VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MODULÁRNÍ WEBHOSTING PRO VÝUKOVÉ ÚČELY

MODULAR WEBHOSTING FOR EDUCATIONAL PURPOSES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **LADISLAV KAŠPÁREK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JIŘÍ PAVELA**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

148154

Ústav: Ústav inteligentních systémů (UITS)
Student: **Kašpárek Ladislav**
Program: Informační technologie
Specializace: Informační technologie

Název: **Modulární webhosting pro výukové účely**
Kategorie: Softwarové inženýrství
Akademický rok: 2022/23

Zadání:

1. Seznamte se s problematikou webhostingu a serverových služeb sftp, git-server, Gitea, Nextcloud, WikiJS, WordPress.
2. Seznamte se s principy softwarových kontejnerů a problematikou jejich orchestrizace.
3. Navrhněte architekturu webhostingového systému (pro výukové účely) založeného na kontejnerech, který bude podporovat php, python, java backendy a alespoň tři serverové služby z bodu 1.
4. Implementujte navržený webhostingový systém včetně konfiguračního rozhraní pro správu uživatelů a jejich kontejnerů.
5. Vyhodnoťte kvalitu dosaženého řešení. Zaměřte se jak na technické hledisko práce (např. vytížení serveru, doba odezvy, míra automatizace nasazení, rozšiřitelnost systému), tak na uživatelskou zkušenost žáků zapojených do testování systému.
6. Diskutujte možná rozšíření systému a případné přínosy orchestrizace kontejnerů.

Literatura:

- Poulton, N. (2017). Docker Deep Dive. ISBN: 978-1521822807
- Poulton, N. (2022). The Kubernates Book. ISBN: 979-8402153776

Při obhajobě semestrální části projektu je požadováno:
První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Pavela Jiří, Ing.**
Konzultant: Ing. Libor Brázda
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 3.11.2022

Abstrakt

Bakalářká práce je zaměřena na návrh webhostingu, jehož cílem je podpořit výuku IT na středních školách. Výsledný webhosting je postaven na kontejnerové platformě Docker, která je ovládána vlastní implementací řídící platformy. Cílem vlastní implementace je dosažení maximální kontroly nad způsobem nasazení služeb, které mají žáci využívat při výuce. Vytvořený webhosting umožňuje izolovaný běh žákovských služeb, jehož výhodou jsou stejná běhová prostředí při zachování oddělenosti jednotlivých služeb. Součástí bakalářské práce je nasazení webhostingu a následná žákovská evaluace zakládající se na zkušenostech využívání služeb platformy.

Abstract

The bachelor's thesis is focused on the design of web hosting, the aim of which is to support IT teaching in secondary schools. The resulting web hosting is built on the Docker container platform, which is controlled by own implementation management platform. The actual implementation achieves maximum control over the deployment of the services that the students are supposed to use during teaching. The created web hosting enables isolated running services, the advantage of which are the same running environments while maintaining the separation of individual services. Part of the bachelor's thesis is the deployment of web hosting and the subsequent student evaluation based on the experience of using the services.

Klíčová slova

kontejner, Docker, webhosting, proměnná prostředí, svazek, síť, obraz, orchestrace, řídící platforma

Keywords

container, Docker, webhosting, environment variable, volume, network, image, orchestration, management platform

Citace

KAŠPÁREK, Ladislav. *Modulární webhosting pro výukové účely.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Pavela

Modulární webhosting pro výukové účely

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jiřího Pavely. Další informace poskytl technický konzultat firmy OptoNet Communication spol. s r.o. Ing. Libor Brázda. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

. . . . . .
Ladislav Kašpárek

Poděkování

Děkuji vedoucímu práce Ing. Jiřímu Pavelovi za cenné rady a odbornou pomoc. Dále děkuji firmě OptoNet Communication, spol. s r.o., za poskytnuté zázemí a možnost využití firemních serverů, kolegovi Ing. Liboru Brázdovi za poskytnuté konzultace a technické vedení. Děkuji též vyučujícím Ing. Michalu Bílkovi a Mgr. Petře Kašpárkové ze Střední školy průmyslové technické a automobilní v Jihlavě za možnost otestování webhostingu ve výuce. Za jazykovou a stylistickou korekturu děkuji Mgr. Marii Zemanové.

Obsah

Kapitola 1

Úvod

Na střední škole se žáci učí základy programování. V příslušných předmětech se učitelé zaměřují zejména na datové typy, řídící konstrukce a základy funkcí. Obsah a rozsah učiva určuje ŠVP školy. Pro ověření těchto znalostí jsou však často využívány předpřipravené konstrukce, které žák pouze doplní. Žáci se tedy při studiu nesetkají s problematikou návrhu nebo nasazení výsledné aplikace, protože namísto toho je jako běhové prostředí většinou používáno samotné vývojové prostředí a při výuce jazyka php pak aplikace jako např. Server2Go nebo Xampp.

Cílem bakalářské práce je zprovoznit webhosting, který bude schopný hostovat aplikace s co největší možnou podporou běhových prostředí. Žáci by poté nemuseli řešit problémy s veřejnou IP adresou nebo serverem, na kterém mají provozovat své aplikace. Škole tento projekt pomůže v usnadnění přístupu k reálnému prostředí používanému v praxi. Pro výuku bude přínosem, že databáze jde jednoduše restartovat, reinicializovat a její připojení k interpretru je snadnější. Žák by tedy mohl jednodušším způsobem testovat své aplikace, strávit méně času řešením problémů spojených s jejich nasazením, a navíc by se setkal s reálným prostředím serverových aplikací. Nasazení služeb pomocí kontejnerů bylo zvoleno pro možnost rychlého vypnutí/zapnutí služby a s cílem průzkumu platformy Docker.

Jako prostředek k vytvoření webhostingu podporujícího výuku poslouží kontejnerové prostředí postavené na platformě Docker. Pro platformu řízení kontejnerů a Dockeru bude vytvořena vlastní implementace v php. Jako základní přístupové protokoly budou využity HTTP a SSH.

Vytvořený webhosting otestují žáci 1. a 4. ročníku *Střední školy průmyslové, technické a automobilní v$_{\text{Jihlavě*}}$ v$_{}$hodinách ICT. Zhodnotí přínos a využitelnost při výuce IT na zmíněné škole.

Kapitola 2

Webhosting

Webhosting je služba, která poskytuje dokumenty na internet za účelem tvorby webové stránky nebo aplikace. [11]

Technologie

Typickými technologiemi jsou webový server, běhové prostředí aplikace a databázový server. Dotaz z internetu je zpracován webovým serverem. Webový server vytvoří proces pro zpracování pomocí běhového prostředí. Výstup procesu je následně poslán zpět, jako odpověď.

Běhové prostředí, typicky scriptovací interpret, zpracovává dotaz podle zdrojových kódů připravené aplikace nebo dodaných zákazníkem. Volba databáze a běhového prostředí závisí na poskytovateli webhostingu. Typickými zástupci běhových prostředí jsou php, Python, .NET. Jako databáze se nejčastěji používá Postgres, MariaDB, Microsoft SQL Server, MongoDB.

Služby

S webhostingem se pojí i doplňkové služby. Tyto služby mohou zahrnovat mail server, doménová jména, doménové aliasy, SSL certifikáty. Rozšiřující službou může být administrační prostředí pro nastavování webhostingu nebo umožňující správu předpřipravených webových aplikací.

Úložiště

Ke každému webhostingu je nabízen i diskový prostor. Velikost diskového prostoru, případně rychlost, se typicky odvíjí od ceny webhostingu. Na toto úložiště se ukládají uživatelská data. V případě hotové aplikace, např. WordPress nebo NextCloud, uživatel ani nepotřebuje přímý přístup k souborům. Zdrojové kódy se nahrávají pomocí aplikací jako je např. GoogleDisk nebo OneDrive. Některé webhostingy umožňují přístup pomocí dříve populárního FTP.

Kapitola 3

Architektura

Architektura serveru bude založená na protokolech HTTP a SSH. Kombinace těchto protokolů zajistí přístup k webovým službám, úložišti a v případě požadavků školy nebo žáka, lze doplňující protokoly tunelovat pomocí SSH.

HTTP

HTTP je bezestavový textový protokol používaný pro přenos dat. Data jsou dělena na hlavičku, která nese upřesňující vlastnosti, a uživatelská data. Uživatelskými daty pak může být prakticky cokoli, s odpovídající hlavičkou. Pro upřesnění povahy uživatelských dat se používá hlavička Content-type.

V 90. letech byl protokol rozšířen o tzv. cookies. Jako cookies se označuje malé množství dat uložené na žádost serveru u klienta. Klient je poté přibalí k příštímu dotazu.

Apache

Program apache je již dlouhou dobu jeden z nejpoužívanějších a nejuniverzálnějších web serverů. Jeho historie sahá do počátku 90. let, kdy vznikal jako sada patchů k webovému serveru *NCSA HTTPd*.

Program se skládá z jádra a modulů. Tyto moduly pak upravují způsob vyhodnocení dotazu. Nejčastěji používanými moduly mohou být *mod_rewrite* pro úpravu url odkazu, *mod_ssl* pro použití šifrování nebo *mod_php* díky kterému lze jednoduše spouštět php skripty.

Pro tvorbu úložiště lze využít protokolu WebDAV, který je rozšířeným protokolem HTTP za účelem vytvoření vzdáleného souborového systému.

Pro konfiguraci používá server Apache globální konfigurační soubor *apache2.conf*, nebo lze upravit chování pomocí souboru *.htaccess* v každé složce zvlášť. Tento způsob konfigurace je velmi univerzální a umožňuje široké použití serveru. Ovšem je to současně i bezpečnostní riziko a zvyšuje se i riziko chyby v nastavení serveru ze strany uživatele.

Htaccess je doporučeno používat jako testovací prostředek pro odladění chování serveru v požadovaném případě. Po odladění chování by se měla konfigurace z *.htaccess* přenést do *apache2.conf* a soubor *.htaccess* smazat. Smazáním *.htaccess* se předejde případným chybám ze strany uživatele a zvýší se tím i rychlost odezvy serveru. *Htaccess* je vyhodnocován pro každý dotaz zvlášť, proto i když je soubor prázdný, server je zdržen jeho zpracováním.

Nginx

Nginx bývá označován jako nejpoužívanější reverzní proxy server. Reverzní proxy server je poddruh webového serveru, který předává obdržené dotazy dalším serverům ke zpracování. Tohoto chování se používá jako vstupní brány. Přímo přístupný z internetu je tedy jen jeden server. Další servery dostávají své dotazy od něj.

Tato topologie ovšem představuje riziko úzkého hrdla, z důvodu možného přetížení proxy serveru.

SSH

SSH je program a současně protokol primárně určený pro přístup ke vzdálenému shellu. Vznikl jako náhrada nezabezpečeného protokolu telnet. Tento protokol se typicky používá pro přístup k příkazové řádce vzdáleného počítače, nicméně má mnoho dalších vlastností, jako je např. přesměrování portů.

SFTP

Název SFTP označuje protokol a současně program pro přenos dat. Program je použitelný i pro jiné protokoly, např. SILC, SSH-1, nejčastěji se však používá v kombinaci s SSH-2. Slouží jako náhrada SCP nebo FTP protokolu.

