Gymnázium, Praha 6, Arabská 14

Programování

ROČNÍKOÝ PROJEKT

Kontejnerizační systém pro školní server

Vypracovali: Havránek Kryštof, Vávra Vladimír, Litoš Josef 3.E

Vedoucí práce: Ing. Daniel Kahoun

Duben 2021

Prohlašuji, že jsme jedinými autory tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V ..... dne ..... Podpisy autorů

Poděkování

Název práce: Kontejnerizační systém pro školní server

Autoři: Havránek Kryštof, Vávra Vladimír, Litoš Josef 3.E

Abstrakt: Cílem práce je vytvořit kontejnerizační systém pro operační systém GNU/-Linux. Ten by měl v jednoduchém a přehledném uživatelském prostředí umožnit žákům vytvořit si vlastný linuxový server. Na kterém si poté mohou hostovat svoje aplikace, webové stránky, či videohry. Systém by se měl postarat o to, aby zdroje serveru byly spravedlivě rozdělené mezi jednotlivé uživatele.

Klíčová slova: linux, lxd, lxc, kontejnery, nodejs, REST API, react js

Title: Containerization system for school server

Authors: Havránek Kryštof, Vávra Vladimír, Litoš Josef 3.E

Abstract: Goal of the work is to create containerization system for GNU/Linux operating system. With its help students should be able to create their own private server in a easy to use webUI. On such server they will be able to host their applications, websites or video games. System should also guarantee fair redistribution of server resources.

Key words: linux, lxd, lxc, containers, nodejs, REST API, react js

Titlel: Containerisierungssystem für Schulserver

Autoren: Havránek Kryštof, Vávra Vladimír, Litoš Josef 3.E

Abstrakt: Ziel des Werk ist ein Containerisierungssystem für das Betriebssystem GNU / Linux zu erstellen. Es sollte den Schülern ihren eigenen Linux-Server in einer einfachen und übersichtlichen Benutzeroberfläche möglicher zu bilden. Auf den Server können sie dann ihre Anwendungen, Websites oder Videospiele hosten, ich mag ein Brot. Das System sollte sich zusichern, damit die Serverressourcen gerecht auf die individuellen Benutzer verteilt würden.

Schlüsselwörter: linux, lxd, lxc, Schulprojekt, nodejs, REST API, react js

Obsah

Úvod

Cílem práce bylo postavit systém pro kontejnerizaci linuxového serveru, které by eventuálně mohl být nasazen na školním serveru. Systém by spravedlivě rozdělil zdroje mezi jednotlivé žáky a poskytl jim již volný prostor, kde mohou testovat svoje aplikace. Aktuálně totiž bylo problematické jen získat přístup na školní server.

O frontend práce se staral Vladimír, backend byl rozdělen mezi Kryštofa a Josefa. Kde Josef se převážně věnoval propojení mezi backendem a lxd deamonem, zatímco Kryštof zpracoval integraci mySQL databáze. Na spojení obou částí backendu a vytvoření route se podíleli oba členové týmu.

Vývoj probíhal na GitHubu – `https://github.com/havrak/AvAvA`, kde je aplikace dostupná pod licencí GPLv3. Projekt jsem testovali na vlastní VPS.

1. Architektura

Jádro práce stojí na lxd, což je kontejnerizační systém zabudovaný přímo do linuxového kernelu. Jedná se tak o nejvíce efektivní řešení, protože jednotlivé kontejnery mohou sdílet společný kernel. Jsou tak menší a rychlejší, než kdyby se jednalo o VPS. Backend s lxd komunikuje prostřednictvím REST API, které lxd přímo podporuje.

Samotný backend je napsán v nodejs a ke svému fungování využívá dva databázové systémy. Většinu informací ukládá do mySQL databáze, kvůli komplikacím s LXD byla později zavedena mongoDB.

Frontend je psán v ReactJS a s backendem komunikuje prostřednictvím vlastního REST API (jehož podrobná specifikace je uvedena v dokumentaci). Na frontendu může uživatel v pohodlném prostředí vytvářet nové kontejnery, projekty, či získat informace o jejich aktuálním stavu a mnohé další.

