Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra Informačních Technologií

Vývoj a nasazování aplikací na platformě Next.js v cloudovém prostředí

Bakalářská práce

Autor práce: DOMINIK RESL
Studijní obor: Aplikovaná Informatika

Vedoucí práce: Mgr. VOJTĚCH VOREL, Ph.D.

Hradec Králové 2023

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

. . . .

Dominik Resl

21. dubna 2023

Poděkování

Rád bych poděkoval svému vedoucímu práce Mgr. Vojtěchu Vorlovi, Ph.D. za konzultace, odbornou i metodologickou pomoc a podporu při zpracování mé bakalářské práce.

Anotace

Cílem práce je představit konkrétní sadu vývojářských a operativních technologií, která by obstála při tvorbě koplexního webu v produkčním prostředí. Vybrané technologie budou představeny a porovnány s alternativami, a to zejména v oblastech: webové platformy pro tvorbu obsahu (byla zvolena Next.js), správa a verzování zdrojového kódu, škálovatelné propojení s databází, kontejnerizace a virtualizace, automatické nasazování (byly zvoleny Docker, Kubernetes, GitLab CI/CD).

Praktická část práce se bude sestávat z ukázkové webové aplikace přiměřeného rozsahu, nasazené v cloudu a využívající vybrané technologie. Jako předmět tohoto ukázkového projektu poslouží data o vytížení serveru, která budou prezentována uživateli na webové stránce.

Anotation

Title: Development and deployment of Next.js applications in cloud environment

The purpose of this thesis is to introduct specific development and operative technologies, which are used to create complex web application in production environment. chosen technologies will be described and compared with alternatives in these areas: web development tools (Next.js was chosen), a versioned code management, scalable connections with database, containers and virtualization, automatic deploys (Docker, Kubernetes, GitLab CI/CD were chosen).

The practical part of the thesis consists of a sample web application deployed to cloud using mentioned technologies. The monitoring server data will be used in order to present them to the web application.

Obsah

1 Úvod 1

2 Cíl práce 2

3 Metodika zpracování 3

4 Teoretická část 4
- 4.1 Webová aplikace . . . 4
  - 4.1.1 Webová aplikace a HTTP . . . 4
  - 4.1.2 Architektura webové aplikace . . . 4
  - 4.1.3 Tvorba webových aplikací . . . 4
  - 4.1.4 Javascriptové knihovny a frameworky . . . 6
- 4.2 Vývoj webových aplikací . . . 7
  - 4.2.1 Správa a verzování zdrojového kódu . . . 7
  - 4.2.2 Nástroje pro vývoj webových aplikací v jazyce Javascript . . . 10
- 4.3 Nasazování webových aplikací . . . 14
  - 4.3.1 Tradiční nasazování . . . 14
  - 4.3.2 Virtualizace . . . 15
  - 4.3.3 VPS a Cloud Computing . . . 15
  - 4.3.4 Kontejnerizace . . . 16
  - 4.3.5 Docker . . . 16
  - 4.3.6 Docker Compose . . . 20
- 4.4 Kubernetes . . . 22
  - 4.4.1 Kubernetes komponenty . . . 22
  - 4.4.2 Kubernetes objekty . . . 23
- 4.5 CI/CD . . . 27
  - 4.5.1 Continuous Integration . . . 27
  - 4.5.2 Continuous Delivery . . . 27
  - 4.5.3 Continuous Deployment . . . 27
  - 4.5.4 GitLab CI/CD . . . 27

5 Praktická část 29
- 5.1 Vývoj Next.js aplikace . . . 29
  - 5.1.1 Struktura aplikace . . . 29
  - 5.1.2 Aplikace, API a proměnné prostředí . . . 32
  - 5.1.3 Dockerizace aplikace . . . 34
- 5.2 Konfigurace Kubernetes . . . 35
  - 5.2.1 Vytvoření objektů pro produkční prostředí . . . 35
  - 5.2.2 Vytvoření objektů pro testovací prostředí . . . 38
  - 5.2.3 Nasazení aplikace . . . 38
- 5.3 Konfigurace GitLab repositáře . . . 38
  - 5.3.1 Docker registr . . . 39
  - 5.3.2 Proměnné prostředí . . . 39
  - 5.3.3 CI/CD . . . 40

6 Souhrn výsledků 44

7 Závěry a doporučení **45**

Literatura **46**

Přílohy **48**

Seznam obrázků

1 Balíčkovací systémy přiřazené k programovacím jazykům 10
2 Inicializace projektu pomocí npm 11
3 Instalace knihovny React pomocí npm 11
4 Sestavení produkčního zdrojového kódu pomocí Webpacku 13
5 Architektura Docker kontejnerů 16
6 Node.js na Docker Hubu 17
7 Ukázka práce s Docker CLI 18
8 Příklad souboru Dockerfile 18
9 Příkaz „docker run“ 19
10 Příkaz „docker network“ 19
11 Ukázka souboru „docker-compose.yml“ 20
12 Použití Docker Compose CLI 21
13 Kubernetes klastr 22
14 Kubernetes prostředky viditelné v každém jmenném prostoru 24
15 Kubernetes prostředky zařazené do jmenných prostorů 24
16 Struktura Next.js webové aplikace 29
17 Soubor „package.json“ 29
18 Soubor „tailwind.config.js“ 30
19 Struktura Next.js komponent 31
20 Soubor „about.tsx“ 31
21 Volání Kubernetes API v Next.js aplikaci 32
22 Výpočet procentuálního zatížení CPU 33
23 Ukázka aplikace „vskp-demo“ 33
24 Dockerfile aplikace „vskp-demo“ 34
25 Vytvoření obrazu a spuštění kontejneru 35
26 Definice tajných dat 36
27 Definice nasazovacího objektu 36
28 Definice objektu služby 37
29 Definice Ingress objektu 37
30 Nasazení aplikace do produkčního prostředí pomocí kubectl 38
31 Nasazení aplikace do testovacího prostředí pomocí kubectl 38
32 Autorizační token do repozitáře 39
33 Proměnné v GitLab CI/CD 40
34 Proměnné použité v definici CI/CD 40
35 CI/CD - testovací krok (kontrola syntaxe) 41
36 CI/CD - vytvoření Docker obrazu v testovací větvi 41
37 CI/CD - GitLab registry s Docker obrazem se dvěma štítky 42
38 CI/CD - definice kroku pro nasazení testovací verze aplikace 42
39 CI/CD - porovnání procesů CI/CD 43

1 Úvod

Tato bakalářská práce se zabývá životním cyklem aplikace od samotného vývoje po nasazení aplikace do produkčního prostředí. Vývoj aplikace je obecně chápán jako část procesu, během kterého dochází ke vzniku určitého softwaru, v této práci se bude jednat konkrétně o webovou aplikaci. Webová aplikace bude postavena na React frameworku Next.js a napojená na serverové API, odkud bude získávat aktuální data o vytížení serveru.

V rámci bakalářské práce budou představeny, porovnány a podrobně vysvětleny technologie a nástroje související s vývojem a nasazováním aplikací. Pro správu zdrojového kódu bude využit nástroj pro správu souborů GIT. Dále bude představena problematika CI/CD (*Continuous Integration a Continuous Delivery/Continuous Deployment*), která pomáhá k automatizaci a správě životního cyklu aplikace.

Dále bude použita technologie Docker s využitím nástroje Kubernetes pro správu jednotlivých kontejnerů a cloudového prostředí.

2 Cíl práce

Cílem práce je ukázat a podrobně vysvětlit problematiku vývoje a nasazování webových aplikací do produkčního prostředí za využití moderních a robustních technologií. Na základě teoretické části bude provedena implementace webové aplikace a nasazení do cloudového prostředí za využití CI/CD a nástroje Kubernetes. Výsledná architektura bude porovnána s alternativami v oblasti nasazování.

3 Metodika zpracování

V rámci teoretické části budou stanovena základní témata a představeny konkrétní technologie potřebné pro vypracování praktické části. K tomu bude použito odborné literatury, článků, diskusních fór a webových stránek. Pro implementaci cloudového prostředí bude využit virtuální server společnosti Hetzner, na kterém je nasazená výsledná webová aplikace na platformě Kubernetes. Ve skutečnosti je na tomto virtuálním serveru nasazeno i několik reálných produkčních webů a aplikací.

4 Teoretická část

4.1 Webová aplikace

4.1.1 Webová aplikace a HTTP

Webová aplikace je softwarový program přístupný přes webové rozhraní, kterým může být webový prohlížeč či jiný nástroj pro práci s protokolem HTTP (*Hypertext Transfer Protocol*). Jedná se o standardizovaný internetový protokol, pomocí kterého komunikují aplikace na webu a webovém prostředí. Slouží pro přenos souborů různého typu, nejčastěji se však jedná o soubory *HTML*, *XHTML*, *XML*, *CSS*, *JS* a v dnešní době stále častěji využívaný formát *JSON*. Číslo portu HTTP protokolu je rezervováno na hodnotě 80. Je zvykem, že webové prohlížeče automaticky číslo tohoto portu doplňují do adresního řádku. Sám o sobě je HTTP nešifrovaný protokol, veškerý provoz tak probíhá ve formě prostého textu, který je z bezpečnostního hlediska velmi rizikový. Vedle HTTP existuje šifrovaná varianta HTTPS, která se stará o šifrovanou komunikaci, z hlediska bezpečnosti se tak jedná o efektivnější variantu komunikace po internetu.

4.1.2 Architektura webové aplikace

Webová aplikace se dělí na klientskou a serverovou část. Serverová část je standardně nainstalována (*či hostována*) na fyzickém nebo virtuálním serveru, který je zpravidla zpřístupněný skrz internet přes protokol HTTP nebo HTTPS. Toto rozdělení dalo základ modelu *client-server*. Princip tohoto modelu spočívá v existenci jedné serverové aplikace (serveru) a klienta. Klient je zde chápán jako webový prohlížeč, jehož chování je definováno klientskou částí aplikace. Více webových prohlížečů (klientů) je pak připojeno na jeden server.

Komunikace mezi serverovou a klientskou částí může probíhat pomocí webového API (*Application Programming Interface*). Níže je popsána API architektura REST, pomocí které se tvoří webová API.

REST

Rozhraní typu REST tvoří komunikační vrstvu mezi klientskou a serverovou částí aplikace. Jedná se o standardizované API, pomocí kterého jsou dostupné všechny CRUD operace (*GET, POST, PUT, DELETE*). Pro přenos dat pomocí této architektury se používá výhradně textový formát JSON.

JSON je jeden z textových formátů souboru, který dokáže uchovat strukturovaná data ve formě slovníků a seznamů za použití primitivních datových typů. Slovníky se zde také nazývají JSON objekty. Pro označení typu těchto souborů se využívá koncovka „.json“.

JSON formát je také 1:1 převoditelný na formát YAML, který využívá stejných typů a objektů, nicméně v úspornější podobě. V rámci této práce bude JSON a YAML formát často využíván.

4.1.3 Tvorba webových aplikací

V této kapitole budou popsány a představeny vybrané technologie pro tvorbu klientské části webové aplikace.

HTML a CSS

Značkovací jazyk HTML je základním nástrojem pro tvorbu webové stránky. Skládá se z několika sémantických značek, které mají svá pravidla a atributy. Značky dohromady tvoří DOM (*Document Object Model*) [1, str. 32]. DOM je objektově orientovaná reprezentace HTML dokumentu. Webový prohlížeč zobrazuje DOM ve formě webové stránky koncovému uživateli.

Pro stylování objektů v DOM se využívájí kaskádové styly neboli CSS (*Cascading Style Sheets*). Existuje mnoho nástrojů, které rozšiřují funkce CSS a zlepšují tak čitelnost kódu a efektivitu vývoje webové stránky. Jednou z nejznámějších technologií, která se v dnešní době velmi využívá, je SASS (*Syntactically Awesome Style Sheets*). SASS patří mezi tzv. CSS preprocesory, které na základě zdrojového kódu vygenerují výsledné CSS za použití proměnných, funkcí a dalších nástrojů používaných v SASS. Alternativou k SASS je možné zmínit LESS, který má velmi podobné vlastnosti.

Javascript

Pro manipulaci a interakce s DOM dokumentu je využíván jazyk Javascript. Jedná se o skriptovací jazyk přinášející interaktivitu do webových stránek za pomocí specifických volání funkcí, které umí pracovat s DOM API. Javascriptový kód, který je připojen k HTML dokumentu, je vždy spouštěn ve webovém prohlížeči na straně klienta [2].

Během času se však Javascript stal i jazykem na straně serveru. Mezi typické představitele programovacích jazyků, které slouží pro programování serverových částí aplikací jsou např. Java, Python, C#, Ruby nebo GoLang. Tyto jazyky slouží pro programování webových serverů, jejichž hlavním cílem je vytvořit standardizované API. Vedle těchto jazyků však vznikla podpora runtime služby ve formě Node.js. Programy ve formě webových serverů jsou v Node.js psány právě v jazyku Javascript. Typický představitel čistě serverového frameworku v Javascriptu je Express.js. Nutno zmínit, že serverovými jazyky se tato práce dále zabývat nebude.

S přesunem Javascriptu do serverové části souvisí pojem *server-side rendering.* Jedná o poměrně novou technologii, která přišla současně s novými frameworky. Hlavní myšlenkou server-side renderingu je zkompletování výsledného HTML na straně serveru a poslat do webového prohlížeče již hotové HTML bez nutnosti spouštět Javascript na klientské části.

TypeScript

Javascript obecně nemá podporu pro datové typy. Všechny objekty v Javascriptu jsou odvozeny od elementů nacházejících se v DOM a definice vlastních typů není možná. V roce 2012 společnost Microsoft představila nový programovací jazyk TypeScript [3]. Jde o nadstavbu nad jazykem Javascript, která rozšiřuje základní Javascript právě o možnosti typování.

Použití TypeScriptu není nutnou podmínkou, nicméně typovaný zdrojový kód poskytuje vývojáři větší kontrolu nad celou aplikací.

4.1.4 Javascriptové knihovny a frameworky

V této kapitole budou popsány frontendové knihovny a frameworky skriptovacího jazyka Javascript. Jedná se o open-source technologie, jejichž hlavním cílem je zjednodušit a zefektivnit vývoj aplikací a zasadit tak charakter aplikace do moderního prostředí.

jQuery

První verze knihovny jQuery má počátky už v roce 2005 a v červnu 2006 byla vydána první verze 1.0 Alpha Release [4]. Na této knihovně je například postaven HTML a Javascript framework Bootstrap ve verzi 4 a předchozí. Nová verze Bootstrap 5 tuto knihovnu přestala používat.

Hlavním cílém této knihovny je usnadnit manipulaci s HTML DOM pomocí podpůrných funkcí. Tyto funkce pomáhají různě vytvářet a manipulovat s elementy DOM. Obsahuje také několik předdefinovaných animací či událostí, se kterými je dále možno pracovat.

V dnešní době se jQuery vyskytuje spíše ve starších projektech a aplikacích, protože jsou k dispozici mnohem efektivnější, výkonnější a modernější knihovny a technologie.