Git

Gitem se rozumí program pro správu verzí využívaný hlavně pro vývoj softwaru. Tento program funguje na principu ukládání průběžných verzí souboru do tzv. repozitáře. Pro uložení nějaké verze souboru se používá příkaz `git commit`. Git pak v repozitáři udržuje všechny uložené změny. Z důvodu ukládání změn do repozitáře se pak může uživatel jednoduše vrátit k libovolné uložené verzi souboru.

Pro distribuci zdrojových kódů se používá tzv. vzdálený repozitář. Tento repozitář je stejný, jako lokální, ale je uložen na serveru. Ke vzdálenému repozitáři pak typicky přistupujeme pomocí protokolu git, který využívá protokolu SSH.

Přesměrování portů

Toto přesměrování může být dvojího typu *remote - vzdálený* nebo *local - místní* přesměrování portů. Princip obou přístupů je ale velmi podobný, a to tvoření tunelu mezi klientem a serverem.

Místní přesměrování portů Uživatel může pomocí SSH klienta otevřít port pro naslouchání, kde druhý konec tunelu je nasměrován na port v síti serveru.

Vzdálené přesměrování portů Uživatel otevře port na straně serveru. Druhý konec tunelu poté přesměruje na svůj port nebo na port zařízení v síti.

Při dotazu na přesměrovaný port data projdou SSH tunelem na jeho druhý konec. Díky zašifrování dat v tunelu, o což se postará SSH, lze posílat internetem jinak nezabezpečená data.

Jump host

SSH server určený k navazování spojení mezi klientem na jedné straně a serverem na straně druhé, fungující v navázané komunikaci jako prostředník, označujeme termínem *jump host*

nebo *jump server*. K navázání spojení prostřednictvím *jump serveru* používáme SSH příkaz *proxy jump* nebo *proxy command*, kdy příkaz *proxy jump* se používá jako jednodušší náhrada k složitějšímu *proxy command*.

Díky použití *jump host* lze zpřístupnit SSH servery v uzavřené síti při použití jedné adresy a jednoho portu jako vstupního bodu. Lze docílit zpřístupnění jinak nedosažitelných služeb díky neexistenci propojení mezi sítí serveru a klientem.

Uživatel má možnost použít *SSH proxy jump* v rámci příkazu SSH
`ssh -J lkasparek@5thbeat.labs.optonet.tech lkasparek@lkasparek`,
nebo jej použije v souboru *.ssh/config*.

.ssh/config

Označením *.ssh/config* myslíme soubor standartně uložený v domovském adresáři uživatele sloužící ke konfiguraci SSH programu. V tomto souboru má uživatel možnost nastavit jména serverů s příslušnými parametry např. port, adresu, *proxy jump*. Použití tohoto souboru velmi zjednodušuje práci při navazování SSH spojení.

Kontejnery

Kontejnery představují zabalený software do standardizovaných jednotek pro vývoj, sdílení a nasazení. Kontejner je standardní jednotka softwaru, která balí kód a všechny jeho závislosti. Obraz kontejneru Docker je samostatný spustitelný balík softwaru, který obsahuje vše potřebné ke spuštění aplikace: kód, běhové prostředí, systémové nástroje, systémové knihovny a nastavení. Obrazy kontejnerů se při spuštění stávají kontejnery. V případě kontejnerů Docker se tak stává, když jsou spuštěny na Docker Engine. Kontejnerový software, který je k dispozici pro aplikace založené na Linuxu i Windows, poběží vždy stejně, bez ohledu na infrastrukturu. Kontejnery izolují software od jeho prostředí a zajišťují, že bude fungovat jednotně navzdory rozdílům, např. mezi vývojem a přípravou. Kontejnery jsou abstrakcí na vrstvě aplikace, která balí kód a závislosti dohromady. Na stejném počítači může běžet více kontejnerů a sdílet jádro operačního systému s jinými kontejnery, přičemž každý běží jako izolovaný proces v uživatelském prostoru. Kontejnery zabírají méně místa než virtuální počítače (obrazy kontejnerů mají obvykle velikost desítek MB), zvládnou více aplikací s menšími požadavky na hardware. [13]

Kontejnerizační platforma poskytuje možnost zabalit a spustit aplikaci ve volně izolovaném prostředí zvaném kontejner. Izolace a zabezpečení umožňuje provozovat mnoho kontejnerů současně na daném hostiteli. Kontejnery jsou lehké a obsahují vše potřebné ke spuštění aplikace, takže se administrátor nemusí spoléhat na to, co je aktuálně nainstalováno na hostiteli. Při práci lze snadno sdílet kontejnery a mít jistotu, že každý, s kým jsou sdíleny, dostane stejný kontejner, který funguje stejným způsobem.

Kontejner vs virtuální stroj

Kontejnery a virtuální stroje mají podobné výhody izolace a alokace zdrojů, ale fungují odlišně, protože kontejnery virtualizují operační systém namísto hardwaru. Kontejnery jsou přenosnější a efektivnější. [13]

Tabulka 3.1: Srovnání kontejneru a virtuálního stroje

Hodnocený parametr	Kontejnery	VMs
Předmět virtualizace	Operační systém	Počítačový hardware
Rychlost startu	Jako běžný proces	Jako počítač
HW požadavky	Jen požadavky procesu	Vysoké požadavky
Knihovny	Sdílí s jádrem hosta	Virtualizuje vlastní
Uchování vnitřního stavu při pádu	Přichází o stav	Zachová vnitřní stav

Kontejnery nevznikly jako náhrada virtuálních strujů. Jejich účelem je zjednodušení správy a izolace služeb.

Docker

Docker je open source platforma pro vývoj, distribuci a spouštění aplikací. Docker umožňuje oddělit aplikace od infrastruktury, s cílem jednodušší administrace aplikací. Využití metodologie Docker pro rychlou distribuci, testování a nasazení kódu může výrazně zkrátit prodlevu mezi napsáním kódu a jeho spuštěním v produkci.

Architektura dockeru

Docker používá architekturu klient-server. Docker klient komunikuje s Docker démonem, který se stará o přípravu, provoz a distribuci Docker kontejnerů. Docker klient a Docker démon mohou běžet na stejném systému, nebo se může Docker klient připojit ke vzdálenému Docker démonovi. Docker klient a Docker démon spolu komunikují pomocí REST API, přes UNIXové sokety nebo síťové rozhraní. Dalším Docker klientem je Docker Compose, který umožňuje pracovat s aplikacemi sestávajícími ze sady kontejnerů.

Docker daemon

Docker daemon (dockerd) naslouchá požadavkům rozhraní Docker API a spravuje objekty Docker, jako jsou obrazy, kontejnery, sítě a svazky. Démon může také komunikovat s jinými démony.

Docker klient

Docker klient (docker) je primární způsob interakce mnoha uživatelů Dockeru s Dockerem. Při použití příkazu, jako je `docker run`, klient odešle tyto příkazy dockerd, který je provede. Příkaz docker používá Docker API. Docker klient může komunikovat s více než jedním démonem.

Docker Desktop

Docker Desktop je snadno instalovatelná aplikace pro prostředí Mac, Windows nebo Linux, která umožňuje vytvářet a sdílet kontejnerizované aplikace a služby. Docker Desktop obsahuje démona Docker (dockerd), klienta Docker (docker), Docker Compose, Kubernetes a další nástroje.

Docker registry

Docker registr ukládá Docker obrazy. Docker Hub je veřejný registr, který může používat kdokoli. Docker je ve výchozím nastavení nakonfigurován tak, aby vyhledával obrazy na

Docker Hub. Uživatel si může dokonce založit i vlastní soukromý registr, např. v rámci aplikace GitLab.

Docker alternativy

Možnými Docker alternativami jsou LXC kontejnery a kontejnery Podman. Přímým konkurentem je kontejnerizační platforma Podman. Z důvodu, dnes již, zastaralejší architektury a způsobu konfigurace nelze považovat LXC za přímého konkurenta.

Podman

Podman je open source daemonless nástroj pro spouštění, sestavování, sdílení a nasazení aplikací pomocí kontejnerů a kontejner obrazů. Podman poskytuje rozhraní příkazového řádku (CLI), které zná každý, kdo používá Docker Container Engine. Většina uživatelů může bez problémů použít alias Docker na Podman `alias docker=podman`. Podobně jako u jiných běžných kontejnerových enginů (Docker, CRI-O, kontejnery) se Podman spoléhá na stejné *Container Runtime*. Díky tomu jsou běžící kontejnery vytvořené Podmanem téměř k nerozeznání od těch vytvořených jakýmkoli jiným běžným kontejnerovým enginem.

Kontejnery pod kontrolou Podman mohou být spuštěny super uživatelem nebo neprivilegovaným uživatelem. Podman spravuje celý ekosystém kontejnerů, kontejnery, obrazy kontejnerů a svazky kontejnerů pomocí knihovny libpod. Podman se specializuje na všechny příkazy a funkce, které pomáhají udržovat a upravovat obrazy kontejnerů, jako je stahování a značkování.

Docker vs Podman

Tabulka 3.2: Srovnání Docker a Podman

Hodnocený parametr	Docker	Podman
Architektura	Daemon	Daemon-less
Práva superuživatele	Nutné	Možné
Sestavení obrazů	Vlastní nástroj	Buildah - externí nástroj
Docker compose	Kompatibilní	Kompatibilní
Docker swarm	Kompatibilní	Nekompatibilní
Přístup	Monolitický	Modulární

Na základě zkušeností firmy *OptoNet Communication, spol. $s_{r.o.*}$ $_{s}$platformou Docker mi technický konzultant doporučil použití této platformy. Dalšími důvody je velikost komunity a samotná architektura Docker.

Orchestrace

Orchestrace je koordinace procesů a služeb za účelem splnění úkolu. Samotné orchestrované procesy se mohou skládat z více podprocesů a některé mohou být automatizované. V IT orchestrace pomáhá zejména při řešení složitých problémů a pracovních postupů.

V oblasti kontejnerizace se orchestrace využívá hlavně k řízení přesunu a škálování kontejnerových služeb. K tomuto účelu využíváme tzv. *orchestrátory*. Jejich úkolem je řídit automatickou údržbu nebo výměnu kontejnerů a řídit nasazení aktualizovaných služeb.

Orchestrace a automatizace

Jako automatizaci označujeme přípravu procesu, který od svého spuštění až po dokončení nepotřebuje zásah od uživatele. Příkladem automatizačního nástroje může být např. make, který je nejčastěji využívaný pro překlad zdrojových kódů a spuštění výsledného programu. Jako automatizační nástroj pro správu více operačních systému a konfigurací lze uvést např. *Ansible*.

Orchestrace je nástroj především pro koordinaci procesů, zatímco cílem automatizace je pro vykonání úkolu nevyžadovat reakci uživatele.

Orchestrační nástroje

Nejčastěji používané nástroje pro ochestraci kontejnerů jsou *Kubernetes* a *Docker Swarm*. Obě platformy mají své výhody a nevýhody. Základní vlastnosti *Docker Swarm* je odlehčenost. Jeho správa je, oproti *Kubernetes*, mnohem jednodušší, ale umožňuje spravovat pouze Docker kontejnery.

Kubernetes oproti tomu je složitější, ale je možné jej využít pro správu i Podman nebo LXC kontejnerů. Narozdíl od *Docker Swarm* umožňuje *Kubernetes self-healing*, což je proces, kdy *Kubernetes* restartuje nebo obnoví poškozený kontejner. Detekce poškozeného kontejneru může být provedena např. pomocí *healthcheck* instrukce v Dockerfile.