1.1 LXD

1.2 Autentifikace

1.3 Sytém kontejnerů

// limity

2. Frontend

3. Backend

3.1 Specifikace vlastního API

3.2 API routy

GET – /combindeDataGET

a

GET – /instances

a

GET – /instances/createInstanceConfigData

a

GET – /instaces/{id}

a

PATCH – /instaces/{id}

a

DELETE – /instaces/{id}

a

GET – /instaces/{id}/stateWithHistory

a

GET – /instaces/{id}/console

a

GET – /instaces/{id}/snapshots

a

POST – /instaces/{id}/snapshots

a

DELETE – /instaces/{id}/snapshots/{snapshotsid}

a

PATCH – /instaces/{id}/restore/{snapshotsid}

a

GET – /instaces/{id}/export

a

PUT – /instaces/import

a

PATCH – /instaces/{id}/start

a

PATCH – /instaces/{id}/stop

a

PATCH – /instaces/{id}/freeze

a

PATCH – /instaces/{id}/unfreeze

a

GET – /projects

a

POST – /projects

a

GET – /projects/stateWithHistory

a

GET – /projects/{id}

a

PATCH – /projects/{id}

a

DELETE – /projects/{id}

a

GET – /projects/{id}/stateWithHistory

a

GET – /projects/user

a

GET – /logout

a

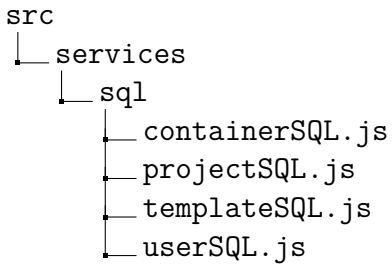
3.3 Databáze

Sytém používá dva databázové systémy – mySQL (mariaDB) a mongoDB. Dle původních představ měl používat pouze mySQL. Kvůli komplikacím s lxd, kde nejde

zjistit stav zastaveného/vypnutého kontejneru, byla později zavedena mongoDB.

3.3.1 mySQL

Následující sekce se zabývá strukturou mySQL databáze a metodami, které s databází zacházejí.



Obrázek 3.1: soubory s programem týkající se mysql databáze

Valnou většinu dat projekt ukládá do mySQL databáze, kde je na tento účel vytvořeno aktuálně 10 tabulek. S databází pracují 4 soubory (viz. **??**) *containerSQL* zpracovává věci týkající se uživatelů, *projectSQL* věci týkající se projektů, *templateSQL* věci týkající se šablon a aplikací, v neposlední řadě *userSQL* věci týkající se uživatele.

Tabulky

Tato sekce spěšně popíše obsah jednotlivých tabulek. A jejich vztah k ostatním. Kaskádové dependence jsou vždy stejné – při smazání se smažou, při update se nic neděje.

appsToInstall | id | name | description | icon_path | package_name |

Tabulka appsToInstall slouží k uložení aplikací, které je možno nainstalovat na kontejner při jeho vytváření. Nemá žádnou vazbu na další tabulky a s jejími daty zachází třída templateSQL. Sloupec name obsahuje jméno jaké se má zobrazit uživateli, package_name je jméno balíčku v repositářích.

containers | id | project_id | name | url | template_id | state | timestamp | time_started |

Tabulka container ukládá všechny kontejnery, které spravujeme. Obsahuje dva cizí klíče a to project_id, které určuje do jakého projektu kontejner patří, a template_id to určuje s jakou šablonou byl kontejner vytvořen.

containersResourcesLimits | container_id | ram | cpu | disk | upload | download |

Tabulka containersResourcesLimits ukládá limity kontejneru, stejně jako u dalších .*ResourcesLimits tabulek byla v rámci normalizačních forem oddělena od tabulky containers. Tabulka obsahuje jeden cizí klíč, který zároveň funguje i jako primární klíč. Veškeré limity jsou uloženy v základních jednotkách, cpu je v abstraktní jednotce herz.

containersResourcesLog | container_id | ram | cpu | number_of_processes | upload | download | timestamp |

Tabulka containersResourcesLog slouží k logování stavu kontejnerů. Container_id je cizím klíčem odkazující na kontejner, ke kterému log patří. V aktuální verzi jsou data uložen v poli, které uložené ve sloupci s typem text. Timestamp odkazuje na datum posledního zapsání, kdy byl zalogován předchozí stav není problém zjistit, jelikož se do tabulky zapisuje v pravidelných intervalech.