React

Open-source knihovna React vznikla v roce 2013 a byla vyvinuta společností Facebook [5]. Je zde využit koncept virtuálního DOM. V tomto konceptu je virtuální DOM synchronizován s DOM skutečným. React pro tuto synchronizaci využívá knihovnu ReactDOM. Aplikace vytvořené pomocí React knihovny jsou zpracovávané na klientské části, tj. ve webovém prohlížeči.

Základní komponentou jsou zde tzv. *React komponenty*. Každá komponenta může mít určité vlastnosti a je možné ji reprezentovat dvěma způsoby.

První způsob, jak vytvořit React komponentu, je pomocí funkcionálních komponent (*Function Component*). Ty jsou tvořeny pouze exportovanou výchozí funkcí, která dále může obsahovat React hooky (*useState, useEffect, useMemo* a podobně), další atributy nebo funkce.

Druhým způsobem vytvoření React elementu je použití předdefinované abstraktní třídy Component, která je součástí knihovny React (*Class Component*). Obdobně jako funkcionální komponenta může obsahovat atributy, funkce a hooky. Hooky jsou zde reprezentovány jako zděděné metody ze třídy Component (např. *componentDidMount* nebo *componentDidUpdate*).

Pro React existuje řada rozšiřujících knihoven, které dohromady tvoří základ pro tvorbu sofistikovaných aplikací. Jako příklad rozšiřující knihovny je možno uvést *react-router*, který se stará a směrování webové aplikace. Použití takového routeru znamená, že výslednou aplikaci bude možné procházet bez nutnosti načítat každou stránku znovu. Na této knihovně vznikl framework Next.js, který obsahuje řadu funkcionalit pro tvorbu komplexní aplikace. Framework Next.js bude popsán v další kapitole.

Vue.js

Vedle knihovny React byl vydán v prosinci v roce 2013 framework s názvem Vue.js [6]. Stejně jako u Reactu jde o framework pro vytváření aplikací na straně klienta a je založen na vlastních komponentách a virtuálním DOM. Liší se zejména syntaxí ve Vue.js šablonách.

Oproti React může být velkou výhodou, že ve zdrojovém kódu se nikde nemusí vyskytovat import deklarace. Všechny části Vue.js jsou automaticky importovány.

Podobně jako React má svoji nadstavbu v podobě frameworku *Nuxt.js*. Zde je však místo použití React knihovny využit framework Vue.js. Oproti Next.js se liší zejména v syntaxi specifické pro Vue.js.

Angular

První verzi open-source projektu Angular představila společnost Google v roce 2010 [7]. Stejně jako React nebo Vue.js je postaven na komponentách, které mají určité vlastnosti. Ve srovnání s Reactem je zde ovšem velký rozdíl v tom, jaká je cílová skupina využívající dotyčný framework či knihovnu. React, jakožto knihovnu, je možné použít i na menší části webové aplikace. Naproti tomu Angular je postaven jako komplexní řešení pro náročné a progresivní webové aplikace.

Progresivní webové aplikace jsou takové, které nabízí mnohem více možností, než klasické webové stránky. Jedná se například o možnost pracovat s aplikací se slabým či žádným internetovým připojením za využití kešovacích služeb. Takové aplikace umí uživateli posílat i tzv. *push notifikace*, které jsou svázány s operačním systémem [8].

Next.js

Protože React je knihovna, která nemá dostatečně sofistikované řešení tvorby komplexních a progresivních aplikací, v roce 2016 byl představen open-source framework Next.js, pro který je základem právě knihovna React [9]. Jedná se o framework sloužící k tvorbě komplexních a robustních aplikací pomocí React komponent.

Velkou výhodou Next.js je možnost poskytovat webové stránky pomocí server-side renderingu.

4.2 Vývoj webových aplikací

Vývoj webových aplikací je komplexní proces, do kterého se často započítává celý vývoj včetně grafického i funkčního návrhu a testování. Tato kapitola se bude zabývat výhradně vývojem z pozice programátora (vývojáře) webových aplikací.

4.2.1 Správa a verzování zdrojového kódu

Každá aplikace, která vyžaduje manuální vývoj ze strany vývojářů, by měla být z hlediska budoucího vývoje udržitelná. Jednak jde o samotný zdrojový kód, který by měl být přehledný, čitelný a měl by dodržovat určitá paradigmata a standardy. Za druhé jde o správu kódu z hlediska časově náročnějšího vývoje nebo v případě vývoje aplikace v týmu, kde se na vytváření aplikace podílí více programátorů. Jak řešit problém správy kódu, aniž by si jednotliví vývojáři upravovali kód navzájem? Na tuto otázku odpoví tato kapitola.

SVN

SVN (zkratka pro *Subversion*) je systém správy verzí pro vývoj softwaru, který byl vytvořen v roce 2000 společností CollabNet, Inc. [10]. Jedná se o open-source nástroj, který umožňuje vývojářům spravovat změny ve zdrojovém kódu v průběhu času.

Pomocí SVN mohou vývojáři sledovat různé verze projektu, spolupracovat s ostatními členy týmu a v případě potřeby se vrátit k dřívějším verzím projektu. SVN také umožňuje vývojářům pracovat na různých verzích projektu současně a později sloučit změny dohromady.

SVN ukládá různé verze projektu do centrálního úložiště, jedná se tak o centralizovaný software pro správu kódu. Vývojáři si mohou prohlédnout nejnovější verzi kódu, provést změny a poté tyto změny odeslat zpět do úložiště. SVN také poskytuje nástroje pro řešení konfliktů, které vznikají, když více vývojářů provádí změny ve stejném kódu současně.

Git

O pět let později v roce 2005 byl vytvořen verzovací systém Git. Autorem tohoto projektu je Linus Torvalds - tvůrce linuxového jádra neboli kernelu [11]. Jedná se o nejpopulárnější verzovací nástroj pro vývoj softwaru. Hlavní rozdíl oproti SVN je v tom, že se různé verze kódu udržují lokálně na uživatelském počítači ve formě větví.

V každé větvi se na základě úprav kódu vytvářejí *commity.* Commit je jednotka, která zachycuje stav kódu oproti předešlé změně. Commit mimo jiné také obsahuje autora a čas, kdy byl daný commit proveden [12].

Pro Git je typické slučování větví dohromady, standardně se jedná o větev *master* (případně *main*), do které se slévají ostatní větve obsahující určité logické celky aplikace. Vedle master větve může existovat i testovací větev, která slouží pro testovací účely aplikace.

Jednotlivé commity a větve se publikují do gitového repozitáře. V dnešní době se používá celá řada poskytovatelů pro správu gitových repozitářů, v dalších kapitolách budou zmíněny tři nejpoužívanější poskytovatelé, kterými jsou *GitHub*, *Bitbucket* a *GitLab*.

GitHub

GitHub je internetová služba, která nabízí po založení účtu možnost vytvářet vlastní privátní i veřejné repozitáře. Na GitHubu se nachází velké množství open-source knihoven, frameworků a softwarů různého druhu. V následnujícím seznamu jsou uvedeny gitové repozitáře se zdrojovým kódem již zmíněných softwarů a nástrojů:

- React.js - https://github.com/facebook/react,
- Vue.js - https://github.com/vuejs/vue,
- Next.js - https://github.com/vercel/next.js/,
- Linux kernel - https://github.com/torvalds/linux

V URL adrese repozítářů je možné si všimnout i organizace, která daný software vyvíjí. Každý repozitář má své určité vlastnosti, které se dají dále konfigurovat. GitHub mimo samotného poskytování repozitářů dále nabízí služby jako *Issue Tracker* (software pro sledování a řešení nalezených problémů) či automatické úlohy (*GitHub Actions*).

GitLab

Ve světě zdarma dostupných poskytovatelů repozitářů existuje již dnes také velmi využívaná služba, kterou je *GitLab*. GitLab nabízí prakticky stejné funkce jako GitHub, včetně automatických úloh zde nazývaných CI/CD, které budou vysvětleny v dalších kapitolách. Nicméně jsou zde k dispozici rozšiřující nastavení, která dodávají vývojářům softwaru specifické možnosti správy zdrojového kódu. Velký rozdíl oproti GitHubu je v tom, že GitLab je možné nainstalovat na vlastní server, tzv. *self-hosted* řešení. Netřeba tak používat veřejně dostupnou instanci, ale využít vlastní instanci schovanou například za VPN pro větší bezpečnost a ochranu zdrojového kódu aplikací.

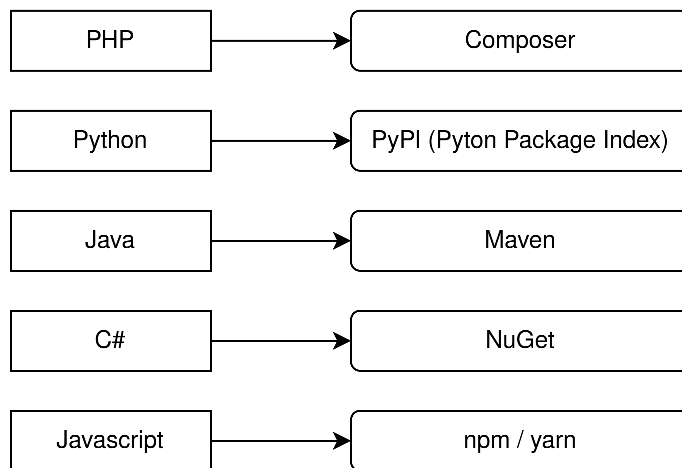
Bitbucket

Na závěr této kapitoly je představen Bitbucket od společnosti Atlassian [13]. Ve srovnání s GitHubem či GitLabem se jedná o placenou alternativu, která ale nabízí nativní možnost integrace například s Jirou - jedním z nejpoužívanějších Issue Trackerů. Bitbucket nabízí opět velmi podobné služby jako jeho alternativy, nicméně cílí spíše na větší firmy a korporáty.

4.2.2 Nástroje pro vývoj webových aplikací v jazyce Javascript

Aplikace v jazyce Javascript využívají v dnešní době řadu přednastavených a standardizovaných nástrojů, jejichž cílem je usnadnit vývoj a správu kódu. Běžnou prakcí ve světě programování je používání knihoven a frameworků. Právě z tohoto důvodu vznikly postupem času balíčkovací systémy, které uchovávají všechny dostupné knihovny a frameworky na jednom místě ve všech vydaných verzích. Tento koncept je společný pro mnoho programovacích jazyků.

Balíčkovací systémy charakterizuje repozitář, ve kterém jsou různé balíčky uloženy. Základní přehled balíčkovacích systémů je zobrazen na obrázku č. 1.



Obrázek 1: Balíčkovací systémy přiřazené k programovacím jazykům

Balíčkovací systémy se vyznačují vlastním CLI nástrojem, také specifickým pro každý jazyk. CLI nástroj je binární program, který je možné používat v terminálovém rozhraní neboli příkazové řádce.

Následující kapitoly se budou věnovat výhradně balíčkovacím systémům a nástrojům jazyka Javascript.

Npm

Npm (*Node Package Manager*) je balíčkovací systém s vlastním repozitářem. Repozitář npm balíčků je možné nalézet na adrese https://registry.npmjs.org/. Zde jsou uloženy takřka všechny používané open-source knihovny a balíčky pro Javascript. Obsahuje více než dva miliony různých knihoven, jedná se tak o největší softwarový repozitář na světě [14].

Používaným CLI nástrojem je zde již z názvu vyplývající nástroj npm. Pro inicializaci projektu využívající javascriptové knihovny slouží příkaz init. Na následujícím obrázku č. 2 je zobrazena práce s tímto nástrojem.

```
root@34e7bd5f4a40:/tmp/test# npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (test)
version: (1.0.0)
description: Test project
entry point: (index.js)
test command:
git repository:
keywords:
author: Dominik Resl
license: (ISC)
About to write to /tmp/test/package.json:

{
  "name": "test",
  "version": "1.0.0",
  "description": "Test project",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Dominik Resl",
  "license": "ISC"
}


Is this OK? (yes) yes
root@34e7bd5f4a40:/tmp/test#
```

Obrázek 2: Inicializace projektu pomocí npm

Při spuštění příkazu *init* se rozhraní zeptá na jméno, verzi, autora a další informace o nově založeném projektu. Příkaz init také dodá informaci o nově vytvořeném souboru s názvem package.json (JSON formát). Tento soubor obsahuje všechny klíčové informace o projektu včetně použitých knihoven a frameworků. Pro instalaci nového balíčku slouží příkaz install (zkráceně „i“), který nainstaluje určitou knihovnou se všemi potřebnými závislostmi. Jako parametr je možné také určit, jestli se má daný balíček nainstalovat globálně nebo jen jako součást lokálního projektu. Pro instalaci globálního balíčku slouží parametr „-g“, pokud se tento parametr vynechá, automaticky se balíček instaluje lokálně do projektu. Dalším používaným parametrem příkazu install je „--save“ a „--save-dev“. První varianta uloží balíček do souboru package.json do sekce balíčků nutných pro běh aplikace. Druhá možnost uloží balíček do sekce pouze pro vývojářské účely, typicky se jedná o balíčky pro testování či formátování kódu. Cílový adresář, kam se balíčky a jejich zdrojové soubory instalují, se jmenuje „node_modules“. Na obrázku č. 3 je znázorněno, jak probíhá instalace balíčku, konkrétně se jedná o knihovnu React.

```
root@34e7bd5f4a40:/tmp/test# npm install react

added 3 packages, and audited 4 packages in 893ms

found 0 vulnerabilities
root@34e7bd5f4a40:/tmp/test#
```

Obrázek 3: Instalace knihovny React pomocí npm

Yarn

Yarn je alternativou ke správci npm. Využívá i stejný package.json soubor pro konfiguraci projektu a stejně jako npm ukládá balíčky do adresáře „node_modules“. CLI nástrojem pro ovládání v konzoli je zde yarn, který používá podobné parametry jako npm.

Hlavním rozdílem těchto dvou nástrojů je v samotné logice instalace jednotlivých balíčků. Yarn dokáže instalovat balíčky paralelně, kdežto npm tyto závislosti instaluje jeden po druhém za sebou. Vzniká tak veliký rozdíl v časové náročnosti správce Yarn oproti npm.

Bootstrap

Počet knihoven, které slouží pro vývoj webových aplikací v Javascriptu je, jak bylo již zmíněno, přes dva miliony. Mezi těmito knihovnami existují však některé, které jsou více používány a upřednostňovány před druhými. Typickým představitelem je knihovna Bootstrap. Jde o HTML, CSS a Javascript framework sloužící pro vytvoření front-end části webové aplikace. Obsahuje několik již předprogramovaných komponent, ze kterých se sestává výsledná aplikace. Od verze 5 již zanikla závislosti na knihovně jQuery, která, jak už také bylo zmíněno, patří mezi méně používané a skoro již zastaralé Javascriptové technologie.

Bootstrap je známý také především díky svému tzv. grid systému, kdy je webová frontendová část aplikace rozdělena do dvanácti stejně širokých sloupců, které je možné využít pro vytvoření responzivního rozvržení. I přesto, že Bootstrap obsahuje velké množství komponent (boxů, navigací, akordeonů, tabulek a další), tak se jedná o celkem omezený framework a pro vytvoření struktury na základě určitého designu je potřeba provést velké úpravy, které přepíší výchozí chování. Jedná hlavně o úpravy ze strany CSS stylů a případného Javascriptu.

