací pro účel rychlejšího prvního použití aplikace, tedy aby uživatel nemusel při prvním spuštění vytvářet tento soubor aktualizací dat z PLCAPI. Uživatel má možnost obsah tohoto souboru kdykoli aktualizovat, a to pomocí možnosti „Aktualizovat seznam serverů (Update server list)“ v menu „Monitorování serverů (Monitor servers) “. Při prvním spuštění aplikace je databáze `internal.db` naplněna daty právě z `default.node`, jak je popsáno v kapitole 3.3.

Dalším adresářem v `~/.plbmng` je adresář `geolocation`, který je po prvním spuštění aplikace prázdný, avšak mohou se v něm vyskytovat dva soubory: `plbmng_server_map.html` obsahující vygenerovanou mapu s uzly a pomocný soubor pro vytvoření této mapy `vega.json`.

Posledním souborem v tomto adresáři je `settings.yaml`, ve kterém je uložena konfigurace aplikace. Tento soubor je zobrazen na výpisu 8 v takové podobě, v jaké jej uživatel nalezne po prvním spuštění. Chybí zde konfigurace v sekci `PLANETLAB`, do které uživatel vyplní název dílu, uživatelské jméno a heslo, které používá v rámci sítě PlanetLab. Poslední požadovanou informaci k dolnění je `SSH_KEY` v sekci `REMOTE_EXECUTION`. Zbylá nastavení v tomto souboru pak popisují názvy souborů, které budou použity pro jednotlivé databáze či mapové soubory.

Pro vytvoření tohoto souboru je použita knihovna *Dynaconf*, která poskytuje abstrakční vrstvu pro konfiguraci aplikací psaných v programovacím jazyce Python. Tato knihovna je integrována do nově vzniklého modulu `plbmng.utils.config`. Ten je importován do všech ostatních modulů aplikace, které využívají některou z hodnot obsažených v konfiguračním souboru. Právě tento modul obstarává existenci složkové struktury zobrazené na výpisu 7 a také poskytuje ostatním modulům pomocné funkce pro získání cesty k databázovým, geolokačním případně jiným souborům.

Výhodou konfigurace skrze *Dynaconf* je snadné rozšíření konfigurace s téměř nulovým úsilím. Knihovna *Dynaconf* navíc nabízí možnost validace konfigurace při jejím načítání pomocí validátorů. Validátory mohou například kontrolovat přítomnost hodnot pro zvolené klíče, případně pokud je klíčem cesta, může *Dynaconf* ověřit zda soubor v této cestě existuje. V případě negativního výsledku validace je uživateli vypsána hláška popisující konkrétní chybu v konfiguraci aplikace. S vytvořením modulu `plbmng.utils.config` byl smazán soubor `plbmng.conf` z adresáře `conf`.

Dalším rozšířením bylo přidání modulu `plbmng.utils.logger`. Tento modul je importován ostatními moduly aplikace plbmng, které informují uživatele pomocí výpisů (logů). Pro vypisování logů byla zvolena knihovna *Loguru* svou jednoduchou implementaci a použití. S vytvořením tohoto modulu byl odstraněn soubor `plbmng.log` nacházející se ve složce `logs`. Momentálně je nastaven výpis logů na standardní výstup a při případném budoucím rozhodnutí o logování do souboru je tato změna pouze otázkou změny jedné hodnoty v modulu.

```yaml
---
plbmng:
  DATABASE:
    LAST_SERVER: last_server.node
    PLBMNG_DATABASE: internal.db
    USER_NODES: user_servers.node
    DEFAULT_NODE: default.node
  GEOLOCATION:
    MAP_FILE: plbmng_server_map.html
  LOAD_DOTENV: true
  PLANETLAB:
    PASSWORD: ''
    SLICE: ''
    USERNAME: ''
  REMOTE_EXECUTION:
    CONNECTION_TIMEOUT: 30
    COMMAND_TIMEOUT: 60
    SSH_KEY: ''
```

Výpis 8: Obsah konfiguračního souboru `settings.yaml` po prvním spuštění aplikace.

Spouštěcí skript ve složce `bin` byl nahrazen souborem `__main__.py`. Tento nově vzniklý soubor byl referencován v instalační konfiguraci jako skript, který se má spouštět po zadání příkazu `plbmng`.

Kód celé aplikace byl naformátován nástrojem *Black* od autorů z Python Software Foundation. Tento nástroj formátuje kód a tím jej činí přehlednějším. *Black* produkuje kód v souladu se standardem PEP 8 definujícím standard formátování kódu [18].

Došlo také ke změně způsobu ukládání závislostí aplikace plbmng a způsobu její instalace. Bylo upuštěno od standardní instalace pomocí knihovny *setuptools*, namísto které byla použita knihovna *Poetry*. K této změně bylo přistoupeno z důvodu možnosti zamknutí verzí závislostí. Je tedy garantováno, že při každé uživatelské instalaci plbmng budou uživateli nainstalovány přesně v těch verzích, jaké jsou specifikovány v konfiguraci v repozitáři aplikace plbmng. *Poetry* bylo do plbmng integrováno pro zjednodušení budoucího vývoje aplikace a také díky benefitu zamykacího mechanismu závislostí.

Do repozitáře aplikace byl také přidán soubor `pyproject.toml`, který sdružuje konfiguraci pro většinu nástrojů v repozitáři plbmng používaných. Tento soubor je v souladu se zněním PEP 518 definující strukturu tohoto souboru. Tento soubor sdružuje informace o projektu samotném, tedy informace o názvu, verzi, licenci a autorech aplikace i odkazech na dokumentaci a stránku s repozitářem aplikace. Nechybí zde právě také definice závislostí zmíněná v předchozím odstavci. Přítomna je zde také konfigurace pro *Black*, *BumpVer* a *Sphinx*, z nichž poslední dva zmíněné nástroje budou popsány.

Do procesu vývoje byly zakomponovány nástroje *BumpVer*, *flake8* a *pre-commit*, jejichž integrace byla provedena za účelem zjednodušení a zpřehlednění práce při vývoji aplikace. První jmenovaný nástroj *BumpVer* je použit pokaždé, když se vývojář plbmng rozhodne vydat novou verzi. Nástroj dle své konfigurace v souboru `pyproject.yaml` aktualizuje všechny textové řetězce, které obsahují informaci o verzi aplikace plbmng a ty následně povýší dle požadavků vývojáře. Použití nástroje *BumpVer* je popsáno v souboru `README.rst` v kořeni repozitáře plbmng. Dalším implementovaným nástrojem je *pre-commit*, který zjednodušuje kontrolu kvality kódu před vydáním revize ve verzovacím systému git. Tento nástroj je spuštěn těsně před vydáním nové revize. Pokud nástroj *pre-commit* najde v část kódu, která je v rozporu s definovaným pravidly pro úpravu kódu, nová verze není vydána a uživateli jsou všechny nalezené chyby prezentovány a očekává se, že je vývojář před vydáním revize opraví. Konfigurace nástroje *pre-commit* se nachází v souboru `.pre-commit-config.yaml`. Zde je aktuálně implementovaná např. kontrola syntaxe různých typů souborů, kontrola přebytečných mezer na koncích řádků aj. Je zde také nakonfigurován nástroj *Black* či *flake8*.

