Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Cloudové riešenie založené na microservice architektúre

Bakalárska práca

Autor: Tomáš Šepeľa
Študijný odbor: Informačný management

Vedúci práce: Mgr. Hana Rohrová

Hradec Králové Apríl 2022

Prehlásenie autora:

Prehlasujem, že som bakalársku prácu spracoval samostatne a s použitím uvedenej literatúry.

Vlastnoručný podpis

V Hradci Králové dne 29.4.2022 Tomáš Šepeľa

Poďakovanie:

Rád by som poďakoval mojej vedúcej bakalárskej práce pani Mgr. Hane Rohrovej za vedenie a konzultácie počas písania bakalárskej práce.

Anotácia

Táto bakalárska práca rieši návrh mikroservisnej architektúry na príklade zjednodušeného ecommerce modelu s použitím moderných jazykov, nástrojov a frameworkov. Pre účely tejto práce bol zvolený programovací jazyk Java 17 a framework Spring Boot. Pre infraštruktúru je zvolený nástroj Terraform a ako orchestrátor sa využíva Kubernetes na platforme Digital Ocean. Všetky konfiguračné súbory sú uložené v kóde, ktorý tvorí neoddeliteľnú časť tejto práce. Práca obsahuje teoretickú časť, v ktorej sú opísané využité technológie, ako aj ich výhody a nevýhody. Praktická časť je rozdelená na analytickú a implementačnú sekciu, v rámci ktorých sú detailnejšie opísané využité knižnice a navrhnuté riešenie. Táto práca môže slúžiť ako podklad pre vývojárov a architektov na založenie nového projektu pre projekty rôznych veľkostí.

Annotation

Title: Cloud solution based on microservice architecture

This bachelor thesis addresses the design of a microservices architecture using the example of a simplified ecommerce model using modern languages, tools and frameworks. For the purpose of this thesis, Java 17 programming language and Spring Boot framework were chosen. Terraform is chosen for the infrastructure and Kubernetes on Digital Ocean platform is used as the orchestrator. All the configuration files are stored in the code that forms an integral part of this thesis. The thesis contains a theoretical part in which the technologies used are described, as well as their advantages and disadvantages. The practical part is divided into analytical and implementation sections, in which the libraries used and the proposed solution are described in more detail. This work can serve as a basis for developers and architects to start a new project for projects of different sizes.

Obsah

Zoznam obrázkov

Zoznam tabuliek

1 Úvod

Informačné systémy rôznych veľkosti si vyžadujú rôzne technologické riešenia. Na každý informačný systém sú kladené iné funkčné či nefunkčné požiadavky, na ktoré musí byť vývojový tím pripravený agilne reagovať. Bežné informačné systémy stredne veľkého podniku v dobe písania tejto bakalárskej práce dosahujú často stovky modelov a databáz, ktoré je potrebné udržiavať. Údržba nespočíva iba v produkčnej podpore a pridávaní nových funkcií, ale aj vo vzdelávaní tímu, ktorý na ňom pracuje. Tento tým sa neskladá iba z programátorov, ale aj analytikov, projektových manažérov, testerov, DevOps, architektov či iných pracovníkov.

V prípade monolitickej architektúry, kde sa systém skladá iba z jednej komponenty – služby – je vyžadované aby celý vývojový tím poznal celú funkcionalitu. S príchodom architektúry orientovanej na služby táto potreba čiastočne zaniká, čo sa vo výslednom efekte prejaví na časovej a finančnej náročnosti riešenia jednotlivých úloh či požiadaviek. Aj napriek tomu má servisne orientovaná architektúra svoje mínusy, ako je napríklad zdieľanie prostriedkov či zdieľanie dát medzi službami. Niektoré z týchto problémov rieši práve mikroservisná architektúra, ktorá je predmetom tejto práce.

Mikroservisná architektúra je zameraná na menšie celky, ktoré sú zamerané na určitú časť biznis procesov, a vzájomne medzi sebou komunikujú. Jednou z výhod mikroservisnej architektúry je práve jednoduchý a pomerne rýchly proces nasadenia, čo vedie k pravidelnému a spoľahlivému dodávaniu komplexných aplikácií.

2 Cieľ práce

Cieľom tejto práce je zoznámiť čitateľa s problematikou mikroservisnej architektúry založenej na frameworku Spring s použitím moderných jazykov, knižníc, nástrojov a frameworkov. Pre plné pochopenie problematiky by mal mať čitateľ základnú znalosť programovacieho jazyka Java, frameworku Spring a servisne orientovanej architektúry.

Táto práca bude rozdelená na 2 časti, teoretickú a praktickú. Praktická bude rozdelená na 2 sekcie, analytickú a implementačnú. Teoretická časť bude pozostávať z teoretických východísk v oblastí kontajnerizácie, orchestrátorov a nasadzovania služieb v cloude. V analytickej sekcii praktickej časti sa budú nachádzať analyzované modely a nástroje, ktoré budú autorom zvolené pre túto prácu. Implementačná sekcia bude opisovať, ako sú jednotlivé komponenty vytvorené, ako fungujú a ako medzi sebou komunikujú. Celá praktická časť bude opísaná na teoretickom ecommerce modeli.

Všetky mikroslužby budú napísané v jazyku Java 17 s využitím MongoDB. Na zabezpečenie komunikácie bude aplikovaný protokol OpenID Connect, o ktorý sa bude starať open-source softvér Keycloak. Databáza pre keycloak, rovnako ako databáza pre služby, budú prenajaté ako služby.

Pre teoretickú aj praktickú časť tejto práce budú využité najmä skúsenosti a vedomosti autora, ktoré získal počas svojho pôsobenia v praxi. Ďalej budú využité aj vedomosti, ktoré získal v priebehu štúdia na vysokej škole.

Cieľom praktickej časti tejto práce je funkčný systém, ktorý je schopný automatického nasadenia, škálovania a testovania. Overenie funkčnosti je možné pomocou používateľského rozhrania vo frameworku Next.js, ktoré autor vypracoval počas svojho pôsobenia na vysokej škole v rámci predmetu Technologie pro publikování na Webu II.

3 Teoretická časť

V tejto časti práce sú vysvetlené základné teoretické východiská s ktorými pracuje praktická časť.

3.1 Cloud

Cloud, alebo často označovaný aj ako cloud computing je spôsob distribúcie výpočtových prostriedkov pre rôzne aplikácie dostupné na internete.

3.1.1 Výhody cloudu

Rôzne počítače alebo zariadenia, prepojené v rámci jednej siete, dokážu spolu komunikovať a vytvárajú tak infraštruktúru, ktorá dokáže prevádzkovať rôzne typy softvéru. Medzi najčastejšie používané druhy softvéru môžeme určite zaradiť databázy či webové aplikácie. Výpočtovú silu týchto počítačov si je možné prenajať. Firma teda nemusí nakupovať svoje vlastné servery (je to často finančne náročnejšie, kvôli nákupnej cene serverov, sieťových prvkov, licencii na softvér a cene práce potrebnej na údržbu), ale využije časť cloudovej infraštruktúry. (1)
Ďalšou výhodou cloudu je decentralizácia. V prípade konvenčnej aplikácie, ktorá má byť dostupná rýchlo a spoľahlivo z rôznych časti sveta by to znamenalo nutnosť zriadenia dátových centier v rôznych oblastiach. Prenájmom cloudu túto nutnosť eliminuje.

3.1.2 Typy cloudu

Podľa toho, kto cloud spravuje, ho môžeme rozdeliť na niekoľko typov:

a) **Public Cloud** – alebo verejný cloud, ako už názov hovorí, ide o verejne dostupný cloud – voľné prostriedky si môže prenajať akákoľvek firma,
b) **Private Cloud** – alebo súkromný cloud – využívaný exkluzívne jednou spoločnosťou na jej vlastné účely – tento typ si nemôže prenajať široká verejnosť,
c) **Community Cloud** – alebo komunitný cloud – využívaný určitou komunitou – napríklad herná komunita (hlasové servery, herné servery, ...),
d) **Hybrid Cloud** – hybridný cloud – časť infraštruktúry je určená na privátne použitie a časť je dostupná širokej verejnosti.

3.1.3 Poskytovatelia public cloudu

Táto práca sa zaoberá výhradne verejným cloudom, ktorý je dostupný širokej verejnosti za poplatok. Medzi najväčších poskytovateľov public cloudu patria:

a) Amazon Web Services (2)
b) Microsoft Azure (3)
c) Digital Ocean (4)
d) Google Cloud Platform (5)

Cenový model u jednotlivých providerov sa môže líšiť. Pri niektorých službách sa platí za rezervovaný čas, u niektorých za počet requestov. Jednotlivé cenníky sú dostupné na ich webových stránkach.

3.2 Mikroslužby

K tomu aby bolo možné definovať pojem mikroslužba je nutná znalosť architektonického prístupu „Architektúra orientovaná na služby“.

3.2.1 Čo je to mikroslužba?

Architektúra orientovaná na služby alebo ang. Service Oriented Architecture (SOA) je architektonický prístup, ktorého hlavným cieľom je oddeliť časti aplikácie na

jednotlivé služby. Služby je možné v tomto prípade chápať ako samostatný funkčný program schopný vykonávať jeden druh činnosti.

Mikroslužba je jedna časť mikroslužbovej architektúry. Tá je do určitej miery rozšírením SOA. Opakom microservice architecture je tzv. monolithic architecture. Monolitická architektúra pozostáva z jednej služby, ktorá pokrýva celú biznis logiku. Nevýhodou monolitickej architektúry je jej komplexita a náročnosť pridávania nových funkcií alebo opravy chýb v existujúcom zdrojovom kóde. Aj najmenšia zmena si vyžaduje opätovanú rekompiláciu, otestovanie a nasadenie celého projektu a na požadované prostredia.

Tento problém je jednoducho odstrániteľný zavedením microservice architektúry. Aplikáciu je možné rozložiť na malé, samostatne funkčné celky – mikroslužby. Tie je možné nasadzovať osobitne bez výpadku ostatných služieb.

Jednou z výhod mikroslužieb je aj nezávislosť na technológii. Pri microservice architektúre je možné každú mikroslužbu postaviť na inej technológii, s iným dátovým úložiskom a prípadne aj iným komunikačným protokolom.

K tomu aby sa služba mohla nazývať mikroslužbou je nutné splniť niekoľko požiadaviek:

a) je jednoduchá na údržbu a testovanie,
b) obsahuje iba slabé väzby (Loose Coupling),
c) je ju možné nasadiť nezávisle,
d) je vytvorená na základe špecifickej business požiadavky,
e) spravuje ju malý tím.

(6)

3.2.2 Ako funguje mikroslužba?

Mikroslužba je samostatný funkčný celok. Má teda vlastnú databázu, pridelený výpočtový výkon, pipeliny, cache, atď.

Ak mikroslužba potrebuje dáta, ktoré sama nemá, tak komunikuje s inou mikroslužbou. Táto komunikácia môže byť synchrónna – očakáva odpoveď, ako operácia dopadla, alebo asynchrónna – očakáva odpoveď, že požiadavka bola zaradená do fronty na spracovanie. O jej priebeh a spracovanie sa stará broker.

V prípade, že mikroslužba nezvláda spracovávať prichádzajúce požiadavky je schopná automatického škálovania.

3.3 Docker a kontajnerizácia

3.3.1 Virtualizácia

Virtualizácia je veľmi obľúbenou súčasťou cloud computingu. Pomocou virtualizácie je možné rozdeliť výkonné servery na menšie celky, tzv. virtuálne servery (VPS). Virtuálny server môže využívať zdieľané alebo dedikované prostriedky. Každý virtuálny server by mal mať priradené CPU, RAM, úložisko a sieťovú priepustnosť. Výhodami virtuálnych serverov je pomerne nízka, izolácia od okolitého prostredia a nízka náročnosť na správu a údržbu.

3.3.2 Kontajnerizácia

Vytvorenie virtuálneho serveru a inštalácia všetkých potrebných knižníc a súčasti nutných k spusteniu aplikácie je zvyčajne časovo náročná. Tento problém ale pomerne efektívne rieši kontajnerizácia. Kontajner si je možné predstaviť ako virtuálny server s preddefinovaným

3.3.3 Docker

Pojem docker je možné chápať niekoľkými spôsobmi:

- Docker Inc – spoločnosť, ktorá udržiava platformu,
- Docker Engine - Kontajnerizačná technológia,
- Docker Hub – verejný repozitár, kde sú dostupné rôzne docker obrazy,
- Docker Desktop – GUI nástroj pre Window / MacOS.

(7)

Docker Engine je open-source platforma udržiavaná rovnomennou spoločnosťou (Docker Inc), slúžiaca na vývoj, nasadenie a správu kontajnerizovaných aplikácií. Vďaka dockeru sú vývojari schopní zabaliť aplikáciu do tzv. docker kontajnera.

Docker kontajner je samostatne spustiteľný celok, ktorý má vlastný operačný systém, knižnice nevyhnutné k spusteniu aplikácie a aplikáciu samotnú. Tento prístup je samozrejme možný aj bez použitia dockeru (napríklad LinuXContainers). V porovnaní s inými nástrojmi je ale docker jednoduchší a lacnejší na prevádzku.

Výhody dockeru:

a) **Prenosnosť**
   Docker kontajnery je možné spustiť na akomkoľvek operačnom systéme.
b) **Granularita**
   V kontajneri je možné spustiť iba jeden proces. Vďaka tomu je developer schopný jednoducho diagnostikovať prípadné chyby.
c) **Automatické vytvorenie**
   Docker dokáže automaticky vytvoriť kontajner zo zdrojového kódu a v prípade problému ho reštartovať.
d) **Verzie**
   Docker je schopný uchovávať verzie obrazov, z ktorých je kontajner vytvorený a vďaka tomu je možné jednoducho nasadiť staršiu verziu aplikácie.
e) **Podpora komunity**
   Okolo dockeru je pomerne široká komunita. Developer si môže vybrať akýkoľvek komunitný obraz a z neho vytvoriť vlastnú, upravenú, verziu.

K tomu aby bolo možné vytvoriť docker kontajner slúži tzv. Dockerfile. Dockerfile je zoznam inštrukcií, podľa ktorých Docker Engine dokáže vytvoriť obraz. Obsahuje informáciu o zdrojovom obraze, dodatočnom softvéri, sieťovej konfigurácii, perzistetnom úložisku a inštrukciách k spusteniu.
Takto pripravený obraz je určený iba k čítaniu. K tomu, aby dokázal vykonávať to, na čo bol navrhnutý, je nutné ho spustiť v tzv. docker kontajneri. Docker kontajner je teda živá inštancia obrazu, ktorá dokáže vykonávať svoj účel.
Pripravený image je možné aj distribuovať. K tomu slúžia docker registre. Sú to verejné alebo súkromné úložiská, ktoré uchovávajú rôzne verzie obrazov. Docker

register funguje podobne ako git. Je možné na neho nahrávať obrazy (push) a sťahovať (pull). Je to jedna z možností, ako automatizovane dostať aplikáciu zo zariadenia vývojára na server.