Tailwind CSS

Tailwind CSS, oproti Bootstrapu, je čistě CSS framework založený na komponentách a CSS třídách, které definují všechny dostupné vlastnosti v CSS. V kombinaci např. s React knihovnou tak lze definovat celý design webové aplikace čistě v Javascriptu bez použití externích CSS souborů. CSS soubor se vygeneruje až při kompilaci a načítají se tak pouze styly, které jsou využitý ve zdrojovém kódu. Tailwind CSS je použit pro tvorbu webové aplikace v praktické části.

Webpack

Pro správu a instalaci knihoven existují již popsané balíčkovací systémy. Nicméně pro vytvoření webové aplikace může být použito několik desítek těchto knihoven a je potřeba výsledný kód správně připravit na spuštění aplikace. Pro svázání zdrojového kódu z různých knihoven slouží Webpack. Webpack vytvoří jeden výsledný javascriptový soubor, který obsahuje všechny použité knihovny včetně samotného zdrojového kódu aplikace. Pro zefektivnění načítání takového souboru je možné Webpack nakonfigurovat tak, aby tento soubor rozdělil na více menších. Stejně tak umí pracovat i s CSS soubory, kdy se ve výsledku spojí všechny použité kaskádové styly do jednoho. Webpack také rozlišuje prostředí, na kterém daná aplikace běží. Lze tak například definovat, že pro nasazení do produkčního prostředí proběhne optimalizace a minifikace výsledných souborů.

Na obrázku č. 4 je ukázána práce s Webpackem, který vytváří produkční kód aplikace.

```
info  - SWC minify release candidate enabled. https://nextjs.link/swcmin
info  - Linting and checking validity of types
Browserslist: caniuse-lite is outdated. Please run:
  npx browserslist@latest --update-db
  Why you should do it regularly: https://github.com/browserslist/browserslist#browsers-data-updating
Browserslist: caniuse-lite is outdated. Please run:
  npx update-browserslist-db@latest
  Why you should do it regularly: https://github.com/browserslist/update-db#readme
info  - Creating an optimized production build
info  - Compiled successfully
info  - Collecting page data
[    ] info  - Generating static pages (0/11)react-i18next:: You will need to pass in an i18next instance by using initReactI18next
react-i18next:: You will need to pass in an i18next instance by using initReactI18next
react-i18next:: You will need to pass in an i18next instance by using initReactI18next
react-i18next:: You will need to pass in an i18next instance by using initReactI18next
react-i18next:: You will need to pass in an i18next instance by using initReactI18next
[=   ] info  - Generating static pages (9/11)react-i18next:: You will need to pass in an i18next instance by using initReactI18next
react-i18next:: You will need to pass in an i18next instance by using initReactI18next
info  - Generating static pages (11/11)
info  - Finalizing page optimization

Page                                       Size     First Load JS
┌ ● /                                      16.8 kB         172 kB
├   /_app                                  0 B             149 kB
├ ○ /404 (307 ms)                          823 B           156 kB
├ ○ /about                                 970 B           156 kB
└ λ /api/data                              0 B             149 kB
+ First Load JS shared by all              149 kB
  ├ chunks/framework-715a76d8b0695da7.js   45.7 kB
  ├ chunks/main-b398ddcfa13e8b14.js        31.7 kB
  ├ chunks/pages/_app-66e628f18f1fe6a2.js  70.7 kB
  ├ chunks/webpack-3433a2a2d0cf6fb6.js     819 B
  └ css/56db2681462b1edb.css               2.38 kB

λ  (Server)  server-side renders at runtime (uses getInitialProps or getServerSideProps)
○  (Static)  automatically rendered as static HTML (uses no initial props)
●  (SSG)     automatically generated as static HTML + JSON (uses getStaticProps)
```

Obrázek 4: Sestavení produkčního zdrojového kódu pomocí Webpacku

Je možné si všimnout, že v tomto případě vznikly celkem 4 javascriptové soubory a jeden CSS soubor se styly webové aplikace. Webpack poskytuje i informace o celkové velikosti souborů, které jsou vygenerovány.

4.3 Nasazování webových aplikací

Lokální vývoj webových aplikací byl popsán v předešlé kapitole. Nicméně takto hotovou webovou aplikace je třeba spustit na určité doméně v produkčním prostředí. Existuje mnoho technologií, který se zabývají problematikou nasazování webových aplikací. V následujících kapitolách budou popsány takové technologie, které jsou v současné době používány pro přesun aplikace do produkčního prostředí.

Pro přechod webové aplikace do produkčního prostředí se mění chování aplikace z hlediska nejrůznějších nastavení, které jsou zohledňovány v celém tomto procesu. Aplikace jsou tak spouštěny specializovanými nástroji, které zajišťují chod aplikace z hlediska vysoké dostupnosti, bezpečnosti či škálovatelnosti. Musejí být připravené na určitou zátěž z hlediska požadavků, které přicházejí na webový server.

Při HTTP požadavku na webovou stránku se přeloží název domény pomocí DNS služby na IP adresu serveru, na kterém běží webová aplikace. První technologií, která zajistí správnou komunikaci se serverem je webový server. Tato práce bude pracovat s webových serverem a load balancerem *NGINX*, který také funguje jako reverzní proxy server [15]. Správná konfigurace NGINX dokáže rozpoznat požadavek podle domény a na základě názvu domény přesměruje požadavek dále na určitý port, na kterém výsledná webová aplikace poslouchá.

U javascriptových aplikací je zvykem, že se spouští pomocí Node.js serveru, které standardně poslouchají na portu 3000, případně 3000 ikrementovaným o jednotky (v závislosti na počtu běžících aplikací). NGINX dokáže přesměrovat HTTP požadavek právě na tento port a Node.js server se postará o správné zobrazení webové aplikace.

Pokud je konfigurace NGINX nastavená na použití šifrovaného přenosu, výsledná aplikace bude dostupná přes HTTPS protokol. K nastavení šifrovaného přenosu se běžně používá certifikační autorita *Let's Encrypt*, která tento šifrovaný přenos zajistí pomocí protokolu SSL.

Popsané technologie výše je možné nakonfigurovat různými způsoby od čistě manuální konfigurace po automatizované kroky, které tento proces zajistí.

4.3.1 Tradiční nasazování

První možností, jak lze nasadit webovou aplikaci, je manuální instalace všech potřebných technologií a služeb, které jsou mezi sebou propojeny. Tyto nástroje jsou nainstalovány přímo na hostitelském serveru (hardwaru), který je připojen do sítě internetu.

Na jednom serveru je nainstalován například Node.js server, který spouští výslednou aplikaci. Zároveň je na stejném serveru nainstalován i NGINX, případně databáze a další služby potřebné pro běh aplikace (keš systémy a podobně).

V případě instalace více aplikací, které jsou napsané i v různých programovacích jazycích, je nutné mít tyto jazyky nainstalované na daném serveru. Může se jednat o Python, Javu či GoLang. Vše je nainstalováno na jednom stroji bez jakékoliv další vrstvy. Správa takto nasazených aplikací může být někdy velmi náročná a v případě nasazení nové aplikace je potřeba dávat pozor na jednotlivé úkony, které se provádějí.

V dnešní době se od tohoto přístupu odstupuje a přechází se na robustnější řešení ve formě virtualizací a kontejnerů.

4.3.2 Virtualizace

Velký posun v oblasti nasazování aplikací přinesla virtualizace operačních systémů. Jedná se o typ virtualizace, který umožňuje paralelní běh více operačních systémů na stejném hardwaru fyzického zařízení.

Tento druh virtualizace je možné využít pro nasazování webových aplikací, kdy jsou jednotlivé aplikace odděleny jednotlivými virtuálními stroji a jsou tak navzájem izolovány. Nicméně komunikace mezi virtualizovanými stroji je dostupná pomocí síťového rozhraní. Vytvoření nové aplikace znamená vytvoření nového virtuálního stroje, přičemž ostatní aplikace nejsou tímto zásahem nijak ovlivněny. Lze tak mít na jednom fyzickém stroji nainstalováno například několik aplikací různého druhu s různými verzemi databází, aniž by docházelo k nežádoucím konfliktům.

4.3.3 VPS a Cloud Computing

Virtuální privátní server (VPS) je typ webhostingové služby, která využívá virtualizační technologii k rozdělení fyzického serveru na více virtuálních serverů. Každý z těchto virtuálních serverů funguje jako nezávislý server s vlastním operačním systémem a úložištěm.

VPS hosting umožňuje uživatelům větší kontrolu nad prostředím serveru ve srovnání se sdíleným hostingem. U hostingu VPS mají uživatelé přístup k virtuálnímu serveru s administrátorskými právy, což znamená, že lze na takový server instalovat a spouštět vlastní softwarové aplikace.

S pojmem VPS úzce souvisí *Cloud* a *Cloud computing*. Jedná se o softwarový model, který je založen na poskytování služeb dostupné pro všechny uživatele skrz síť internet. Na tyto služby je možné se připojit například pomocí webového prohlížeče.

Takovou infrastrukturu lze vytvořit například pomocí open-source nástroje OpenStack. OpenStack umožňuje správu desítek a stovek virtuálních strojů, které mohou, ale nemusí, být navzájem propojeny. OpenStack má vlastní uživatelské webové rozhraní, pomocí kterého lze během krátké chvíle vytvořit virtuální stroj se spuštěnou webovou aplikací.

Jedním z poskytovatelů virtuálních privátních serverů je společnost Hetzner, jejichž VPS byl použit pro hostování cloudového prostředí webové aplikace, která je součástí této práce.

4.3.4 Kontejnerizace

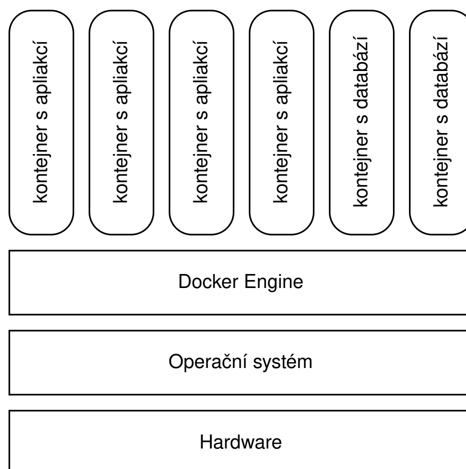
Kontejnerizace je zabalení výsledného programu spolu s knihovnami a závislostmi operačního systému potřebnými ke spuštění aplikace a vytvoření jediného spustitelného objektu (kontejneru) [16]. Výsledný kontejner dokáže běžet spolehlivě na jakékoli infrastruktuře. Tento druh virtualizace také velmi snižuje nároky na běžící aplikaci, protože spouštěný operační systém mnohdy nepřesahuje více než několik menších desítek MB a režie takového operačního systému je oproti klasickému virtuálnímu stroji velmi nízká.

Hlavní výhodou kontejnerizace spočívá v oddělení jednotlivých aplikací na úrovni kontejnerů, není tak nutné instalovat kompletně celý operační systém.

4.3.5 Docker

Nejznámější správce kontejnerů je v dnešní době Docker, který vznikl v roce 2013 [17]. Jde o software, který dokáže vytvářet spustitelné jednotky ve formě Docker kontejnerů. Pro spouštení kontejnerů se využívá Docker Engine.

Na obrázku č. 5 je vyobrazena architektura kontejnerizovaných aplikací v Dockeru.



Obrázek 5: Architektura Docker kontejnerů

Pro práci s kontejnery slouží CLI nástroj *docker*, pomocí kterého se ovládá kompletní architektura Dockeru. V současnosti je dostupný produkt Docker Desktop, který obaluje CLI nástroj grafickým rozhraním.

V dalších podkapitolách budou popsány nejdůležitější komponenty Docker infrastruktury a jejich vzájemná komunikace.

Docker obraz a repozitář

Nejzákladnější komponentou Docker architektury je obraz (*image*). Ten funguje jako šablona pro vytvoření kontejneru. Obraz v sobě obsahuje operační systém a zdrojový kód výsledné aplikace. Nejčastěji jsou založeny na linuxovém jádru, nicméně společnost Microsoft publikovala i obrazy, které jsou založené na Windows Serveru.

Docker obrazy se vytvářejí pomocí CLI nástroje a publikují do Docker registru (knihovny). Oficiální registr obrazů se jmenuje *Docker Hub* dostupný na internetové

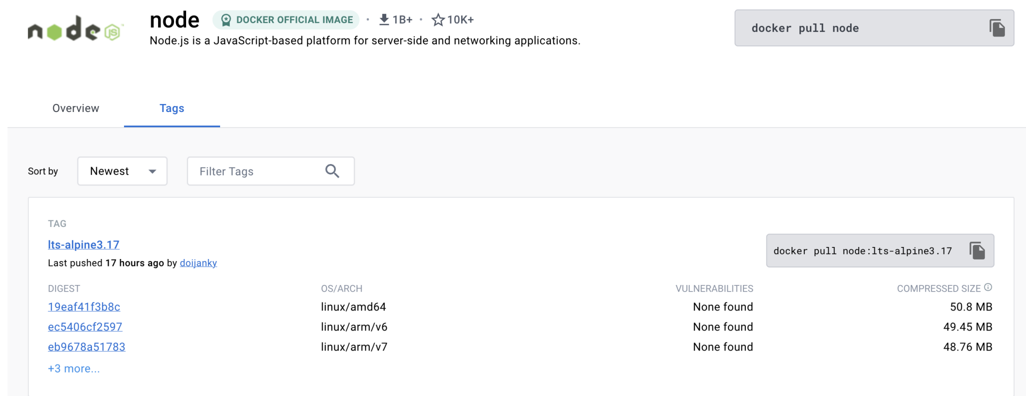
adrese `https://hub.docker.com`. Po vytvoření účtu je zdarma přístupný a je možné jej používat i pro publikování vlastních Docker obrazů.

Každý obraz má svůj vlastní štítek (*tag*), přičemž výchozí štítek se jmenuje „latest“. Takto je možné verzovat výsledné aplikace a jejich obrazy. Typicky se jedná o štítky s inkrementálním charakterem verzí doplněné případným názvem verze. U Docker štítků je dodržována určitá konvence s názvem „slim“, která značí, že se jedná o minifikovaný obraz s minimálním počtem předinstalovaných nástrojů. Jedná se tak často o čistý Linuxový systém bez dalších nepotřebných závislostí.

Na Docker Hubu je možné využít několik již přednastavených obrazů. V následujícím výčtu jsou uvedeny zdarma dostupné Docker obrazy včetně štítku a jejich velikostí.

- alpine (3.17.2) - 3,25 MB,
- ubuntu (22.04) - 28,17 MB,
- node (16.13.2-slim) - 60,75 MB.

Na obrázku č. 6 je zobrazena domovská stránka obrazu Node.js, kde je i mimo jiné možné nálezt příkaz, kterým lze stáhnout tento obraz.