Nástroj *flake8* byl v aplikaci již implementován dříve, avšak na jiném místě, a to v konfiguraci průběžné integrace služby GitLab. Tento nástroj slouží k vynucování konzistence stylů v Python projektech. Mimo výchozí kontroly, provádí také kontroly, které jsou definovány pluginy pro *flake8*. Pluginy, které jsou v plbmng využívány lze konfigurovat v souboru `pyproject.toml` v sekci `tool.poetry.dev-dependencies`. Konfigurace *flake8* se nachází v souboru `.flake8`, který specifikuje maximální délku řádků, moduly vyloučené z kontroly, styl dokumentačních řetězců a seznam ignorovaných chyb.

Zásadní změnou bylo také přidání chybějících dokumentačních řetězců ke všem funkcím v projektu plbmng. Příklad takového dokumentačního textového řetězce je uveden na výpisu 9.

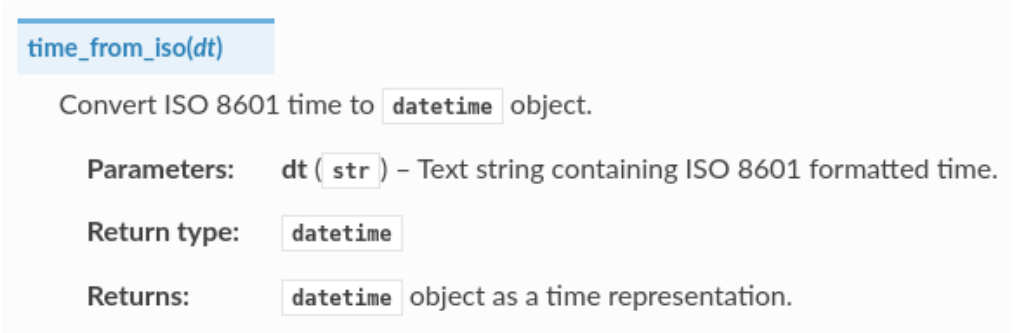
Na prvním řádku řetězce je uveden stručný popis funkce a její účel. Dále následuje sekce s popisem vstupního parametru a návratové hodnoty funkce. Každá funkce v projektu plbmng je nyní takto dokumentována. Pro generování dokumentace byl použit nástroj *Sphinx*, který vytváří z těchto dokumentačních textových

```python
def time_from_iso(dt: str) -> datetime:
    """
    Convert ISO 8601 time to :py:class:`datetime
    <datetime.datetime>` object.

    :param dt: Text string containing ISO 8601 formatted time.
    :return: :py:class:`datetime <datetime.datetime>` object as
    a time representation.
    """
    return datetime.strptime(dt, "%Y-%m-%dT%H:%M:%S.%f")
```

Výpis 9: Příklad dokumentačního textového řetězce u funkce.

řetězců výslednou dokumentaci ve zvoleném formátu. Výchozím generovaným formátem je HTML, kdy výsledek generování dokumentace nástrojem *Sphinx* je možné prohlédnout prostřednictvím webového prohlížeče. Výslednou podobu dokumentace pro funkci `time_from_iso` zobrazenou na výpisu 9 lze vidět na obrázku 3.18.

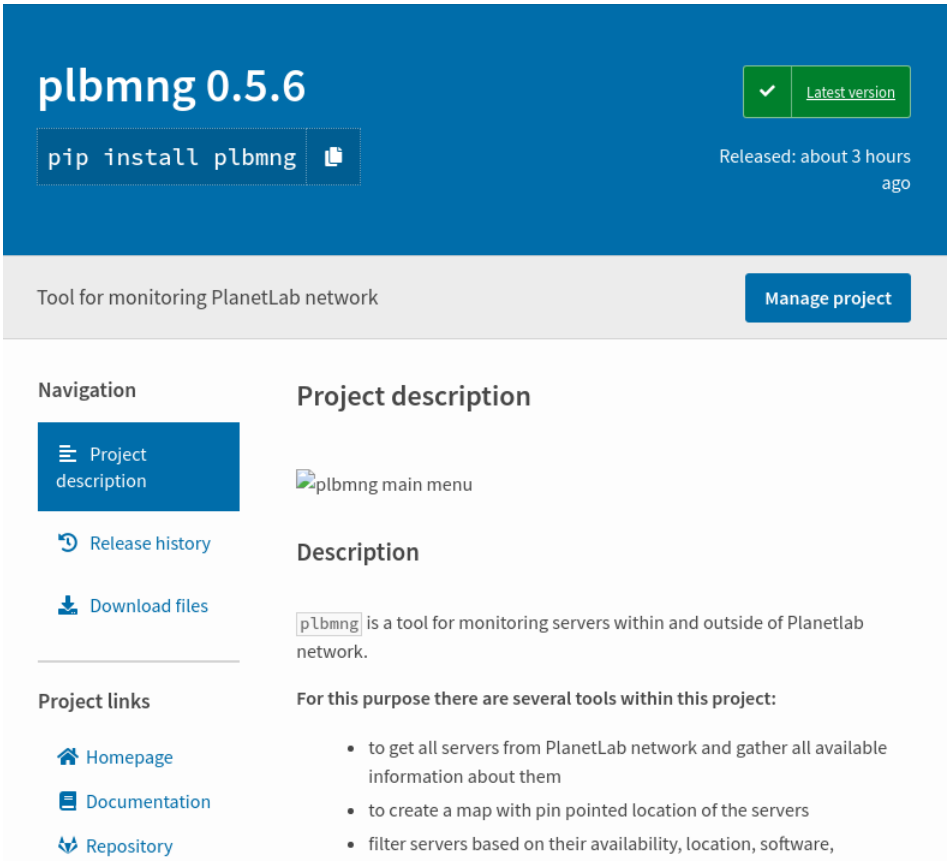


Obr. 3.18: Výsledná podoba dokumentace generované nástrojem *Sphinx*.