Metoda na updateování stavu (updateLogsForContainer(id, state)) si jednoduše data rozdělí dle znaku: **,**. Do pole se přidají nové hodnoty a odebere se první prvek, nový stav se pak uloží do databáze.

Data se aktuálně do databáze uloží každých 10 minut, o což se stará cronjob (potažmo schedule) definovaný v app.js. Aktuálně si databáze pamatuje dvanáct záznamů, takže dvě hodiny do minulosti. Fakt, že log časový rozdíl mezi zápisy nemusí být přesně 10 minut, příliš nevadí, grafy, které se z těchto dat vykreslují jsou spíše orientační.

projects | id | name | owner_email | timestamp |

Tabulka projects si pamatuje všechny projekty, které spravuje náš systém. Cizím klíčem je owner_email, což je email vlastníka projektu, tedy člověka, který ho vytvořil. Timestamp je čas vytvoření projektu.

projectsCoworkers | project_id | user_email |

Tabulka projectsCoworkers zporstředkovává M:N vazbu mezi users a projects. Slouží k uložení lidí, kteří jsou spolupracovnící na jednom projektu. K datu odevzdání práce, nejsou spolupracovnící implementované, takže tabulka je aktuálně zbytečná. Řada metoda nepočítá s existencí spolupracovníků.

projectsResourcesLimits | project_id | ram | cpu | disk | upload | download |

Tabulka projectsResourcesLimits slouží k uložení limitů projektu. Nerozdíl od containersResourcesLimits můžou zde mít limity hodnotu null. V takovém případě je kon-

tejnerům dostupný vešerý volný prostor, který uživatel má.

templates | code | id | profile_name | image_name | version | profile_description | image_description | profile_path | min_disk_size |

Tabulka templates slouží k uložení šablon, lde kterých se má vytvořit kontejner. Šablona je koncept, který se nenachází přímo v lxd, ale integruje dvě věci – profily a image. Image je distribuce jaká se na systém má nainstalovat. Profile je koncept z lxd, který obsahuje konfiguraci kontejneru. Eventuálně by měl uživatel možnost vytvářet si vlastní profily, které by obsahovali například konfiguraci networků.

users | id | email | given_name | family_name | icon | role | coins |

Tabulka users slouží k uložení uživatelů. Svoje data dostává z dat, které zasílá google auth 2.0. Sloupce role a coins aktuálně nemají význam, role bude dělit uživatele mezi standardního uživatel, admina a superadmin. Admin a superadmin by měli právo zasahovat do kontejnerů jiných uživatelů. Coins měl být sloupec sloužící u ekonomiky systému, k její implementaci však nedošlo.

usersResourcesLimits | user_id | ram | cpu | disk | upload | download |

Tabulka usersResourcesLimits ukládá limity uživatelů. Aktuálně má uživatel k dispozici 2.8 GHz[1], 1 GB ram, 8GB volného místa na disku, a download a upload 800kb/s.

3.3.2 mongoDB

3.4 LXD

3.5 Proxy

Následující sekce se zabývá networkingem kontejnerů a proxy, které umožňuje přístup na ně z venčí.

3.5.1 Konfigurace networků

Pomineme li loopback (standardní *lo*), tak každý kontejner má v aktuální verzi právě jeden networkový interface. Tím je *eth0*, který je na hostovacím stroji napojen

[1]server kde byl program vyvíjen měl 2 jádra o 2.8 GHz, jedná se tak o polovinu výkonu

na bridge *lxdbr0*. Přes něj mají kontejnery přístup na internet. Zároveň jsou všechny na stejné síti (prostřednictvím *lxdbr0*)

Jelikož jsou veškeré kontejnery na stejné síti, tak mezi sebou mohou komunikovat. Lxd pro tento účel samo nastavuje jejich interní doménu, ta je ve tvaru *c{id}.lxd*. Pochopitelně však nejsou dostupné z internetu, doména je pouze interní a veřejnou ip adresu nemají. Aby bylo možno na kontejner přistupovat bylo nutné nastavit proxy, které mu databáze bude přeposílat.