3.3.4 Orchestrácia

Udržiavanie malého počtu kontajnerov je nenáročné. Problém nastáva, pokiaľ ich k fungovaniu aplikácie potrebujeme stovky či tisíce. Spravovať také množstvo kontajnerov je fyzicky nemožné. Na zjednodušenie tohto procesu bolo vyvinutých niekoľko nástrojov:

3.3.5 Docker Compose

Najjednoduchším nástrojom na správu a údržbu je docker compose. Jeho základným setom inštrukcií je YAML súbor, zvyčajne nazývaný docker-compose.yaml. Tento YAML súbor definuje, z akých kontajnerov je aplikácia tvorená, aké sú jej závislosti, ako komunikujú, či aké majú pridelené úložisko.

3.3.6 Docker Swarm

Docker Swarm je možné považovať za najjednoduchší orchestračný softvér. Je jednoduchý na inštaláciu a poskytuje 3 základné prvky:

- nody,
- služby a úlohy,
- load balancery.

Je vhodný pre malé, nenáročné prostredia tvorené malým počtom kontajnerov.

3.4 Kubernetes

Kubernetes je open-source orchestračný nástroj na správu a údržbu najzložitejších prostredí. Vďaka svojej robustnosti je vhodný pre produkčné použitie. (8) Narozdiel od Docker Swarm-u, kubernetes dokáže fungovať na viacerých serveroch súčasne.

3.4.1 Časti kubernetes clustra

Kubernetes sa skladá z dvoch základných častí:

a) **Master Component**
   Úlohou master komponent je riadiť stav kubernetes clustra a manažovať worker nody.
b) **Worker Nodes**
   Slúžia na prevádzkovanie kontajnerov.

3.4.2 Typy objektov

K tomu aby bolo kubernetes schopné obslúžiť aj najnáročnejšie požiadavky produkčných aplikácií slúži niekoľko základných stavebných častí:

a) Pod
   Pod je najzakladanejšou funkčnou jednotkou v kubernetes. Skladá sa z 1 alebo viacerých kontajnerov, ktoré sú umiestnené na rovnakom node. Každý pod ma svoju vlastnú internú adresu a pridelený diskový priestor.
b) Service
c) Deployment
d) ReplicaSet
e) ConfigMap
f) Secret

3.5 Spring Boot

Spring je populárny open-source framework vyvinutý v programovacom jazyku Java. S príchodom servisne orientovanej architektúry ale vznikla potreba na sofistikovanejšie riešenie, ktoré ešte viac uľahčí prácu vývojárom. Preto vznikol

Spring Boot. Spring Boot je súbor nástrojov, ktorý uľahčuje vývoj webových aplikácii, vďaka svojim 3 základným prvkom:

a) Automatická konfigurácia
b) Voliteľné moduly
c) Prednastavený server

(9)

3.5.1 Spring aplikácie v kubernetes

K tomu, aby bolo možné spring aplikácie jednoduchšie kontajnerizovať a nasadzovať do distribuovaných systémov slúži set nástrojov zvaný Spring Cloud. Ten za nás rieši niekoľko základných problémov, ako napríklad:

a) Registrácia mikroslužieb a ich vyhľadávanie v rámci siete (discovery)
b) Smerovanie (routing)
c) Volania medzi službami
d) Rozdeľovanie výkonu (load balancing)

Spring Cloud má niekoľko implementácii. Každá implementácia je vhodná pre iný typ distribuovania aplikácie. Medzi najčastejšie používané patria:

a) Spring Cloud Azure
b) Spring Cloud Alibaba
c) Spring Cloud AWS
d) Spring Cloud Gateway
e) Spring Cloud Kubernetes
f) Spring Cloud Security
g) Spring Cloud Sleuth
h) Spring Cloud Vault

3.5.2 Spring Boot Admin

Spring Boot Admin je webová služba určená na správu a monitorovanie mikroslužieb využívajúcich framework Spring. Na pozadí každá mikroslužba vystavuje pomocou actuatora endpoint, z ktorých Spring Boot Admin sťahuje dáta, alebo naň posiela príkazy.

3.6 Komunikácia medzi mikroslužbami

3.6.1 Synchrónna komunikácia

V servisne orientovanej architektúre má každá mikroslužba svoju doménu modelov, s ktorými pracuje. Často ale nastane situácia, že v rámci API musí poslať aj informácie, ktorými sama nedisponuje. K tomu aby tieto dáta získala musí zavolať endpoint inej mikroslužby a dáta si vyžiadať. Táto komunikácia musí prebehnúť v reálnom čase, nakoľko jej klient čaká na odpoveď. V prípade komunikácie, ktorá sa deje v reálnom čase a volajúca služba čaká na spracovanie a následnú odpoveď, hovoríme o synchrónnej komunikácií.

3.6.2 REST API

Synchrónna komunikácia môže byť implementovaná rôznymi návrhovými vzormi, či protokolmi. Jedným z nich je aj REST. REST je súbor architektonických pravidiel, ktoré rozširujú HTTP komunikáciu. Základným stavebným prvkom REST-u je zdroj (resource). Zdrojom môže byť akýkoľvek dokument, obrázok, služba alebo akýkoľvek iný objekt. Tieto zdroje je možné čítať (GET), vytvárať (POST), upravovať (PUT), upravovať čiastočne (PATCH) alebo mazať (DELETE).

3.6.3 OpenAPI Špecifikácia

OpenAPI špecifikácia pomáha definovať RESTful API tak, aby mu lepšie rozumeli jeho klienti. Klientom sa rozumie webová aplikácia, či server, ktorý API volá alebo človek, ktorý API študuje. OpenAPI nevystavuje žiadne implementačné detaily ani kusy zdrojového kódu. Definuje iba rozhranie, na ktorom klient a server komunikujú.

Pomocou OpenAPI je možné generovať dátové objekty, či servisné rozhrania. Vďaka tomuto prístupu nie je nutné, aby frontend developer mal akúkoľvek znalosť backendu a naopak.

OpenAPI špecifikáciu je možné písať v 2 formátoch: JSON alebo YAML. V oboch prípadoch musí byť zachovaná štruktúra podľa dokumentácie, ktorá je verejne dostupná. (10)

3.6.4 Asynchrónna komunikácia

V praxi ale služby vykonávajú komunikáciu, kde nie je nutné čakať na odpoveď. Typickým príkladom môže byť hromadné spracovanie informácií, kde operácia môže trvať aj niekoľko hodín.

3.6.5 Fronty

V prípade spracovania veľkého množstvá dát je možné využiť tzv. fronty. Fronty sú poradovníky, do ktorých jedna služba vkláda požiadavky na spracovanie a message broker tieto požiadavky priradzuje klientom, ktorý ich spracujú.

3.6.6 RabbitMQ

Najjednoduchším message brokerom, ktorý je možné v servisne orientovanej architektúre použiť je open-source projekt RabbitMQ. RabbitMQ je napísaný v jazyku Erlang a bol pôvodne vyvinutý spoločnosťou Rabbit Technologies Ltd. (11)

3.6.7 AMQP Protokol

AMQP, alebo Advanced Message Queuing Protocol je aplikačný protokol, ktorý slúži na zasielanie, smerovanie, zabezpečenie a zaraďovanie správ do fronty. (12)

3.7 Bezpečnosť

Bezpečnosť a je možné na backende zabezpečiť rôznymi spôsobmi.

3.7.1 Autorizácia a autentifikácia

Autorizácia a autentifikácia je opakovaný problém mnohých webových aplikácií. Bezpečnosť je jedným z najdôležitejších prvkov, ktorým je potrebné venovať pozornosť. Preto nie je vhodné vyvíjať vlastné riešenie (službu, protokol, štandard) ale naopak siahnuť po hotovom riešení. Hotové riešenie môže mať formu open-

source projektu (Keycloak) alebo softvéru ako služby – SaaS (Okta, Auth0, Amazon Cognito, Azure AD, atď.).

3.7.2 OIDC

OIDC, alebo OpenID Connect je identifikačná vrstva postavená na základe OpenID 2.0. OIDC umožňuje klientom autorizáciu používateľov voči autorizačnému serveru, či získavanie základných informácii o používateľovi.
Použitím Spring Bootu je možné implementovať OIDC. Podľa autorizačného serveru často stačí definovať 2 konfiguračné premenné a automatická konfigurácia Spring Bootu sa postará o zvyšné nastavenia.
OIDC je možné implementovať rôznymi spôsobmi. Jedným z nich je aj Authorization Code Flow. V tomto prípade sa ID Token aj Access Token získajú v odpovedi z token endpointu.

3.7.3 JWT Token

JWT Token je štandard – RFC-7519 (13), ktorý umožňuje bezpečný prenos JSON informácií medzi 2 klientmi.
JWT Token ma najpočetnejšie využitie práve v rámci autorizácie. Každá požiadavka klienta voči serveru obsahuje v hlavičkách autorizačnú informáciu v podobe JWT Tokenu.

3.8 Testovanie

Implementácia servisne orientovanej architektúry je voči monolitickému prístupu vďaka svojim závislostiam na iných službách náchylnejšia na chyby. Z tohto dôvodu je vhodné dbať na dostatočné pokrytie testami.

3.8.1 Unit Testy

Unit testy sú testy, ktoré pokrývajú malú časť kódu. Tieto testy by mali byť automatické a mali by prebiehať ešte pred samotným buildom aplikácie.

3.8.2 Integračné test

Integračné testy rozširujú unit testy a ich cieľom je pokryť väčšiu časť zdrojového kódu. Ich cieľom je overiť či 2 a viac časti aplikácie spolu fungujú. Integračné testy by mali byť spustené automaticky po unit testoch.

3.8.3 E2E Testy

Testovanie aplikácie by malo byť zakončené E2E testami. E2E testy je vhodné vykonávať na pred-produkčnom prostredí po nasadení aplikácie.

4 Zadanie praktickej časti

V praktickej časti tejto práce budú využité nadobudnuté teoretické znalosti v oblasti kontajnerizácie, softvérového vývoja, mikroservisnej architektúry, Spring Bootu, Spring Cloudu, Keycloaku, Spring Security, React-u, Node JS a kubernetes. Praktická časť bude pozostávať z 2 hlavných celkov: analýzy a implementácie.
Nasledujúce podkapitoly za budú zaoberať ukážkou architektúry na jednoduchom ecommerce modeli. Model je dostatočne zjednodušený, aby odpovedal rozsahu tejto bakalárskej práce, a preto sú niektoré modely, ako napríklad košík, profil zákazníka, či detail produktu, vynechané. V tomto zjednodušenom modeli je možné vytvoriť hromadnú objednávku produktov ako anonymný, či prihlásený zákazník. Do systému vstupujú 3 typy používateľov:

ANONYMOUS – anonymný používateľ, bez prihlásenia, smie prezerať produkty a vytvoriť objednávku

CUSTOMER – zákazník, smie všetko, čo anonymný používateľ a má naviac možnosť prihlásenia a zobrazenie histórie svojich objednávok

ADMIN - administrátor, smie vykonávať všetky CRUD operácie nad objektami typu objednávka a produkt

Používateľské rozhranie tohto systému sa rozdeľuje na 2 aplikácie. Prvou aplikáciou je hlavný frontend (webapp), ktorý obsahuje administráciu pre dátové objekty typu Produkt a Objednávka. Druhým používateľským rozhraním je rozhranie keycloak-u. To umožňuje prihlásenie používateľov do systému ako SSO (Single-sign-on) a zároveň poskytuje aj administračné rozhranie pre nastavenie Keycloaku ako IP (Identity Provider) – zoznam klientov, používateľov, aktívnych sedení a ďalšie.

V rámci praktickej časti bude vytvorená jednoduchá cloud aplikáciu, ktorá bude implementovať princípy architektúry orientovanej na mikroslužby. Jadro bude tvorené z týchto služieb:

1. Produktová služba,
2. Objednávková služba,
3. Štatistická služba,
4. API Gateway ,
5. Keycloak,
6. Webapp (Frontend).

Zadanie je doplnené akceptačnými kritériami, ktoré je dostupné v prílohe č. 1.

5 Analýza zadania

Implementácii zadaného problému predchádza analýza, ktorej výsledkom budú navrhnuté dátové modely a použité nástroje.

5.1 Analýza použitých nástrojov

V tejto práci budem uchovávaná čo najväčšia možná časť architektúry v kóde. V prípade infraštruktúry sa tento prístup nazýva Infrastructure as a Code, alebo skrátene IaC. Na automatizované vytvorenie kubernetes infraštruktúry bude použitý Terraform. Terraform je nástroj, ktorý slúži na automatizované vytvorenie infraštruktúry a vykonávanie zmien v nej za pomoci konfiguračných súborov.
Zdrojový kód bude verzovaný a uchovávaný na GitLabe, pomocou technológie git. Okrem zdrojového kódu budú na GitLabe uložené aj Maven artefakty a Docker obrazy. CI/CD procesy budú tiež vykonávané v pipelinach na GitLabe.
Z OpenAPI špecifikácie budú pomocou swagger generátora generované zdrojové súbory pre klienta aj pre server. Backendová mikroslužba môže byť v roli klienta pre inú miroslužbu a serveru pre frontend alebo mikroslužbu. V prípade serverovej časti pôjde o DTO objekty, rozhrania pre API controllery a reaktívny webklient pre synchrónnu komunikáciu medzi mikroslužbami.

5.2 Analýza dátových objektov

Ako už bolo v úvodnej sekcii naznačené, v tomto ecommerce modeli bude vystupovať niekoľko jednoduchých objektov:

- používateľ – model, súčasť identity providera, obsahuje základné informácie o používateľovi – používateľské meno, heslo, celé meno,
- produkt – model, obsahuje informácie o fyzickom produkte – názov, SKU, popis, cena,
- objednávka – model, obsahuje informácie o objednávke, skladá sa z položiek objednávky – produkt, cena (klonovaná pre prípad zmeny ceny produktu v čase), počet,
- rola - model, súčasť identity providera, hodnoty: CUSTOMER, ADMIN,
- štatistika – agregát, vypočítaný zo štatistických udalosti.

5.3 Analýza služieb

Jednotlivé modely opísané v predchádzajúcej sekcii sú rozdelené podľa zodpovednosti na jednotlivé kontexty – mikroslužby.

API Gateway bude zabezpečovať overovanie prichádzajúcich Authorization hlavičiek, kontrolovať ich platnosť voči keycloaku a propagovať vlastné hlavičky do ďalšej distribúcie (X-User-Id, X-Roles). Tieto dáta budú získané z JWT tokenu a mikroslužby za API GW budú považovať tieto hlavičky za dôveryhodné. Táto služba bude postavená na reaktívnej Spring Cloud Gateway.

V smere z frontendu k databáze sa za API GW budú nachádzať ostatné mikroslužby – produktová služba, objednávková služba a štatistická služba.

Produktová služba bude spracovávať informácie o produktoch a ich kategóriách. Na frontend bude vystavovať REST API, ktoré bude pozostávať z 1 zdroja - Produktu.

Objednávková služba bude spracovávať informácie o objednávkach. Na frontend bude vystavovať RESTový zdroj Objednávka, ktorá v rámci svojich položiek bude obsahovať aj informácie o produktoch – ich identifikátory.