3.6 Zveřejnění na repozitáři PyPI

Posledním krokem při vydávání nové verze aplikace je její zveřejnění v repozitáři PyPI. Vývojáři zde publikují aktuální verze svých aplikací a knihoven napsaných v jazyce Python. Uživatelům těchto aplikací slouží PyPI jako zdroj, odkud si je stahují a instalují pomocí příkazu `pip install` ve formě balíčků. Každý zde publikovaný balíček má svou stránku ve webovém rozhraní PyPI, kde jsou obsaženy nejdůležitější informace jako je jeho popis, historie vydání, odkazy na dokumentaci,

domovskou stránku a dokumentaci balíčku. Lze zde nalézt také klasifikátory, které aplikaci zařazují do určitých kategorií např. podle cílové skupiny lidí, pro něž je aplikace určena, dále pak podle podporovaného operačního systému, licence aplikace či tématu, kterému se aplikace věnuje. Tato stránka je zobrazena na obrázku 3.19. Zobrazené tlačítko „Manage project (Spravovat projekt)“ je viditelné pouze pro administrátory projektu a nabízí možnosti pro administraci projektu.



Obr. 3.19: Stránka projektu plbmng ve webovém rozhraní PyPI.

Pomocí platformy Google BigQuery byla sesbírána data obsahující počet stažení aplikace plbmng z repozitáře PyPI za posledních 6 měsíců. Data jsou zobrazena v tabulce 3.1. Postup získání těchto metrik je popsán v manuálové stránce [17] přímo na stránkách jazyka Python.

Počet stažení	Kalendářní měsíc
1040	05-2021
824	04-2021
1020	03-2021
970	02-2021
939	01-2020
723	12-2020
1066	11-2020

Tab. 3.1: Počet stažení aplikace plbmng za posledních 6 měsíců.

3.7 Zveřejnení aplikace a její dokumentace na platformě GitLab

Většina aplikací licencovaná pod open-source licencemi, tedy licencemi s otevřeným kódem, je zveřejněna na veřejných platformách pro sdílení kódu a kolaboraci. Aplikace plbmng je zveřejněna ve stejnojmenném repozitáři [1] na platformě GitLab, jež je domovem pro velké množství projektů a poskytuje velké množství funkcionalit pro podporu vývoje aplikací verzovaných verzovacím systémem *Git*. Aplikace zde byla již zveřejněna před započetím práce na ní, proto zveřejnění finální verze aplikace na stejném místě je logickým krokem.

Hlavním důvodem pro použití platformy GitLab je její funkce průběžné integrace a také jiné nástroje, jako například systém pro hlášení problémů s aplikací. Proces průběžné integrace souhrn vývojářských nástrojů pro urychlení a zpřehlednění vývoje software. Slouží pro urychlení nalezení nedostatků aplikace během jejího vývoje. V kořenovém adresáři plbmng se nachází soubor `.gitlab-ci.yml`, psaný v jazyce YAML, definující seznam činností, které se mají v rámci průběžné integrace provádět. Tyto činnosti jsou rozděleny do tří logických fází: „lint“, „build“ a „deploy“, z nichž každá obsahuje jednu nebo více úloh. Tyto fáze jsou spuštěny ve stejném pořadí, v jakém byly vyjmenovány. Proces průběžné integrace je spouštěn při každé nové revizi publikované v kterékoli větvi repozitáře. Podmínky pro spuštění jednotlivých úloh se nachází v jejich definici.

Konfigurace průběžné integrace byla díky změnám uvedených v předchozí kapitole značně obměněna a byly v ní použity právě tyto nově integrované nástroje.

[1] https://gitlab.com/utko-planetlab/plbmng/

Úloha lint-check

První spouštěnou úlohou v rámci průběžné integrace je úloha „lint-check“ – jediná definovaná ve fázi „lint“. Tato má za cíl zkontrolovat kvalitu kódu pomocí nástroje *flake8* a ostatních nástrojů definovaných v konfiguraci nástroje *pre-commit*. Selže-li tato kontrola a jsou nalezeny části kódu, které nejsou v souladu s definovanými pravidly, je tato skutečnost uživateli oznámena v samotném webovém rozhraní GitLab. V závislosti na nastavení konkrétního uživatele může být tato skutečnost oznámena také emailem tomu, kdo revizi provedl.

Úloha build-sdist

V rámci této úlohy je spuštěn příkaz pro sestavení balíčku aplikace plbmng. Tento balíček obsahuje všechny soubory potřebné k sestavení balíčku pro konkrétní platformu. Jmenovitě `setup.py`, `pyproject.toml`, `README.rst`, licence a všech zdrojových souborů aplikace. Sestavení balíčku je podmínkou pro úspěšnou publikaci aplikace plbmng na repozitář PyPI, které se provádí ve úloze „publish“. Pokud tedy neproběhne sestavení balíčku úspěšně, poslední fáze – „deploy“ se neprovede.

Úloha pages

Účelem této úlohy je sestavení dokumentace pomocí nástroje *Sphinx* a její publikace na webovou stránku [2], kterou GitLab bezplatně poskytuje. Poté, co je dokumentace úspěšně sestavena, jsou její artefakty, tedy vygenerovaná dokumentace ve formátu HTML, zveřejněny na zmíněné webové stránce. Úloha je spouštěna pouze tehdy, je li přidána nová revize do větve *master*. Tím je zajištěna přítomnost aktuální verze dokumentace na této stránce. Úloha „pages“ se nachází v poslední fázi – „deploy“.

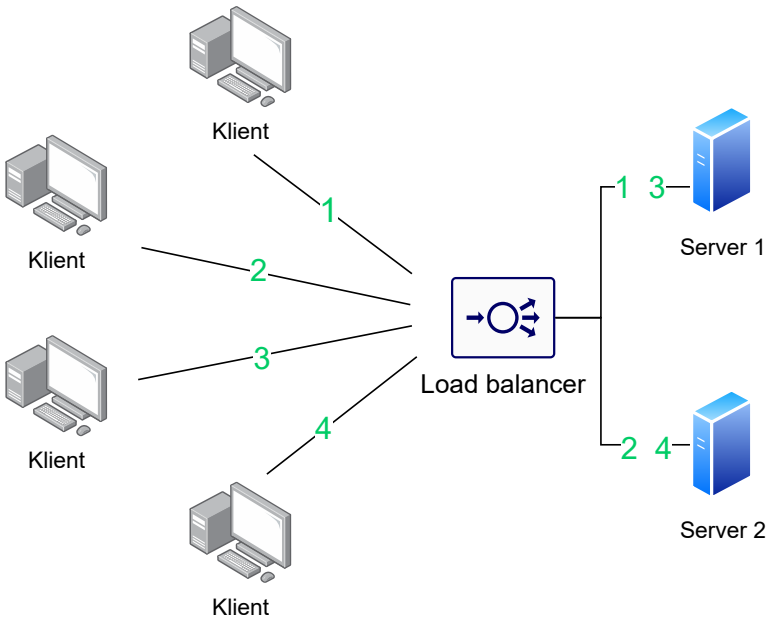
Úloha publish

Poslední úloha - „publish“ provádí sestavení balíčku. Tato úloha je spouštěna pouze tehdy, pokud úloha „build-sdist“ proběhla úspěšně, tzn. balíček byl úspěšně sestaven. Pomocí nástroje *Poetry* je balíček následně nahrán na repozitář PyPi. Pro nahrání je potřeba mít v konfiguraci tajných proměnných pro průběžnou integraci přidanou proměnnou `PYPI_TOKEN`, ve které je uložen autentizační token vygenerovaný administrátorem projektu plbmng na PyPI. Bez této proměnné úloha selže. Spuštěna je pokaždé, když je do repozitáře publikován nový štítek s číslem verze aplikace. Tato úloha značně snižuje úsilí vývojáře potřebné pro vydání nové verze.