Monitoring

Spojením *IT monitoring* označujeme procesy a nástroje pro sledování stavu zařízení či služby v IT. Nástroje pro monitorování IT mohou zahrnovat vše od základních nástrojů až po pokročilejší řešení založených na technikách umělé inteligence. Cílem monitoringu je detekce výpadku, upozornění na výpadek nebo předpověď výpadku zařízení či služby.

Informace od monitorovacího systému lze rozdělit do dvou kategorií.

Poplachy Poplachové hlášky generované systémem, např. přílišné zaplnění disku, selhání zálohování. Informace je většinou předána zodpovědné osobě pomocí emailu, SMS nebo dalších komunikačních kanálů.

Dlouhodobé záznamy Dlouhodobě uchované záznamy nasbírané systémem, např. vytížení CPU, využití paměti. Informace jsou vyhodnocovány zpětně a v širším kontextu. Cílem je získat souvislejší představu o stavu systému.

Před zahájením analýzy dat za účelem monitoringu je potřeba daná data získat. K tomuto účelu byla vybrána zejména aplikace Prometheus, jejíž architektura je založená na exporterech, a tak umožňuje sběr různorodých dat za účelem monitoringu. Data lze získat přímo jejich uložením do nějaké databáze. Z tohoto důvodu byla vyvinuta aplikace InfluxDB, která umožňuje ukládání *time series* dat.

Nejdůležitější částí monitoringu je interpretace nasbíraných dat. Firma *OptoNet Communication, spol. s r.o.* pro zobrazení a interpretaci monitoringu používá aplikaci Grafana, proto byla použita i pro interpretaci dat spojených s monitoringem aplikací nasazených v rámci této práce.

Grafana

Grafana Cloud je vysoce dostupná, výkonná a škálovatelná platforma pro sledování aplikací a infrastruktury. Poskytuje centralizovaný pohled na monitorovací data pocházející z mnoha zdrojů, jako jsou Prometheus, InfluxDB, Elasticsearch a Amazon CloudWatch.

Umožňuje i tvorbu poplachů pomocí analýzy dat a nastavením prahových hodnot. Pro správu poplachů lze použít *Grafana alert manager* nebo i externí správce poplachů, např. *Prometheus alert manager*.

InfluxDB

InfluxDB Cloud je platforma vytvořená pro účely sběru, ukládání, zpracování a vizualizace *time series dat*. *Time series data* jsou sbírkou pozorování (chování) pro jeden subjekt (entitu) v různých časových intervalech (obecně uniformně rozložených jako v případě metrik, nebo neuniformně rozložených jako v případě událostí). Datové body se obvykle skládají z po sobě jdoucích měření provedených ze stejného zdroje a používají se ke sledování změn v průběhu času.

Prometheus

Prometheus je sada nástrojů pro monitorování zařízení a služeb v oblasti IT, původně vytvořená ve společnosti SoundCloud. Od svého založení v roce 2012 přijalo Promethea mnoho společností a organizací a projekt má velmi aktivní komunitu vývojářů a uživatelů. Nyní je to samostatný open source projekt, který je spravován nezávisle na jakékoli společnosti.

Prometheus shromažďuje a ukládá své metriky jako *time series data*. Monitorovací hodnoty získává pomocí HTTP dotazu na tzv. *exportéry*. Exporter bývá samostatný proces, většinou samostatný kontejner, který sleduje požadované skutečnosti. Rozhraní mezi exporterem a Prometheem je zdokumentované a dostupné např. v jazyku Python. Díky této otevřenosti je možné si dopsat vlastní exportér sledující požadované metriky.

Node exporter

Node exporter je základní Prometheus exporter používaný pro sledování pro hardware i software dostupný na *NIX jádrech. Jádro je napsané pomocí jazyka GO s možností přidat další rozšíření podle potřeby. Nejpoužívanější metriky jsou pro sledování vytížení paměti *node_memory_MemTotal_bytes*, *node_memory_MemFree_bytes*, *node_memory_MemAvailable_bytes* nebo pro sledování vytížení procesoru *node_cpu_seconds_total*, případně sledování zaplnění úložiště *node_filesystem_avail_bytes*, *node_filesystem_size_bytes*.

cAdvisor

Prometheus exporter cAdvisor se používá pro sledování využívání zdrojů v kontejnerovém prostředí. Nativně podporuje Docker, ale vývojáři se snaží i o rozšíření podpory na další kontejnerizační platformy. Umožňuje sledovat zatěžování procesoru jednotlivými kontejnery pomocí metriky *container_cpu_usage_seconds_total* nebo využití paměti *container_memory_rss*, nebo lze pomocí metriky *container_network_receive_bytes_total* a *container_network_transmit_bytes_total* měřit používání sítě.

Kapitola 4

Implementace

Tato kapitola pojednává o implementaci platforem řízení, jejich srovnání a vzniklé architektuře webhostingu.

Portainer

Aplikace Portainer představuje univerzální platformu pro management kontejnerů nezávislý na technologii nebo zprostředkovateli.

Vlastní implementace

Řízení kontejnerů, tedy Docker prostředí, je implementováno pomocí architektury klient-server. Komunikace mezi klientem a serverem probíhá pomocí html/json odpovědí serveru a form-url-encoded/form-multipart dotazů. Celé rozhraní potom umožňuje použití REST.

Klientská část

Klientskou část představuje soubor skriptů komunikující se serverem pomocí libcurl z prostředí php. Skripty vznikly za účelem připravení služeb pro žáky.

Druhou část představuje html prostředí dostupné samotným serverem. Cílem html prostředí je možnost zobrazení aktuálního nastavení a možnost doupřesnění jednotlivých atributů. Rozhraní má sloužit hlavně jako ulehčení vývoje serveru a jako jednoduchý přístup k objektům v případě nutnosti rychlé korekce kontejnerů.

Serverová část

Nasazení kontejnerů není jediná část, kterou je zapotřebí v práci vyřešit. Proces přípravy služeb a kontejnerů začíná přípravou obrazu pro budoucí službu a končí jejím spouštěním. Ovšem tento proces přípravy zahrnuje i možnost přesunu na jiný server nebo aktualizace jednotlivých služeb.

Z tohoto důvodu bylo řízení služeb rozděleno do tří fází.

1. Sestavení obrazu - v této fázi se z připraveného Dockerfile sestaví a pojmenuje obraz
2. Globální příprava - fáze init je pro přípravu obecně dostupného prostředí, např. sdílené sítě, svazky, atd.
3. Spouštění - platforma pro správu spouští jednotlivé služby

Implementace obsluhy těchto fází byla provedena ve dvou variantách. Prototypovací verze *ServAdmin* byla použita pro otestování navržené architektury a ujasnění požadavků na řízení i nasazení webhostingu. Nasazená verze, pojmenovaná *Platform*, zahrnuje znalosti nabyté používáním *ServAdmin* a bylo přidáno nastavení pomocí proměnných prostředí pro větší znovupoužitelnost.

ServAdmin

Při postupném seznamování s Docker kontejnery bylo potřeba automatizovat některé procesy. Jedním z těchto procesů byla např. tvorba docker-compose souboru pro spuštění služeb. V docker-compose souboru musely být popsány jaké služby, který uživatel je spouští a jaké zdroje serveru mají využívat. Dalším procesem, který bylo třeba zautomatizovat, byl proces sestavení obrazů služeb. Obrazy je těžké udržovat v konzistentním stavu, pokud je administrátor spravuje ručně.

Pro vyřešení prvních problémů byla připravována řídící platforma *ServAdmin*. Cílem platformy byla správa uživatelů a jejich služeb. Jako jazyk pro implementaci bylo zvoleno php, z důvodu předešlých zkušeností s jazykem a jednoduchého vytvoření uživatelského rozhraní pomocí html formulářů.

Architektura

Zvolená architektura komunikace kontejnerů pomocí sítě a přístupu k datům pomocí svazků byla založená na společném sdílení. Byly připraveny dvě sítě, jedna pro používání protokolu HTTP a druhá využívaná protokolem SSH. Všechny aplikace využívající jeden nebo druhý protokol byly připojeny do příslušné sítě.

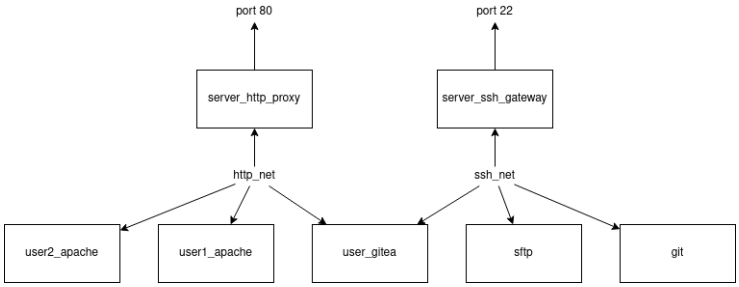
Páteřní sítě a svazky umožňovaly kontejnerům přístup ke sdíleným datům a komunikaci mezi službami:

ssh_net síť dostupná použitím SSH protokolu a umožňující přístup k SSH službám, do této sítě se uživateli připojuje pouze aplikace Gitea

http_net služby v této síti využívají protokol HTTP, standartně jsou do této sítě připojovány aplikace WordPress, apache servery, atd.

home svazek, ve kterém je dostupná struktura home adresářů, standartně je připojován do */home*

etc tento svazek sdružuje soubory *passwd*, *shadow*, *groups* a je připojován do */mnt/etc* v používaných kontejnerech je místo souboru */etc/passwd* použit symbolický odkaz směřující na */mnt/etc/passwd*, stejné odkazy jsou použity pro *groups* a *shadow*



Obrázek 4.1: Diagram *ServAdmin* sítí

Diagram ukazuje zapojení kontejnerů do sdílených sítí.

Jako základní služby dostupné všem uživatelům bylo připraveno:

SFTP kontejner umožňující připojení úložiště pomocí SFTP protokolu

git git server společný pro všechny uživatele

servreg služba umožňovala navázat vzdálené přesměrování portů

SSH_gateway kontejner přístupný ze sítě hostitele, pro navázání spojení od klientů

HTTP_gateway proxy server přesměrovávající HTTP dotazy na služby, používaný mimo jiné pro správu SSL certifikátů

Sdílení zdrojů serveru je, v tomto případě, jednodušší na implementaci, ale představuje bezpečnostní hrozbu. Dále by bylo velmi těžké pro každou službu přesně uvádět absolutní cestu přes strukturu domovských adresářů pro načtení konfigurace nebo zdrojových kódů. Proto za tímto účelem vznikl svazek *app*, který není sdílený a tvoří se pro každou službu zvlášť. Svazek *app* je připojován do */app*.

Řízení

Jako způsob konfigurace bylo stanoveno deklarativní rozhraní, umožňující komunikaci pomocí yaml formátu. Výběr rozhraní byl ovlivněn programem docker compose, který jej používá.

Jádro platformy tvoří třídy *Analyzer*, *Interpret*, *Users/BUser* a *Services/Service*.

Uživatelé

Uživatelé jsou v platformě reprezentováni dvěma třídami, a to *Manager* a *User*. Implementace pomocí dvou tříd byla zvolena s cílem umožnit učiteli vytvoření uživatelských účtů potřebných k jeho výuce, u kterých by měl přímý přístup k jejich souborům, ale oproti tomu nedovolit žákům tvorbu nového uživatele.