Obrázek 6: Node.js na Docker Hubu

Docker CLI

Docker CLI je velmi komplexní a sofistikovaný nástroj pro práci s Dockerem a jeho infrastrukturou. Obsahuje příkazy pro přihlášení do registru, vytvoření vlastního obrazu, štítkování a publikaci obrazu, spouštění kontejneru za základě obrazu a další operace. Na následujícím obrázku č. 7 je zobrazena práce s operačním systémem Ubuntu spuštěném v Dockeru.

```
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s$ docker pull ubuntu:22.04
22.04: Pulling from library/ubuntu
d0a4bfa485d1: Pull complete
Digest: sha256:2adf22367284330af9f832ffefb717c78239f6251d9d0f58de50b86229ed1427
Status: Downloaded newer image for ubuntu:22.04
docker.io/library/ubuntu:22.04
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s$ docker run -it ubuntu:22.04 bash
root@2cce15e1cd90:/# ls /
bin  boot  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@2cce15e1cd90:/#
root@2cce15e1cd90:/#
```

Obrázek 7: Ukázka práce s Docker CLI

Parametr „-it“ způsobí automatický SSH přístup do kontejneru po jeho zapnutí. Dalším, velmi užitečným, parametrem je „-d“ (detach), který spustí kontejner na pozadí. Takový kontejner je pak možné ovládat pomocí Docker CLI, tedy tento kontejner vypnout, restartovat, zapnout, smazat a podobně.

Vytváření vlastních Docker obrazů

Princip vytváření obrazů se zdrojovým kódem webové aplikace je založen na souboru, který se jmenuje *Dockerfile*. Každá dockerizovaná aplikace obsahuje svůj vlastní Dockerfile, podle kterého se vytvoří finální obraz. Struktura tohoto souboru je zobrazena na obrázku č. 8.

```
1   FROM node:16.13.2-slim
2
3   WORKDIR /app
4   COPY package.json /app
5   RUN yarn install
```

Obrázek 8: Příklad souboru Dockerfile

První řádek obsahuje notaci `FROM`, která určuje, jaký obraz se zvolí pro definici výsledného obrazu. Řádek číslo 3 obsahuje příkaz `WORKDIR`, který říká, v jaké složce se bude pracovat (totožné s linuxovým příkazem „cd“). O řádek dále se využívá stěžejní příkaz `COPY`, který kopíruje soubory do výsledného obrazu. Na posledním pátém řádku se pak provádí instalace všech balíčků, které jsou definovány v souboru „package.json“.

Dockerfile obsahuje několik příkazů, kterými lze sestavit výsledný obraz tak, aby korektně spouštěl výslednou aplikaci. Pro iniciaci vytváření nového obrazu se používá příkaz „docker build“ a pro publikování „docker push“. Více se tomuto tématu bude věnovat praktická část.

Proměnné prostředí

Spuštění kontejneru podle daného obrazu provede příkaz „docker run“. Jeden z parametrů tohoto příkazu je právě obraz, podle kterého se má kontejner spustit. K dispozici je celá řada parametrů, která definují charakteristiku výsledného kontejneru. Uvnitř kontejneru je ale také možné definovat proměnné prostředí (*Environmental Variables*) s použitím parametru „-e“.

Na obrázku č. 9 je zobrazeno použití environmentální proměnné „`TEST_VAR`“, která nabývá hodnotu řetězce.

```
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s$ docker run -it -e TEST_VAR=Test1 ubuntu:22.04 bash
root@b76362218602:/# echo $TEST_VAR
Test1
root@b76362218602:/#
```

Obrázek 9: Příkaz „docker run“

Docker volumes

Při zastavení či odstranění kontejneru dochází ke ztrátě dat, které byly v rámci kontejneru vytvořeny. *Volume* je objekt v rámci Dockeru, který slouží k ukládání perzistentních dat v rámci vytvořeného kontejneru. Jako příklad je možné uvést databázový kontejner, u kterého je nezbytně nutné držet aktuální stav databáze i po zániku kontejneru. Bez využití volume by například po upgradu na novou verzi databázového stroje došlo ke smazání všech dat uložených v databázi. Ke správě těchto objektů slouží příkaz „docker volume“.

Síťování v Dockeru

Při spuštění nového kontejneru dojde zároveň k přiřazení IP adresy. V rámci Docker infrastruktury je k dispozici možnost vytvoření vlastních sítí, které jsou od sebe odděleny. U jednotlivých sítí je možné zadefinovat, o jaký typ sítě se jedná. Pro práci se síťovými rozhraními uvnitř Dockeru se používá příkaz „docker network“. Jednotlivé možnosti dostupné pomocí toho příkazu jsou ukázány na obrázku č. 10.

```
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s$ docker network help

Usage:  docker network COMMAND

Manage networks

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s$ docker network list
NETWORK ID     NAME      DRIVER    SCOPE
0b62c1bd6c88   bridge    bridge    local
2188596b71f6   host      host      local
32b01cf6e301   none      null      local
19bfb7d6db91   storm     bridge    local
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s$
```

Obrázek 10: Příkaz „docker network“

4.3.6 Docker Compose

Samotný Docker sice umožňuje efektivně oddělit jednotlivé aplikace a další podpůrné systémy od sebe, nicméně správa komplexnější infrastruktury může být pouze pomocí Docker CLI obtížná. Tato problematika se týká zejména budoucí správy a škálovatelnosti aplikací.

Z tohoto důvodu vznikl Docker Compose, což je nadstavba klasického Dockeru. Účelem Docker Compose je sjednotit kontejnery, které jsou navzájem v určitém vztahu. Typicky se jedná například o webovou aplikaci, databázi a keš server. Takové Docker kontejnery jsou nazývány v rámci Docker Compose jako služby a jsou nadefinovány ve speciálním souboru „docker-compose.yml“ ve formátu YAML.

Pro každou službu je specifikován obraz a případně další parametry, které udávají, jakým způsobem se má služba spustit. Definice tak může obsahovat například proměnné prostředí, sítě, volumes nebo port, na kterém služba běží.

Taková definice je ukázána na obrázku č. 11 v případě nasazení dvou databází - PostgreSQL a MySQL pomocí jednoho souboru.

```
1  version: '3.8'
2  services:
3    psql:
4      image: postgres:14.1
5      restart: always
6      environment:
7        - POSTGRES_USER=postgres
8        - POSTGRES_PASSWORD=postgres
9      ports:
10       - '5432:5432'
11     volumes:
12       - persistent_psql:/var/lib/postgresql/data
13   mysql:
14     image: mysql:8.0
15     cap_add:
16       - SYS_NICE
17     restart: always
18     environment:
19       - MYSQL_DATABASE=mysql
20       - MYSQL_ROOT_PASSWORD=mysql
21     ports:
22       - '3306:3306'
23     volumes:
24       - persistent_mysql:/var/lib/mysql
25 volumes:
26   persistent_psql:
27     driver: local
28   persistent_mysql:
29     driver: local
```

Obrázek 11: Ukázka souboru „docker-compose.yml“

Na hostitelském počítači tak stačí mít nainstalovaný pouze Docker a jeho potřebné závislosti. Stejně tak to platí i pro webové aplikace napsané v Node.js, případně v jiných programovacích jazycích.

Pro správu vytvořených kontejnerů se používá Docker Compose CLI, který má podobné vlastnosti jako Docker CLI. Na následujícím obrázku č. 12 je ukázána práce s Docker Compose CLI.

```
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/databases$ docker compose up -d
[+] Running 12/12
 ⠿ mysql Pulled
   ⠿ 8ff0c9d62930 Pull complete
   ⠿ 8c7ab176cd9d Pull complete
   ⠿ 3f6283edca86 Pull complete
   ⠿ 74489f3832cb Pull complete
   ⠿ 68fdb433cd6e Pull complete
   ⠿ ed704471ee10 Pull complete
   ⠿ 8017dfacae02 Pull complete
   ⠿ 9c01cb78ffdd Pull complete
   ⠿ aa238b7fd3ad Pull complete
   ⠿ 3a86d0b8c1cc Pull complete
   ⠿ 326b54668f8b Pull complete
[+] Running 5/5
 ⠿ Network databases_default             Created
 ⠿ Volume "databases_persistent_mysql"   Created
 ⠿ Volume "databases_persistent_psql"    Created
 ⠿ Container databases-psql-1            Started
 ⠿ Container databases-mysql-1           Started
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/databases$ docker compose ps
NAME                IMAGE              COMMAND                  SERVICE             CREATED             STATUS              PORTS
databases-mysql-1   mysql:8.0          "docker-entrypoint.s…"   mysql               7 seconds ago       Up 5 seconds        0.0.0.0:3306->3306/tcp, 33060/tcp
databases-psql-1    postgres:14.1      "docker-entrypoint.s…"   psql                7 seconds ago       Up 5 seconds        0.0.0.0:5432->5432/tcp
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/databases$
```

Obrázek 12: Použití Docker Compose CLI

V případě, že obrazy neexistují, Docker Compose automaticky tyto obrazy stáhne z Docker Hubu a následně je spustí. Zároveň jsou spuštěny na pozadí a pomocí příkazu „docker compose ps“ jsou spuštěné kontejnery zobrazeny se všemi dalšími informacemi. Mezi poskytnuté informace patří název kontejneru a použitého obrazu, jaký příkaz byl uvnitř kontejneru spuštěn, o jakou službu se jedná a na jakém portu je tato služba k dispozici. Docker také zobrazuje informaci o stavu kontejneru, v tomto případě jsou databázové kontejnery ve stavu „UP“, tedy že jsou spuštěné a připraveny pro produkční použití.

Pro správu jednotlivých aplikací je Docker Compose dostačující technologie. V omezené podobně také nabízí možnost škálovat jednotlivé aplikace a vytvořit nad nimi *Load Balancer*, který v případě výpadků nebo velkého počtu požadavků automaticky tyto požadavky směruje na instance dalších kontejnerů stejné aplikace pro rozložení případné zátěže.

Pokud se ale jedná o infrastrukturu, která se skládá například ze dvou VPS, na kterých mají být nasazené desítky aplikací různého typu včetně monitoringu těchto služeb, Docker Compose přestane být dostačujícím řešením. Docker Compose není možné nastavit tak, aby spravoval kontejnery, které jsou odděleny jednotlivými VPS. Právě z tohoto důvodu vznikly postupem času orchestrační nástroje a technologie, které umožňují takovou infrastrukturu spravovat. V další kapitole bude popsán jeden z představitelů těchto nástrojů, kterým je *Kubernetes*.

4.4 Kubernetes

Kubernetes - také označovám jako *k8s* - byl vytvořen v roce 2014 jako open-source správce a orchestrační nástroj kontejnerů společností Google. Pro vybudování tohoto projektu Google využil svých 15 let zkušeností v oblasti produkčního nasazování aplikací [18]. Kubernetes nabízí v oblasti nasazování webových aplikací několik výhod a specifických vlastností.

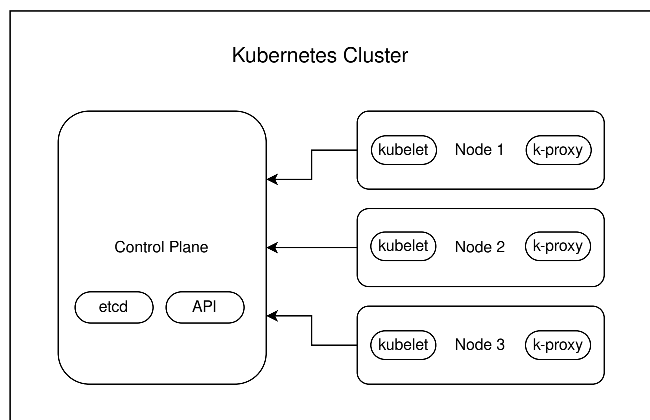
V základu poskytuje vyrovnávání zátěže jednotlivých kontejnerů pomocí Load Balancingu, i v případě velkého množství požadavků jsou tak nasazené aplikace dále stabilní a v plném provozu. Pomocí určitých definic lze popsat stav kontejneru, do kterého se má při nasazování daná aplikace dostat. Může se tak jednat o upgrade na novou verzi nebo například návrat k verzi starší. Jednoduše tak lze přepínat mezi verzemi aplikace v případě, že se v konkrétní verzi například vyskytne bezpečnostní chyba. Pro udržení konzistence jednotlivých kontejnerů Kubernetes využívá samozotavovací mechanizmus, který spravuje životní cyklus kontejnerů. Dokáže restartovat kontejnery, které selžou, případně je nahradit nebo úplně odstranit - vše v závislosti na konfiguraci nasazovací strategie.

Zároveň tak disponuje plnou správou přidělovaných prostředků ke kontejnerům, lze tak přidělit jednotlivým kontejnerům určité množství RAM či CPU. A v neposlední řadě také obsahuje plnou podporu pro bezpečnou komunikaci napříč kontejnery uvnitř infrastruktury. Nabízí tak možnost bezpečně ukládat obecně citlivé údaje jako jsou hesla, autorizační tokeny či SSH klíče. Uložené bezpečnostní údaje je možné aktualizovat napříč celou infrastrukturou.

4.4.1 Kubernetes komponenty

Kubernetes také umožňuje správu více oddělených VPS, které dohromady tvoří klastr (*cluster*). Klastr se skládá z hlavní řídící vrstvy (*Control Plane*) a pracovních serverů, které jsou na řídící vrstvu napojeny pomocí API. Tyto pracovní servery se označují jako uzly (*Node*). Na každém uzlu jsou nainstalovány služby *kubelet* a *k-proxy*, které zprostředkovávají komunikaci s hlavním serverem [19].

Základní schéma Kubernetes klastru je vyobrazeno na obrázku č. 13.



Obrázek 13: Kubernetes klastr

Kubernetes API (*api-server*) tvoří hlavní ovládací rozhraní řídící vrstvy, pomocí kterého je možné spravovat celý klastr.

4.4.2 Kubernetes objekty

Objekty v Kubernetes reprezentují persistentní data, která popisují stav klastru a aplikací v něm, včetně stavu kontejnerů, aplikačních prostředků a strategií, podle kterých jsou dané aplikace nasazovány. Vytvořeným objektům se také někdy říká prostředky (*resources*).

Pro popis objektů se využívá textový formát YAML (lze i JSON), který má standardizovanou strukturu, kterou je nutné dodržovat. Výsledný soubor se označuje jako *manifest.* Mezi povinné klíče patří:

- *apiVersion* - definuje, jaká verze Kubernetes API se má použít,
- *kind* - druh objektu, který se má vytvořit,
- *metadata* - popis dat, která jednoznačně identifikují vytvořený objekt (název, ID),
- *spec* - popis stavu vytvořeného objektu.

Přesná struktura definice objektu se liší na základě druhu objektu, který má být vytvořen. Nicméně existují společné atributy, mezi které patří *labels* a *selector*, které dodávají objektu štítky identifikující vlastnost objektu. Tyto štítky mohou být libovolně zvoleny a je možné vytvářet vlastní. Jako příklad je možné uvést klíč „environment“ s hodnotou „production“ nebo s hodnotou „test“.

Mezi další atributy patří anotace (klíč *annotations*). Ty představují další konfiguraci objektu, typicky se jedná o konfiguraci specifickou pro nasazovanou aplikaci.

Pro práci s Kubernetes objekty se používá konzolový nástroj *kubectl*, který obsahuje všechny potřebné funkce ke správě objektů v klastru. Pro vytváření a aplikování objektů slouží příkaz „apply“. Vedle tohoto příkazu jsou k dispozici dále příkazy pro odstranění objektu či jeho aktualizaci. Dohromady je k dispozici přes čtyřicet různých konfiguračních příkazů tohoto nástroje [20].