[2] https://utko-planetlab.gitlab.io/plbmng/

4 Použití aplikace a ověření její funkčnosti

Tato kapitola poukazuje na důkaz funkčnosti aplikace plbmng. V rámci tohoto experimentu jsou využity všechny nově přidané funkce aplikace. Pro tento experiment byl vybrán scénář, při němž bude nainstalována distribuovaná aplikace, které je předřazen server, rovnoměrně rozkládající zátěž mezi skupinu serverů, na nichž běží zvolená aplikace (*Load Balancer*). Technika vyvažování zátěže je dnes hojně používána u služeb, které obsluhují stovky či desetitisíce uživatelů najednou. Důvodem jejich použití je dosažení optimálního využití, prostupnosti nebo času odezvy služby. Použití této techniky může také zvýšit odolnost vůči výpadkům služby [19]. Existují různé techniky vyvažování zátěže, pro tento experiment byla však vybrána jedna z nejjednodušších – Round Robin. Tato technika předává příchozí žádosti o připojení cyklicky mezi všechny servery ve skupině. Technika nijak nerozlišuje výkonnost jednotlivých uzlů ve skupině. To ji činí nenáročnou na implementaci. Její použití je vhodné v případech, kdy všechny uzly ve skupině mají stejný výkon a dokážou tedy zpracovat požadavky za přibližně stejný čas [19]. Zjednodušené schéma metody Round Robin je zobrazeno na obrázku 4.1.



Obr. 4.1: Zjednodušené schéma metody Round Robin.

Pro účel demonstrace je aplikací stojící za load balancerem webový server poskytující statickou webovou stránku. Obsahem stránky je jednoduchý text, který obsahuje FQDN serveru, jež tuto stránku poskytuje. Pokud umístíme takovéto webové servery do skupiny za load balancer, pak poskytovaná stránka bude vždy jiná v závislosti na tom, jaký server ji sestavil. V praxi by měl webový server, na nějž je

přistupováno skrze load balancer, poskytovat jednu a tutéž stránku, nehledě na to, který server za load balancerem stránku sestavil a poskytl. Jak webový server, tak load balancer představují reálné produkční aplikace. Pro účel experimentu jsou však aplikace implementované jako jednoduché skripty. Webový server je implementován jako jednoduchý skript v jazyce Python využívající implementaci HTTP serveru, jež je distribuována se standardní instalací Pythonu. Load balancer je implementován také jako prostý skript v tomtéž jazyce. Tento byl získán ze služby GitHub[1]. Oba tyto skripty jsou uvedeny v příloze, load balancer pod jménem `lb.py` a webový server pod jménem *plbmng_demo_server.py*.

Pro tento experiment byly ze sítě PlanetLab vybrány následující servery:

- Webové servery
  - planetlabeu-1.tssg.org,
  - planetlabeu-2.tssg.org,
- Load balancer
  - vicky.planetlab.ntua.gr.

Experiment si klade za cíl pomocí aplikace plbmng provést následující:

- Nainstalovat webové servery na dva hosty,
- nainstalovat službu pro vyvažování zátěže – load balancer na třetího hosta,
- spustit webové servery a load balancer,
- otestovat dostupnost obsahu poskytovaného webovým serverem,
- otestovat funkčnost load balanceru,
- zastavit všechny spuštěné služby,
- zkontrolovat výpis z load balanceru.

Pro nastavení těchto webových serverů a load balanceru bude využito aplikace plbmng. Aplikaci plbmng lze nainstalovat příkazem `pip` následovně;

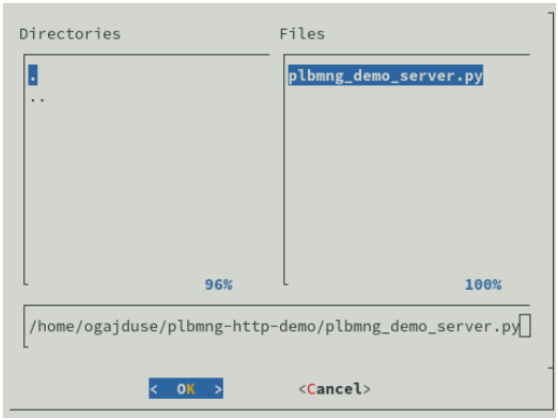
```
pip install plbmng
```

Předpokladem je nainstalovaný interpret jazyka Python a knihovna setuptools. Dalším krokem je spuštění aplikace pomocí následujícího příkazu:

```
plbmng
```

Následně je vykresleno hlavní menu aplikace, které je zobrazeno na obrázku 3.5. Z této nabídky je vybrána možnost „Spouštění úloh na serverech“, jejíž podoba je na obrázku 3.6. Veškeré zbylé kroky experimentu jsou prováděny z tohoto menu. Před samotným spuštěním webového serveru, je nutné jej nejdříve nainstalovat na cílové servery. V tomto případě je instalace jednokroková – zkopírování skriptu webového serveru na cílové servery. Pro provedení této operace byla v aplikaci plbmng otevřena nabídka „Kopírování souborů na server(y)“, v dialogu pro výběr souborů byla vybrána cesta ke skriptu, jak znázorňuje obrázek 4.2.

[1] https://gist.github.com/zhouchangxun/5750b4636cc070ac01385d89946e0a7b



Obr. 4.2: Dialog pro výběr souboru pro zkopírování.

Následně pomocí možnosti „Hledat podle DNS“ byly vyfiltrovány servery podle regulárního výrazu „*tssg*“. V zobrazené nabídce byly vybrány všechny servery, tedy *planetlabeu-1.tssg.org* a *planetlabeu-2.tssg.org* a následně byla vybrána cesta pro uložení skriptu na vzdálených serverech. Pro účel experimentu byl soubor uložen v cestě `/home/cesnetple_vut_utko/plbmng_demo_server.py`. O úspěšné operaci kopírování je uživatel informován v dialogovém okně. Vybrané servery lze vidět na obrázku 4.3.

Obr. 4.3: Výběr serverů vyfiltrovaných podle jejich FQDN.