Struktury uživatelů jsou využívány zejména pro vytvoření souboru *etc/passwd* a *etc/shadow*. Příkladem takových atributů může být *UID*, *password*, *home*, *name*, kde poslední dva atributy jsou zděděny od třídy *base*. Dalším klíčovým atributem uživatele je seznam jeho služeb, které má k dispozici. Uživatel typu *Manager* má navíc oproti *User* seznam svých uživatelů.

Příklad vytvoření nového uživatele a nové služby typu apache:

```
users:
  testuser:
    manager: "yes"
    home: /home/testuser
    services:
      test:
        home: /home/testuser/Apps/test
        type: apache
```

Odpověď od serveru po vytvoření uživatele a služby:

```
  testuser:
    name: testuser
    home: /home/testuser
    domain: ~
    uid: 1033
    services:
      test:
        name: testuser_test
```

```
    home: /home/testuser/Apps/test
    domain: ~
    type: apache
    state: stop
    port: ~
    vhosts: []
password: '!'
quota: ~
max_serv: ~
app_folder: ~
manager: "yes"
defaults: []
users: []
```

Služby

Platforma *ServerAdmin* reprezentuje služby pomocí implementovaných tříd, které dědí třídu Services/Service. Třídu následně doplňuje šablona docker-compose souboru ve formátu yaml. Cílem této implementace bylo umožnění jednoduššího nastavení dílčích parametrů pro jednotlivé aplikace. Administrátor může naimplementovat třídu a šablonu podle vlastních preferencí.

Pro vytvoření docker-compose souboru třída připraví proměnné, které jsou dosazeny do připravené šablony. Tento způsob tvorby byl zvolen na základě rychlosti implementace a potřeb měnit v souboru jen části.

Na služby se mohou vztahovat kvóty, které byly navrženy s cílem nedovolit uživatelům tvořit příliš mnoho nevyužívaných služeb a nastavit omezení diskového prostoru. Od implementace se nakonec upustilo kvůli složitosti implementace.

Interpret a Analyzátor

Tyto dvě třídy představují logiku vyhodnocující konfiguraci dostupnou z databáze a příchozího dotazu.

Cílem Analyzátoru je zkontrolovat a ověřit nebo nastavit požadované parametry vstupní konfigurace. Základní kontroly ověřují vztahy mezi uživateli a typy uživatelů v těchto vztazích. V druhém kromu ověří existenci uživatelova UID, které případně vytvoří. Třetím krokem se na základě možných kvót zkontrolují dostupnosti uživatelových zdrojů, zvláště počet spustitelných služeb. V posledním kroku se připravují adresáře vyžadované službou pro vytvoření jejích svazků. Tyto adresáře se vytvářejí v adresáři Apps uloženým ve vlastníkově domovském adresáři.

V průběhu analýzy objekty, v případě změny, informují Intepret o tom, jaká změna nastala. Po úspěšné kontrole hodnot analyzátorem dojde ke spuštění Interpretru, který na základě uložených zpráv od objektů tyto změny vykoná. Nejčastějšími úkony interpretru je zapínání / vypínání služeb nebo tvorba složek, uživatelských nebo služeb.

Důvodem implementace Interpretru a Analyzátoru bylo oddělení kontroly konzistence konfigurace a následné aplikace konfigurace.

Externí části

Proces sestavení obrazů a počátečního vytvoření sdílených sítí a svazků je implementován pomocí php pouze částečně. Samotné spuštění procesu probíhá pomocí bash skriptu.

Sestavení obrazů

Sestavení obrazů probíhá za použití docker compose. Za účelem vytvoření těchto docker compose souborů byl implementován php script, který načte a zpracuje adresářovou strukturu. Cílem prohledávání adresářové struktury je nalezení Dockerfile souborů a složek *Children*, které obsahují závislé obrazy. Po sestavení závislostí v php reprezentovaných adresářovou strukturou dojde k vygenerování nejmenšího možného počtu docker-compose souborů umožňujících postupné nasazení pomocí příkazu docker compose.

Tvorba sdílených prostředků

Tvorba prostředků proběhne spuštěním příkazu docker compose. Vytvoření potřebného docker compose probíhá pomocí php skriptu, který uchovává všechny potřebné informace o kontejnerech a prostředcích. Implementace tohoto scriptu je založená na slučování asociativních polí. Pomocí této metody byla následně implementována platforma *Platform*.

Platform

Na základě analýzy nedostatků řídící platformy *ServAdmin* byla jako náhrada vyvinuta platforma *Platform*. Cílem této platformy bylo vyřešit problémy související s používáním *ServAdmin*, s přidáváním nových druhů služeb nebo sestavováním obrazů.

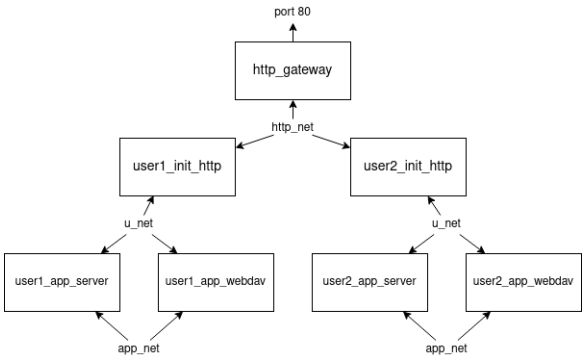
Architektura

Architektura komunikace kontejnerů a přístupu k datům pomocí svazků byla založena za účelem oddělování služeb i uživatelů. Z implementačních důvodů musely být vytvořeny i sdílené sítě. Tyto sítě jsou:

ssh_net síť dostupná pomocí SSH protokolu, připojují se kontejnery typu *SSH* pro vytvoření *jump host*

http_net síť je dostupná prostřednictvím HTTP proxy serveru, připojují se uživatelské HTTP proxy

Za účelem maximálního oddělení uživatelů je pro každého uživatele připravena síť, tato síť je označována obecně *u_net*. Do této sítě jsou uživateli krom běžných služeb, např. WordPress, Gitea, připojovány kontejnery zprostředkovávající komunikaci s hlavními proxy servery.



Obrázek 4.2: Síťový diagram Platform

Diagram znázorňuje zapojení HTTP serverů a použití *u_net*.

Připojení aplikace do *u_net* je specifikováno druhem služby, ale lze připojení doplnit.

Sdílený svazek dostupný všem službám je *etc*, který uchovává soubory *passwd* a *shadow*. Druhý svazek, od jehož využití se upustilo, je *homes*. Tento svazek představuje celou uživatelskou adresářovou strukturu.

Řízení

Za účelem implementace byl vytvořen framework v jazyce php umožňující rychlou implementaci datových modelů pomocí MVC architektury. Cílem frameworku bylo obecné zjednodušení generování konfiguračních souborů podle implementovaných pravidel.

Framework je založen na REST API využívající html nebo json formáty. Server očekává data ve standartním formátu *form-**.

Uživatelé

Informace o uživatelích platforma uchovává pomocí instancí třídy *App/Client*. Na základě zkušenosti s platformou *ServAdmin* byl implementován pouze jeden typ uživatele. Tato implementace umožnila širší možnosti tvorby a správy uživatelů. Znemožnění vytvořit nového potomka lze dodatečně doplnit po dokončení implementace politik.

Atributy instance *App/Client* jsou např.

name jméno uživatele pro vytvoření *login*

password hash hesla vytvořený funkcí *crpypt*

parent jméno rodičovského uživatele

Pokud je uživatel tzv. *root_user*, hodnotou je prázdný řetězec.

domain základ doménového jména pro uživatelovy aplikace

Standartně se vytvoří jako *name.<doména předka>*.

home absolutní cesta v kontextu kontejneru k domovskému adresáři

apps seznam uživatelových služeb

children seznam uživatelových potomků

Výchozím uživatelem je uživatel *Server*, který je rodič všech klientských účtů. V implementaci se tento uživatel označuje jako *root_user*. Jméno výchozího uživatele lze dopředu nastavit pomocí proměnné prostředí *ROOT_USER*.

Služby a aplikace

Aplikace je v platformě reprezentována jako instance třídy *App/Compose*. Základním atributem aplikace je seznam instancí třídy *App/Service* představující jednotlivé služby. Tyto služby vycházejí z druhu služby reprezentovaného instancí třídy *App/Image*.

Pro spuštění aplikace jsou generovány soubory *docker-compose* a *.env*. Soubory jsou následně uloženy do složky pojmenované jménem aplikace. Tato implementace umožňuje jednduché použití `docker compose` a využití proměnných prostředí v *docker-compose*. Vygenerování souboru *docker-compose* je implementováno pomocí slučování asociativních polí. Tento proces má několik fází:

1. aplikace připraví uživatelovy svazky, sítě a verzi
2. každá ze služeb aplikace vytvoří vlastní pole *docker-compose*
3. aplikace rekurzivně sloučí pole jednotlivých služeb
4. zkontroluje a opraví chyby vzniklé slučováním
5. vrátí pole reprezentující *docker-compose* celé aplikace

Kód ukazuje implementaci těchto fází

```
public function to_compose() {
```

```
        $client = $this->get_client()->name();
        $ret = [
            'version' => '3.0',
            'volumes' => $this->compose_volumes($client),
            'networks' => $this->compose_nets($client)
        ];
        foreach ($this->get_services() as $s) {
            $ret = array_merge_recursive($ret, $s->to_compose());
        }

        return $this->compose_correct($ret);
    }
```

Implementace využívá šablony služby, nikoli celé aplikace, jak to bylo v případě *ServAdmin*. Tento druh implementace byl zvolen z důvodu větší možnosti správy služeb, které aplikace využívá. Implementace současně umožňuje upravit službu podle konkrétní aplikace, případně uživatele, při zachování jednotné formy.

Tvorba docker-compose tímto způsobem navíc umožňuje sestaveným aplikacím předání hodnot proměnných prostředí potřebných pro běh. Práce s těmito proměnnými je implementována na úrovni služby a je rozdělena na dvě části:

env_export proměnné jsou sdíleny v celé aplikaci

environment proměnné jsou použity jen pro danou službu

Nejčastější využití takovéhoto předání hodnot se předpokládá při použití aplikace s databází. Aplikace nastaví sdílení proměnných s hodnotami uživatele, hesla a jménem databáze, touto aplikací může být např. WordPress.

Zpracování Dockerfile

Uložení Dockerfile probíhá při vytvoření instance *App/Image*, a to do vnitřní struktury platformy. Platforma při sestavování obrazů vychází z uložených Dockerfile souborů, a to i v případě obrazů z registrů. Při ukládání instance *App/Image* je současně uloženo jméno obrazu. Díky získanému zobecnění pomocí *App/Image* nemusí administrátor znát přesnou hodnotu instrukce `FROM`, platforma si ji vytvoří sama.

Kontrolované uložení Dockerfile umožňuje jednodušší implementaci sestavovacího procesu. Tento proces je implementován ve třídě *App/Image* pomocí příkazu, který sestaví příkaz příkazové řádce.

```
$cmd = 'set -o pipefail ; cat ' . $this->get_dfile_path()
. ' | docker build -t ' . $this->get_image() . ' - 2>&1 | tee -a '
. \App\env()->build_log();
```

Řízené sestavení pomocí závislostí mezi obrazy je implementováno ve třídě *App/Http/-Controller/Builder* metodami:

building_process($data) metoda spustí sestavení obrazu pro každou položku v seznamu *$data*

build_tree($image, $build_tree) metoda do seznamu *$build_tree* přidá rodiče požadovaného obrazu, obraz a následně potomky obrazu

Díky možnosti sestavit obraz pomocí platformy lze všechny takto sestavené obrazy označkovat jednotným prefixem.