Objednávková služba bude zbierať štatistické dáta vo forme výšok objednávok. Nakoľko prichádzajúca požiadavka na vytvorenie objednávky je synchrónna a dáta o štatistickom evente nie sú pre odberateľa dôležité, je možné využiť asynchrónnu komunikáciu. Pomocou protokolu AMQP budú správy zaslané do messsage brokera, ktorý ich následne nasmeruje na štatistickú službu. Štatistická služba následne vytvorí záznam v databáze. K prehliadaniu dát zo štatistickej služby bude slúžiť API endpoint, ktorý vytvorí agregovaný pohľad.

Ako podporné služby budú v tomto projekte použité:

1. RabbitMQ

   - rola message brokera – príjem, spracovanie a zaradenie správy do fronty.

2. Keycloak

   - rola identity managera – uchovávanie informácii o používateľoch, vydávanie tokenov, overovanie tokenov.

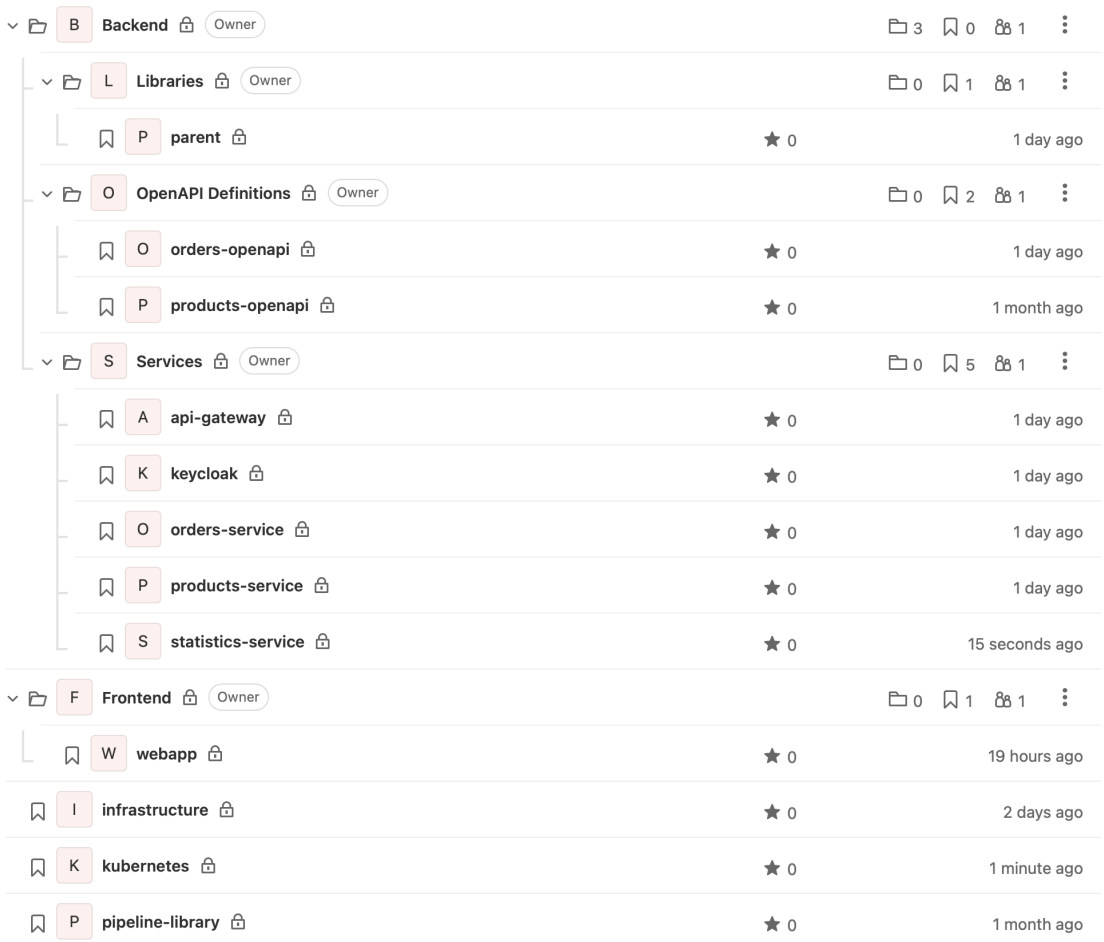
3. NGINX Ingress Controller
   - smerovanie prichádzajúcich požiadaviek na API Gateway.
   - možné rozšíriť o frontend, či iné služby.
4. Cert Manager
   - automatizovaný manažment Lets's enrypt SSL certifikátov.

5.4 Analýza rozdelenia repozitárov

V rámci prístupu „separation of concerns", alebo oddelenia zodpovednosti, je vhodné oddeliť jednotlivé služby a ich závislosti do osobitných repozitárov. Toto rozdelenie dovoľuje definovať rôzne pipeliny pre rôzne typy operácii. Napríklad, výsledkom pipeliny pre mikroslužbu je docker kontajner nasadený v kubernetes a pre repozitár s OpenAPI špecifikáciou maven artefakt, ktorý je možne importovať v 2 rôznych službách a automaticky tak vygenerovať zdrojový kód pre klient a server.

Podľa opísaného prístupu je vhodné následné rozdelenie:

1. Maven Parent – maven artefakt, z ktorého dedia ostatné maven projekty,
2. OpenAPI – pre každú službu vlastný repozitár,
3. mikroslužba – pre každú mikroslužbu vlastný repozitár,
4. infraštruktúra – terraform scripty a základné kubernetes scripty,
5. knižnica pre pipeliny – šablóny, z ktorých dedia jednotlivé konfigurácie.



Obr. 1 Štruktúra repozitárov na GitLab.com
Zdroj: Záznam obrazovky z GitLab.com – vlastné spracovanie

6 Implementácia ecommerce modelu

Nasledujúca sekcia sa zaoberá zvolenými technológiami a implementačnými detailmi jednotlivých mikroslužieb a architektúry, ktorej sú súčasťou.

6.1 Voľba technológií

Na základe teoretických znalostí bol pre túto prácu vybraný programovací jazyk Java 17 a framework Spring. Pre mikroslužby bola použitá nerelačná databáza MongoDB, ktorá bola prenajatá ako služba od MongoDB Atlas. Pre Keycloak bola použitá relačná databáza PostgreSQL, ktorá bola tiež prenajatá ako služba od Digital Ocean. Jej vytvorenie je zdokumentované v zdrojových súboroch infraštruktúry. Systém je zabezpečený JWT Tokenom za využitia OIDC protokolu.

Z frameworku Spring Boot boli použité nasledujúce balíčky:

- **spring-boot-starter-actuator** – slúži na reportovanie stavu mikroslužby – v tomto projekte sú z neho použité health/readiness check-y pre kubernetes,
- **spring-boot-starter-security** – slúži na zabezpečenie aplikácie,
- **spring-security-oauth2** – slúži na dekódovanie a overenie JWT tokenov,
- **spring-boot-starter-data-mongodb** – slúži na komunikáciu s MongoDB,
- **spring-boot-starter-web** – zjednodušuje prácu s REST API,
- **spring-boot-starter-test** – slúži na zjednodušené písanie JUnit testov,
- **spring-cloud-starter-gateway** – slúži na smerovanie prichádzajúcich požiadavok na jednotlivé mikroslužby a zabezpečenie – validáciu JWT tokenov a propagáciu bezpečnostných hlavičiek.

Ostatné závislosti:

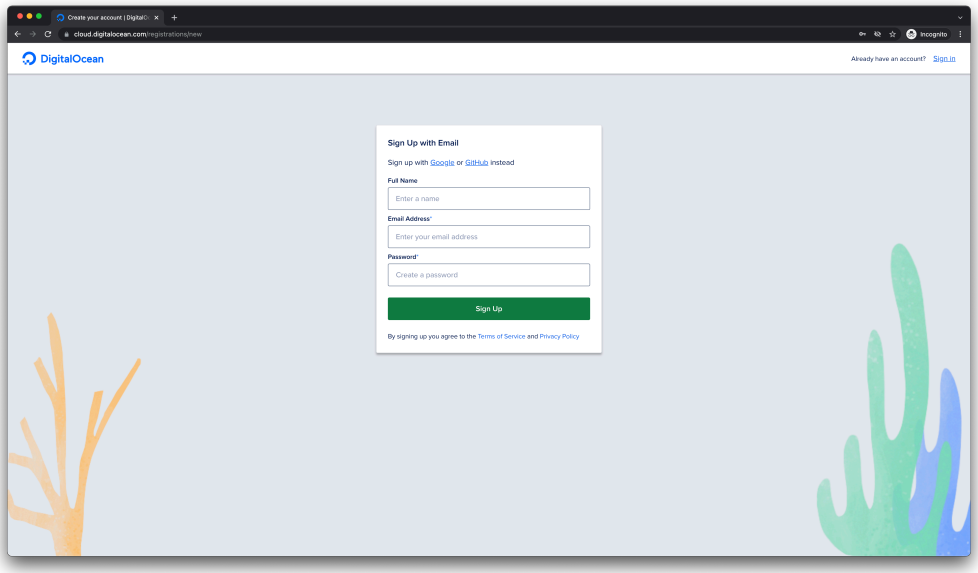
- **javax.validation.validation-api** – validácia DTO (Data Transfer Object) na úrovni Spring Beanov,
- **mapstruct** – slúži na generovanie mapovacích tried,
- **lombok** – slúži na automatické generovanie metód v dátových triedach,

- **OpenAPI Generátor a jeho závislosti** – slúži na generovanie zdrojového odu z OpenAPI dokumentácie.

6.2 Nastavenie prostredia

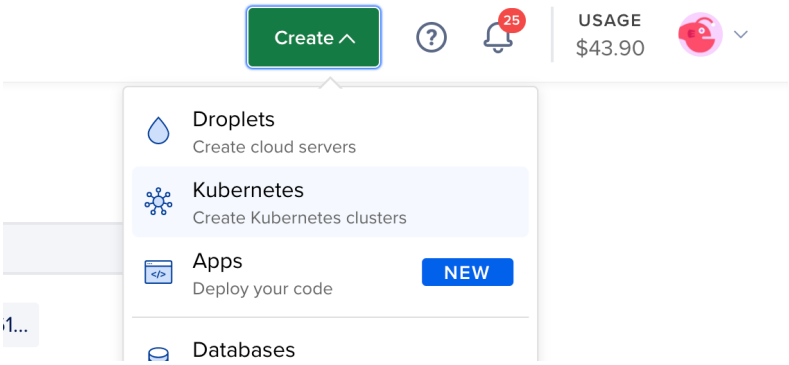
6.2.1 Ručné vytvorenie kubernetes clustera na platforme Digital Ocean

Prvým krokom k vytvoreniu kubernetes clustera na digital ocean je registrácia zákazníckeho účtu. Registrácia je možná na adrese [...].
https://cloud.digitalocean.com/registrations/new
Pri registrácií sú vyžadované: email, heslo a meno



Obr. 2 Registrácia na portáli Digital Ocean
Zdroj: Záznam obrazovky z digitalocean.com – vlastné spracovanie

V prípade nedostatku skúsenosti s programom Terraform je možné vytvoriť cluster aj ručne a nasledujúcu časť preskočiť. Po prihlásení je v pravom hornom rohu dostupné tlačidlo „Create“, ktorým je možné rýchlo vytvoriť niekoľko základných objektov, ktoré Digital Ocean ponúka.



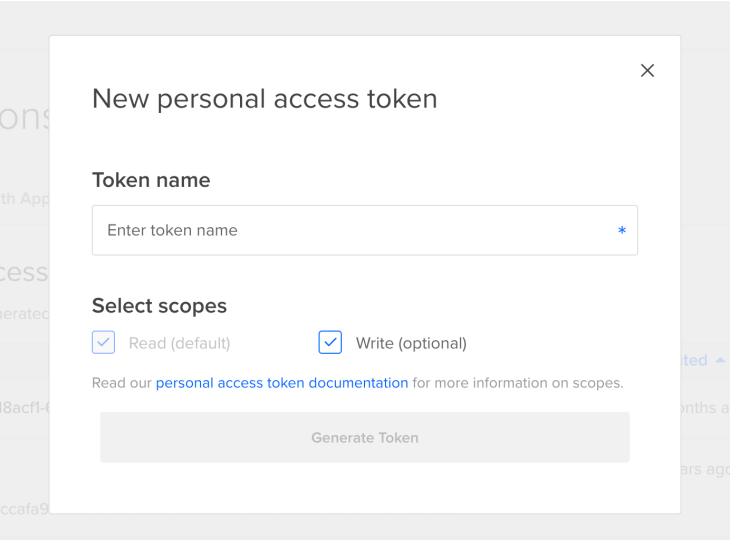
Obr. 3 Rychlé vytvorenie Kubernetes Clustera na Digital Ocean
Zdroj: Záznam obrazovky z digitalocean.com – vlastné spracovanie

6.2.2 Vytvorenie kubernetes clustera pomocou Terraform-u

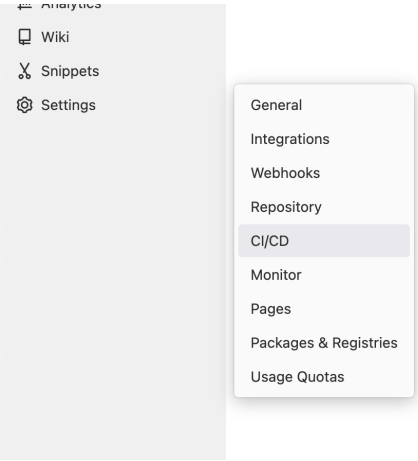
V koreňovom adresári sa nachádza zložka „infrastrucutre“. Táto zložka obsahuje konfiguračné súbory, ktoré je možné použiť pre automatizované vytvorenie infraštruktúry. Tento prístup sa v rôznych publikáciách vyskytuje pod skratkou IaaC, teda infraštruktúra ako kód. V tomto prípade je pre infraštruktúru použitý nástroj Terraform.