Následně je nutné tyto webové servery spustit. Toho bylo dosaženo pomocí funkcionality poskytované pod možností „Plánování vzdálených úloh“ v menu plbmng. Příkaz pro spuštění webového serveru „python3 plbmng_demo_server.py“ byl naplánován na takové datum a čas, aby byl spuštěn ihned po jeho naplánování. Cílové servery byly vybrány stejným způsobem jako při operaci kopírování. Dalším krokem je otestování funkčnosti obou spuštěných webových serverů. Toho bylo dosaženo pomocí webového prohlížeče, ve kterém byla načtena tato stránka. Úspěšně načtené stránky lze vidět na obrázku 4.4. Oba servery poslouchají na portu 8000.

Pro dodatečné ověření správné funkčnosti webových serverů je možné v aplikaci zobrazit stav běžící úlohy, která spustila webový server. Nejprve bylo nutno

Obr. 4.4: Webová stránka poskytovaná spuštěnými webovými servery.

aktualizovat stav naplánovaných, běžících úloh. Toho bylo docíleno výběrem možnosti „Aktualizace stavu úloh“ v menu aplikace plbmng. Po aktualizaci byl zobrazen stav obou úloh pomocí možnosti „Zobrazit nedokončené úlohy“, následného výběru serveru a konkrétní úlohy. Dialog zobrazující detailní informace o běžící úloze je zobrazen na obrázku 4.5. Dialog informuje o času spuštění úlohy, stavu běžící úlohy a jeho výsledku, který stále není znám. Ten bude znám až po skončení úlohy.

Obr. 4.5: Detaily o spuštěné úloze, jež spustila webový server.

Dalším krokem po úspěšném ověření funkčnosti obou webových serverů je instalace load balanceru a jeho spuštění. Před jeho distribucí na vzdálené servery je nejdříve zapotřebí upravit jeho konfiguraci. Tato konfigurace se nachází v hlavičce skriptu `lb.py`. V konfiguraci je třeba definovat skupinu serverů, na něž má load balancer rozkládat zátěž. Tato konfigurace je zobrazena na výpisu 10. Taktéž je nutno upravit doménové jméno serveru, na němž load balancer běží. Řádek definující toto doménové jméno se nachází na samotném konci skriptu, viz výpis 11. V konfiguraci load balanceru je pro naslouchání klientským požadavkům nastaven port 5555.

Stejně jako v případě instalace webových serverů, i zde bylo využito funkcionality menu „Kopírování souborů na server(y)“. Postup je totožný jako v předchozím případě, tedy požadovaný server *vicky.planetlab.ntua.gr* byl nalezen pomocí dialogu

```
SERVER_POOL = [
    ('planetlabeu-1.tssg.org', 8000),
    ('planetlabeu-2.tssg.org', 8000)
]
```

Výpis 10: Konfigurace skupiny serverů pro rozložení zátěže.

```
if __name__ == '__main__':
    try:
        LoadBalancer('vicky.planetlab.ntua.gr', 5555, 'round
        ↪  robin').start()
    except KeyboardInterrupt:
        print "Ctrl C - Stopping load_balancer"
        sys.exit(1)
```

Výpis 11: Úprava doménového jména load balanceru.

„Hledat podle DNS“ a byl na něj nakopírován skript load balanceru s patřičně upravenou konfigurací.

Tento skript byl nakopírován do cesty `/home/cesnetple_vut_utko/lb.py`. Následně byl pomocí dialogu na něm naplánován příkaz `python lb.py`, jež load balancer spustí. Bylo postupováno stejně jako v případě spouštění webových serverů, tedy skrze průvodce naplánováním v nabídce „Plánování vzdálených úloh“, s rozdílem ve výběru cílového serveru. Ověření, zda je load balancer spuštěn lze v plbmng provést stejně jako v případě kontroly spuštěných webových serverů. Ověření funkčnosti z uživatelského pohledu bylo provedeno pomocí programu `curl` v příkazové řádce. Příkazem

`curl http://vicky.planetlab.ntua.gr:5555`

bylo ověřeno, že load balancer skutečně poskytuje obsah stránek z obou serverů, mezi něž rozkládá zátěž, a že mezi nimi cyklicky přepíná. Uvedené tvrzení je dokázáno na výpisu 12. Stejný test lze provést také prostřednictvím webového prohlížeče, tedy přistoupení na URL `http://vicky.planetlab.ntua.gr:5555` a následným obnovováním stránky. Po každé by měl load balancer poskytnout stránku z rozdílného serveru. V našem případě by se měl obsah stránek měnit tak, jak je zobrazeno na obrázku 4.4.

```
$ curl http://vicky.planetlab.ntua.gr:5555
<html><head></head><body><h1>Hello from
↪  planetlabeu-1.tssg.org!</h1></body></html>
$ curl http://vicky.planetlab.ntua.gr:5555
<html><head></head><body><h1>Hello from
↪  planetlabeu-2.tssg.org!</h1></body></html>
$ curl http://vicky.planetlab.ntua.gr:5555
<html><head></head><body><h1>Hello from
↪  planetlabeu-1.tssg.org!</h1></body></html>
$ curl http://vicky.planetlab.ntua.gr:5555
<html><head></head><body><h1>Hello from
↪  planetlabeu-2.tssg.org!</h1></body></html>
```

Výpis 12: Test funkčnosti load balanceru pomocí programu curl.