Jako sdílené prostředky platforma využívá především svazek *etc*, sítě *ssh_net*, *http_net* a samotný kontejner s řídící platformou. Spuštěním platformy pomocí předpřipraveného

docker-compose dojde k vytvoření těchto sdílených prostředků. Ostatní služby vždy vlastní určitý uživatel. Cílem implementace bylo jednodušší ukládání souborů potřebných pro běh služby a určení vlastnictví a správce těchto souborů.

Politiky a validátory

V rámci frameworku jsou implementovány podpůrné třídy umožňující ověření vstupních dat, tzv. validátory. Tyto validátory velmi zjednodušují implementaci logiky modelů, protože zaručují korektnost vstupních dat.

Politiky umožňují ověřit, jestli uživatel smí manipulovat s daným objektem nebo konkrétním atributem. Tyto politiky přístupu ale nebyly implementovány. Jejich doplnění by umožnilo platformu nasadit i pro použití dalšími uživateli.

ServAdmin vs Platform

Následující tabulka srovnává implementaci obou platforem v klíčových směrech.

Tabulka 4.1: Srovnání vyvinutých platforem řízení

Hodnocený parametr	ServAdmin	Platform
rozhraní	deklarativní yaml	JSON/html REST
sestavení obrazů	externí bash script	interní součást
příprava zdrojů platformy	externí bash script	interní součást
spouštění služeb	interní součást	interní součást
šablona služeb	textový yaml	asociativní pole
uživatelský prostor	sdílená síť i data	oddělená síť u_net a volume u_home
proměnné prostředí služeb	nepřipravuje	nastavuje
chování podle prostředí	vždy stejné	proměnné prostředí, konfigurační soubor

Ze srovnání vyplývá, že platforma *Platform* byla vyvíjena jako náhrada za *ServAdmin*.

Systemd

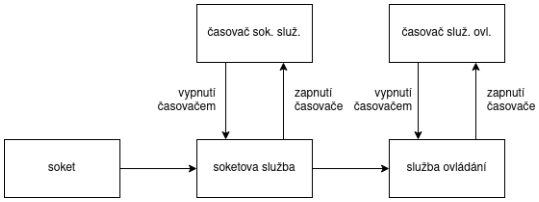
Tato kapitola popisuje základy implementace řízení kontejnerů na dotaz pomocí služeb systemd. Klíčovou část implementace tvoří použití *systemd-socket-proxyd*, který umožňuje vytvořit obecný soketový proxy server aktivovaný na základě soketu.

Návrh tvoří služby a dva časovače.

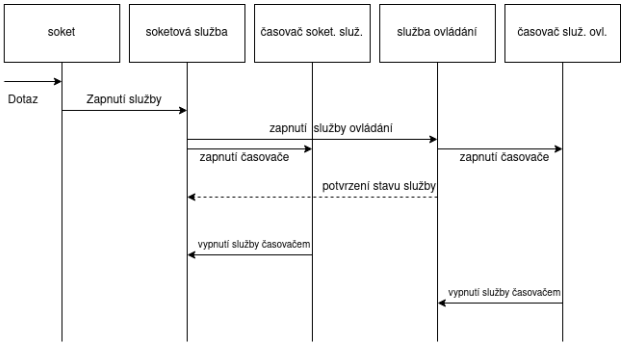
1. soket - vstupní bod
2. soketová služba - aktivuje službu ovládání a časovač konfliktní soketové služby
3. časovač a konfliktní služba soketové - zastaví soketovou službu
4. služba ovládání - zapíná / vypíná kontejnery při zapnutí / vypnutí služby
5. časovač a konfliktní služba ovládání dockeru - zastaví službu ovládání

Diagram ukazuje závislosti služeb. Pro zjednodušení diagramu byly vynechány konfliktní služby.

Aktivace služeb v čase:



Obrázek 4.3: Diagram služeb systemd-socket-proxyd



Obrázek 4.4: Časový diagram systemd-socket-proxyd

Diagram ukazuje časovou návaznost zapínání služeb. Pro zjednodušení diagramu byly vynechány konfliktní služby.

Součástí implementace byla i příprava Docker kontejneru. Po několika testech se ukázalo, že systemd lze použít v kontejneru využívající distribuci Ubuntu. Tomuto kontejneru byl následně nastaven příkaz spuštění `/lib/systemd/systemd`. Po nastartování kontejneru a spuštění systemd se spustila předpřipravená služba *doc_init.service*, která měla vytvořit konfigurační soubory služeb a skripty ovládání Docker pro konkrétní aplikaci. Před dokončením zapínání služby byl spuštěn vygenerovaný skript, který zapnul naslouchání na soketech představující porty jednotlivých služeb. Posledním krokem bylo potřeba restartovat *docker.service* pro správné načtení *docker.socket* sdíleného s hostem.

Kapitola 5

Nasazení a vyhodnocení

1. Nejprve byla připravena platforma pro řízení kontejnerů a jejich obrazů.
   Tato nově vznikající platforma dostala později název *ServAdmin*.
   Prvním projektem nasazeným pomocí platformy *ServAdmin* byla aplikace Gitea.
2. Byla přidána ještě možnost soukromých repozitářů zcela v režii uživatele.
   Zároveň byla zvažována a prozkoumávána možnost vytvoření / zapnutí služby na dotaz.
3. Byl nasazen exeprimentálně systém pro ITUIIS projekt.
4. Bylo domluveno s vyučující ze školy *Střední školy průmyslové, technické a automobilní Jihlava* vytvoření webhostingu pro ni a 16 žáků.
   Pomocí *ServAdmin* tedy byla nasazena první verze webhostingu pro výuku IT.
5. Po nasazení kontejnerů pro výuku bylo zjištěno příliš mnoho nedostatků, např. nepřehledná konfigurace, nutnost spustit nepotřebné služby, v platformě *ServAdmin*.
   Proto byl zahájen vývoj platformy *Platform*. Tato platforma byla později produkčně nasazena.
6. S nasazením *Platform* byla spojená migrace *ServAdmin* služeb do nového prostředí.
7. Pomocí *Platform* byly nasazeny služby pro *SŠPTA Ji* za účelem uživatelského testování.
8. Mezi prvním a druhým testováním byly provedeny rozsáhlé změny v řízení platformy a mimo jiné byla vypnuta aplikace cAdvisor.

Používání *ServAdmin*, *Platform*, migrace, systemd

Tato část se zabývá problémy vzniklými při používání jednotlivých platforem řízení. Součástí je popis administrace uživatelů, služeb a vytvoření nového druhu služby. Závěr této kapitoly se zabývá nasazením a používáním systemd pro řízení kontejnerů na dotaz.

Používání *ServAdmin*

Konfigurace pomocí yaml formátu, zapisovaného do textového pole, se pro administraci platformy neosvědčila. Velký problém byl s přehledností celého zobrazení nastavení a pro žáky by byl tento způsob konfigurace nepoužitený. *ServAdmin* pomocí yaml vyjadřoval celkovou konfiguraci serveru, to mělo za následek zobrazení všech informací najednou.

Správa uživatelů

Uživatelé se dělí na dva druhy, správce a potomka. Správce může spravovat uživatele, ovšem pouze typu potomek. Nemůže se tedy stát, aby rodičem správce byl správce.

Od tohoto principu se odvíjí i přidávání nových uživatelů. Pokud chce administrátor přidat uživatele typu správce, stačí v yaml formátu do pole *users* přidat klíč, uživatelovo jméno s přiřazenou hodnotou prázdného pole. Přidání uživatele typu potomek lze pomocí atributu *parent*, kde administrátor vyplní jméno rodiče, nebo přidáním uživatele do rodičova pole *users*. Po odeslání dotazu na server jej server zpracuje a nevyplněným parametrům nastaví výchozí hodnoty.

Aktualizace údajů uživatele, např. hesla, byla velmi nebezpečná operace. Z důvodu nespolehlivosti platformy při potvrzení aktualizovaného údaje - platforma změnu nezaregistrovala vůbec nebo špatně, nebo provedla aktualizaci jen ve vlastních strukturách, ale neinformovala ostatní systémy - bylo jednodušší používat tlačítko reset. Smazání celé databáze služeb a uživatelů kvůli změně hesla je velmi neefektivní a riskantní přístup. Proto byl pro novou platformu *Platform* následně zvolen přístup pomocí REST.

Správa služeb a hromadné nasazení

Každá aplikace byla v systému reprezentovaná pomocí php třídy a šablony docker-compose souboru napsané v yaml formátu. Z implementačního hlediska to byl velmi jednoduchý způsob, jak vygenerovat potřebný konfigurační soubor. Z pohledu používání a následné administrace se ukázalo, že je velmi neefektivní a nepřehledný. Navíc konfigurace pomocí yaml je velmi složitá, administrátor nemá k dispozici nápovědu ani vodítka a je tedy vysoká šance, že potřebné parametry zadá chybně nebo vůbec.

Za účelem hromadné tvorby uživatelů a služeb pro výuku HTML byl vytvořen php skript, jehož výstupem byla yaml konfigurace pro platformu. Tato konfigurace byla následně zkopírována do textového pole a odeslána na server. Napoprvé se konfigurace nezdařila z důvodu přednastavených výchozích kvót na počet spustitelných kontejnerů. Musela tedy být smazána částečná konfigurace, následně smazání potvrdit pomocí reset, upravit kvótu při generování, a pak teprve bylo možné vytvořit uživatele a služby k výuce. Nicméně výsledný produkt nebyl zcela optimální, z důvodu šablon byla žákům spuštěna i databáze, přestože ji vůbec nepotřebovali. I toto spuštění bylo následně motivující k tvorbě nové platformy.

Příprava nového druhu služby

Přidání nového druhu služby bylo velmi obtížné. Celý proces představoval naimplementování nové php třídy představující službu. V případě služeb pro výuku HTML by navíc bylo ideální mít oddělenou třídu reprezentující pouze Apache server, ale pro výuku php by musela jiná třída mít Apache server a databázi. V případě aktualizace Apache pak bylo velmi složité najít, kde je daný obraz použit, a doupravit potřebnou konfiguraci.

Reprezentace druhů služeb pomocí tříd měla ještě velké nevýhody při práci s databází. Jako databáze byla od začátku používána vždy jen Mariadb10, ale využití této přípravy bylo velmi obtížné. Služby např. WordPress, které využívají databázi, nebylo možné připravit stejným způsobem, jako php pro žáky. Pokusem vyřešit tento problém bylo dědit třídy mezi sebou, to ale celou situaci udělalo ještě více nepřehlednou.

Největší nevýhodou při tvorbě nového druhu služby je nutnost vytvoření yaml šablony pro danou službu. Třída pak jen doplňuje proměnné do šablony. I právě z tohoto důvodu je velmi těžké aktualizovat chování všech služeb centrálním nebo hromadným způsobem.

Shrnutí

Funkcionalita navíc, oproti pozdější *Platform* (která se časem také neosvědčila), byla možnost nastavení pro Apache druhou doménu. Při konfiguraci aplikace šla nastavit doména a kořen serveru pro vhost. Cílem bylo ušetřit zdroje severu a pro aplikace s podobným

chováním, lišící se pouze doménou, nasadit jen jeden apache server. Příkladem mohou být php skripty bez použití databáze. Pro nastavování služeb se ale ukázalo, že je výrazně jednodušší použít pro každou doménu zvláštní službu.