Pro správnou funkčnost připojení je nutné mít vytvořený konfigurační soubor, který se nazývá *kubeconfig.* Tento soubor obsahuje autorizační token a další potřebné údaje k připojení se na řídící vrstvu. Pomocí parametru „--kubeconfig“ příkazu *kubectl* je možné definovat cestu k tomuto souboru.

V následujících podkapitolách budou představeny nejpoužívanější typy objektů a jejich využití.

Jmenný prostor

Prvním představeným objektem je jmenný prostor (*namespace*), který poskytuje mechanismus izolace jednotlivých prostředků v rámci klastru. Lze tak přiřazovat jmenné prostory objektům a roztřídit je do logických celků, například podle druhu projektu. Pouze prostředky v rámci stejného jmenného prostoru jsou navzájem viditelné a mohou spolu komunikovat.

Ve výchozím nastavení existují čtyři základní jmenné prostory - „default“, „kube-system“, „kube-node-lease“ a „kube-public“. Ve jmenných prostorech s prefixem „kube-“ jsou nainstalovány všechny důležité součásti pro správný běh Kubernetes klastru. Jmenný prostor „default“ je ve výchozím stavu zcela prázdný.

Ne všechny objekty je však možné zařadit do určitého jmenného prostoru. Existují objekty, které jsou viditelné ve všech jmenných prostorech. Na následujícím obrázku 14 jsou tyto objekty zobrazeny.

Pro zobrazení dostupných prostředků je použit nástroj *kubectl* s konkrétními parametry.

```
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/kubernetes$ kubectl --kubeconfig=master-kubeconfig api-resources --namespaced=false
NAME                              SHORTNAMES   APIVERSION                             NAMESPACED   KIND
componentstatuses                 cs           v1                                     false        ComponentStatus
namespaces                        ns           v1                                     false        Namespace
nodes                             no           v1                                     false        Node
persistentvolumes                 pv           v1                                     false        PersistentVolume
mutatingwebhookconfigurations                  admissionregistration.k8s.io/v1        false        MutatingWebhookConfiguration
validatingwebhookconfigurations                admissionregistration.k8s.io/v1        false        ValidatingWebhookConfiguration
customresourcedefinitions         crd,crds     apiextensions.k8s.io/v1                false        CustomResourceDefinition
apiservices                                    apiregistration.k8s.io/v1              false        APIService
tokenreviews                                   authentication.k8s.io/v1               false        TokenReview
selfsubjectaccessreviews                       authorization.k8s.io/v1                false        SelfSubjectAccessReview
selfsubjectrulesreviews                        authorization.k8s.io/v1                false        SelfSubjectRulesReview
subjectaccessreviews                           authorization.k8s.io/v1                false        SubjectAccessReview
clusterissuers                                 cert-manager.io/v1                     false        ClusterIssuer
certificatesigningrequests        csr          certificates.k8s.io/v1                 false        CertificateSigningRequest
flowschemas                                    flowcontrol.apiserver.k8s.io/v1beta2   false        FlowSchema
prioritylevelconfigurations                    flowcontrol.apiserver.k8s.io/v1beta2   false        PriorityLevelConfiguration
ingressclasses                                 networking.k8s.io/v1                   false        IngressClass
runtimeclasses                                 node.k8s.io/v1                         false        RuntimeClass
podsecuritypolicies               psp          policy/v1beta1                         false        PodSecurityPolicy
clusterrolebindings                            rbac.authorization.k8s.io/v1           false        ClusterRoleBinding
clusterroles                                   rbac.authorization.k8s.io/v1           false        ClusterRole
priorityclasses                   pc           scheduling.k8s.io/v1                   false        PriorityClass
csidrivers                                     storage.k8s.io/v1                      false        CSIDriver
csinodes                                       storage.k8s.io/v1                      false        CSINode
storageclasses                    sc           storage.k8s.io/v1                      false        StorageClass
volumeattachments                              storage.k8s.io/v1                      false        VolumeAttachment
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/kubernetes$
```

Obrázek 14: Kubernetes prostředky viditelné v každém jmenném prostoru

Na obrázku č. 15 jsou zobrazeny naopak prostředky, které jsou zařazeny do jmenných prostorů.

```
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/kubernetes$ kubectl --kubeconfig=master-kubeconfig api-resources --namespaced=true
NAME                        SHORTNAMES   APIVERSION                     NAMESPACED   KIND
bindings                                 v1                             true         Binding
configmaps                  cm           v1                             true         ConfigMap
endpoints                   ep           v1                             true         Endpoints
events                      ev           v1                             true         Event
limitranges                 limits       v1                             true         LimitRange
persistentvolumeclaims      pvc          v1                             true         PersistentVolumeClaim
pods                        po           v1                             true         Pod
podtemplates                             v1                             true         PodTemplate
replicationcontrollers      rc           v1                             true         ReplicationController
resourcequotas              quota        v1                             true         ResourceQuota
secrets                                  v1                             true         Secret
serviceaccounts             sa           v1                             true         ServiceAccount
services                    svc          v1                             true         Service
challenges                               acme.cert-manager.io/v1        true         Challenge
orders                                   acme.cert-manager.io/v1        true         Order
controllerrevisions                      apps/v1                        true         ControllerRevision
daemonsets                  ds           apps/v1                        true         DaemonSet
deployments                 deploy       apps/v1                        true         Deployment
replicasets                 rs           apps/v1                        true         ReplicaSet
statefulsets                sts          apps/v1                        true         StatefulSet
localsubjectaccessreviews                authorization.k8s.io/v1        true         LocalSubjectAccessReview
horizontalpodautoscalers    hpa          autoscaling/v2                 true         HorizontalPodAutoscaler
cronjobs                    cj           batch/v1                       true         CronJob
jobs                                     batch/v1                       true         Job
certificaterequests         cr,crs       cert-manager.io/v1             true         CertificateRequest
certificates                cert,certs   cert-manager.io/v1             true         Certificate
issuers                                  cert-manager.io/v1             true         Issuer
leases                                   coordination.k8s.io/v1         true         Lease
endpointslices                           discovery.k8s.io/v1            true         EndpointSlice
events                      ev           events.k8s.io/v1               true         Event
ingresses                   ing          networking.k8s.io/v1           true         Ingress
networkpolicies             netpol       networking.k8s.io/v1           true         NetworkPolicy
poddisruptionbudgets        pdb          policy/v1                      true         PodDisruptionBudget
rolebindings                             rbac.authorization.k8s.io/v1   true         RoleBinding
roles                                    rbac.authorization.k8s.io/v1   true         Role
csistoragecapacities                     storage.k8s.io/v1beta1         true         CSIStorageCapacity
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/kubernetes$
```

Obrázek 15: Kubernetes prostředky zařazené do jmenných prostorů

Některé prostředky mají také vlastní zkratku, kterou je možné používat pro práci s těmito objekty. U jmenného prostoru se jedná o zkratku „ns“.

Pod

Pod charakterizuje nejmenší jednotku v rámci nasazovacího procesu. Reprezentuje jeden či několik kontejnerů, které mohou mít sdílené disky nebo sítě. Manifest podu se skládá z několika klíčů, z nichž nejdůležitější je pole „containers“, který definuje kontejnery se jménem, použitým obrazem a případnými dalšími atributy. Podu lze nastavit další parametry, které zajišťují vyhovující chování aplikace. Pokud se změní a aplikuje nová definice stejného podu, Kubernetes vytvoří pod nový a neaktuální se tímto podem nahradí, nedochází tak k aktualizaci již stávajícího.

Pod se může nacházet celkem ve třech stavech. Pokud se kontejner spouští a probíhají operace nezbytné ke spuštění kontejneru, pod se nachází ve stavu „Waiting“. Pokud požadované operace proběhnou úspěšně, pod se dostane do stavu „Running“, který indikuje, že start kontejneru proběhl bez komplikací a je připraven k použití. Pokud dojde při inicializaci kontejneru k nějaké chybě, přiřadí se stav „Terminated“ [21]. Kubernetes poskytne informace o důvodu selhání startu kontejneru.

Na základě stavu lze také definovat, jakým způsobem se má Kubernetes snažit restartovat kontejnery. Pomocí klíče „restartPolicy“ lze definovat hodnoty „Always“, „onFailure“ nebo „Never“. Výchozí hodnota je nastavena na „Always“.

Pro pod lze definovat dále inicializační kontejnery, které jsou spuštěny ještě před finálním kontejnerem. Tyto kontejnery obsahují nástroje či instalační skripty pro správný běh výsledné aplikace. Typicky se jedná například o nastavení správných práv souborů. Pokud dojde k selhání inicializačního kontejneru, na základě „restartPolicy“ se provedou další kroky.

Pro práci s pody pomocí *kubectl* nástroje lze využívat zkratku „po“.

Služba

Služba je v rámci Kubernetes jednotka, která vystavuje pod na určitém portu do sítě v rámci klastru. Definuje cílový port (na kterém běží aplikace v rámci podu) a port, na kterém bude služba dostupná s výslednou aplikací. Pomocí selektoru lze definovat přesně daný pod, pro který se má služba vytvořit. Zamezí se tak případnému konfliktu, pokud by běželo více aplikací na stejném portu.

Replikační sada

Vytvořené pody lze v Kubernetes škálovat. Hlavním účelem replikační sady (*ReplicaSet*) je zajistit toto škálování pomocí replik. Replikační sada se stará o udržení replikovaných podů ve stavu „Running“. Definice manifestu se podobá definici podu, nicméně je rozšířená o možnost nastavení počtu replik pomocí klíče „replicas“ s celočíselnou hodnotou.

Nasazovací objekt

Pro správu jednotlivých aplikací by byla replikační sada dostačující, nicméně Kubernetes nabízí ještě jednu vrstvu, která nabízí sofistikovanější řešení při pohledu na tzv. deklarativní aktualizaci, kterým je nasazovací objekt (*Deployment*). Tento objekt vytváří na pozadí replikační sady a kontroluje, zda jsou všechny pody ve spuštěném stavu. Při vynucení nové definice nasazení se vytvoří nové replikační sady a čeká se na jejich spuštění. Pokud se nové

repliky spustí, staré se odstraní. Každé vytvoření nového setu replik zároveň aktualizuje revizi daného nasazení.

Definice nasazovacího manifestu obsahuje klíč „replicas“ a „strategy“, který udává strategii nasazovacího procesu. Pokud je nastavená strategie na hodnotu „RollingUpdate“, Kubernetes při nasazení nové verze aplikace (nebo restartování nasazení) umožní současného běhu více verzí a zajistí tak dostupnost aplikace i během přechodu na novou verzi aplikace. Tento druh nasazování je označován jako *proporcionální škálování* [22].

Ingress

Nasazené aplikace v podech jsou dostupné pomocí Kubernetes služeb na určitém portu v rámci klastru. Pokud má však být aplikace dostupná přes webový prohlížeč, je nutné vystavit proxy server na portu 80 (případně 443). Pro tyto účely slouží Ingress. Ingress namapuje existující službu na určitou doménu a postará se o komunikaci pomocí HTTP protokolu (případně HTTPS). Důležitou součástí pro správné fungování je nutné mít nainstalován *Ingress Controller*, který celou tuto problematiku řeší. Mezi nejpoužívanější kontrolery patří *ingress-nginx*, využívající NGINX proxy server.

Tajná data

Již popsané objekty jsou dohromady schopné efektivně nasadit webovou aplikaci. V případě, že aplikace obsahuje citlivá data (hesla či tokeny), Kubernetes tato data zapouzdří do tajného objektu *Secret*. Tento objekt může být následujících typů [23]:

- Opaque - libovolně zadaná data,
- kubernetes.io/service-account-token - token pro Kubernetes účet,
- kubernetes.io/dockercfg - konfigurační soubor „dockercfg“ pro Docker,
- kubernetes.io/dockerconfigjson - konfigurační soubor „config.json“ pro Docker,
- kubernetes.io/basic-auth - údaje pro HTTP Basic Auth autentifikaci,
- kubernetes.io/ssh-auth - uložení SSH klíče,
- kubernetes.io/tls - data pro TLS zabezpečení,
- bootstrap.kubernetes.io/token - pro uložení speciálního tokenu.

V definici nasazení je možné odkazovat na určitá tajná data pomocí názvu, lze tak například předat přihlašovací token do aplikace pomocí proměnných prostředí bez přímého zadávání citlivých údajů, které by bylo možné zneužít.

Role v klastru

Pro práci s Kubernetes klastrem se využívá nástroj *kubectl*, který, jak již bylo zmíněno, používá pro připojení do klastru konfigurační soubor kubeconfig. Ten je napojen na určitý Kubernetes služební účet (*Service Account*). Pomocí role klastru (*Cluster Role*) a vazby rolí v klastru (*Cluster Role Binding*) lze omezit práva jednotlivým účtům.

Pro vytvoření omezujících pravidel se používá role klastru, která má definováno, na jaké prostředky se vztahuje a jaké operace s nimi jsou povoleny. Mezi operace, které je možné omezit, patří *create*, *delete*, *get*, *list*, *patch*, *update* a *watch* [24].

4.5 CI/CD

Pro vývojáře je velmi důležité, aby se většinu času věnovali vývoji dané aplikace a nezabývali se zbytečně problematikou nasazování. Nasazování pomocí Kubernetes nepřináší pro programátora příliš velkou režii, nicméně při vyskytnutí problému může být jeho řešení obtížné. Pro snížení rizika takového problému právě existuje nasazovací strategie CI/CD (*Continuous Integration a Continuous Delivery/Continuous Deployment*).

Jedná se o automatizační metodu, jejíž cíl je automaticky otestovat a nasadit aplikace do produkčního prostředí. K tomuto procesu zpravidla dochází, jakmile vývojář publikuje změny do gitového repozitáře.

4.5.1 Continuous Integration

První částí automatizační metody je *Continuous Integration*. V tomto úseku automatizačního procesu dochází k otestování aplikace. Pro každý programovací jazyk existují nástroje, které jsou už připravené na vytváření specifických testů pro danou aplikaci. V tomto kroku dochází ke kontrole, že změny zdrojového kódu, které vývojář provedl, nijak negativně neovlivní běh aplikace. Kontrola se může týkat jak syntaxe zdrojového kódu, tak i pouštění aplikačních testů.

4.5.2 Continuous Delivery

V tomto kroku procesu dochází k zabalení už otestované aplikace do balíčku, která je možná nasadit do produkčního prostředí (například binární soubor či Docker obraz). Nasazení může probíhat manuálním zásahem nebo za použití dalšího kroku, kterým je *Continuous Deployment*.

4.5.3 Continuous Deployment

Tento krok je automaticky spuštěn (případně ručně vynucen), jakmile se úspěšně dokončí předchozí krok. Zde dochází k automatickému nasazení aplikace pomocí určitých nasazovacích skriptů. Tato práce bude využívat k nasazovacímu procesu Kubernetes API pomocí příkazu *kubectl*.