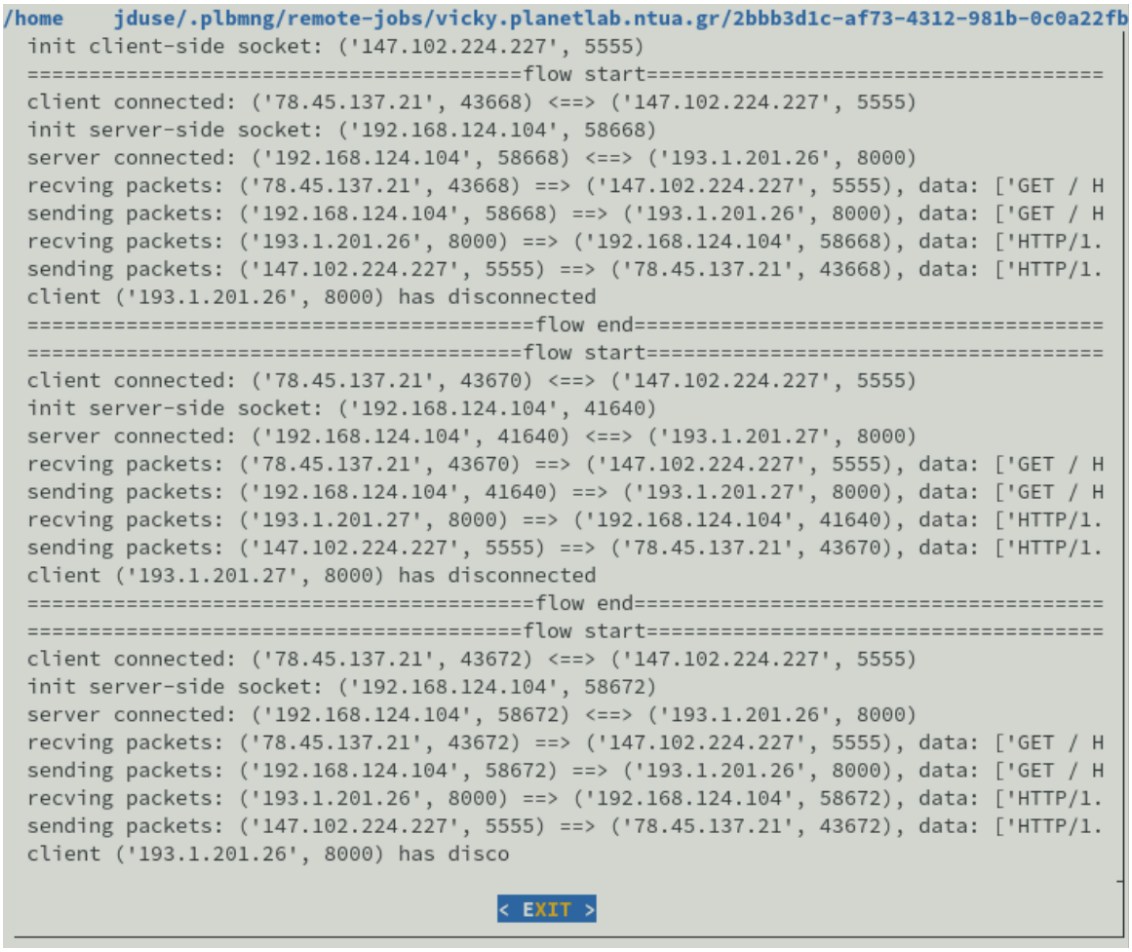
V této fázi experimentu bylo docíleno spuštění distribuované služby, která poběží do doby, než bude zastavena. Pro její zastavení je nutno ukončit proces s touto službou asociovaný. Tohoto lze opět docílit pomocí aplikace plbmng. V této fázi se může uživatel rozhodnout, zda chce, aby služba běžela do určitého času a poté byla ukončena nebo ji chce ukončit okamžitě. V prvním případě uživatel naplánuje spuštění příkazů pro ukončení procesu služby na cílových serverech na určitý čas, kdežto v případě druhém služby ukončuje v daný okamžik. Pro účel experimentu byla zvolená druhá možnost ukončení služby. Pro okamžité ukončení služby bylo použito funkcionality, jež poskytuje nabídka „Spuštění jednorázového příkazu“ v menu plbmng. Sled dialogů v tomto menu je velice podobný jako u menu pro kopírování souborů. Nejdříve je zvolen příkaz pro spuštění a následně je vybrána skupina serverů, kde je spuštěn. Pro ukončení webových serverů byl použit příkaz `pkill` následujícím způsobem;

```
pkill -f "python3␣plbmng_demo_server.py$"
```

Příkaz `pkill` má za cíl ukončit proces, který byl spuštěn spouštěcím skriptem plbmng, nikoli skript samotný. Po provedení příkazu je webový server zastaven, spouštěcí skript uloží výpisy, jež příkaz vygeneroval a aktualizuje záznam o úloze v lokální databázi úloh na vzdáleném serveru. Po ukončení webových serverů zbývá ukončit load balancer. Postup jeho ukončení je stejný jako u webových serverů s rozdílem v použitém příkazu, který v tomto případě je:

```
pkill -f "python␣lb.py$"
```

Služba byla ukončena a nyní zbývá zkontrolovat její výpisy. Pro ověření výpisů z load balanceru je nutno nejdříve aktualizovat stav úloh možností „Aktualizace stavu úloh“ v menu plbmng. Po provedení aktualizace stavu úloh bylo ověřeno, že všechny tři naplánované úlohy, tedy 2 úlohy pro webový server a 1 úloha pro load balancer, byly ukončeny. Následně bylo v menu plbmng vstoupeno do dialogu „Výstupy úloh“ a v něm zvoleno „Stažení výpisů“, čímž došlo pro dokončené úlohy ke stažení výpisů. Pro demonstraci je na obrázku 4.6 zobrazen výpis ze skriptu load balanceru, kde lze vidět jednotlivá připojení a přepínání klienta mezi zvolenou skupinou serverů.



Obr. 4.6: Detail výpisu úlohy.

Opomenuto však není ani využití funkcionality aplikace plbmng pro mazání úloh. Jelikož došlo k úspěšnému ověření výsledků úloh, je možné výsledky smazat, aby uživateli nepřekážely v pracovním prostředí. V menu plbmng je pro tento účel volba „Vymazání úloh“. V tomto dialogu byly vyfiltrovány ukončené úlohy a servery, na nichž v rámci experimentu běžela nasazená služba. Pomocí volby „Vymazat úlohy“ byl potvrzen výběr úloh k vymazání. Následně byly úlohy i s jejími výpisy vymazány

se stanice, na níž běží aplikace plbmng a také ze vzdálených serverů.

Na základě výsledků experimentu vyplývá, že je možno prohlásit ho za zdařilý. Všech stanovených cílů bylo v rámci něj dosaženo. Pomocí aplikace plbmng byla nasazena distribuovaná služba na skupinu serverů v síti PlanetLab. Nasazená služba je svou povahou velmi podobná službám produkčním. Experiment poukazuje na možnost reálného využití aplikace plbmng v praxi pro nasazování a práci s distribuovanými aplikacemi.

5 Navržená budoucí zlepšení aplikace

V této kapitole jsou navrženy potenciální zlepšení aplikace, které je možno v budoucnu implementovat. Všechna uvedená zlepšení by měla zlepšit design aplikace, případně k němu alespoň přispět. Návrhy vznikaly v průběhu práce na aplikaci plbmng. Tato vylepšení je možné je implementovat v rámci dalších prací, jež budou na tuto práci navazovat.

Vylepšení jsou rozdělena do dvou kategorií podle předpokládané náročnosti jejich implementace.

Potenciálně lehká vylepšení

Tato vylepšení byla vyhodnocena jako nenáročná na implementaci do stávající aplikace plbmng. Jejich implementace by znamenala změnu jen malé části kódu, většinou sady funkcí.

Implementace validátorů pro Dynaconf

Knihovna Dynaconf, jež v plbmng zajišťuje konfiguraci aplikace dle konfiguračního souboru, při jejím spuštění umožňuje použití tzv. validátorů. Tyto validátory jsou programové konstrukce definující podmínky, jež musí konfigurační soubor aplikace splňovat. Pokud tyto podmínky nesplňuje, pak je zobrazena výjimka a aplikace není spuštěna. Uvažujme konfigurační soubor zobrazený na výpisu 13, tedy soubor s jedním klíčem a hodnotou. Pokud chceme zajistit, aby aplikace nebyla spuštěna, pokud tento klíč v konfiguraci nebude přítomen, použijeme validátor zobrazený na výpisu 14.

```
---
plbmng:
  DATABASE:
    LAST_SERVER: last_server.node
```

Výpis 13: Příkladná konfigurace aplikace v souboru `settings.yaml`.