Používání *Platform*

Díky JSON REST API je používání *Platform* velmi přímočaré. Navíc umožňuje přístup pomocí html formulářů. Lze tedy rychle zkontrolovat, nebo aktualizovat, vazby a vlastnosti objektů systému.

Správa uživatelů

Zjednodušení dvou typů uživatele na jeden se osvědčilo při administraci. Při tvorbě nového uživatele není potřeba zvažovat jakého bude typu. Navíc lze kdykoli využít možnosti vytvořit uživateli potomka.

Klíčový parametr při tvorbě uživatele je rodič, vyplněním tohoto parametru se uživatel přiřadí rodiči.

API platformy navíc nabízí pohodlný přístup k seznamu všech uživatelů, takže je možné velmi pohodlně vyhledávat.

Smazání uživatele lze provést pomocí tlačítka delete, kdy metodou DELETE je odeslán dotaz na server.

Narozdíl od *ServAdmin* jsou hesla šifrována až na serveru, což velmi zjednodušuje celý proces zadání nového hesla.

Správa služeb a hromadné nasazení

Díky zvolené implementaci, kdy každá služba je v systému reprezentována samostatným objektem, lze velmi pohodlně ze služeb skládat aplikace.

Pro vytvoření služby je potřeba znát jméno aplikace, které má být služba přiřazena, a jméno šablony, ze které služba vychází. Pokud je to nutné, může administrátor přidat síť, svazek, proměnnou prostředí nebo nastavit příkaz pro spuštění. Tento přístup celkově velmi zjednodušuje administraci. Administrátor má možnost připojit aplikaci do další sítě, např. kvůli sdílení služby, využívající SSH s jiným uživatelem, nebo připojit další svazek, např. pro ukládání logů, v případě žákovských projektů.

Službu lze z aplikace odstranit zasláním příkazu delete.

Aplikace přijímá instrukce start, stop, restart, pomocí kterých lze ovládat Docker za účelem vypnutí / zapnutí aplikace. Příkaz restart byl implementován na základě zkušenosti nabyté při práci s platformou *ServAdmin*, kde byly pouze příkazy running a stop.

Největší výhodou při přípravě aplikací je práce s proměnnými prostředí. Při registrování druhu služby administrátor specifikuje, jaké proměnné služba ke svému běhu potřebuje. Například Mariadb server pro své spuštění vyžaduje proměnné *MYSQL_ROOT_PASSWORD*, *MYSQL_DATABASE*, *MYSQL_USER*, *MYSQL_PASSWORD*. Administrátorem bylo specifikováno, že hodnoty proměnných budou, pro zjednodušení názvu, přejaty z jiných proměnných *DB_USER*, *DB_PASSWORD*, *DB_DATABASE*. Při tvorbě aplikace WordPress byl následně nastaven export těchto proměnných s hodnotami *DB_USER: WordPress*, *DB_PASSWORD: WordPress*, *DB_DATABASE: WordPress*. Díky automatickému předání hodnot mezi službami není potřeba zasahovat přímo do textové konfigurace. Výhodou použití proměnných *DB_* je zjednodušený a přehlednější přístup. Proměnnou *DB_PASSWORD* lze přiřadit proměnné *MYSQL_ROOT_PASSWORD* a *MYSQL_PASSWORD*.

Příprava nového druhu služby

Registrace nového druhu služby spočívá hlavně v nahrání Dockerfile a potřebných metadat na server. Toho lze dosáhnout vytvořením instance Image, která reprezentuje druh služby. Pro vytvoření instance je potřeba vyplnit Dockerfile, který může zůstat prázdný, pokud se administrátor rozhodne přejmenovávat instance Image. Atribut *from* se vyplňuje pouze v případě, že se vychází z jiné instance Image, v tom případě se vyplní její jméno. Pokud je vyplněn atribut *from*, nemusí již být uvedeno FROM v Dockerfil, při ukládání souboru bude nahrazeno jménem obrazu předka. Pojmenování instance se rozhodne na základě atributu *comp_name*. Atribut *comp_name* nese název, pod kterým bude aplikace uvedena v docker-compose. Pokud atribut *comp_name* chybí, platforma nedovolí pro danou instanci vytvořit službu. Této vlastnosti bylo využito např. při vytvoření obecného obrazu deb, který představuje základní instalaci distribuce Debian. Cílem atributu *from* je uchovat stromovou strukturu závislostí obrazů mezi sebou.

Sestavení obrazu lze zahájit také z prostředí platformy. Platforma sama následně sestaví strom závislostí a nechá obrazy sestavit v takovém pořadí, aby byly zachovány všechny návaznosti. Cílem je, aby se nesestavil potomek dříve, než bude rodič ve finálním stavu. V takovém případě by musel být potomek sestaven znovu.

Administrátor má dále možnost specifikovat doplňující atributy např. sítě, svazky, proměnné prostředí, spouštěcí příkaz. Tyto doprovodné vlastnosti určují základní vlastnosti a chování druhu služby. Uživatel při tvorbě služby může tyto vlastnosti doplnit nebo, v případě proměnných prostředí, změnit hodnoty. Nastavení sítí slouží k určení dostupnosti služby, obecně je nastaveno, že s databází může komunikovat pouze aplikace, která ji využívá. Oproti tomu Apache servery jsou dostupné i$_z$jiných aplikací.

Shrnutí

HTML rozhraní se osvědčilo v případě aktualizace nastavení jedné služby. Rozhraní je velmi jednoduché, proto je určeno především pro zjednodušení vývoje platformy, aby měl vývojář snadný přístup k datům. Nevyužívá AJAX, a jedna stránka představuje vždy jen jeden datový objekt. Pro přiblížení Platformy žákům by bylo vhodné udělat lepší a přehlednější uživatelské rozhraní, které by umožňovalo např. aby si uživatel vytvořil služby sám podle nějakých šablon.

Navíc pro usnadnění vývoje nejsou implementovány žádné politiky přístupu. Uživatele ani nelze ověřit, nebo přihlásit. Tento problém lze vyřešit, neboť při návrhu bylo počítáno s budoucími politikami. Nehodí se proto pro použití běžným uživatelem, ani není nasaditelné veřejně. Prozatím je platforma obsluhována autorem práce, jako správcem.

Migrace služeb *ServAdmin* na *Platform*

Služby a uživatele, které byly uložené v rámci *ServAdmin*, byly podle jejich vzoru vytvořeny v prostředí *Platform*. Platforma sama si vytvořila potřebné vztahy a zdroje pro jednotlivé služby a uživatele.

Uživatelská data a data aplikací byla následně zkopírována mezi servery bez použití kontejnerů.

Nasazení systemd-socket-proxyd

Systemd byl použit pro vytvoření možnosti zapnout službu na dotaz a po uplynutí předdefinované časové lhůty ji vypnout. Implementace pomocí systemd se ale nakonec neosvědčila z důvodu složitosti konfigurace služeb a nespolehlivosti celého řešení.

Aplikační proxy server

Konfigurace proxy serveru, který představovalo systemd, byla udělaná automaticky na úrovni aplikace. Pro dodatečné nasazení proxy serveru ale bylo potřeba změnit již existující záznamy v uživatelských proxy serverech. Tato změna byla udělána automaticky pomocí podpůrných php skriptů. Pro běžného uživatele by časem mohlo být matoucí, že všechen provoz aplikace je přesměrováván na jeden kontejner s použitím různých portů.

Problém nastával s přidáním sítě pro připojovaný kontejner. Z důvodu toho, že všechna komunikace prochází systemd, je potřeba mít je připojeno do všech sítí, do kterých mají mít přístup služby, které zprostředkovává. Tento problém pravděpodobně nepůjde vyřešit ani použitím jiné aplikační proxy.

Vazba na platformu

Systemd byl vyvíjen při používání Linuxové distribuce Ubuntu a instalován do kontejneru využívajícího systém Debian. Při základním testování bylo zjištěno, že systemd není možné použít v kontejnerech využívajících systém Debian. Po změně distribuce využité v kontejneru na Ubuntu systemd začal spouštět procesy a chovat se podle očekávání.

V době prvních testů a vývoje systemd byl server řízen platformou *ServAdmin* se systémem Debian. Právě používání systému Debian znesnadňovalo nasazení kontejneru se systemd. Po provedení testů, kombinace systém hostitele a systém v kontejneru, se ukázalo, že systemd funguje nejspolehlivěji při používání kombinace Ubuntu-Ubuntu.

Proč systemd lze použít pouze při kombinaci Ubuntu-Ubuntu nemá jasné odůvodnění. Celá platforma a řešení se tím stávají závislé na systému Ubuntu.

Nastavení intervalu

Interval nečinnosti služby, po jehož uplynutí měla být služba vypnuta, byl stanoven na 10 min. Interval byl stanoven na základě odhadu možství času, po jehož uplynutí nebude vadit zpoždění odpovědi na první dotaz v důsledku startování služeb.

Testování prokázalo, že interval byl příliš krátký. Následná konzultace ukázala, že situace, kdy vyučující vykládá výuku a žáci nepracují s aplikacemi, může být i mnohem delší, než 10 minut.

Chyba nastaveného intervalu se ještě více projevila v případě úložiště, které se mělo po době nečinnosti vypnout. Služby úložiště ale nebyly dostupné včas a OS Windows nebyl schopen opětovně připojit síťovou jednotku. V důsledku automatického vypínání úložiště byla práce velmi nekomfortní a pro žáky matoucí.

Řízení kontejnerů při testování

Spolehnutím se na dynamické řízení kontejnerů pomocí systemd docházelo k zbytečnému zapínání služeb. Příkladem mohou být SSH a SFTP služby pro žáky. V době tvorby testu byla zvažována i možnost testovat dostupnost úložiště pomocí SFTP, ale po uvážení bylo od tohoto testu upuštěno z důvodu jeho časové a znalostní náročnosti. Kontejnery ale zůstaly připravené s tím, že v případě nepoužívání budou pomocí systemd vypnuty.

V důsledku toho v průběhu testování mohlo být v provozu až 318 kontejnerů, z nichž 240 bylo zapnutých. Spouštěním služeb žáky došlo k přetížení a následnému selhání monitorovací aplikace cAdvisor. Následné používání aplikací vedlo k stále většímu zatížení systému, které mohlo být monitorováno pouze jako celkové vytížení hostujícího systému. Situaci se podařilo zmírnit vypnutím 96 kontejnerů, z nichž bylo vypnuto 32, spojených s výukou HTML, které v době testování nebyly využívány.

V průběhu testování dále vznikla potřeba kontejnery restartovat, důvodem byla snaha o hotfix chybné interpretace souboru .htaccess WebDAV serverem. Následné zapínání služeb, které trvalo asi 20 min, vedlo k nepoužitelnosti systému po dobu zapínání kontejnerů. Následné vypínání služeb po uplynutí intervalu neproběhlo korektně. U poloviny aplikací došlo k vypnutí jen jedné služby ze dvou vypnutelných, v případě druhé poloviny nedošlo k vypnutí vůbec. Analýza vypnutých a nevypnutých kontejnerů ukázala, že aplikace zapnuté dříve se vypnuly z poloviny a nevypnuté aplikace se zapínaly v době, kdy se spustilo vypínání.