Nutno podotknout, že i traffic mezi kontejnery je aktuálně omezen limitem. V budoucno se tedy nabízí možnost implementovat možnost vytvářet další bridge a kontejnery si propojovat. Ty by se mohly ukládat do profilů, uživatel by si tak rovnou mohl vytvořit kontejnerů izolovaný od ostatních.

Fakt, že kontejnery jsou na stejné síti není rizikové ani problematické. Standartě se nemůžou nijak ovlivňovat.

3.5.2 Proxy

Pro proxy využívá projekt volně dostupnou haproxy, jejíž instance běží ve stejnojmenném kontejneru. Hostovací stroj totiž standardně nemůže přistupovat na kontejnery prostřednictvím .lxd domény. Bylo by možné sice mít proxy na hostovacím stroji, ale z bezpečnostních důvodů tak nebylo učiněno.

Do kontejneru haproxy je aktuálně přesměrována traffic ze čtyř portů – 80, 443, 2222 a 3000. Port 80 pochopitelně slouží na webové stránky pomocí protokolu http. 443 je pro https, proxy má nastavený ssl certifikát. Ten stačí jeden pro celý systém, musí se však jednat o certifikát typu wildcard[2]. Systém byl testován na standardním certifikátu, čily připojení sice bylo zabezpečené, ale prohlížeč hlásil podezření z fishningu, jelikož se doména a doména na certifikátu neschodovali. Port 2222 slouží na připojení přes ssh, port 22 používá hostovací stroj a z pochopitelných důvodů není přesměrován do proxy. Port 3000 je dostupný pro REST API aplikací, které na serveru budou běžet.

Kotejnerům je automaticky přirazena doména v následujícím tvaru *{jméno kotejneru}.{jméno projekty}.{část emailu uživatel, před @}.{doména serveru}*. Například kontejner pojmenovaný *rumburak* v projektu *web* uživatele *belzebub@email.com* na serveru s doménou *avava.cz* bude mít doménu rumburak.web.belzebub.avava.cz. Mezery

[2]certifikát, který zahrnuje i subdomény

ve jméně kontejneru, či projektu se domény odstraní. Pokud by měl nový kontejner stejné jméno, jako jiný systém vyhodí výjimku. Tak je zaručena unikátnost domén, u mailu není takový problém nutno řešit, jelikož se jedná o školní emaily, které jsou vždy unikátní.

Bohužel je možno elegantně forwardovat pouze http protokol. Většina TCP protokolů totiž neoperuje s SNI a řídí se pouze doménu. Z packetu tak nejde zjistit na jakou doménu se uživatel snaží dostat a tedy jakému kontejneru mý být protokol přiřazen. To by se týkalo i ssh, příkaz ssh však podporuje nastavit pomocí flagu -o ProxyCommand (viz. **??**), packety poté sni v hlavičce mají a je možno poznat jakému kontejneru patří.

```
ssh -o ProxyCommand="openssl s_client -connect $SERVERURL:2222
    -servername $CONTAINERURL" blank -l $USERNAME
```

Obrázek 3.2: příkaz k připojení na server[3]

Porty jako 8443 aktuálně forwardovány nejsou, pokud uživatel potřebuje přistupovat do datáze běžící na kontejneru je možno použít ssh local port forwarding. Tento princip je i bezpečnější.

Konfigurační soubor HAProxy je generován v containerSQL.js. Po jeho vygenerování se prostřednictvím lxd.postFileToInstance pošle nová konfigurace to HAProxy kontejneru. Restartování proxy je téměř instantní operace i s velkým konfiguračním souborem obsahujícím desítky kontejnerů. Downtime je tak minimální. Podoba konfigurace byla realizována za pomocí dokumentace HAProxy[4]. Aktuálně má však formát configu jisté rezervy, do jeho délka. Je však o trochu rychlejší než, kdyby byly použity pokročilé metody mapování, které HAProxy podporuje.

[3] $SERVERURL je url hostovacího server, $CONTAINERURL je url kontejneru, $USERNAME je jméno uživatel, argument dummy je ingnorován, je však potřeba

[4]

Závěr

Seznam obrázků