Oprava filtrování uzlů se spuštěnou službou SSH server

Aplikace aktuálně nabízí filtrování uzlů sítě PlanetLab dle dvou kritérií – „Ping“ a „SSH“. Druhý zmíněný filtr neodfiltruje uzly, na něž se nelze připojit pomocí SSH

```
settings = Dynaconf(
    validators=[Validator('DATABASE.LAST_SERVER',
    ↪  must_exist=True)]
)
```

Výpis 14: Příklad validátoru.

klienta. I po aplikaci tohoto filtru se ve výběru uzlů objevují i ty, k nimž se nelze přihlásit. Aplikace provádí kontrolu, zda na daném uzlu běží SSH server, už ale však netestuje, zda je uživatel schopen se na tento uzel přihlásit. Toto chování by bylo vhodné zlepšit ve prospěch větší uživatelské přívětivosti.

Vytvoření korelace mezi úlohami

Je-li prostřednictvím aplikace plbmng naplánována v jedné dávce více než jedna úloha, tedy pokud jsou použity dva nebo více serverů, všechny úlohy z této dávky budou spouštět stejný příkaz, avšak každá úloha bude mít rozdílné ID. Je tedy téměř nemožné s jistotou určit, které úlohy byly spuštěny v jedné dávce. Tento potenciální nedostatek by mohlo vyřešit přidání dodatečného identifikátoru všem úlohám naplánovaným v jedné dávce.

Vylepšení s netriviální implementací

Následující vylepšení byla vyhodnocena jako netriviální, tedy vylepšení, jejichž implementace vyžaduje výrazné změny ve struktuře kódu.

Ukládání dalších výpisů prostřednictvím spouštěcího skriptu

Spouštěcí skript v současnosti `executor.py` podporuje ukládání výpisů ze standardních proudů. Může však nastat případ, kdy uživatel bude potřebovat pomocí spouštěcího skriptu načíst také obsah souboru, jež spuštěný příkaz vytvořil. Tento soubor by byl uložen do složky `artefacts` k příslušné úloze a následně by mohl být stažen do aplikace plbmng. Také samotný spouštěcí skript by potenciálně mohl ukládat své výpisy do této složky. Jeho výpis by pak mohl sloužit pro účely ladění.

Abstrakce uzlů

V aplikaci je na mnoha místech operováno s datovou strukturou typu slovník, ve které jsou uloženy informace o uzlu sítě PlanetLab. Operace s touto strukturou by mohla zjednodušit implementace třídy *PlbmngServerNode*, která by nesla všechny atributy uzlu sítě.

Implementace vícekrokového filtrování uzlů

V současné implementaci aplikace není možné volit více než jeden typ filtru při provádění jakékoli operace, při níž je vybírána skupina serverů. Tento design by mohl být vylepšen tak, aby při jedné operaci se skupinou serverů mohlo být použito více typů filtrů. Tedy např. filtrování pomocí DNS a zároveň lokality serveru.

Použití objektově relačního mapování

Současná implementace modulu pro manipulaci s databází není řešena komplexně a jeho současný design otevírá prostor chybám. Další rozšíření databázového modulu by mohlo zvýšit jeho komplexitu. Vhodnou úpravou lze upravit modul zajišťující operace s databází tak, aby pro každou tabulku v databázi existovala jedna třída, která tuto tabulku popisuje. Při použití této techniky by bylo možné také realizovat upgrady aplikace bez rizika zničení databáze uživatele, případě nefunkčnosti aplikace po upgradu.

Závěr

Cílem této diplomové práce bylo přidání funkcionality do aplikace PlanetLab Server Manager implementované v jazyce Python. Tato aplikace usnadňuje vývojářům aplikací práci s distribuovanými síťovými službami. Aplikace z důvodu provozu na stanicích bez grafického uživatelského rozhraní implementuje textové uživatelské rozhraní.

První kapitola popisuje a analyzuje stávající stav aplikace, veškerou její funkcionalitu a její použití. Funkcionalita implementovaná v rámci této práce zahrnuje plánování úloh na vzdálených serverech, možnost monitorování stavu úloh. Distribuce software na vybrané servery je realizována pomocí kopírování souborového archivu a instalačního skriptu na vzdálený server. Tento skript je následně pomocí příkazu z aplikace plbmng spuštěn a software je nainstalován. Jeho následné spuštění v uživatelem stanovený čas je rovněž realizováno pomocí aplikace. Při plánování a jiných operacích podporujících hromadné akce, je uživateli prezentován dialog pro filtrování cílových serverů, pomocí kterého vybere výslednou skupinu serverů, které budou použity pro zvolenou akci.

Při implementaci procesu plánování úloh nebyla opomenuta problematika časových zón, tedy zajištění spuštění tohoto software ve stejný okamžik bez ohledu na časovou zónu, ve které se daný systém nachází.

Aplikace uživateli nabízí možnost následné kontroly výsledků spuštěných úloh, která je taktéž možná přímo z rozhraní aplikace. U skončených úloh aplikace nabízí možnost prohlížení výpisů, jež byly úlohou vyprodukovány. Tím je uživateli poskytnut jednoduchý způsob sběru těchto výpisů.

Poslední přidanou funkcionalitou do aplikace je mazání úloh dle uživatelem zvolených kritérií. Tímto je seznam úloh možno udržovat aktuální a přehledný. V aplikaci došlo k utřízení položek v dialozích z důvodu zvýšení přehlednosti aplikace.

Funkcionalita aplikace byla demonstrována na experimentu, který využívá veškeré implementované funkcionality a ukazuje fungování aplikace v praxi. Všechny vytyčené cíle experimentu byly splněny a z toho vyplývá, že všechna nově implementovaná rozšíření jsou funkční.

Práce také popisuje realizovaná zlepšení a jejich opodstatnění. Nově přidaná funkcionalita byla řádně dokumentována pomocí dokumentačních textových řetězců. Výsledná dokumentace je automaticky generována a zveřejňována na veřejně přístupné webové stránce. Verze aplikace byla povýšena a celá aplikace byla zveřejněna na repozitáři Python balíčků – PyPI a na platformě GitLab.