Závěr

Systemd se pro dynamické řízení kontejnerů neosvědčilo. Hlavně kvůli složité a těžko čitelné implementaci spojené se závislostí na distribuci Ubuntu, která nemá zcela předvídatelné chování. Následné problémy spojené s přesměrováváním provozu na aplikační proxy server nejsou přímou nevýhodou implementace pomocí systemd, ale souvisí s použitím aplikační proxy.

Problematika dynamického řízení služeb by v budoucnu stála za další prozkoumání. Na základě získaných zkušeností se systemd bych doporučil vlastní implementaci aplikačního proxy serveru.

Testování

Testování prototypu i výsledného produktu bylo provedeno po částech v několika formách. Popis závěru testování jednotlivých aplikací je rozdělen do kategorií podle charakteru aplikace.

Webové služby

Následující aplikace pro své použití vyžadují protokol HTTP. Aplikace jsou dostupné pomocí dvou reverzních proxy serverů přesměrovávajících dotaz.

Hotové aplikace

Existující převážně open-source aplikace s oficiálním vývojem a komunitou. Nejedná se tedy o žákovské nebo studentské projekty.

WordPress

Redakční systém WordPress je určený k tvorbě a údržbě webových stránek napsaný v jazyce php.

Nasazení a testování bylo provedeno v případě vývoje testovacích aplikací. Uživatelé účastnící se testu měli připravenou instalaci WordPress, kde prošli úvodní instalaci a následně zkoušeli tvorbu webu z šablon.

Při testování nebyla zaznamenána žádná zpoždění ani nedostupnost aplikace.

Grafana

Aplikace grafana je využívána pro zobrazení monitorovaných parametrů serveru na adrese grafana.5thbeat.labs.optonet.cz. Nasazení aplikace pomocí kontejnerů bylo velmi jednoduché. Samotný běh je plynulý a neovlivněný prostředím dockeru. Aplikace má lepší možnosti přístupu ke zdrojům dat, ať už v podobě Promethea, nebo pomocí přesměrovaných portů navázaného spojení k externímu php skriptu.

Gitea

Gitea je služba určená pro podporu vývoje software využívaná jako git repozitář pro týmovou spolupráci. Vzhledem a používáním se podobá aplikaci GitHub.

Aplikace byla testována při vývoji projektů do předmětů ITU, IIS a IVS. V prvním testování vývoje aplikace *ITUIIS* byla využívána několika kolegy pro přístup k repozitářům protokolu SSH. Toto řešení bylo složité na nastavení, bylo třeba spolužákům správně umístit SSH klíče a připravit jim *.ssh/config*. Pro běžného uživatele mohla být velmi matoucí adresa serveru. Gitea generuje adresu repozitáře, jenže tato adresa byla dostupná až po nakonfigurování SSH. Kolegové tedy nemohli nastavit přístup bez znalosti vnitřních propojů na serveru. Po nastavení přístupů byla aplikace používána bez dalších problémů.

Při druhém testování, v rámci vývoje projektu *ArithmeticCalculator* do předmětu IVS, byl ve spolupráci s kolegy z prvního ročníku otestován přístup pomocí HTTP. Přístup pomocí protokolu HTTP nebyl tak náročný jako v prvním případě testování. Nyní kolegům stačilo zkopírovat odkaz, který aplikace za tímto účelem vygenerovala, a beze změn jej vložit do gitu. Při používání byli spokojení, nic dalšího jako chybu ani nedostatek nepopsali.

Na základě těchto zkušeností lze usoudit, že pro výuku by bylo možné aplikaci použít, a to jak s HTTP, tak s SSH protokolem.

Adminer

Z důvodu obtížného nasazení aplikace phpMyAdmin v prostředí vytvořeného webhostingu, a po prodiskutování s technickým konzultantem, byla jako alternativa zvolena aplikace Adminer. Tato aplikace dostatečně nahrazuje phpMyAdmin z pohledu administrace databáze pro výukové účely.

Žáci při testování porovnali Adminer s vlastními zkušenostmi s phpMyAdmin, a rozdíl jim nevadil. Vzhled aplikace jim sice přišel zastaralý, ale neshledali to jako závažný problém pro použití při výuce.

Apache server

Testování web server Apache bylo rozděleno do tří částí.

php

Spouštění php skriptů za pomoci mod_php bylo vyzkoušeno s žáky 4. ročníku. Žáci dostali za úkol napsat skripty v php, kterými měli postupně otestovat:

Jednoduchou odpověď parafráze *Hello World*
Cookies vytvořit počítadlo přístupu na stránku pomocí cookies
Session přihlásit/odhlásit uživatele při použití session
Databáze použít staré cvičení z výuky pro otestování spojení s databází

Všechna zadání žáci úspěšně splnili, v případě připojení databáze bylo třeba jen ujasnit, jaké měli nastavit parametry pro připojení.

Obecně by žáci doporučili použití webhostnigu k další výuce na střední škole. Jako velký přínos hodnotili vzdálený přístup na server a absenci flash disku s externí aplikací pro spuštění lokálního webového serveru.

CGI

CGI neboli *Common Gateway Interface* označuje způsob tvorby dynamického obsahu webových stránek při používání webového serveru Apache. Tvorba dynamického obsahu je prováděna spuštěním externího programu nazývaného *CGI program* nebo *CGI skript*. Pro

správné fungování CGI je nutné v odpovědi generované skriptem vytvořit hlavičky HTTP protokolu. Dále je nutné nastavit, aby byl CGI skript spustitelný webovým serverem.

Java kalkulačka

Za účelem otestování a získání představy o náročnosti použití cgi pro tvorbu odpovědi byla v jazyce Java naimplementována kalkulačka. Tato kakulačka pomocí html formuláře od uživatele načte dvě čísla, která sečte a následně uživateli vrátí výsledek v html podobě. Takto vytvořená kalkulačka byla následně přeložena a zabalena do spustitelného *index.jar*.

Aby mohl být spuštěn jar soubor, byl vytvořen skript index.sh. Tento skript je pro apache spustitelný.

Vytvoření kalkulačky bylo oproti původnímu očekávání mnohem těžší. V průběhu nasazení aplikace chybí logy. Dopad absence logů je nejvíce cítit při tvorbě index.sh. Apache zapisuje chyby do souboru */var/log/apache2/error.log*, jenže tento soubor je pro běžného uživatele nečitelný. Navíc běžný žák o souboru neví. Apache server předává dotaz jako hodnoty v proměnných prostředí, s čímž nemají žáci na střední škole zkušenosti. Tyto hodnoty jsou v javě reprezentovány typem String, pro zpracování vstupu je tedy potřeba přečíst proměnné prostředí a následně je převést na číselný datový typ. Problém zpracování se dá vyřešit výběrem jiného jazyka, ale předání informací pomocí proměnných prostředí zůstane.

Žákovské cgi

Použití cgi za účelem tvorby web stránek bylo s žáky testováno dvakrát.

Během prvního pokusu testování bylo žákům vysvětleno, co bude jejich cílem a jak mají postupovat. Žáci pak v rámci řešení úlohy vytvořili požadovaný soubor .htaccess, kde měli mimo jiné nastavit jméno spouštěného cgi skriptu, jenže tento soubor přečetl i jiný Apache server a to ten, který žákům poskytoval úložiště pomocí WebDAV. Po založení souboru *.htacces*, téměř s jakýmkoli obsahem, došlo na straně úložiště k chybě a adresáře se staly nečitelnými. Při používání *.htaccess* nebyla tato chyba odhalena dříve, protože používáním protokolu SFTP k ní nedojde.

Druhé testování proběhlo později, dopředu byly připraveny servery tak, aby tvorba *.htaccess* nebyla pro vyřešení nutná a současně server s úložištěm tento soubor ignoroval.

V úvodu hodiny byli žáci požádáni, aby vytvořili php skript, který jim přečte a vypíše */var/log/apache2/error.log*. Po zkušenostech s vývojem java kalkulačky byly běžnému uživateli zpřístupněny apache logy.

V rámci samotného testování dostali studenti za úkol vytvořit bash skript, který jim vrátí libovolný pozdrav.

Tento úkol byl poměrně obtížný.

Největším problémem studentů bylo, že pracovali s operačním systémem Windows, ve kterém nejde jednoduše změnit příznak spustitelnosti souboru pro Linux a jako konec řádku se používají znaky 13 a 10.

Po zjištění, že nemohou ve Windows nastavit příznak spustitelnosti souboru, jim byly příznaky přiděleny správcem serveru.

Pro vyřešení 1310 vs 13 si žáci stáhli NotePad++ nebo PSPad.

Velkým nedostatkem bylo, že si žáci nezkusili spustit skript jen pro test na lokálním počítači. Při testování na lokálním pc by zjistili dříve, že mají špatné kódování textu, než vyčítáním z chybového logu apache.

Další věc, která byla pro žáky velmi obtížná, bylo správné nastavení HTTP hlaviček. Žáci je buďto neodeslali vůbec nebo špatně.

Problém se spustitelností souboru lze vyřešit pomocí php, kde lze voláním funkce *shell_exec* spustit příkaz příkazového řádku. Registrování jmen spustitelných souborů bylo po zkušenostech z prvního testování provedeno v apache2.conf, v případě nestandartní situace je ale stále možné přepsat toto chování pomocí *.htaccess*. Apache hledá soubory ke spuštění v pořadí *Makefile index index.sh index.php index.html*, kde *Makefile index a index.sh* jsou spouštěny pomocí cgi. Problém s hlavičkami a nepřehledností shell skriptu bylo vyřešeno pomocí Makefile. Generování odpovědi pomocí Makefile se ukázalo jako uživatelsky nejpohodlnější.

Tento způsob zpracování dotazu lze hodnotit jako velmi náročný a určený pro pokročilé uživatele nebo pro zábavu. Původně bylo zamýšleno pomocí cgi demonstrovat použití Javy jinak, než žáci znají, ale celá problematika je pro ně velmi složitá.

Vlastní server

Vlastním serverem je v tomto kontextu myšlen proces spuštěný za účelem zpracování HTTP dotazu. Do této kategorie potom spadá i php Development Server.

V době nasazení první služby tohoto typu služby byla použita pro řízení platforma *ServAdmin*, takže nasazení bylo velmi chaotické, nečitelné a zdlouhavé. Samotná aplikace ale fungovala bez větších potíží. V případě použití *Platform* by nasazení bylo značně jednodušší, z důvodu absence nutnosti implementovat nové třídy.

Možnost spuštění vlastní implementace web serveru byla vyzkoušena pouze pomocí nasazení projektu do IPK. Tématem projektu bylo vytvořit webserver, který bude odpovídat různé HTTP statuty. Pro testovací účely je projekt k dispozici na adrese ipk.kaspi.personal.5thbeat.labs.optonet.tech, kde jsou dostupné cesty /teapod, /cpu-name, /load a /hostname.

Pro spuštění projektu byl do souboru Makefile doplněn cíl *run*, který projekt nechá sestavit a následně jej spustí. Jiné zásahy do struktury projektu nebyly nutné.

Nasazení vlastní implementace serveru je tedy možné. Vlastní proces vývoje by mohl být limitovaný nutností proxyserveru ze strany serveru a absencí logů, případně chybových hlášek systému. Dalším limitem může být absence možnosti nastavit individuální spouštěcí příkaz. Uživatelé by museli respektovat spouštění např. pomocí shell skriptu start.sh, který by museli implementovat.