O celý proces nasadenia infraštruktúry sa stará gitlab-terraform. Pre jeho správne fungovanie je nutné špecifikovať v GitLab CI/CD (obrázok č. 5) settings premennú TF_VAR_DO_TOKEN (obrázok č. 6). (14)

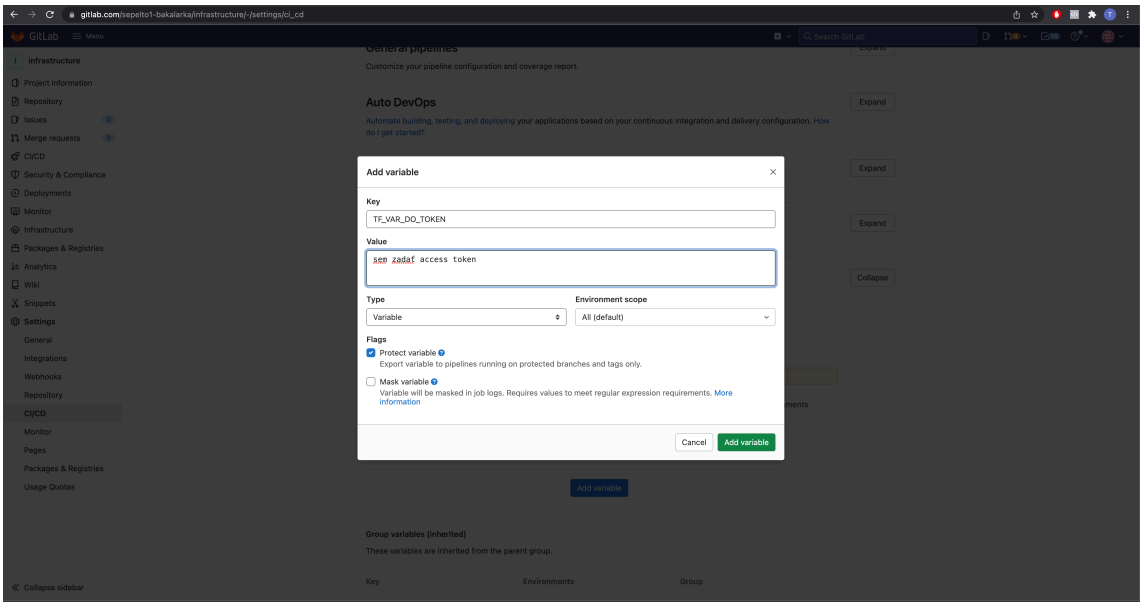
Hodnota tejto premennej je Access Token z Digital Ocean. Access Token je možné vygenerovať na stránke Digital Ocean – obrázok č. 4. (15)



Obr. 4 Vytvorenie Personal Access Token-u
Zdroj: Záznam obrazovky z digitalocean.com – vlastné spracovanie



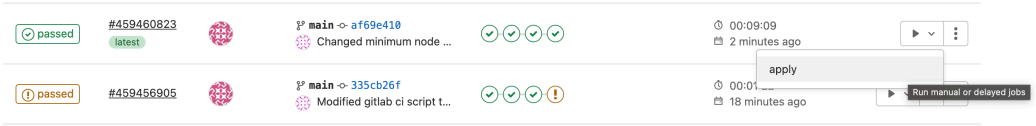
Obr. 5 CI/CD nastavenia v menu na GitLab.com
Zdroj: Záznam obrazovky z gitlab.com – vlastné spracovanie



Obr. 6 Pridanie premennej k skupie/projektu na GitLab.com
Zdroj: Záznam obrazovky z gitlab.com – vlastné spracovanie

Tento repozitár je navrhnutý tak, aby pri odoslaní nového zdrojového kódu na server (GitLab) bola automaticky spustená pipeline. Prvým krokom v pipeline je validácia konfiguračného súboru podľa vstavaných validátorov. Následne sa automaticky spustí plánovanie. V tomto kroku Terraform vytvorí metadata, v ktorých drží informáciu o zmenách v infraštruktúre. Posledný krok pipeliny je

dostupný iba pre zdrojový kód vo vetve main. Pri pipelinach spustených v tejto vetve je dostupné tlačidlo na manuálne nasadenie naplánovaných zmien.

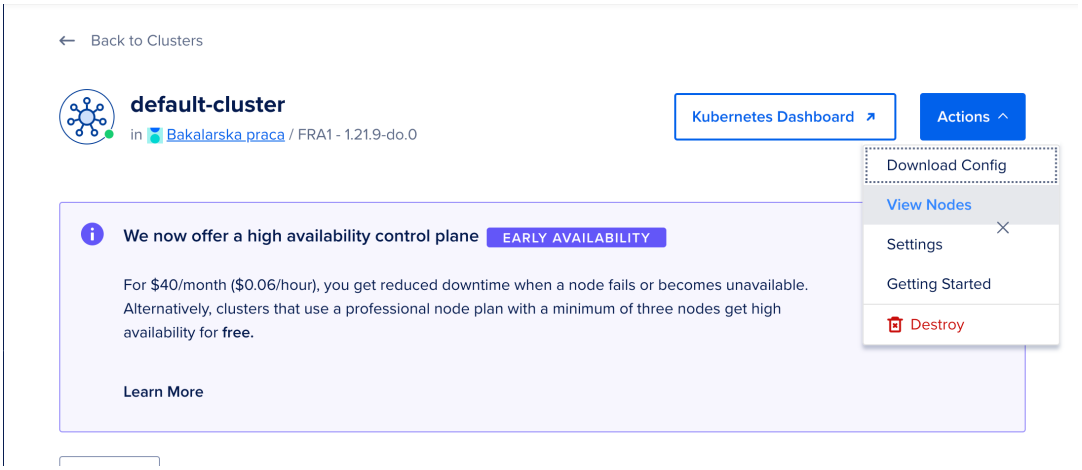


Obr. 7 Manuálne spustenie pipeline na GitLab.com
Zdroj: Záznam obrazovky z gitlab.com – vlastné spracovanie

Výsledok je možné validovať v sekcii Infrastructure > Terraform. V tejto sekcii je možné nájsť všetky pipeliny, ktoré vykonali zmeny v stave infraštruktúry. Kliknutím na tri bodky je možné stiahnuť JSON, ktorý obsahuje výsledok z terraformu.

6.3 Stiahnutie certifikátov

Aby sme boli schopní komunikovať s kubernetes clustrom z lokálneho prostredia pomocou Kubernetes CLI, je nutné stiahnuť certifikáty. Certifikáty sú dostupné v administračnom rozhraní Digital Ocean v detaile clustra.



Obr. 8 Stiahnutie certifikátov na prístup ku kubernetes clustru na Digital Ocean
Zdroj: Záznam obrazovky z digitalocean.com – vlastné spracovanie

Takto stiahnutý konfiguračný súbor je nutné presunúť do zložky ~/.kube/

6.4 Kubectl

Aby bolo možné mať lokálne viacero konfigurácií, tak je odporúčané definovať aktívny kubernetes cluster pomocou lokálnej environmentálnej premennej:

```
export KUBECONFIG=~/.kube/config:~/.kube/<názov configu>
```

Zoznam užitočných príkazov:

kubectl config get-contexts - zobrazí zoznam dostupných kontextov,

kubectl config use-context <názov kontextu> - vyberie kontext,

kubectl apply -f <názov súboru> - vytvorí objekty definované v zdrojovom súbore,

kubectl get <typ objektu> - zobrazí zoznam vybraných objektov,

kubectl describe <typ objektu> <názov objektu> - zobrazí konfiguráciu vybraného objektu.

6.5 Namespace

Keď už máme pripravený kubernetes cluster, tak môžeme vytvoriť nevyhnutné objekty, bez ktorý servisne orientovaná architektúra nemôže fungovať.

Prvým krokom nevyhnutným k segregácii objektov je nutné vytvoriť osobitné prostredie – *namespace*. Namespace je možné chápať ako virtuálne prostredie v rámci clustra. K jeho vytvoreniu slúži konfiguračný súbor uložený v prílohe tejto práce: *kubernetes/namespace.yaml*

Nasadenie namespace je možné vykonať pomocou nástroja kubectl:

```
kubectl apply -f default/namespace.yaml
```

Tento príkaz vytvorí 2 namespaces: *dev* a *ingress-nginx*

6.6 Inštalácia NGINX Ingress Controllera a Load Balancer

Keďže používame klaster kubernetes spravovaný Digital Ocean, na nasadenie je k dispozícii vopred pripravený helm chart:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.34.1/deploy/static/provider/do/deploy.yaml
```

Po úspešnej inštalácii je možné nasadiť vlastný ingress-controller:

```
kubectl apply -f default/ingress-controller.yaml
```

Týmto sa vytvorí aj load balancer na platforme Digital Ocean. Vytvorenie load balancera je možné overiť prostredníctvom administrátorskej konzoly. K tomu, aby bolo možné vykonať ďalšie krok, je potrebné vytvoriť DNS záznamy smerujúcy na LB.
IP Adresa load balancera je dostupná v jeho detaile v administrátorskej konzole. Následne je potrebné vytvoriť DNS záznam typu A na ľubovoľnej subdoméne smerujúcej na danú IP adresu.

6.7 Cert Manager

Rovnako ako ingress controller, tak aj cert manager je možné nainštalovať z preddefinovaného helm chartu:

```
kubectl apply --validate=false -f https://github.com/jetstack/cert-manager/releases/download/v0.16.1/cert-manager.yaml
```

Po úspešnej inštalácii je možné nasadiť vlastný cert manager:

```
kubectl apply -f default/cert-manager.yaml
```

6.8 GitLab Registry Secrets

Aby sa kubernetes cluster bol schopný pripojiť k privátnemu úložisku docker obrazov, je nutné vytvoriť kubernetes secret. Secret je uložený v súbore *default/regcred.yaml*
Hodnota secretu je nasledujúci json, šifrovaný vo formáte base64:

```
{"auths":{"registry.gitlab.com":{"username":"gitlab+deploy-token-<ID>","password":"<password>"}}}
```

Hodnoty je nutné nahradiť vlastným deploy tokenom.
Tento secret je nutné nasadiť do každého namespace, ktorý bude v budúcnosti sťahovať docker obrazy z GitLabu:

kubectl apply -f default/regcred.yaml

kubectl apply -n dev -f default/regcred.yaml

kubectl apply -n prod -f default/regcred.yaml

6.9 Kontrola

V tomto momente by v kubernetes mali byť dostupné všetky základné objekty, bez ktorých architektúra nemôže fungovať. Či všetko správne funguje môžeme overiť nasledujúcimi krokmi:

Validácia namespace-ov:

kubectl -n ingress-nginx get pods

Tabuľka 1 – Očakávané „namespaces“ v kubernetes clusteri

Názov	**Stav**
cert-manager	Active
default	Active
dev	Active
ingress-nginx	Active
kube-node-lease	Active
kube-public	Active
kube-system	Active

Validácia podov v namespace ingress-nginx:

kubectl -n ingress-nginx get pods

Tabuľka 2 – Očakávané „pody“ v namespace ingress-nginx

NAME	**READY**	**STATUS**	**RESTARTS**	**AGE**
ingress-nginx-admission-create-jzhx5	0/1	Completed	0	17m
ingress-nginx-admission-patch-2z29r	0/1	Completed	1	17m

ingress-nginx-controller-6595dc7ff4-bm4dd	1/1	Running	0	17m

Validácia podov v namespace cert-manager:
kubectl -n cert-manager get pods

Tabuľka 3 – Očakávané pody v namespace cert-manager

NAME	READY	STATUS	RESTARTS	AGE
cert-manager-cainjector-fc6c787db-rq94v	1/1	Running	0	13m
cert-manager-d994d94d7-d8lmm	1/1	Running	0	13m
cert-manager-webhook-5c5ddb797c-s9d47	1/1	Running	0	13m

6.10 Pipeline

V rámci teoretickej časti tejto práce bolo opísaných niekoľko krokov, ktoré by mali predchádzať nasadeniu aplikácie. Tieto kroky sú pomerne rutinné a ich manuálne vykonávanie by bolo časovo a finančne nevýhodné. Preto je vhodné využiť automatizačné nástroje, ktoré tieto kroky dokážu vykonávať automatizovane.

6.10.1 GitLab CI/CD

Jedným z takýchto nástrojov je aj GitLab CI/CD. Je to nástroj, ktorý je zabudovaný v platforme GitLab a umožňuje spúšťať rôzne procesy podľa konfigurácie. Konfigurácia by sa mala nachádzať v git repozitári, ktorého sa procesy týkajú a mala by byť zapísaná v súbore gitlab-ci.yaml.

6.10.2 Pipeline pre Maven Knižnicu

Pipeline pre Maven Knižnicu pozostáva iba z 1 kroku – *deploy*. V tomto kroku sa vloží maven konfigurácia do pomocného súboru *ci_settings.xml* a následne sa spustí príkaz *mvn deploy*, ktorý odošle maven image do maven repozitára. Obsah

premennej, ktorá sa vkladá do pomocného súboru je Maven Konfigurácia. Táto konfigurácia je opísaná detailne v súbore *pipeline-library/README.md* a rovnako aj v časti 3.3.4 – maven parent (*~/.m2/settings.xml*)

6.10.3 Pipeline pre službu

Pipeline pre službu je o niečo komplexnejšia ako pre knižnicu a teda pozostáva z 3 krokov: build, package, deploy.

Build

V tomto kroku sa podobne ako v predchádzajúcej sekcii vloží premenná do pomocného súboru, no nasleduje príkaz *mvn package*. Tento príkaz spustí build aplikácie, ktorá sa nachádza vo formáte *.jar* v zložke *target/*. Obsah zložky target je dostupný ako artifact.

Package

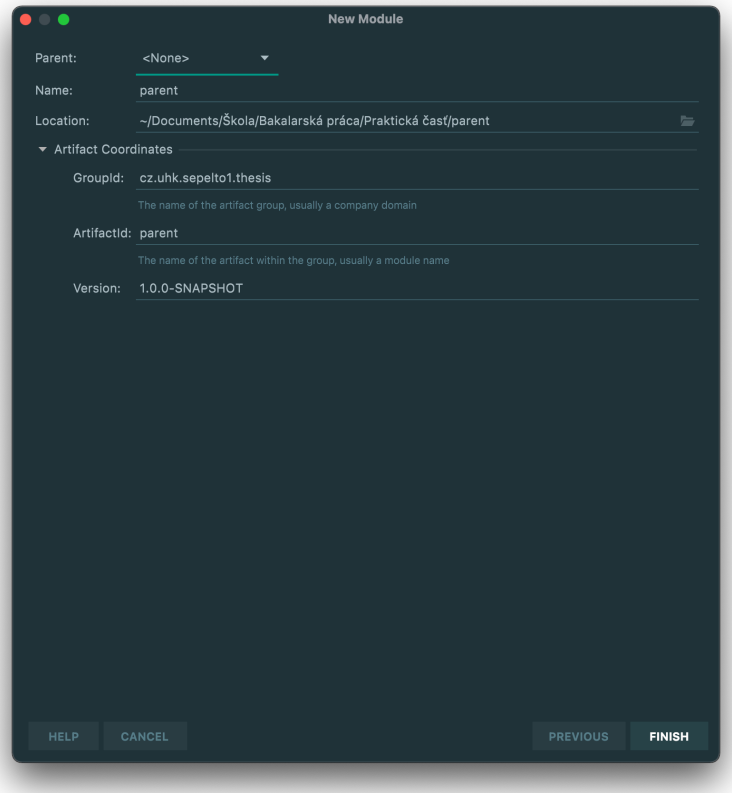
V tomto kroku sa vytvorí docker image z pripraveného artifaktu podľa definície v súbore Dockerfile. Po jeho úspešnom vytvorení sa označí tento image tagom – referenčným číslom pipeliny – a odošle sa do docker registra.

Deploy

Posledným krokom pipeliny je deploy. V tomto kroku sa vytvoria objekty podľa konfigurácie v súboroch v zložke *k8s/<prostredie>* v projektovom repozitári. Jedným zo súborov je *service.yaml*, ktorý obsahuje informáciu o obraze, ktorý sa ma stiahnuť. Súčasťou tohto kroku je nahradenie *VERSION_PLACEHOLDER* hodnoty referenčným číslom pipeliny – toto číslo sa zhoduje s referenčným číslom v predchádzajúcom kroku a teda sa zhoduje aj verzia docker image-u.

6.11 Maven Parent

Nový maven projekt si môžeme inicializovať pomocou IntelliJ IDEA. Sekciu Project Structure otvoríme stlačením ⌘ + ;. Kliknutím na tlačidlo + sa zobrazia možnosti Import Module a New Module. Vyberieme New Module, čím sa otvorí dialógové okno, ktoré môžeme vyplniť podľa predlohy na obrázku č. 9.



Obr. 9 Vytvorenie maven parent projektu
Zdroj: Záznam obrazovky z Intellij IDEA – vlastné spracovanie

Týmto procesom inicializujeme minimálny maven projekt. Následne v pom.xml definujeme niekoľko nastavení:

```
<packaging>pom</packaging>
```

Toto nastavenie definuje najjednoduchší typ maven projektu. Jeho životný cyklus sa skladá iba z 2 operácií: install a deploy. Výsledkom je maven artefakt, ktorý je možné použiť ako agregátor alebo rodič.