4.5.4 GitLab CI/CD

Pro vytvoření procesu CI/CD existuje celá řada open-source nástrojů, kterou je možné využít. GitLab ve výchozím nastavení obsahuje plnou podporu pro vytvoření tohoto procesu pomocí tzv. *pipeline*. Pipeline obsahuje další podprocesy (*Jobs*), které obsahují určité kroky a skripty k úspěšnému dokončení celého procesu. Pokud některý z těchto podprocesů selže, celá pipeline se zastaví a čeká se na opravu aplikace.

Pro popis pipeline se používá speciální soubor „.gitlab-ci.yml“ nacházející se typicky v kořenovém adresáři gitového repozitáře [25]. Při tvorbě definice CI/CD lze používat proměnné prostředí, případně proměnné definované přímo v rámci GitLab repozitáře.

Při správném nastavení celého CI/CD tak vývojář pouze publikuje změny do určité větve v repozitáři a během několika minut je aplikace nasazena do produkčního prostředí, aniž by musel provádět další kroky. CI/CD může být omezeno pouze na určité větve, lze tak například při publikování změn do testovací větve automaticky nasadit změny pouze do testovacího prostředí (pokud existuje).

5 Praktická část

Praktická část se bude věnovat vytváření samotné webové aplikace za využití balíčkovacího systému yarn, frameworku Next.js a vše zkompletované pomocí Webpacku. Zdrojový kód bude udržován v GitLab repozitáři za využití několika větví.

Hotová aplikace bude ukazovat aktuální vytížení Kubernetes klastru. Tato data bude získávat z Kubernetes API pomocí služebního účtu s omezenými právy vytvořeného pro tyto účely. V rámci klastru bude vytvořen testovací a produkční jmenný prostor s testovací a produkční verzí aplikace. Veškeré tokeny a přístupové údaje budou zapouzdřeny do tajných dat, aby se zabránilo úniku citlivých údajů. Napříč celou infrastrukturou budou také využity proměnné prostředí, pomocí kterých budou citlivá data přenášena.

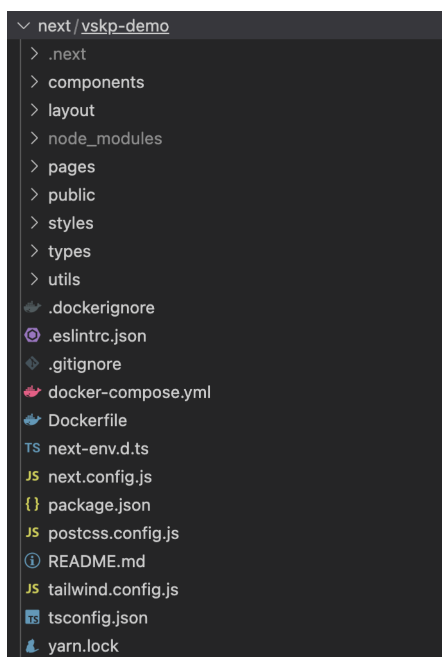
5.1 Vývoj Next.js aplikace

Tato kapitola se bude věnovat vývoji Next.js aplikace s názvem „vskp-demo“.

5.1.1 Struktura aplikace

Pro vygenerování struktury projektu byl použit nástroj *yarn* s nainstalovaným Next.js frameworkem, Tailwind CSS a dalšími potřebnými závislostmi. Při instalaci balíčků zároveň vznikl adresář „node_modules“ se zdrojovým kódem všech nainstalovaných balíčků. V kořenové složce Next.js aplikace se nachází řada konfiguračních souborů, které budou popsány v dalších kapitolách. Vedle těchto souborů jsou adresáře se samotným zdrojovým kódem výsledné aplikace. Tato struktura je vyobrazena na obrázku č. 16.

Obrázek č. 17 vyobrazuje výsledný soubor „package.json“ se všemi použitými závislostmi.



Obrázek 16: Struktura Next.js webové aplikace

Obrázek 17: Soubor „package.json“

Konfigurace aplikace

Hlavním konfiguračním souborem v každé Next.js aplikaci je „next.config.js“ obsahující nezbytné parametry ke spuštění projektu. Jelikož je v aplikaci využit zároveň framework Tailwind CSS, v kořenovém adresáři se nachází speciální konfigurační soubor „tailwind.config.js“. Na obrázku č. 18 je tento soubor zobrazen.

```
1  module.exports = {
2    content: [
3      "./pages/**/*.{js,ts,jsx,tsx}",
4      "./components/**/*.{js,ts,jsx,tsx}",
5      "./layout/**/*.{js,ts,jsx,tsx}",
6    ],
7    theme: {
8      fontSize: {
9        base: ['15px', '26px'],
10       'h1': ['53px', '68px'],
11       'h2': ['40px', '50px'],
12       'h3': ['30px', '40px'],
13       'h4': ['22px', '27px'],
14       'h5': ['17px', '28px'],
15       'h6': ['15px', '25px']
16     },
17     colors: {
18       primary: '#101f2f',
19       white: '#fff',
20       'white-transparent': 'rgba(255, 255, 255, 0.6)',
21       green: '#1d7a16',
22     },
23     extend: {},
24   },
25   plugins: [],
26 }
```

Obrázek 18: Soubor „tailwind.config.js“

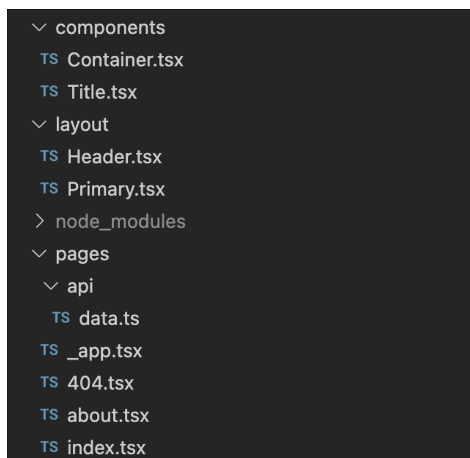
Proměnné a parametry definované v tomto souboru jsou dostupné napříč celou Next.js aplikací.

V souboru „package.json“ jsou dále definované speciální skripty („dev“, „build“, „start“ a „lint“). Skript „dev“ slouží pro spuštění lokálního vývoje, v tomto režimu jsou při změně zdrojového kódu veškeré změny ihned zkompilovány a projeveny. Skript „build“ vytvoří výsledný optimalizovaný zdrojový kód, který je připraven na nasazení do produkčního prostředí. Příkaz „start“ slouží ke spuštění aplikace v produkčním módu. Posledním dostupným skriptem je „lint“, který provede kontrolu, zda se ve zdrojovém kódu nevyskytují syntaktické chyby. Tuto kontrolu je vhodné provést před každým spuštěním příkazu „build“.

Pro spuštění skriptu se používá nástroj *yarn*, například pro spuštění kontroly zdrojového kódu se využije příkaz „yarn lint“.

Komponenty

Adresáře „components“, „layout“ a „pages“ obsahují komponenty, které slouží pro definování prvků na klientské části. Tyto komponenty jsou psány pomocí jazyka TypeScript a speciální syntaxe *JSX*, která umožňuje psát HTML elementy uvnitř Javascript souborů. Koncovka těchto souborů je ve výchozím nastavení „.jsx“, nicméně ve spojení s TypeScriptem je používána koncovka „.tsx“. Struktura komponent je dále vidět na obrázku č. 19.



Obrázek 19: Struktura Next.js komponent

V adresáři „pages“ se nacházejí soubory, pro které framework Next.js automaticky vytvoří URL, na kterých budou stránky dostupné. Příkladem je zdrojový kód stránky „About“ na obrázku č. 20.

```
1   import { NextPage } from "next";
2   import Head from "next/head";
3   import Header from "layout/Header";
4   import Title from "components/Title";
5
6

7   interface AboutProps {
8     environment: string;
9   }
10
11  const About: NextPage<AboutProps> = ({
12    environment
13  }) => {
14    return (
15      <>
16        <Head>
17          <title>About this project</title>
18        </Head>
19        <Header environment={environment}>
20          <Title className="font-bold max-w-5xl mx-auto" as="h1" content="About this project" />
21          <p>This project was made with purpose to make an introduction to running Next.js application in Kubernetes
22          <p>It is used for the presenting of the main goal of University Qualification Thesis 2022/23, FIM UHK.</p>
23        </Header>
24      </>
25    )
26  }
27
28  export const getServerSideProps = async({ locale }: { locale: any }) => {
29    return {
30      props: {
31        environment: process.env.ENVIRONMENT
32      },
33    };
34  }
35
36  export default About;
```

Obrázek 20: Soubor „about.tsx“

5.1.2 Aplikace, API a proměnné prostředí

Pro zajištění získání dat z Kubernetes klastru je využito Kubernetes API. Logika zpracování dat z tohoto API se nachází v souboru „pages/api/data.tsx“. Pro spojení komunikace byl použit nástroj *Axios*, pomocí kterého se provede požadavek na získání dat. Způsob volání tohoto požadavku je zobrazen na obrázku č. 21.

```
13  const cluster = {
14    name: 'local-config',
15    server: 'https://49.12.186.221:6443',
16    cert: process.env.CLUSTER_CERT || ''
17  };
18
19  const user = {
20    name: process.env.API_USER,
21    token: process.env.API_USER_TOKEN
22  };
23
24  const httpsAgent = new https.Agent({
25    rejectUnauthorized: false,
26    cert: b64_to_utf8(cluster.cert)
27  });
28
29  const instance = axios.create({ httpsAgent });
30
31  const getK8sData = async(node: '01' | '02' | string = '01') => {
32    const { data } = await instance.get(`${cluster.server}/api/v1/nodes/hetzner-eu-central-compute-${node}/proxy/stats/summary`, {
33      headers: {
34        "Authorization" : `Bearer ${user.token}`,
35        'Access-Control-Allow-Origin': '*',
36      }
37    })
38    return data;
39  }
40
41  const handler = async(
42    req: NextApiRequest,
43    res: NextApiResponse<Data>
44  ) => {
45    // @ts-ignore
46    const { node } : { node: string } = req.query;
47    const data = await getK8sData(node);
48    res.status(200).json(data)
49  }
```

Obrázek 21: Volání Kubernetes API v Next.js aplikaci

Jelikož aplikace využívá Kubernetes API, přístupové údaje pro komunikaci s tímto API jsou uloženy v proměnných prostředí. Konkrétně se jedná o „CLUSTER_CERT“, „API_USER“ a „API_USER_TOKEN“. Tyto údaje tak nejsou přímo ve zdrojovém kódu, ale je možné je definovat pomocí exportování proměnné. Stejným způsobem je definován například řetězec v proměnné „ENVIRONMENT“, který ukazuje, na kterém prostředí je aplikace spuštěna.

Stěžejní je metoda „getK8sData“, která volá koncový bod s názvem pracovního serveru. Návratovým typem této metody je vlastní typ „K8sDataType“, který odpovídá datům získaného z Kubernetes API.

Zajímavostí je, že Kubernetes API vrací údaje o zatížení procesoru ve dvou proměnných - „usageCoreNanoSeconds“ a „usageNanoCores“. Hodnoty těchto proměnných nabývají například:

- usageCoreNanoSeconds - 18448001302166896,
- usageNanoCores - 967412318.

Výpočet procentuálního zatížení se provádí na základě následujícího vzorce [26]:

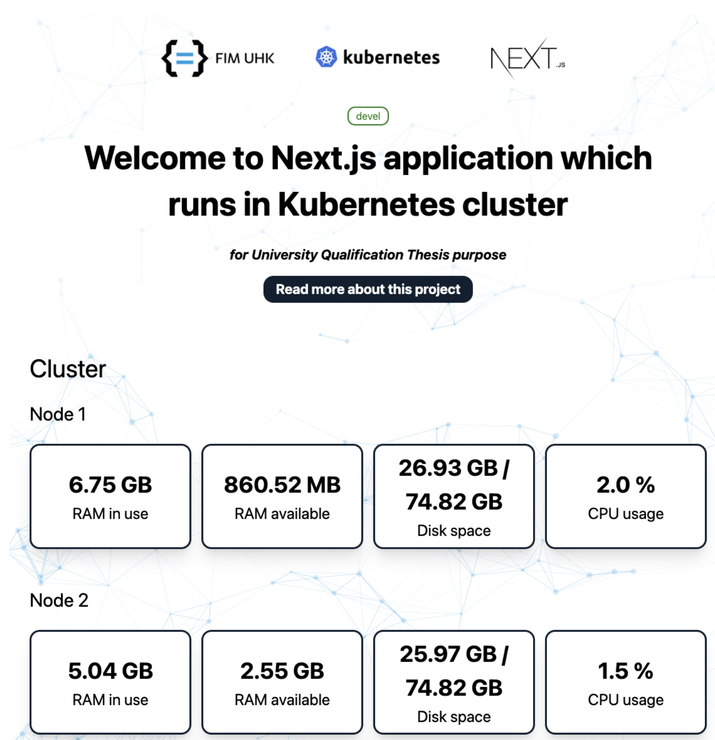
$$CPUusage = \frac{usageCoreNanoSeconds}{usageNanoCores \cdot 1e9}$$

V Javascriptu vypadá výpočet podle obrázku č. 22.

```
`${((nodeData?.node?.cpu?.usageCoreNanoSeconds / (nodeData?.node?.cpu?.usageNanoCores * 1e9)) * 100).toFixed(1)} %`
```

Obrázek 22: Výpočet procentuálního zatížení CPU

Výsledná aplikace zobrazující data o zatížení procesoru, zaplnění úložiště a využití RAM je ukázána na obrázku č. 23.



Obrázek 23: Ukázka aplikace „vskp-demo“

5.1.3 Dockerizace aplikace

Pro nasazení aplikace v cloudovém prostředí na platformě Kubernetes je potřeba vytvořit Dockerfile pro již hotovou aplikaci. Vytvořený Dockerfile je zobrazen na obrázku č. 24.

```
1   FROM node:16.13.2-slim AS deps
2
3   WORKDIR /app
4   COPY package.json ./
5   RUN yarn install --frozen-lockfile
6
7
8   FROM node:16.13.2-slim AS builder
9
10  WORKDIR /app
11  COPY --from=deps /app/node_modules ./node_modules
12  COPY . .
13  RUN yarn build
14
15
16  FROM node:16.13.2-slim AS runner
17
18  WORKDIR /app
19  ENV NODE_ENV production
20
21  COPY --from=builder /app/public ./public
22  COPY --from=builder /app/.next/standalone ./
23  COPY --from=builder /app/.next/static ./.next/static
24
25  EXPOSE 3000
26
27  CMD ["node", "server.js"]
```

Obrázek 24: Dockerfile aplikace „vskp-demo“

Dockerfile je rozdělen celkem do tří částí, z nichž každá je zvlášť pojmenována („deps“, „builder“ a „runner“). V první části se zkopíruje soubor „package.json“ obsahující všechny potřebné balíčky a ty se nainstalují (řádek 4 a 5). Ve druhém kroku se nainstalované balíčky zkopírují z předešlého kroku a spustí se příkaz „yarn build“ pro zhotovení výsledné aplikace (řádek 11, 12 a 13). V poslední části se zkopírují finální soubory, které optimalizovanou tvoří finální aplikace připravenou na produkční spuštění.

Deklarace „EXPOSE“ udává, na jakém portu bude aplikace v kontejneru dostupná. Na posledním řádku se pomocí deklarace „CMD“ určí, jaký příkaz má být automaticky spuštěn po vytvoření kontejneru. V tomto případě se jedná o zapnutí Node.js serveru.