SSH služby

Použitelnost SSH služeb pro výuku je těžké jednoznačně určit. Příprava přístupu k SSH službám je obtížnější, než u protokolu HTTP. Další použití služeb je ovšem možné a jednoduché.

Pro zlepšení použitelnosti kolegové navrhli vytvoření generátoru *.ssh/config*. Jako největší problém se ukázalo nahrávání SSH klíčů na server. Kolegové tyto klíče nahráli v rámci aplikace Gitea a tyto klíče byly nakonec rozkopírovány do uživatelských složek.

Git

Možnost soukromého git repozitáře je zajímavá, ale v praxi pro žáky méně použitelná, než bylo předpokládáno na začátku. Založení git je velmi jednoduché: po připojení úložiště stačí vstoupit do adresáře a pomocí příkazu `git init --bare` lze vygenerovat kostru repozitáře. Nevýhodou tohoto řešení je možnost přístupu pouze majiteli (nelze tedy časem začít repozitář sdílet) potřeba nastavit SSH a neexistence dalších rozšíření.

Přesměrování portů

Přesměrování portů je použitelné v obou variantách na základě vlastní zkušenosti s použitím této služby. Pro monitorování serverů 5thbeat a BPS byl použit remote port forwarding navazovaný pomocí služby systemd. Navázání pomocí docker kontejneru bylo použito v případě služby poskytující data o stavu periferií a zařízení v datovém centru vizualizační aplikaci Grafana. Místní přesměrování portů je pak používáno pro přístup k administračnímu prostředí *Platform*.

Do budoucna je v plánu připojení InfluxDB běžící v Docker kontejneru na server 5thbeat pomocí místního přesměrování portů za účelem monitoringu Proxmoxu, který zapisuje data do InfluxDB.

Proxy jump

Tato služba je klíčová pro používání dalších SSH služeb. Díky zpřístupňování dalších serverů je nastavení této služby klíčové. Pro úspěšné nastavení doporučuji použít SSH klíče a *.ssh/config*.

Kolegům, kteří spolupracovali na projektu do předmětu ITU, byla poskytnuta pomoc s přípravou a nakopírováním *.ssh/config* a nakonec se jim podařilo používat další služby. Nastavení hodnotili jako složitější, ale s předpřipraveným *.ssh/config* použitelné.

Úložiště

Přístup k úložišti je umožňen pomocí dvou protokolů, a to WebDAV a SFTP.

SFTP

Použití SFTP bylo testováno autorem, protože příprava SSH přístupu vyžaduje znalost konfigurace SSH. Dále je použití SSH jednoduší pro systém Linux. Nicméně po nastavení SSH přístupu lze úložiště používat běžným způsobem. Dlouhodobě byla maximální rychlost připojení 50 MB/s, což je přibližně polovina 1Gbps pásma. Rychlost bez použití proxyJump však byla nižší, než původní předpoklad, a to pouhých 80 MB/s.

WebDAV

Přístup pomocí WebDAV lze jednoduše nastavit na počítačích s operačním systémem Windows. Spolehlivé nasazení bylo možné až po změně typu autentizace. Operační systém Windows odmítal připojení síťové jednotky při použití Basic Auth. Pro vyřešení problému byl na serveru změněn typ ověření na Diggest Auth. Alternativním řešením mohla být změna registů OS Windows, ale nutnost zásahu do registrů by značně ztížilo nasazení pro školu nebo běžné uživatele.

V rámci vyhodnocení byl WebDAV otestován několika uživateli z autorova okolí. Při tomto testování bylo zjištěno, že rychlost přenosu je nízká. Přenos větších souborů trval dlouho. Dosahovaná rychlost se pohybovala v řádech desítek KB.

Při testování ve škole toto však nebylo omezující. Žáci ve svém hodnocení zmínili delší odezvu serveru, ale nebylo to pravidlem. Lze tedy říci, že při kopírování malých souborů to není limitující.

SFTP WebDAV

Následující tabulka ukazuje základní srovnání protokolů sloužících k přístupu k úložišti.

Tabulka 5.1: Srovnání přístupových protokolů

Hodnocený parametr	SFTP	WebDAV
použitelné pro systém	Linux	Windows
dosahovaná rychlost	50 MB/s	10 KB/s
náročnost přípravy	vysoká	nízká
zabezpečení	SSH klíče/heslo	heslo/žádné

Ze srovnání vyplývá, že protokol SFTP by měl být lepší. Ale při používání je příprava WebDAV značně jednodušší.

Monitoring

V této kapitole budou shrnuta naměřená data. Sledovanými parametry byly především využití operační paměti a zatížení procesoru. Vedlejším parametrem bylo sledování využití sítě.

V průběhu sledování zátěže měl server přiděleno 16 GB operační paměti a 16 jader procesoru. Cílem sledování bylo zjistit, jestli výkon serveru bude dostačující pro nasazení a použití při výuce.

Sběr dat měl poběhnout ze dvou exportérů, a to *node exporter* a *cAdvisor*. cAdvisor ale nakonec použit nebyl z důvodu jeho přetížení a následného poškození dat. Graf popisující naměřené hodnoty získané po zapnutí 260 kontejnerů ukazuje, že cAdvisor má výpadky.

Obrázek 5.1: Průběh výpadků *cAdvisor*

Následující graf zachycující stejný časový úsek ukazuje výpadky i v měření pomocí *node exporter*. Výpadky *cAdvisor* byly zapříčiněny nedostatkem paměti a jeho následným ukončením pomocí *oom_killer*. Následující výpadky v měření *node exporter* jsou zapříčiněny přetížením Promethea, který byl též ukončen pomocí *oom_killer*.

Následující grafy ukazují vytížení serveru při výuce HTML. Z grafů lze vyčíst, že vytížení procesoru ani paměti nezaznamenalo prudké změny. Lze se tedy domnívat, že samotným využíváním služeb nedošlo k výraznému navýšení zatížení serveru. Graf současně zachycuje vytížení serveru při testování použití ve výuce prováděné s žáky 4. ročníku. Žáci během výuky pracovali s webovým serverem využívajícím php, databází a úložištěm připojeným pomocí protokolu WebDAV.

Z naměřených hodnot lze soudit, že zatížení serveru představuje více běh služeb než jejich přímé využívání výukou. Měřením se ukázalo, že výkon serveru pro použití ve výuce je

Obrázek 5.2: Doložení výpadků *cAdvisor* pomocí *node exporter*

Obrázek 5.3: Využívání serveru s vypnutým *cAdvisor* při testování i výuce

omezován nedostatkem operační paměti. Její využití dlouhodobě převyšovalo 60 %. Dopady tohoto problému lze redukovat pravidelným zapínáním/vypínáním služeb podle rozvrhu, nebo nastavením jejich spouštění na dotaz. Obecně z naměřených dat vyplývá, že po stránce zatížení serveru je možné využívat služby pro výuku minimálně dvou skupin žáků současně.

Kapitola 6

Závěr

Cílem práce bylo zprovoznit wehosting, který podpoří výuku IT na střední škole, tento cíl byl splněn a jeho výsledkem je nový webhosting podporující výuku IT na střední škole. Vytvořené řešení webhostingu bylo otestováno ve výuce na Střední škole průmyslové, technické a automobilní v Jihlavě žáky prvního a čtvrtého ročníku.

V rámci práce byla připravena platforma řízení kontejnerů, pomocí které byly nasazeny služby podporující výuku na střední škole. Tato platforma řízení byla implementována s možným rozšířením o uživatelské rozhraní v budoucnu.

Dlouhodobé testování při výuce html na *Střední škole průmyslové, technické a automobilní v Jihlavě* prokázalo, že žáci mají zájem o veřejný webhosting. Tento trend potvrdilo i testování s žáky 4. ročníků, kteří ocenili dostupnost platformy a její možnosti. Žáci nejvíce oceňovali možnost vzdáleného úložiště a jednoduché zobrazení chybových logů.

Monitoring serveru ukázal nedostatek operační paměti, v důsledku čehož byly vypínány monitorovací služby. V průběhu výuky ale nebyla zjištěna nárazová navýšení ve využívání serveru. Z toho lze soudit, že po rozšíření operační paměti by mohl být webhosting nasazen i pro více skupin žáků.

Zpracovaná témata by si zasložila rozšíření v oblasti spouštění služeb na dotaz, ale zejména v oblasti orchestrace. Množství kontejnerů vytvořených v rámci webhostingu může velmi ulehčit budoucí průzkum a nasazení orchestrace.

Zajímavé by také bylo testování provést nejen v příslušných předmětech na oborné střední škole, ale také ve výuce informatiky na vyšším gymnáziu.

Literatura

[1] *Apache Module mod_cgi*. Dostupné z: https://httpd.apache.org/docs/2.4/mod/mod_cgi.html.

[2] *Apache Module mod_dav*. Dostupné z: https://httpd.apache.org/docs/2.4/mod/mod_dav.html.

[3] *Apache Module mod_rewrite*. Dostupné z: https://httpd.apache.org/docs/2.4/mod/mod_rewrite.html.

[4] *Apache Tutorial: Dynamic Content with CGI*. Dostupné z: https://httpd.apache.org/docs/2.4/howto/cgi.html.

[5] *Systemd-socket-proxyd — Bidirectionally proxy local sockets to another (possibly remote) socket*. Dostupné z: https://www.freedesktop.org/software/systemd/man/systemd-socket-proxyd.html.

[6] *Systemd(1) — Linux manual page*. Dostupné z: https://man7.org/linux/man-pages/man1/init.1.html.

[7] *What is Orchestration?* Dostupné z: https://www.databricks.com/glossary/orchestration.

[8] *What is time series data?* Dostupné z: https://www.influxdata.com/what-is-time-series-data/.

[9] *Wordpress*. Dostupné z: https://hub.docker.com/_/wordpress.

[10] *What is Podman?* 2019. Dostupné z: https://docs.podman.io/en/latest/.

[11] *Co je webhosting a jak vybrat ten nejlepší*. 2020. Dostupné z: https://www.active24.cz/jak-na-tvorbu-webu/tvorba-stranek-pokrocila/co-je-web-hosting-a-jak-vybrat-ten-nejlepsi.

[12] *Docker overview*. 2023. Dostupné z: https://docs.docker.com/get-started/overview/.

[13] *Use containers to Build, Share and Run your applications*. 2023. Dostupné z: https://www.docker.com/resources/what-container.

[14] Authors, T. G. *What is Gitea?* 2023. Dostupné z: https://docs.gitea.io/en-us/.

[15] Bigelow, S. J. *How to use Kubernetes' self-healing capability*. 2022. Dostupné z: https://www.techtarget.com/searchitoperations/tip/How-to-use-Kubernetes-self-healing-capability.

[16] brandon.lee. *Kubernetes LXC Containers Configuration – Lab Setup*. 2021. Dostupné z: https://www.virtualizationhowto.com/2021/07/kubernetes-lxc-containers-configuration-lab-setup/.

[17] Ganapathi, D. *Kubernetes VS Docker Swarm – What is the Difference?* 2023. Dostupné z: https://www.knowledgehut.com/blog/devops/docker-swarm-vs-kubernetes.

[18] Hira, Z. *Kubernetes VS Docker Swarm – What is the Difference?* Dostupné z: https://www.freecodecamp.org/news/kubernetes-vs-docker-swarm-what-is-the-difference/.

[19] Smith, S. *On-demand activation of Docker containers with systemd*. 2015. Dostupné z: https://blog.developer.atlassian.com/docker-systemd-socket-activation/.