```
<properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <java.version>17</java.version>
        ...
</properties>
```

V tomto bloku budeme definovať konfiguračné premenné, ako napríklad verzie použitých balíčkov, či iných premenných použitých v subprojektoch.

```
<repositories>
    <repository>
        <id>gitlab-maven</id>
        <url>https://gitlab.com/api/v4/projects/.../packages/maven</url>
    </repository>
</repositories>
```

Repozitár je miesto, kde sa ukladajú maven artefakty. Maven sa prv pokúsi vyhľadať artefakt v lokálnom maven repozitári (~/.m2/) a následne vo vzdialenom. Táto sekcia nakonfiguruje, kde má maven hľadať uložené artefakty. Do URL je potrebné doplniť ID projektu v GitLab-e. Dôležité je, aby mal maven prístup k tomuto repozitáru. V prípade použitia GitLab-u je možné v lokálnom konfiguračnom súbore definovať autorizáciu formou HTTP hlavičky:

```
nano ~/.m2/settings.xml
<settings ...>
        <servers>
            <server>
            <id>gitlab-maven</id>
            <configuration>
                <httpHeaders>
                    <property>
                        <name>Private-Token</name>
                        <value>{PAT}</value>
                    </property>
                </httpHeaders>
            </configuration>
 ...
```

Kde {PAT} je potrebné nahradiť Personal Access Token-om z GitLabu. Token musí mať prístup ka čítanie a zápis do registrov. PAT je možné nahradiť aj deploy tokenom, či CI tokenom (len pre pipeliny).
Podobne ako <*repositories*> je potrebné nastaviť aj <*pluginRepositories*> a <*distributionManagement*>. Viď priložené súbory.

6.12 Zabezpečenie komunikácie medzi frontendom a API GW

Pre zabezpečenie aplikácie bol zvolený softvér Keycloak. Keycloak je Identity Provider aktívne vyvíjaný spoločnosťou RedHat. Pre tento projekt bola zvolená najnovšia verzia 17.0, ktorá je postavená na Quarkuse. Predchádzajúce verzie tohto softvéru boli postavené na aplikačnom serveri WildFly. Keycloak podporuje SSO (Single sign on), protokoly SAML, OIDC a ďalšie, ktoré nie sú pre túto prácu dôležité. Základným prvkom keycloaku je realm. Realm je prostredie, ktoré obsahuje set klientov, používateľov a oprávnení. Okrem API Keycloak poskytuje aj používateľské rozhranie pre prihlásenie a administráciu.
Používateľ, ktorý v prehliadači otvorí podstránku, ktorá vyžaduje prihlásenie bude automatický presmerovaný na prihlasovací formulár a nebude mu umožnené načítať obsah bez prihlásenia. Po prihlásení sa do cookies uloží token, ktorý je následne nutné odoslať ako Bearer autorizačnú hlavičku na API GW. API GW následné overí autorizačný token voči Keycloaku.
Pre nasadenie keycloaku na kubernetes server je nutné vytvoriť Dockerfile, ktorý pozostáva z 2 krokov: build, run. Celý proces je dostupný na stránke Keycloaku. (16)

6.13 OpenAPI

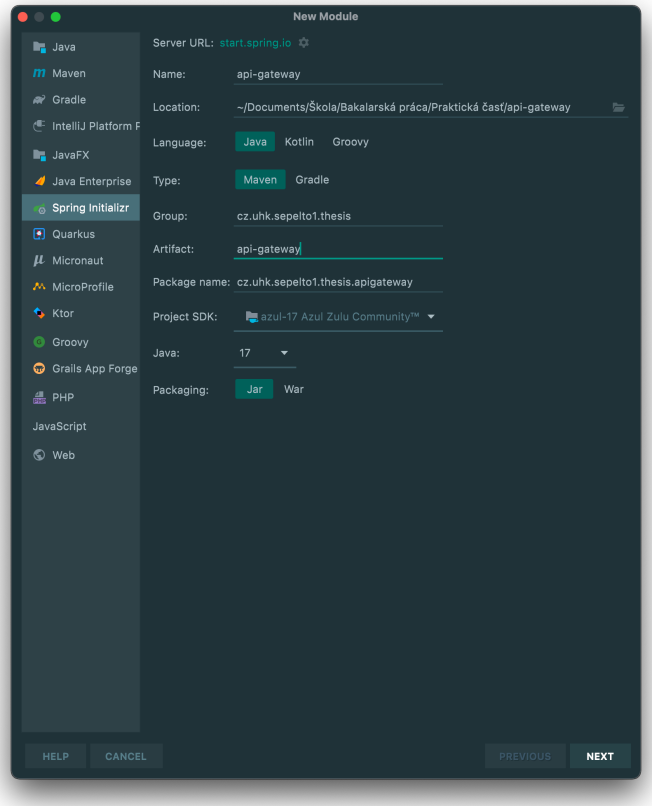
OpenAPI je štandard, ktorý slúži na opis REST API. V jeho definícií sa nachádzajú dostupné endpointy, dátové modely, informácie o ich zabezpečení a metadata o API (autor, licencia, popis, ...).
Pre účely tejto práce bol použitý tento štandard v kombinácií s API-first dizajnom. API-first znamená, že vývojár alebo analytik najprv definuje, ako budú jednotlivé API vyzerať a následne sa z tejto definície automaticky vygenerujú zdrojové kódy potrebné pre chod aplikácie. V prípade serverových služieb sa vygenerujú DTO

a interface pre controllery. V prípade klienta sa vygeneruje klient (axios/webclient) a DTOs.

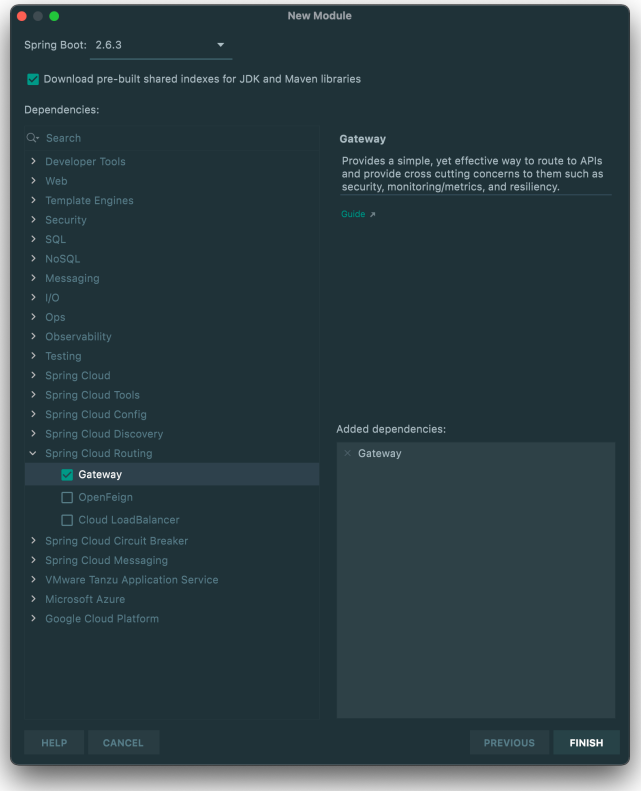
6.14 Spring API Gateway

Princíp fungovania tejto služby je opísaný v sekcií 3.3 tejto práce. Táto sekcia slúži ako opis implementačných detailov tejto služby.

Pre vytvorenie Spring Boot aplikácie je možné použiť Spring Initializr, ktorý je súčasťou Intellij IDEA. Postup je zaznamenaný na nasledujúcich obrázkoch.



Obr. 10 Vytvorenie projektu pomocou nástroja Spring Initializr
Zdroj: Záznam obrazovky z Intellij IDEA – vlastné spracovanie



Obr. 11 Výber závislostí pre projekt
Zdroj: Záznam obrazovky z Intellij IDEA – vlastné spracovanie

Trieda SecurityConfig

Táto trieda slúži k nastaveniu bezpečnostných filtrov.

@EnableWebFluxSecurity

Táto anotácia pridá podporu reaktívnej WebFlux Security

Metóda springSecurityFilterChain vytvorí Bean typu SecurityWebFilterChain a zabezpečí filtrovanie prichádzajúcich požiadaviek. V prvej časti tejto funkcie je definovaný reaktívny JWT Converter Adapter, ktorý konvertuje role z JWT tokenu na kolekciu GrantedAuthority. Tie je možné používať ďalej v aplikácii.

Trieda SecurityFilterFactory

Táto trieda definuje vlastný filter, ktorý slúži na propagáciu nových hlavičiek z pôvodného JWT Tokenu.

application.yaml

Konfigurácia Spring Cloud Gateway sa nachádza v resources zložke, v súbore application.yaml

Tento konfiguračný súbor obsahuje nasledujúce informácie:

```
spring:
 security:
  oauth2:
   resourceserver:
    jwt:
     jwk-set-uri: http://keycloak:8080/realms/sepelto1/protocol/openid-connect/certs
```

Tento blok konfigurácie definuje adresu JWK Setu, voči ktorému prebieha validácia JWT Tokenu.

```
 cloud:
  gateway:
   globalcors:
    cors-configurations:
     "[/**]":
      allowedOrigins: "*"
      allowedHeaders: "*"
      allowedMethods:
       - GET
       - POST
       - PUT
       - PATCH
       - DELETE
       - OPTIONS
```

Táto sekcia definuje konfiguráciu CORS hlavičiek – v tomto prípade povolí prístup odkiaľkoľvek.

```
default-filters:
 - App
 - RemoveRequestHeader=Authorization,Cookie,Set-Cookie
 - StripPrefix=1
```

Táto sekcia definuje prednastavené filtre. RemoveRequestHeader zabezpečuje, že hlavičky Authorization, Cookie a Set-Cookie nie sú ďalej propagované na ďalšie mikroslužby.

```
routes:
 - id: products
   uri: http://products-service:8080
   predicates:
    - Path=/products/**
 - id: orders
   uri: http://orders-service:8080
   predicates:
    - Path=/orders/**
 - id: statistics
   uri: http://statistics-service:8080
   predicates:
    - Path=/statistics/**
```

V tejto časti sú definované jednotlivé cesty k mikroslužbám, rovnako aj adresy jednotlivých mikroslužieb a pravidlá pre ich smerovanie.

```
management:
 endpoints:
  web:
   exposure:
    include: '*'
 endpoint:
  health:
   show-details: always
```

```
server:
  port: 8081
```

Tento posledný blok kódu definuje konfiguráciu actuatora.

6.15 Spring mikroslužby

Táto sekcia sa bude venovať generovaniu modelov a controllerov z OpenAPI špecifikácie pre Spring mikroslužby a procesu ich nasadenia.

6.15.1 Generovanie DTO

Základom každej mikroslužby je interface pre RESTovú komunikáciu, ktorý je generovaný z OpenAPI dokumentácie. O generovanie týchto zdrojových súborov sa stará maven plugin *openapi-generator-maven-plugin*. Konfigurácia tohto pluginu vyzerá nasledujúco:

```
<plugin>
        <groupId>org.openapitools</groupId>
        <artifactId>openapi-generator-maven-plugin</artifactId>
        <version>${version.openapi-generator-plugin}</version>
        <configuration>
          <generatorName>spring</generatorName>
          <library>spring-mvc</library>
          <modelNameSuffix>DTO</modelNameSuffix>
          <generateApiTests>false</generateApiTests>
          <generateModelTests>false</generateModelTests>
          <generateSupportingFiles>true</generateSupportingFiles>
          <output>${project.build.directory}/generated-sources</output>
          <configOptions>
            <java8>true</java8>
            <dateLibrary>java8</dateLibrary>
            <interfaceOnly>true</interfaceOnly>
            <useTags>true</useTags>
            <sourceFolder>swagger</sourceFolder>
            <serializableModel>true</serializableModel>
```

```
            </configOptions>
            <typeMappings>
              <typeMapping>OffsetDateTime=Instant</typeMapping>
            </typeMappings>
            <importMappings>