Literatura

[1] PlanetLab Migrating to LXC. *PlanetLab* [online]. 2012, 19. září 2012 [cit. 2020-12-10]. Dostupné z: https://planetlab.cs.princeton.edu/node/263.html

[2] VERMEULEN, Kevin, Burim LJUMA, Olivier FOURMAUX, Timur FRIEDMAN a Rick MCGEER. Internet measurements on EdgeNet. *CNERT: Computer and Networking Experimental Research using Testbeds in conjunction with IEEE INFOCOM 2019* [online]. Paris, France, 2019, 20 Mar 2019, , 1-4 [cit. 2020-12-10]. Dostupné z: https://hal.archives-ouvertes.fr/hal-02073681

[3] NAVRÁTIL, Jiří. PlanetLab - model budoucího Internetu. *Zpravodaj ÚVT MU* [online]. 2006, **16**(5), 1-5 [cit. 2020-12-10]. ISSN 1212-0901. Dostupné z: http://webserver.ics.muni.cz/bulletin/articles/525.html

[4] PETERSON, Larry. It's Been a Fun Ride. In: *Systems Approach* [online]. 2020, 3/19/2020 [cit. 2020-12-10]. Dostupné z: https://www.systemsapproach.org/blog/its-been-a-fun-ride

[5] PlanetLab Terminology. *PlanetLab Europe* [online]. [cit. 2020-12-10]. Dostupné z: https://planet-lab.eu/doc/guides/user/terminology

[6] PlanetLab Central API Documentation. *PlanetLab Europe* [online]. [cit. 2020-12-10]. Dostupné z: https://www.planet-lab.eu/db/doc/PLCAPI.php

[7] PlanetLab Node Manager API Documentation. *PlanetLab Europe* [online]. [cit. 2020-12-10]. Dostupné z: https://www.planet-lab.eu/planetlab/doc/NMAPI.pdf

[8] KELEMEN, Ondřej. *Detekce SSH serverů na počítačové síti* [online]. Brno, 2018 [cit. 2020-12-10]. Dostupné z: https://is.muni.cz/th/pvyqe/. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Pavel Čeleda.

[9] ANDRAŠOV, Ivan. *Měření experimentální sítě PlanetLab* [online]. Brno, 2018 [cit. 2020-12-10]. Dostupné z: http://hdl.handle.net/11012/70280. Bakalářská práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Dan Komosný.

[10] KAČMARČÍK, Martin. *Aplikace pro monitorování serverů s operačním systémem Linux* [online]. Brno, 2019 [cit. 2020-12-10]. Dostupné z: http://hdl.handle.net/11012/177604. Diplomová práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Dan Komosný.

[11] SCHEEPERS, Mathijs Jeroen. Virtualization and containerization of application infrastructure: A comparison. *21st twente student conference on IT*. 2014, **21**, 3.

[12] ŠUBA, Filip. *Monitorování serverů s OS Linux* [online]. Brno, 2018 [cit. 2020-12-10]. Dostupné z: http://hdl.handle.net/11012/82182. Bakalářská práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Dan Komosný.

[13] ŠUBA, Filip. *Správa serverů s operačním systémem Fedora* [online]. Brno, 2020 [cit. 2020-12-10]. Dostupné z: http://hdl.handle.net/11012/189108. Diplomová práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Dan Komosný.

[14] MŮČKA, Jan. Typy virtualizace serverů — je lepší KVM nebo LXC? In: *MasterDC* [online]. c2020, 05. 02. 2020 [cit. 2020-12-11]. Dostupné z: https://www.master.cz/blog/typy-virtualizace-serveru-kvm-nebo-lxc/

[15] MARTIN, John Paul, A. KANDASAMY a K. CHANDRASEKARAN. Exploring the support for high performance applications in the container runtime environment. *Human-centric Computing and Information Sciences* [online]. 2018, **8**(1), 4-6 [cit. 2020-12-11]. ISSN 2192-1962. Dostupné z: doi:10.1186/s13673-017-0124-3

[16] PAPAZIS, Kon a Naveen CHILAMKURTI. Detecting indicators of deception in emulated monitoring systems. *Service Oriented Computing and Applications* [online]. 2019, **13**(1), 26 [cit. 2020-12-11]. ISSN 1863-2386. Dostupné z: doi:10.1007/s11761-018-0252-2

[17] Analyzing PyPI package downloads. *Python Packaging User Guide*. Python Software Foundation, 2019. Dostupné také z: https://packaging.python.org/guides/analyzing-pypi-package-downloads/

[18] VAN ROSSUM, Guido, Barry WARSAW a Nick COGHLAN. PEP 8 – Style Guide for Python Code. *Python Developer's Guide: PEP Index* [online]. Python Software Foundation, 2013, 5 Jul 2001 [cit. 2021-5-24]. Dostupné z: https://www.python.org/dev/peps/pep-0008/

[19] PONT, Martin. *Vyvažovač zátěže pro protokol SIP* [online]. Praha, 2018 [cit. 2021-5-24]. Dostupné z: http://hdl.handle.net/10467/79417. Bakalářská práce. České vysoké učení technické v Praze. Vedoucí práce Pavel Troller.

Seznam symbolů, veličin a zkratek

PlanetLab Experimentální Síť PlanetLab

plbmng PlanetLab Server Mangager
Aplikace pro management silverů v experimentální síti PlanetLab

SSH Secure Shell
Kryptografický síťový protokol

LXC Linux Containers
Virtualizace na úrovni operačního systému

OS Operační systém

VM Virtual Machine

RST ReStructuredText

AMQP Advanced Message Queuing Protocol

TUI Text-based User Interface
Textové uživatelské rozhraní

UUID Universally unique identifier

JSON JavaScript Object Notation

FQDN Fully Qualified Domain Name

PyPI Python Package Index

MIT Massachusetts Institute of Technology

A Seznam odevzdaných souborů

- / ..... kořenový adresář
  - experiment ..... složka s aplikacemi pro experiment
    - lb.py
    - plbmng_demo_server.py
  - plbmng ..... adresář aplikace plbmng
    - pyproject.toml
    - LICENSE
    - .flake8
    - README.rst
    - requirements.txt
    - .gitlab-ci.yml
    - poetry.lock
    - .pre-commit-config.yaml
    - .gitignore
    - plbmng
      - executor.py
      - __init__.py
      - engine.py
      - __main__.py
      - database
        - user_servers.node
        - default.node
      - lib
        - full_map.py
        - icmp_map.py
        - ssh_map.py
        - ssh.py
        - library.py
        - port_scanner.py
        - planetlab_list_creator.py
        - map_one.py
        - database.py
      - utils
        - config.py
        - logger.py
    - source
      - plbmng.lib.rst
      - README.rst
      - plbmng.utils.rst
      - index.rst
      - plbmng.rst
      - conf.py
      - images
        - target.png

/ ..... kořenový adresář
- plbmng ..... adresář aplikace plbmng
  - source
    - images
      - plot.png
      - select.png
      - add_server.png
      - extras.png
      - checklist.png
      - access_servers.png
      - plbmng.png
      - monitoring.png
      - run_jobs_on_servers.png