Příkazem „docker build“ s dalšími parametry se spustí proces vytvoření obrazu. Příkazem „docker run“ lze spustit kontejner s tímto obrazem. Tyto dva kroky jsou zobrazeny na obrázku č. 25.

```
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/next/vskp-demo$ docker build -t vskp-demo .
[+] Building 1.1s (15/15) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 37B
 => [internal] load .dockerignore
 => => transferring context: 34B
 => [internal] load metadata for docker.io/library/node:16.13.2-slim
 => [internal] load build context
 => => transferring context: 1.27kB
 => [deps 1/4] FROM docker.io/library/node:16.13.2-slim@sha256:f527a6118422b888c35162e0a7e2fb2febced4c85a23d96e1342f9edc2789fe
 => CACHED [deps 2/4] WORKDIR /app
 => CACHED [deps 3/4] COPY package.json ./
 => CACHED [deps 4/4] RUN yarn install --frozen-lockfile
 => CACHED [builder 3/5] COPY --from=deps /app/node_modules ./node_modules
 => CACHED [builder 4/5] COPY . .
 => CACHED [builder 5/5] RUN yarn build
 => CACHED [runner 3/5] COPY --from=builder /app/public ./public
 => CACHED [runner 4/5] COPY --from=builder /app/.next/standalone ./
 => CACHED [runner 5/5] COPY --from=builder /app/.next/static ./.next/static
 => exporting to image
 => => exporting layers
 => => writing image sha256:bcd456c8f287e6d96338a082adcd45644600f541b2f471cf98401aab71c8a0f8
 => => naming to docker.io/library/vskp-demo

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/next/vskp-demo$
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/next/vskp-demo$ docker image list | grep vskp-demo
vskp-demo                                           latest         bcd456c8f287   59 seconds ago   191MB
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/next/vskp-demo$
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/next/vskp-demo$ docker run -p 3000:3000 vskp-demo
Listening on port 3000
```

Obrázek 25: Vytvoření obrazu a spuštění kontejneru

Podle výpisu lze vidět, že vytvoření obrazu proběhlo úspěšně, obraz má štítek „latest“ s velikostí 191 MB. Spuštění kontejneru podle tohoto obrazu proběhlo také úspěšně a aplikace skutečně poslouchá na portu 3000. Tímto je aplikace připravená na nasazení do Kubernetes klastru.

5.2 Konfigurace Kubernetes

Pro aplikaci byly vytvořeny dva jmenné prostory „uhk-fim“ a „uhk-fim-test“ - pro testovací a produkční prostředí. V rámci klastru existuje vytvořený služební účet s názvem „uhk-sa“. Zároveň jsou tomuto služebnímu účtu přidělena omezující práva pro práci s klastrem. Pro vytvoření práv byl použit následující příkaz:

```
kubectl --kubeconfig=master-kubeconfig create clusterrole uhk-role \
--verb=get --verb=list --verb=watch --verb=patch \
--resource=ns,pods,deployments,deployments/scale,nodes/proxy
```

Služebnímu účtu je přiděleno právo pouze číst a upravovat vyjmenované prostředky. Na základě tohoto účtu byl také vytvořen nový kubeconfig soubor obsahující název uživatele, token a certifikační autoritu do klastru.

5.2.1 Vytvoření objektů pro produkční prostředí

V tuto chvíli jsou již nastaveny i přístupové údaje do Kubernetes API přes služební účet pomocí tokenu a certifikační autority. Zbývá vytvořit a aplikovat zbývající objekty pro nasazení Next.js aplikace.

Tajná data

Pro uložení tokenu a certifikátu byla vytvořena tajná data s názvem „uhk-fim-secret“. Definice tohoto objektu pro produkční prostředí je zobrazena na obrázku č. 26.

```yaml
10  apiVersion: v1
11  stringData:
12    api_user_token:
13    cluster_cert:
14  kind: Secret
15  metadata:
16    name: uhk-fim-secret
17    namespace: uhk-fim
18  type: Opaque
```

Obrázek 26: Definice tajných dat

Nasazení

Nasazovací objekt, jak již bylo řečeno, obsahuje definici výsledného kontejneru včetně nasazovací strategie a počtu replik. Na obrázku č. 27 je tato definice manifestu zobrazena.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: vskp-demo
  name: vskp-demo
  namespace: uhk-fim
spec:
  replicas: 2
  selector:
    matchLabels:
      app: vskp-demo
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: vskp-demo
    spec:
      containers:
        - name: vskp-demo
          image: registry.gitlab.com/dresl/vskp-nextjs-on-k8s:latest
          imagePullPolicy: Always
          env:
            - name: API_USER
              value: uhk-fim-vskp-user
            - name: API_USER_TOKEN
              valueFrom:
                secretKeyRef:
                  name: uhk-fim-secret
                  key: api_user_token
            - name: CLUSTER_CERT
              valueFrom:
                secretKeyRef:
                  name: uhk-fim-secret
                  key: cluster_cert
            - name: ENVIRONMENT
              value: production
          ports:
            - containerPort: 3000
              protocol: TCP
      restartPolicy: Always
      imagePullSecrets:
        - name: uhk-gitlab-registry
```

Obrázek 27: Definice nasazovacího objektu

Jsou zde definovány dvě repliky výsledné aplikace. Tokeny a ostatní data se předávají do kontejneru pomocí proměnných prostředí s využitím tajných dat s názvem „uhk-fim-secret“.

Služba

Aby byla aplikace dostupná na portu i v rámci Kubernetes klastru, je nutné vytvořit službu, která bude mapovat tuto aplikaci na definovaný port. Novým zvoleným portem, na kterém bude výsledná aplikace v rámci klastru dostupná, je 8080. Definice je zobrazena na obrázku č. 28.

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    labels:
5      app: vskp-demo
6    name: vskp-demo-service
7    namespace: uhk-fim
8  spec:
9    ports:
10     - port: 8080
11       protocol: TCP
12       targetPort: 3000
13       name: service
14   selector:
15     app: vskp-demo
16   type: NodePort
```

Obrázek 28: Definice objektu služby

Ingress

Všechny objekty potřebné ke spuštění aplikace jsou vytvořené. Posledním objektem, který je třeba vytvořit, je Ingress. Ten zajistí dostupnost aplikace na určité URL adrese za využití již vytvořené služby „vskp-demo-service“. V tomto případě se jedná o adresu „`https://vskp-demo.resldominik.cz`“. Ingress definice je zobrazena na obrázku č. 29.

```
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    annotations:
5      cert-manager.io/cluster-issuer: letsencrypt
6    name: vskp-demo-ingress
7    labels:
8      app: vskp-demo
9    namespace: uhk-fim
10 spec:
11   ingressClassName: nginx
12   rules:
13     - host: vskp-demo.resldominik.cz
14       http:
15         paths:
16           - path: /
17             pathType: Prefix
18             backend:
19               service:
20                 name: vskp-demo-service
21                 port:
22                   number: 8080
23   tls:
24     - hosts:
25         - vskp-demo.resldominik.cz
26       secretName: vskp-demo-ingress
```

Obrázek 29: Definice Ingress objektu

5.2.2 Vytvoření objektů pro testovací prostředí

Vytvoření pro testovací prostředí probíhá stejným způsobem jako pro produkční prostředí. Jediný rozdíl je v tom, že jsou objekty definovány v odlišném jmenném prostoru, v nasazovacím objektu se používá obraz se štítkem „test“ a testovací doména je nastavena na „https://vskp-demo.test.resldominik.cz“.

5.2.3 Nasazení aplikace

Zatím jsou objekty pouze vytvořené lokálně, nikoli aplikované v Kubernetes klastru. Pro nasazení a aplikování objektů je potřeba použít příkaz „kubectl apply“ s cestou k manifest souborům, případně složce s těmito soubory. Ukázka procesu nasazení aplikace včetně kontroly, zda nasazení proběhlo úspěšně, je zobrazena na obrázku č. 30.

```
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/kubernetes$ kubectl --kubeconfig=master-kubeconfig apply -f demo/prod
deployment.apps/vskp-demo created
ingress.networking.k8s.io/vskp-demo-ingress created
secret/uhk-gitlab-registry created
secret/uhk-fim-secret created
service/vskp-demo-service created
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/kubernetes$ kubectl --kubeconfig=master-kubeconfig get all,ingress -n uhk-fim
NAME                             READY   STATUS    RESTARTS   AGE
pod/vskp-demo-687959b75f-fr7lb   1/1     Running   0          43s
pod/vskp-demo-687959b75f-vzqtg   1/1     Running   0          43s

NAME                        TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/vskp-demo-service   NodePort   10.101.208.219  <none>        8080:32325/TCP   42s

NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/vskp-demo   2/2     2            2           43s

NAME                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/vskp-demo-687959b75f   2         2         2       43s

NAME                                        CLASS   HOSTS                     ADDRESS                       PORTS     AGE
ingress.networking.k8s.io/vskp-demo-ingress nginx   vskp-demo.resldominik.cz  49.12.186.221,78.46.250.73    80, 443   43s
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/kubernetes$
```

Obrázek 30: Nasazení aplikace do produkčního prostředí pomocí kubectl

Stejným způsobem byla nasazena i testovací verze, ale pouze s jednou replikou kontejneru, viz obrázek č. 31.

```
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/kubernetes$ kubectl --kubeconfig=master-kubeconfig apply -f demo/test
deployment.apps/vskp-demo created
ingress.networking.k8s.io/vskp-demo-ingress-test created
secret/uhk-gitlab-registry created
secret/uhk-fim-secret created
service/vskp-demo-service created
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/kubernetes$ kubectl --kubeconfig=master-kubeconfig get all,ingress -n uhk-fim-test
NAME                            READY   STATUS    RESTARTS   AGE
pod/vskp-demo-699b656759-fkj8m  1/1     Running   0          22m

NAME                        TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/vskp-demo-service   NodePort   10.99.196.196  <none>        8080:31482/TCP   22m

NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/vskp-demo   1/1     1            1           23m

NAME                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/vskp-demo-699b656759   1         1         1       23m

NAME                                             CLASS   HOSTS                          ADDRESS                      PORTS     AGE
ingress.networking.k8s.io/vskp-demo-ingress-test nginx   vskp-demo.test.resldominik.cz  49.12.186.221,78.46.250.73   80, 443   23m
dominik@drenb:~/work_uhk/vskp-nextjs-on-k8s/kubernetes$
```

Obrázek 31: Nasazení aplikace do testovacího prostředí pomocí kubectl

5.3 Konfigurace GitLab repositáře

Pro vývoj aplikace je využit GitLab jako hosting gitových repozitářů. Byl vytvořen repozitář s názvem „vskp-nextjs-on-k8s“. Pro produkční i testovací prostředí byly vytvořeny speciální větve - „master“ a „test“. Veškerý zdrojový kód aplikace včetně manifestů Kubernetes objektů jsou uloženy právě v tomto repozitáři v určitých adresářích.

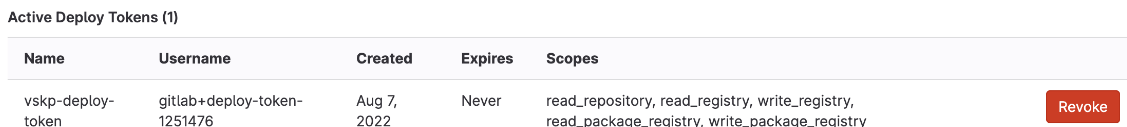
Pro vývoj aplikace je tento způsob ideální, nicméně pro nasazení nové verze je stále nutný manuální zásah ze strany vývojáře. Cílem je, aby při publikování nové verze do testovací větve automaticky proběhlo nasazení na testovací prostředí. Pro produkční prostředí by bylo vhodné, aby proběhly automaticky přípravy na nasazení s tím, že kdokoliv s přístupem do GitLab rozhraní by mohl manuálně vynutit nasazení do produkce kliknutím na jedno tlačítko.

Pro zajištění takové architektury je potřeba automaticky vytvořit nový obraz ze zdrojového kódu publikovaný na GitLab a zároveň tento nový obraz stáhnout v Kubernetes klastru a vynutit restartování Next.js aplikace ve formě kontejneru.

Popsané chování lez pomocí GitLab rozhraní nakonfigurovat.

5.3.1 Docker registr

Pro publikování Docker obrazů je využit *GitLab Container Registry*. Pro získání autorizace do tohoto registru je potřeba získat tzv. *Deploy Token*, který lze vytvořit v nastavení repozitáře. Tento token je zobrazen na obrázku č. 32.


Active Deploy Tokens (1)

Name	Username	Created	Expires	Scopes	
vskp-deploy-token	gitlab+deploy-token-1251476	Aug 7, 2022	Never	read_repository, read_registry, write_registry, read_package_registry, write_package_registry	Revoke

Obrázek 32: Autorizační token do repozitáře

Jak lze z obrázku vidět, jsou nastavena práva jak pro čtení a zápis do Docker registru tak i pro čtení samotného repozitáře.

5.3.2 Proměnné prostředí

Ještě před vytvořením konfigurace pro CI/CD je nutné zajistit konektivitu pomocí tokenů pro přístup do Docker registru a pro přístup do Kubernetes klastru. Pro tyto účely je využito proměnné prostředí v oblasti GitLab CI/CD.

V předchozí kapitole byl vytvořen token, pro který se musejí manuálně přidat proměnné přes GitLab rozhraní. Jedná se o proměnné „`CI_DEPLOY_PASSWORD`" a „`CI_DEPLOY_USER`". Stejným způsobem je přidána proměnná „`KUBECONFIG`", která obsahuje YAML definici kubeconfig souboru napojenou na služební účet „uhk-sa". Takto definované proměnné jsou zobrazeny na obrázku č. 33.

Variables

Variables store information, like passwords and secret keys, that you can use in job scripts. Each project can define a maximum of 8000 variables. Learn more.

Variables can have several attributes. Learn more.

- `Protected:` Only exposed to protected branches or protected tags.
- `Masked:` Hidden in job logs. Must match masking requirements.
- `Expanded:` Variables with `$` will be treated as the start of a reference to another variable.

Environment variables are configured by your administrator to be protected by default.

Type	↑ Key	Value	Options	Environments
Variable	CI_DEPLOY_PASSWORD	*****	Expanded	All (default)
Variable	CI_DEPLOY_USER	*****	Expanded	All (default)
File	KUBECONFIG	*****	Protected, Expanded	All (default)

Add variable   Reveal values

Obrázek 33: Proměnné v GitLab CI/CD

5.3.3 CI/CD

Posledním krokem ke zprovoznění procesu automatizovaného nasazování je potřeba vytvořit speciální soubor „.gitlab-ci.yml“, který obsahuje definici jednotlivých kroků v rámci nasazovacího procesu.

Nejdříve je potřeba přidat základní nastavení, tj. proměnné, které se budou v souboru používat a části procesu (*stages*), viz obrázek č. 34. Jsou zde definovány logicky oddělené části procesu „test“, „build“ a „deploy“. Dále se definují názvy Docker obrazů, které se používají v rámci CI/CD.

```
1  variables:
2    NEXT_ROOT: next/vskp-demo
3    NODE_IMAGE: node:16.13.2-slim
4    DOCKER_IMAGE: docker:20.10.16
5    CONTAINER_TEST_IMAGE: $CI_REGISTRY_IMAGE:test
6    CONTAINER_RELEASE_IMAGE: $CI_REGISTRY_IMAGE:latest
7
8  stages:
9    - test
10   - build
11   - deploy
```

Obrázek 34: Proměnné použité v definici CI/CD

Automaticky lze použít proměnnou „`CI_REGISTRY_IMAGE`“, která nabývá v tomto případě hodnotu „registry.gitlab.com/dresl/vskp-nextjs-on-k8s“. Tento název je také použit v definici nasazovacího objektu v Kubernetes.

První krok, který je součástí všech publikovaných změn, je „lint“. Tento krok nainstaluje balíčky a spustí příkaz „yarn lint“, jak je ukázáno na obrázku č. 35.

```
13  lint:
14    image: $NODE_IMAGE
15    script:
16      - cd $NEXT_ROOT
17      - rm yarn.lock
18      - yarn install
19      - yarn lint
20    stage: test
21    only:
22      changes:
23        - next/**/*
```

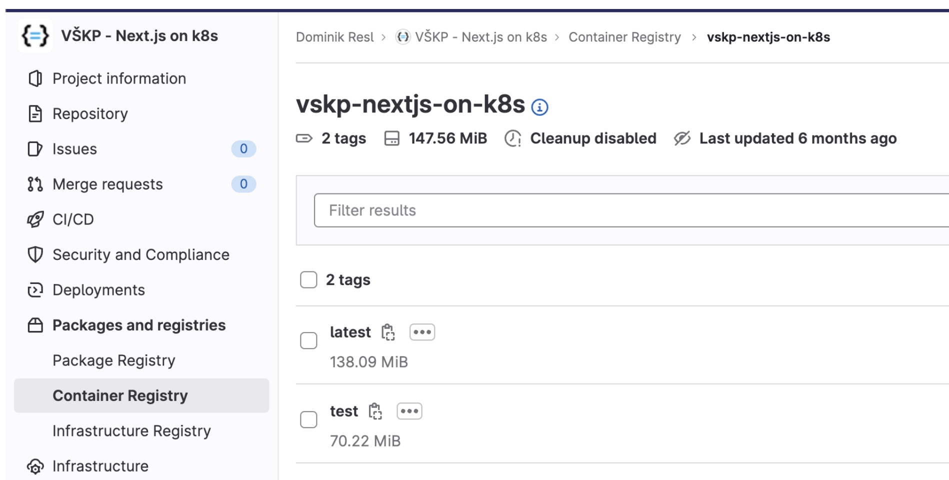
Obrázek 35: CI/CD - testovací krok (kontrola syntaxe)

Pokud tento krok proběhne úspěšně, proces se přesune do další části, a to vytvoření Docker obrazu. K tomuto kroku je využit speciální obraz „docker“ ve verzi 20.10.16, pomocí kterého proběhne vytvoření obrazu a zároveň publikování do GitLab registru za využití GitLab proměnných. Tato část procesu je zobrazena na obrázku č. 36.

```
25  build-test:
26    image: $DOCKER_IMAGE
27    services:
28      - $DOCKER_IMAGE-dind
29    stage: build
30    script:
31      - cd $NEXT_ROOT
32      - docker login -u "$CI_DEPLOY_USER" -p "$CI_DEPLOY_PASSWORD" registry.gitlab.com
33      - docker build -t $CONTAINER_TEST_IMAGE .
34      - docker push $CONTAINER_TEST_IMAGE
35    needs: ["lint"]
36    only:
37      refs:
38        - test
39      changes:
40        - next/**/*
```

Obrázek 36: CI/CD - vytvoření Docker obrazu v testovací větvi

Po úspěšném dokončení této části procesu se výsledný obraz publikuje do registru, který je zobrazen na obrázku č. 37. Pro produkční prostředí je používán obraz se štítkem „latest“, pro testovací se štítkem „test“.



Obrázek 37: CI/CD - GitLab registry s Docker obrazem se dvěma štítky

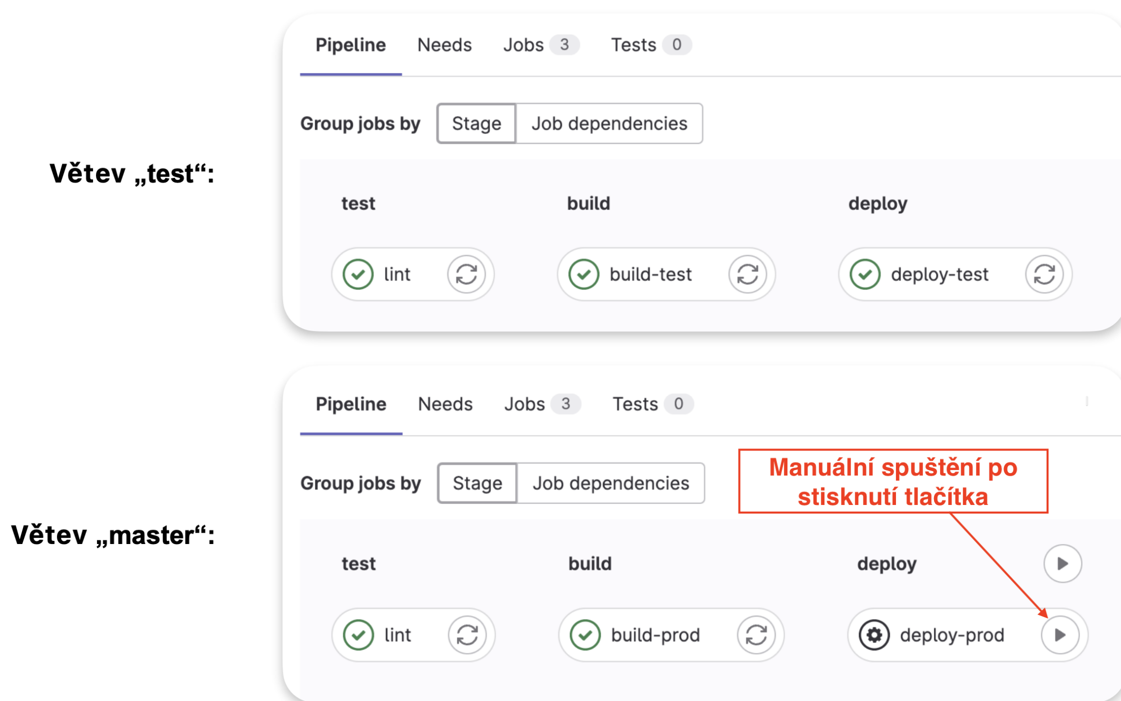
V testovací větvi se po dokončení předchozí části procesu přesune pipeline do poslední části - nasazení vytvořeného obrazu do Kubernetes klastru. Definice tohoto kroku je zachycena na obrázku č. 38.

```
42  deploy-test:
43    image:
44      name: bitnami/kubectl:latest
45      entrypoint: ['']
46    stage: deploy
47    before_script:
48      - export KUBECONFIG=$KUBECONFIG
49    script:
50      - kubectl rollout restart deployment vskp-demo -n uhk-fim-test
51    needs: ["build-test"]
52    only:
53      refs:
54        - test
55      changes:
56        - next/**/*
```

Obrázek 38: CI/CD - definice kroku pro nasazení testovací verze aplikace

Pro nasazení nového obrazu byl použit příkaz „kubectl rollout restart“, kterému se jako parametr zadá název nasazovacího objektu a jmenný prostor, ve kterém je potřeba aplikaci restartovat. Jelikož definice nasazovacího objektu obsahuje klíč „imagePullPolicy“ s hodnotou „Always“, nový obraz se stáhne automaticky. Jinak by se musel přidělit obrazu speciální štítek, například s konkrétní verzí daného obrazu, odlišný při každém novém zahájení procesu CI/CD.

Pro produkční větvi, tj. „master“, je navíc definován klíč „when“ s hodnotou „manual“. Finální průběh procesů se zohledněním manuálního nasazení v produkční větvi je zobrazen na obrázku č. 39.



Obrázek 39: CI/CD - porovnání procesů CI/CD

6 Souhrn výsledků

Praktická část ukázala reálný příklad nasazení webové aplikace do cloudového prostředí s využitím Kubernetes klastru. Bylo vytvořeno produkční a testovací prostředí pro jednoduchou a efektivní údržbu aplikace. Pro vývojáře byly potřebné kroky k nasazení nové verze aplikace automatizovány pomocí GitLab CI/CD procesů, nad kterými má plnou kontrolu.

Veškerá citlivá data nejsou použita přímo ve zdrojovém kódu, ale jsou skryta pomocí speciálních Kubernetes objektů a nehrozí tak únik těchto dat.

Nasazenou aplikaci je možné škálovat v případě zjištění vysoké zátěže a zajistit tak co nejvyšší dostupnost aplikace.

Výsledné nasazené aplikace pro obě prostředí jsou dostupné na adresách „`https://vskp-demo.resldominik.cz`“ a „`https://vskp-demo.test.resldominik.cz`“.

7 Závěry a doporučení

Využitá architektura je robustní a dokáže spravovat i několik desítek či stovek aplikací. Nicméně jsou zde určité limity, které by bylo možné vylepšit.

Výsledná infrastruktura neobsahuje téměř žádné monitorovací služby, které by ukazovaly, v jakém stavu jsou nasazené aplikace. V případě nedostupnosti aplikace by mohlo být zasláno upozornění ve formě notifikace. Pro tyto účely jsou k dispozici nástroje jako *Prometheus* či *Grafana*, které řeší kompletně problematiku monitoringu a souvisejících oblastí.

Literatura

[1] Keith, J.; Sambells, J. *DOM scripting*. Berlin, Germany: APress, second edition, 2010, ISBN 978-1430233893.

[2] Lindley, C. *DOM Enlightenment*. Sebastopol, CA: O'Reilly Media, 2013, ISBN 978-1-449-34284-5.

[3] Microsoft. Announcing TypeScript 0.8.1. [online]. [cit. 2023-03-03]. Dostupné z: https://devblogs.microsoft.com/typescript/announcing-typescript-0-8-1/

[4] jQuery Foundation. History. [online]. [cit. 2023-03-03]. Dostupné z: https://jquery.org/history/

[5] React. Releases · facebook/react · GitHub. [online]. [cit. 2023-03-04]. Dostupné z: https://github.com/facebook/react/releases?page=10

[6] Vue.js. Releases · vuejs/vue · GitHub. [online]. [cit. 2023-03-04]. Dostupné z: https://github.com/vuejs/vue/releases?page=25

[7] Angular. Angular. [online]. [cit. 2023-03-04]. Dostupné z: https://angular.io

[8] Sheppard, D. *Beginning progressive web app development*. New York, NY: APRESS, first edition, 2017, ISBN 978-1-4842-3089-3.

[9] Vercel. Releases · vercel/next.js · GitHub. [online]. [cit. 2023-03-04]. Dostupné z: https://github.com/vercel/next.js/releases?page=178

[10] Apache Subversion. Enterprise-class centralized version control for the masses. [online]. [cit. 2023-03-04]. Dostupné z: https://subversion.apache.org/

[11] Git. Git - A Short History of Git. [online]. [cit. 2023-03-04]. Dostupné z: https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git

[12] Git. Git - About. [online]. [cit. 2023-03-04]. Dostupné z: https://git-scm.com/about/branching-and-merging

[13] Atlassian. Bitbucket | Git solution for teams using Jira. [online]. [cit. 2023-03-04]. Dostupné z: https://bitbucket.org/

[14] npm. npm. [online]. [cit. 2023-03-08]. Dostupné z: https://www.npmjs.com/

[15] Nginx. Nginx - Advanced Load Balancer, Web Server & Reverse Proxy - NGINX. [online]. [cit. 2023-03-11]. Dostupné z: https://www.nginx.com/

[16] IBM. IBM - What is containerization? [online]. [cit. 2023-03-11]. Dostupné z: https://www.ibm.com/topics/containerization

[17] Docker. What is a Container? [online]. [cit. 2023-03-12]. Dostupné z: https://www.docker.com/resources/what-container/

[18] Kubernetes. Overview. [online]. [cit. 2023-03-19]. Dostupné z: https://kubernetes.io/docs/concepts/overview/

[19] Kubernetes. Kubernetes Components. [online]. [cit. 2023-03-20]. Dostupné z: https://kubernetes.io/docs/concepts/overview/components/

[20] Kubernetes. Understanding Kubernetes Objects. [online]. [cit. 2023-03-20]. Dostupné z: https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/

[21] Kubernetes. Pod Lifecycle. [online]. [cit. 2023-03-22]. Dostupné z: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/

[22] Kubernetes. Deployment. [online]. [cit. 2023-03-22]. Dostupné z: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

[23] Kubernetes. Secret. [online]. [cit. 2023-03-23]. Dostupné z: https://kubernetes.io/docs/concepts/configuration/secret/

[24] Kubernetes. Using RBAC Authorization. [online]. [cit. 2023-03-23]. Dostupné z: https://kubernetes.io/docs/reference/access-authn-authz/rbac/

[25] GitLab Docs. GitLab CI/CD. [online]. [cit. 2023-03-24]. Dostupné z: https://docs.gitlab.com/ee/ci/

[26] Stack Overflow. Converting kubernetes kublet API usageNanoCore or usageCoreNanoSeconds to CPU utilization % Kubernetes kublet API. [online]. [cit. 2023-03-26]. Dostupné z: https://stackoverflow.com/questions/38798601/converting-kubernetes-kublet-api-usagenanocore-or-usagecorenanoseconds-to-cpu-ut

Přílohy

A Zadání bakalářské práce

Zadání bakalářské práce

Autor: **Dominik Resl**

Studium: I2000407

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Vývoj a nasazování aplikací na platformě Next.js v cloudovém prostředí**

Název bakalářské práce AJ: Development and deployment of Next.js applications in cloud environment

Cíl, metody, literatura, předpoklady:

Cílem práce je představit konkrétní sadu vývojářských a operativních technologií, která by obstála při tvorbě koplexního webu v produkčním prostředí.
Vybrané technologie budou představeny a porovnány s alternativami, a to zejména v oblastech: webové platformy pro tvorbu obsahu (byla zvolena Next.js), správa a verzování zdrojového kódu, škálovatelné propojení s databází, kontejnerizace a virtualizace, automatické nasazování (byly zvoleny Docker, Kubernetes, Gitlab CI/CD).
Praktická část práce se bude sestávat z ukázkové webové aplikace přiměřeného rozsahu, nasazené v cloudu a využívající vybrané technologie. Jako předmět tohoto ukázkového projektu poslouží data o vytížení serveru, která budou prezentována uživateli na webové stránce.

[1] Mastering GitLab 12: Implement DevOps culture and repository management solutions Autor: Joost Evertse, Rok vydání: 2019
[2] Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud Autor: John Arundel, Justin Domingus, Rok vydání: 2019
[3] React in Action Autor: Mark Tielens Thomas, Rok vydání: 2018

Zadávající pracoviště: Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu

Vedoucí práce: Mgr. Vojtěch Vorel, Ph.D.

Datum zadání závěrečné práce: 26.1.2021