<importMapping>java.time.OffsetDateTime=java.time.Instant</importMapping>
            </importMappings>
          </configuration>
        </plugin>
```

Konfiguračné parametre:

- **generatorName** – názov generátora použitého pre vygenerovanie kódu,
- **library** – informácia o šablóne, ktorá sa má použiť pre generovanie,
- **modelNameSuffix** – prípona k názvu súboru,
- **generateApiTests** – true/false hodnota, konfiguruje generovanie testov pre API rozhranie,
- **generateModelTests** – true/false hodnota, konfiguruje generovanie testov pre modely,
- **generateSupportingFiles** – true/false hodnota, konfiguruje generovanie podporných súborov,
- **output** –cesta, kam sa vygenerujú súbory ,
- **typeMappings –** informácia o pretypovaní dátových typov.

6.15.2 Dockerfile

Pre všetky mikroslužby je spoločný dockerfile, ktorý pozostáva z 8 krokov:

- **FROM** – informácia o základnom Docker obraze,
- **ENV** – nastavenie env premennej, ktorá slúži pre nastavenie Spring prostredia,
- **RUN** – nainštaluje nástroje tini a bash,
- **SHELL** – nastaví prednastavený shell na bash,

- **VOLUME** – nastaví zložku, v ktorej sa vykonávajú ďalšie príkazy,
- **ENTRYPOINT** – nastaví vstupný bod pre kontajner tini,
- **COPY** – skopíruje aplikáciu z target zložky do rootu,
- **CMD** – spustí java aplikáciu.

6.15.3 Deployment

Deployment Springovej mikroslužby pozostáva zo 4 kubernetes objektov:

- Service pre aktuátor
- Service pre aplikáciu
- Deployment
- Configmap

Keycloak, API GW a Webapp sú dostupné z internetu a preto potrebujú aj ingress.

7 Zhodnotenie výsledkov

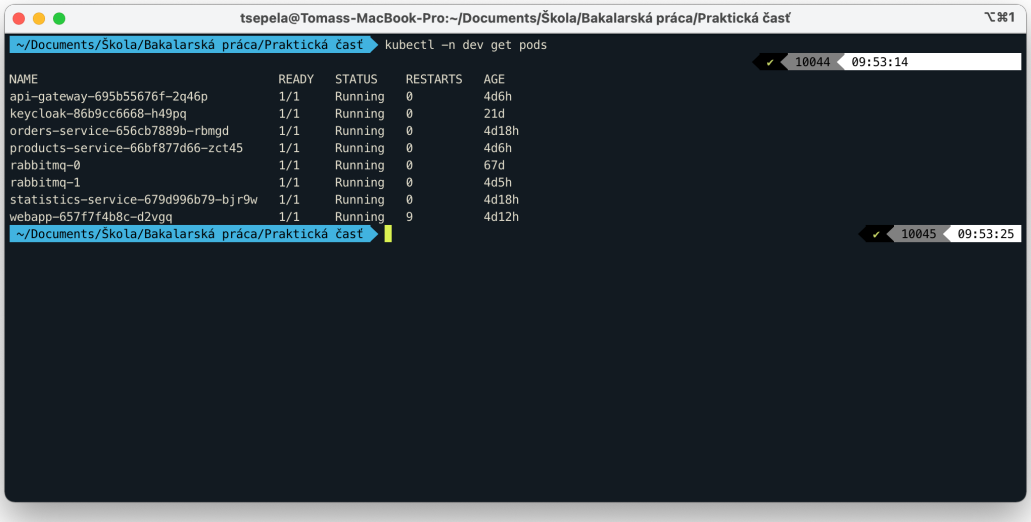
V nasledujúcej sekcii sa nachádza zhodnotenie funkčnosti systému. Zhodnotenie je možné vykonať formou porovnania akceptačných kritérií a reality.

Kritérium: Mikroslužby sú nasadené v kubernetes a sú v stave „Running“.

Overenie tohto akceptačného kritéria je možné príkazom

kubectl -n dev get pods

ktorý zobrazí všetky pody (mikroslužby v jednej replike) v pracovnom namespace, ako je možné vidieť na obrázku 12.



Obr. 12 Výpis podov v namespace "dev"
Zdroj: Záznam obrazovky z iTerm (terminál) – vlastné spracovanie

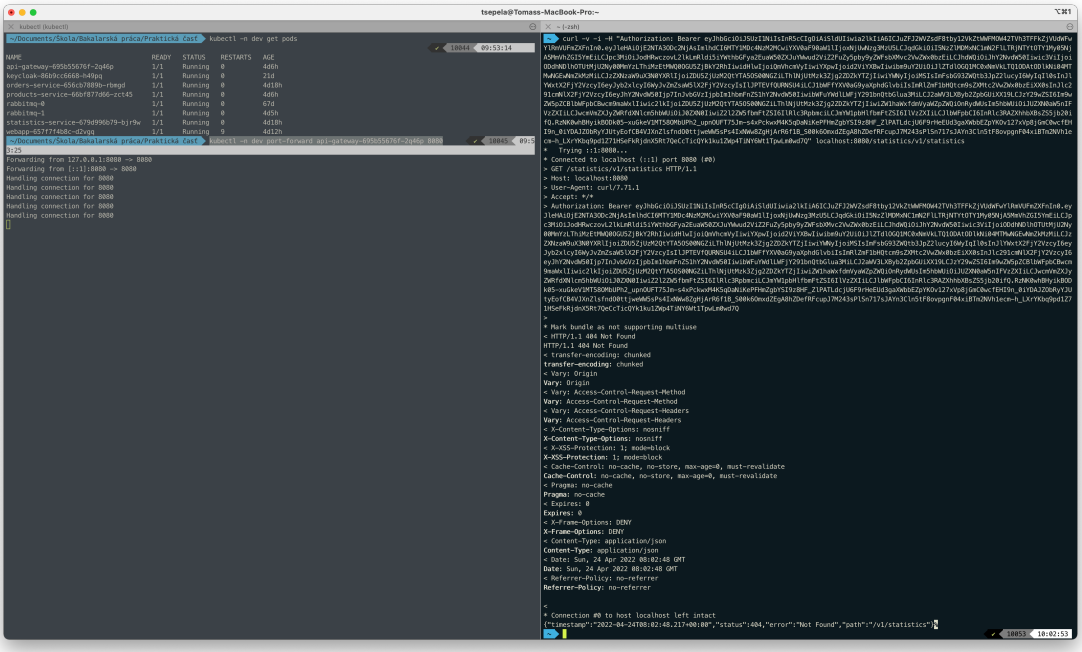
Kritérium: Mikroslužby môžu prijímať requesty z API Gateway.

Overenie tohto kritéria spočíva v dvoch krokoch:

1. Vytvorenie tunelu medzi lokálnym prostredím a podom v kubernetes clustri za pomoci príkazu

kubectl -n dev port-forward api-gateway-695b55676f-2q46p 8080

2. Zavolanie koncového bodu niektorej mikroslužby, napríklad cez curl (obrázok č. 13)



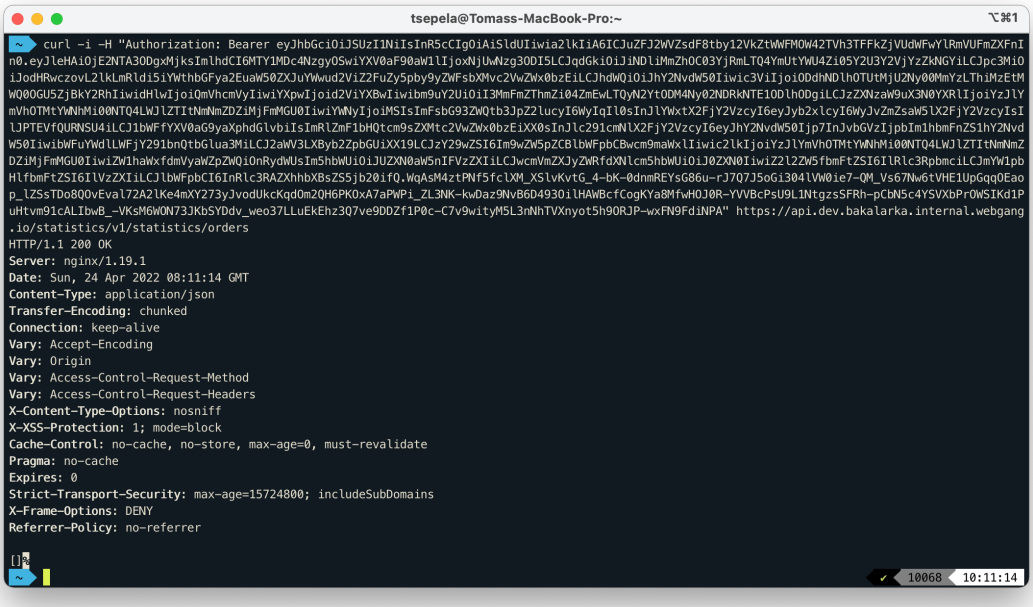
Obr. 13 Zavolanie endpointu štatistik cez terminál príkazom curl
Zdroj: Záznam obrazovky z iTerm (terminál) – vlastné spracovanie

Kritérium: API Gateway je dostupná z internetu.

Overenie je možné zavolaním endpointu pomocou príkazu curl.

curl -i -H "Authorization: Bearer ACCESS_TOKEN"
https://api.dev.domena.tld/statistics/v1/statistics/orders

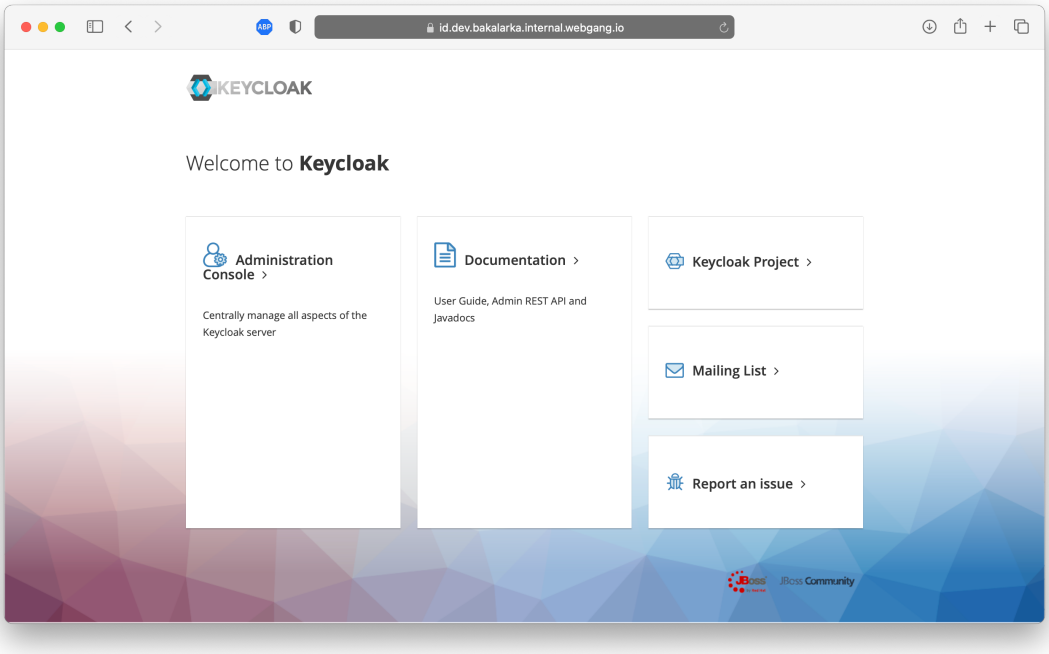
V odpovedí je očakávaná hlavička 200 OK.



Obr. 14 Zavolanie externého endpointu štatistik cez terminál príkazom curl
Zdroj: Záznam obrazovky z iTerm (terminál) – vlastné spracovanie

Kritérium: Keycloak je dostupný z internetu.

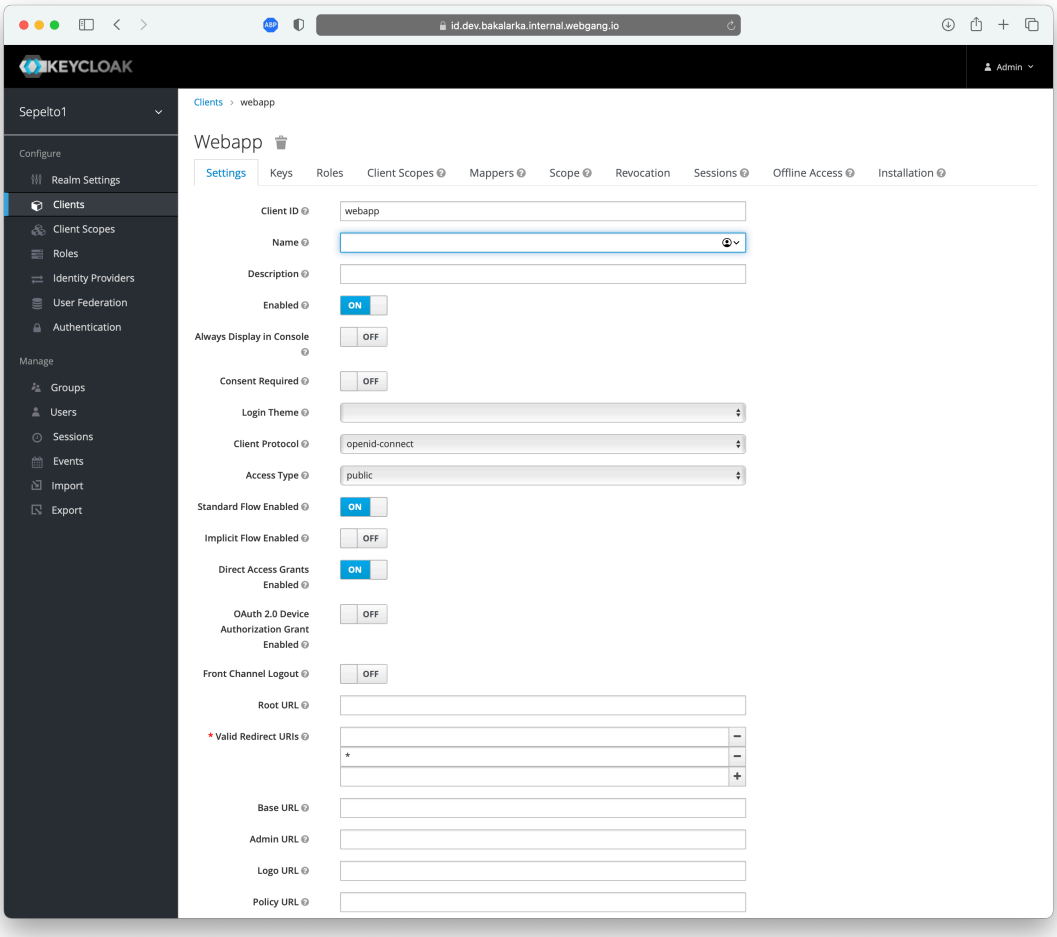
Toto akceptačné kritérium je možné overiť priamo z webového prehliadača navštívením adresy, na ktorú boli nasmerované DNS záznamy keycloaku a na ktorej je nastavené reverzné proxy. Na domovskej obrazovke keycloaku by sa mala zobraziť obrazovka ako na obrázku č. 15. V produkčnom prostredí je odporúčané pomocou ingressu zablokovať prístup na túto podstránku.



Obr. 15 Uvítacia obrazovka keycloaku
Zdroj: Záznam obrazovky zo Safari (Keycloak) – vlastné spracovanie

Kritérium: V rozhraní keycloak sa nachádza klient pre frontend:

Rovnako ako pri predchádzajúcom overení aj toto je možné vykonať pomocou webového rozhrania. Po kliknutí na odkaz „Administration Console“ z predchádzajúceho kroku a následného prihlásenia prihlasovacími údajmi špecifikovanými v deploymente Keycloak-u je možné zobraziť v administrácii kartu Clients, v ktorej sa zobrazia všetci klienti registrovaní na príslušnom realme. V prípade tejto práce sa tam nachádza client „webapp“, ktorého detail je možné vidieť na obrázku č. 16.



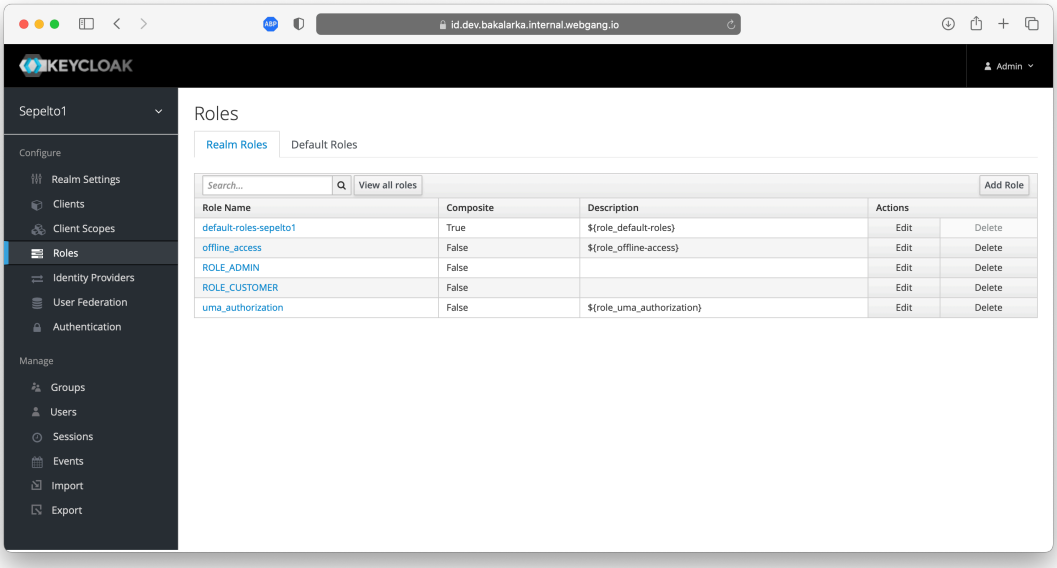
Obr. 16 Detail client-a "webapp" v administrácii Keycloak-u
Zdroj: Záznam obrazovky zo Safari (Keycloak) – vlastné spracovanie

Kritérium: V rozhraní keycloak sa nachádzajú minimálne role:

ROLE_ADMIN (ďalej len Administrátor),

ROLE_CUSTOMER (ďalej len Zákazník).

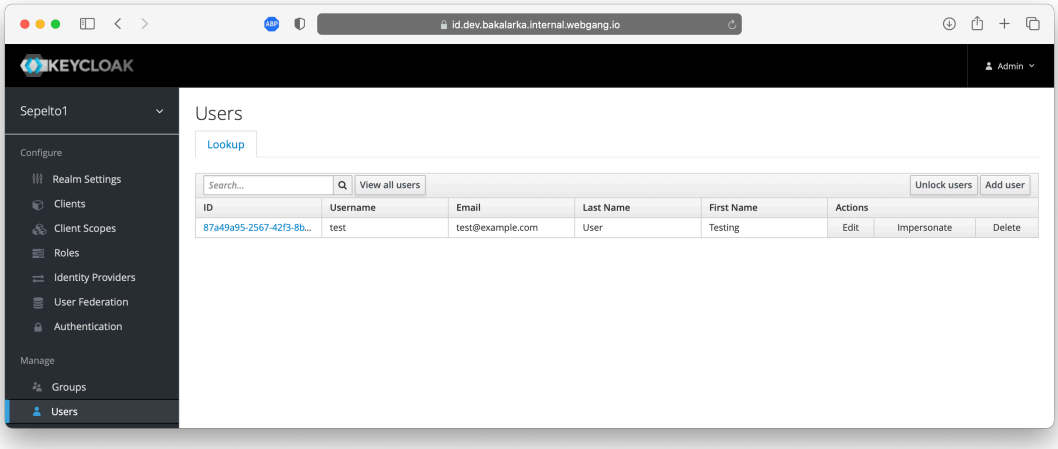
Overenie: Po prihlásení do administračného rozhrania Keycloak-u, podobne ako v predchádzajúcich krokoch, je možné zobraziť kartu Roles, v ktorej sa nachádzajú administrátorom definované role. Okrem iných sa tu nachádzajú aj role definované akceptačným kritériom – viditeľné na obrázku č. 17.



Obr. 17 Zoznam roli v administrácii Keycloak-u
Zdroj: Záznam obrazovky zo Safari (Keycloak) – vlastné spracovanie

Kritérium: Administrátor môže vytvoriť používateľský profil v rozhraní keycloak.

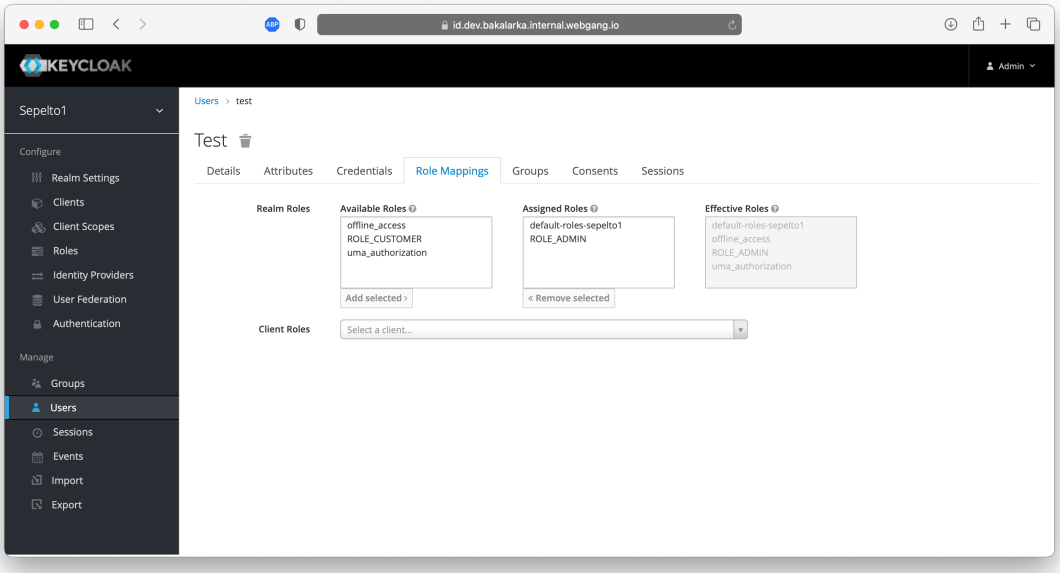
Overenie: Po prihlásení do administračného rozhrania Keycloak-u, podobne ako v predchádzajúcich krokoch, je možné zobraziť kartu Users, v ktorej sa nachádzajú používatelia – viditeľné na obrázku č. 17. Vytvorenie používateľa je možné kliknutím na tlačidlo „Add user“.



Obr. 18 Zoznam používateľov v administrácii Keycloak-u
Zdroj: Záznam obrazovky zo Safari (Keycloak) – vlastné spracovanie

Kritérium: Administrátor môže priradiť role používateľovi v rozhraní Keycloak.

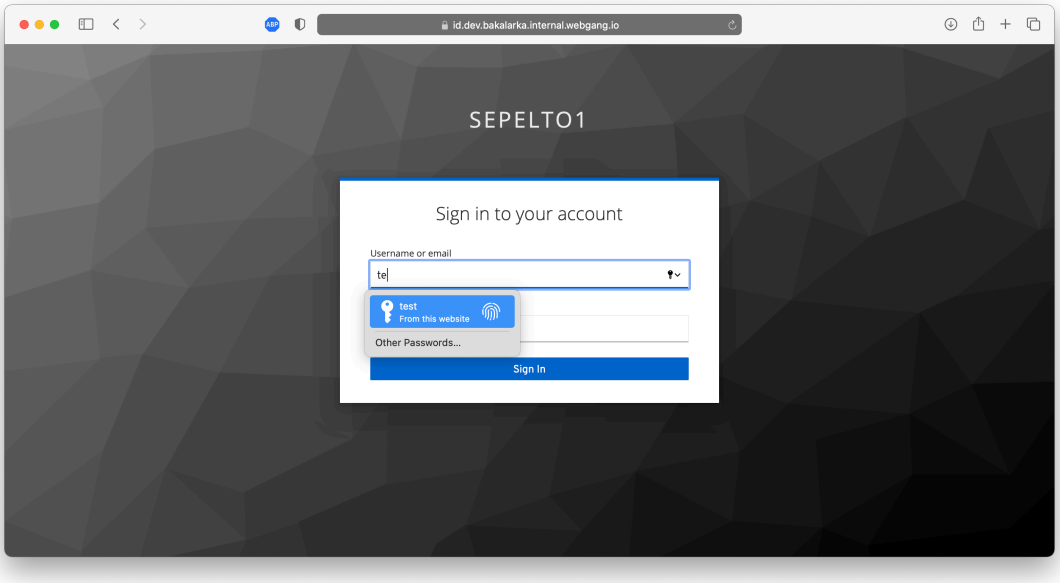
Overenie: Po kliknutí na ID používateľa sa zobrazí formulár na jeho úpravu. Na vrchnej lište sa nachádza karta „Role Mappings“ v ktorej je možné priradiť používateľovi preddefinované role. Používateľ na obrázku č. 19 má priradenú rolu Administrátor.



Obr. 19 Priradenie role používateľovi v rozhraní Keycloak
Zdroj: Záznam obrazovky zo Safari (Keycloak) – vlastné spracovanie

Kritérium: Používateľ sa môže prihlásiť.

Pri pokuse navštíviť podstránku /admin prostredníctvom frontend-u, ktorý je súčasťou tento práce v zložke webapp, dôjde k automatickému presmerovaniu na stránku prihlásenia pre používateľa – obrázok č. 20. Prihlásenie do tejto časti je možné pre používateľov, ktorí majú rolu Administrátor.



Obr. 20 Prihlásenie do systému pre Administrátora
Zdroj: Záznam obrazovky zo Safari (Webapp - Keycloak) – vlastné spracovanie

Nasledujúce akceptačné kritéria budú validované len za pomoci CURL príkazov. Alternatívnou metódou overenia je Webapp frontend, ktorý je súčasťou tejto práce, a ktorého záznamy obrazoviek sú dostupné v prílohe č. 2. Hlavičky doplnené prehliadačom sú pre prehľadnosť vynechané.

Kritérium: Anonymný zákazník môže vytvoriť objednávku.

CURL:

```
curl 'https://api.dev.domena.tld/orders/v1/orders' \
-X 'POST' \
--data-binary
'{"items":[{"productId":3,"quantity":1},{"productId":5,"quantity":2}],"fullName":"Tomas"}'
```

Odpoveď:

```
HTTP/1.1 200 OK
```

Obr. 21 Zoznam objednávok obsahuje aj novo vytvorenú objednávku
Zdroj: Záznam obrazovky zo Safari (Network Inspector) – vlastné spracovanie

Na obrázku č. 21 je možné vidieť, že vytvorená objednávka bola zapísaná do databázy a vrátená vo výpise. Objednávková služba, okrem iného, vráti aj cenu produktov, ktorú získala REST volaním na produktovú službu.

Kritérium: Administrátor vidí objednávky.

CURL:

```
curl --location --request GET 'https://api.dev.domena.tld/orders/v1/orders'
--header 'Authorization: Bearer ...TOKEN...‘
```

Odpoveď JSON (200 OK):

```
{
  "results": [
    {
      "id": 1,
      "fullName": null,
      "items": [
        {
          "id": "e7646e72-692a-47bf-93ee-199db256281a",
          "productId": 1,
          "quantity": 1,
          "price": 350.0
        },
        {
          "id": "c9257a16-e249-4778-ba4d-82dbb80f8971",
          "productId": 3,
          "quantity": 2,
          "price": 1200.0
        },
        ...
```

Kritérium: Administrátor môže upravovať objednávky.

CURL:

```
curl --location --request PUT 'https://api.dev.domena.tld/orders/v1/orders/10' \
--header 'Authorization: Bearer ...TOKEN...“ ' \
--data-raw '{
        "fullName": "Any customer",
   "items": [
      {
         "id": "556343fa-dc22-43b0-9524-00790f99f8ed",
         "productId": 3,
         "quantity": 10,
         "price": 1200.0
      },
      {
         "id": "de8c90ca-3e94-40e3-81bb-ae96cd4c1239",
         "productId": 4,
         "quantity": 10,
         "price": 900.0
      }
   ]
}'
```

Odpoveď JSON (200 OK):

```
{
   "id": 10,
   "fullName": "Any customer",
   "items": [
      {
         "id": "556343fa-dc22-43b0-9524-00790f99f8ed",
         "productId": 3,
         "quantity": 10,
```

```
        "price": 1200.0
      },
      {
        "id": "de8c90ca-3e94-40e3-81bb-ae96cd4c1239",
        "productId": 4,
        "quantity": 10,
        "price": 900.0
      }
    ],
    "totalPrice": 21000.0,
    "createdDate": "2022-04-19T18:42:54.416Z",
    "lastModifiedDate": "2022-04-24T09:56:49.912929528Z"
}
```


Kritérium: Administrátor môže mazať objednávky.

CURL:

```
curl --location --request DELETE 'https://api.dev.domena.tld/orders/v1/orders/10' \
--header 'Authorization: Bearer ...TOKEN...“ '
```

Odpoveď:

```
HTTP 204 No Content
```

Kritérium: Administrátor vidí produkty.

CURL:

```
curl --location --request GET 'https://api.dev.domena.tld/products/v1/products'
--header 'Authorization: Bearer ...TOKEN...“ '
```

Odpoveď:

```
{
   "results": [
      {
         "id": 5,
         "name": "Šál",
         "description": "Šál - produkt s diagritikou",
         "sku": "sal",
         "price": 13.0,
         "createdDate": "2022-04-20T06:09:54.719Z",
         "lastModifiedDate": "2022-04-20T06:09:54.719Z"
      },
      ...
   ],
   "paging": {
      "pageNumber": 0,
      "pageSize": 20,
      "totalElements": 3
   }
}
```

Kritérium: Administrátor môže upravovať produkty.

CURL:

```
curl --location --request PUT 'https://api.dev.domena.tld/products/v1/products/5' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: Bearer ...TOKEN...' \
--data-raw '{
        "name": "Upravený produkt",
   "description": "Šál po úprave",
   "sku": "upraveny-produkt",
   "price": 17.0
}'
```

Odpoveď:

```
{
   "id": 5,
   "name": "Upravený produkt",
   "description": "Šál po úprave",
   "sku": "upraveny-produkt",
   "price": 17.0,
   "createdDate": "2022-04-20T06:09:54.719Z",
   "lastModifiedDate": "2022-04-24T10:04:26.578687647Z"
}
```

Kritérium: Administrátor môže mazať produkty.

CURL:

```
curl --location --request DELETE
'https://api.dev.domena.tld/products/v1/products/5' \
--header 'Authorization: Bearer  ... TOKEN ...‘
```

Odpoveď:

```
HTTP 204 No Content
```

Kritérium: Administrátor môže vytvárať produkty.

CURL:

```
curl --location --request POST 'https://api.dev.domena.tld/products/v1/products' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: Bearer ...TOKEN... \
--data-raw '{
        "name": "dolore elit aliqua sed deserunt",
        "description": "occaecat dolor reprehender",
        "sku": "sed",
        "price": 123
}'
```

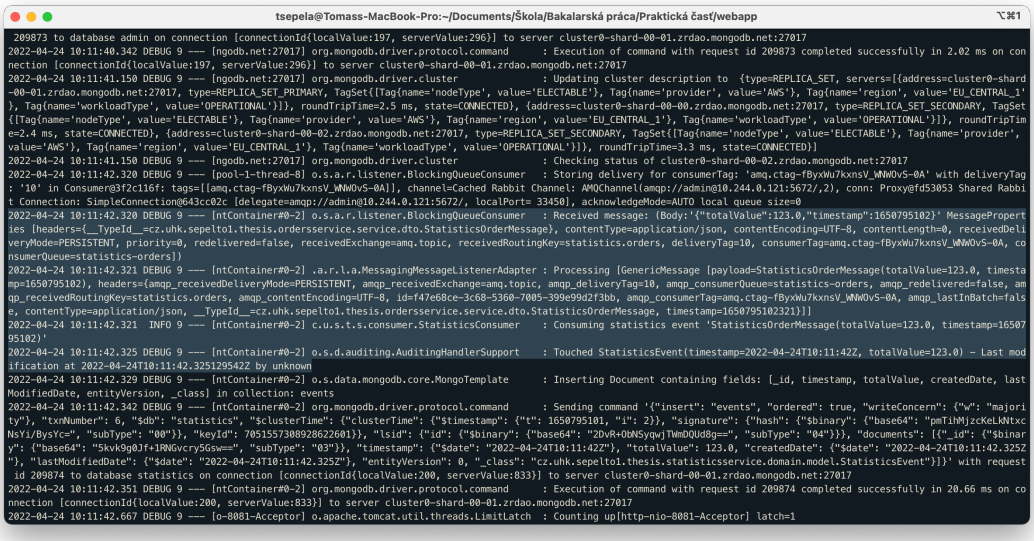
Odpoveď:

```
{
   "id": 6,
   "name": "dolore elit aliqua sed deserunt",
   "description": "occaecat dolor reprehender",
   "sku": "sed",
   "price": 123.0,
   "createdDate": "2022-04-24T10:07:23.619008636Z",
   "lastModifiedDate": "2022-04-24T10:07:23.619008636Z"
}
```

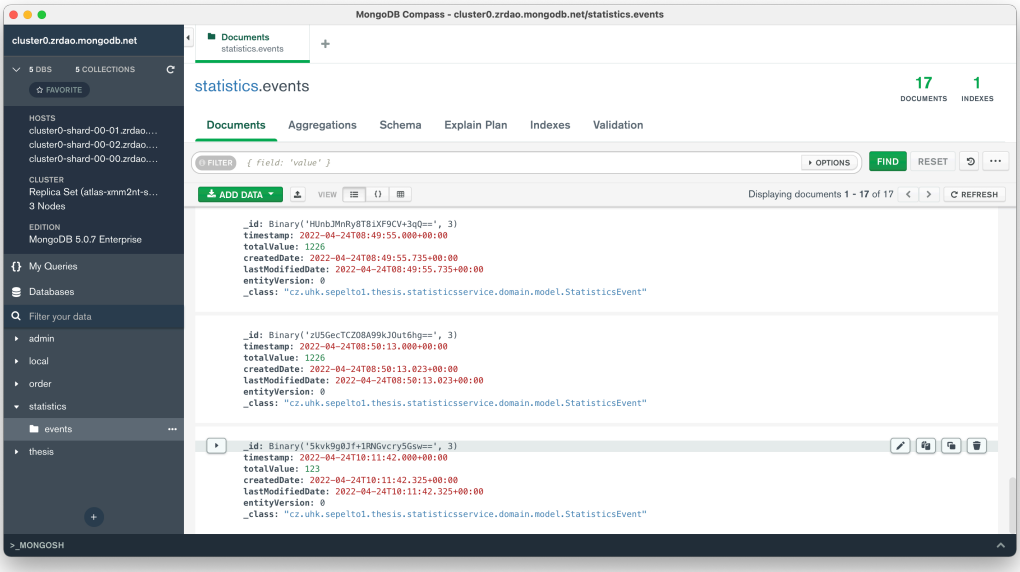
Na odpovedi je možné vidieť, že novému produktu bolo automaticky vygenerované ID a hodnoty auditor-a – createdDate, lastModifiedDate.

Kritérium: Systém asynchrónne vytvorí udalosť – štatistiku pri vytvorení objednávky.

Po zobrazení logov zo štatistickej mikroslužby je možné vidieť, že služba prijala správu z RabbitMQ (obraázok č. 22) a následne ju uložila do databázy (obrázok č. 23).



Obr. 22 Štatistická služba prijala správu z RabbitMQ a uložila ju do DB
Zdroj: Záznam obrazovky z iTerm (terminál) – vlastné spracovanie



Obr. 23 Zoznam štatistických události v databáze
Zdroj: Záznam obrazovky z MongoDB Compass – vlastné spracovanie

Kritérium: Štatistické eventy sú perzistentné a imutabilné.

Overenie: Štatistický event je uložený v DB (obrázok č. 23) a neexistuje API Endpoint, ktorým by ho bolo možné upraviť.

Kritérium: Administrátor vidí štatistiky.

CURL:

```
curl --location --request GET
'https://api.dev.domena.tld/statistics/v1/statistics/orders?from=0&until=1650795896754' \
--header 'Accept: application/json' \
--header 'Authorization: Bearer ...TOKEN...'
```

Odpoveď:

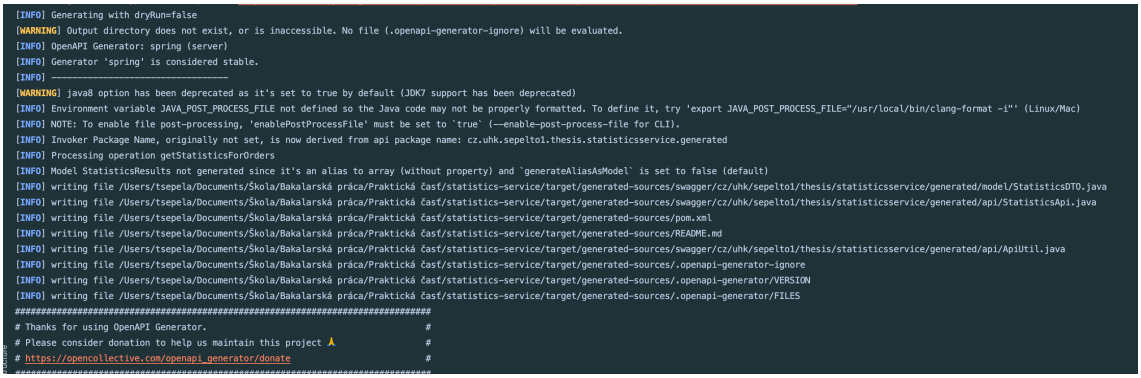
```
[
  {
    "timestamp": "2022-04-19T13:08:58Z",
    "totalPrice": 1800.0
  },
  {
    "timestamp": "2022-04-19T13:25:36Z",
    "totalPrice": 9000.0
  },
  {
    "timestamp": "2022-04-19T18:32:48Z",
    "totalPrice": 0.0
  },
  ...
]
```

Kritérium: DTO sú generované.

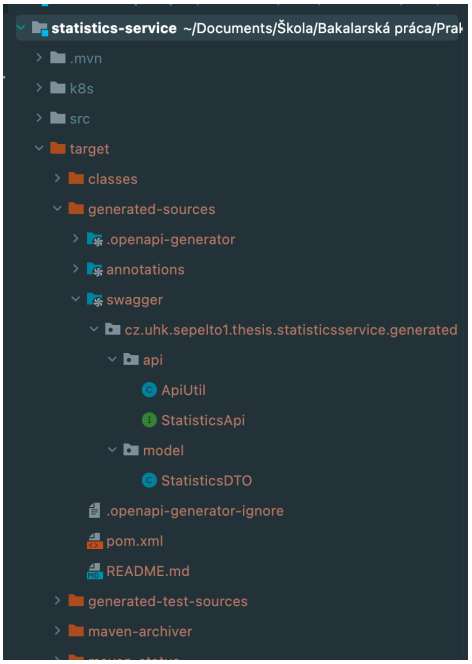
Po spustení príkazu

mvn clean install

by sa mali vygenerovať (obrázok č. 24) dátové objekty a interface pre controllery do zložky target (obrázok č. 25).



Obr. 24 Výpis terminálu při generovaní DTO z OpenAPI špecifikácie na statistics-service
Zdroj: Záznam obrazovky z Intellij IDEA – vlastné spracovanie



Obr. 25 Vygenerované DTO v zložke target pre statistics-service
Zdroj: Záznam obrazovky z Intellij IDEA – vlastné spracovanie

8 Záver

Cieľom tejto práce bolo naštudovanie teoretických pojmov, ktoré sa zvyčajne používajú spoločne s pojmom „cloudové riešenie založené na mikroservisnej architektúre“. Na základe teoretických východísk z tretej kapitoly bolo vyskúšané na praktickom príklade v čom spočíva vývoj cloudovej aplikácie rozdelenej do troch mikroslužieb, ktoré vzájomne komunikujú nielen synchrónnym ale aj asynchrónnym spôsobom. Ako to už pri vývoji softvéru býva zvykom, implementácii predchádzala analýza riešenia a návrh použitých technológií. Počas samotnej implementácia došlo k niekoľkým komplikáciám spojeným so zabezpečením integrity dát, ktoré sa ale podarilo odstrániť. Výsledné riešenie by teda malo spĺňať základné požiadavky na bezpečnosť a zabezpečenie dát.

Samotná implementácia riešenia je súčasťou tejto bakalárskej práce vo forme zdrojových súborov. K tomuto zdrojovému kódu sa v kapitole č. 6 nachádzajú popisy, ktorým by mal byť schopný porozumieť programátor alebo softvérový architekt s pokročilou znalosťou využitých technológii. Nakoľko riešenie, ktoré bolo implementované, si vyžaduje časovo náročný deployment a nutnosť prenajať si cloudový priestor od Digital Ocean, je kapitola č. 7 venovaná práve kontrole a testovaniu fiktívnych akceptačných kritérií, na ktorých je dokázané, že implementované riešenie funguje.

Autor tejto práce by sa chcel ďalej venovať práve optimalizácií využívaných zdrojov (CPU, RAM) a to napríklad výmenou frameworku Spring za novší Quarkus.

9 Zoznam použitej literatúry

1. What is the cloud? | Cloud definition. *Cloudflare.* [Online] [Dátum: 20. 04 2022.] https://www.cloudflare.com/en-gb/learning/cloud/what-is-the-cloud/.

2. Amazon AWS. [Online] [Dátum: 20. 04 2022.] https://aws.amazon.com.

3. Microsoft Azure. [Online] [Dátum: 20. 04 2022.] https://azure.microsoft.com/.

4. The developer cloud. *Digital Ocean.* [Online] [Dátum: 20. 04 2022.] https://www.digitalocean.com.

5. Google Cloud Platform. [Online] [Dátum: 20. 04 2022.] https://cloud.google.com.

6. What are microservices? [Online] [Dátum: 20. 04 2022.] https://microservices.io.

7. Docker. [Online] [Dátum: 20. 04 2022.] https://www.docker.com.

8. Kubernetes. [Online] [Dátum: 20. 04 2022.] https://kubernetes.io.

9. Spring Boot - Introduction. *tutorialspoint.* [Online] [Dátum: 20. 04 2022.] https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm.

10. OpenAPI Specification. *Swagger.* [Online] [Dátum: 20. 04 2022.] https://swagger.io/specification/ .

11. Launch of RabbitMQ Open Source Enterprise Messaging. [Online] February 2007. [Dátum: 18. 03 2022.] https://www.rabbitmq.com/resources/RabbitMQ_PressRelease_080207.pdf.

12. O'Hara, John. Toward a Commodity Enterprise Middleware. *ACM queue : tomorrow's computing today.* [Online] 2007. [Dátum: 23. 03 2022.]

13. Jones, M., Bradley, J., and N. Sakimura. JSON Web Token (JWT). *RFC Editor.* [Online] [Dátum: 20. 04 2022.] https://www.rfc-editor.org/rfc/rfc7519.html.

14. GitLab-managed Terraform state. *GitLab Docs.* [Online] [Dátum: 12. 03 2022.] https://docs.gitlab.com/ee/user/infrastructure/iac/terraform_state.html.

15. New personal access token . *Digital Ocean.* [Online] [Dátum: 15. 03 2022.] https://cloud.digitalocean.com/account/api/tokens/new .

16. Running Keycloak in a container . *Keycloak.* [Online] [Dátum: 20. 04 2022.] https://www.keycloak.org/server/containers .

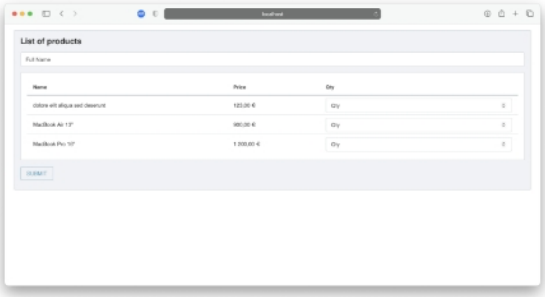
10 Prílohy

1) Akceptačné kritéria praktickej časti
2) Záznamy obrazovky z frontendu na účely validácie

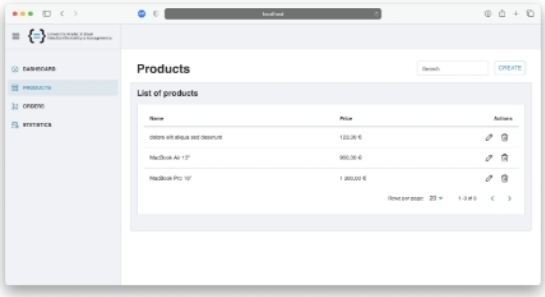
Akceptačné kritéria praktickej časti:

1. Mikroslužby sú nasadené v kubernetes a sú v stave „Running“.
2. Mikroslužby môžu prijímať requesty z API Gateway.
3. API Gateway je dostupná z internetu.
4. Keycloak je dostupný z internetu.
5. V rozhraní keycloak sa nachádzajú minimálne role:
    a. ROLE_ADMIN (ďalej len Administrátor),
    b. ROLE_CUSTOMER (ďalej len Zákazník).
6. Administrátor môže vytvoriť používateľský profil v rozhraní keycloak.
7. Administrátor môže priradiť role používateľovi v rozhraní keycloak.
8. Používateľ sa môže prihlásiť.
9. Anonymný zákazník môže vytvoriť objednávku.
10. Administrátor vidí objednávky.
11. Administrátor môže upravovať objednávky.
12. Administrátor môže mazať objednávky.
13. Administrátor môže vytvárať objednávky.
14. Administrátor vidí produkty.
15. Administrátor môže upravovať produkty.
16. Administrátor môže mazať produkty.
17. Administrátor môže vytvárať produkty.
18. Systém asynchrónne vytvorí udalosť – štatistiku pri vytvorení objednávky.
19. Štatistické eventy sú perzistentné a imutabilné.
20. Administrátor vidí štatistiky.
21. DTO sú generované.

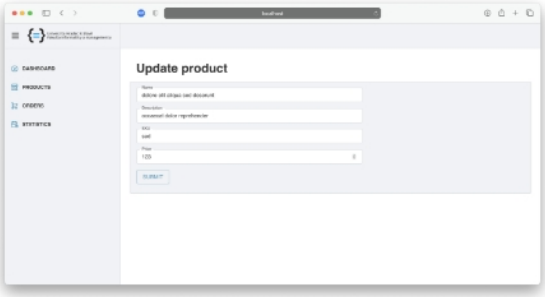
Záznamy obrazovky z frontendu na účely validácie



Objednávkový formulár

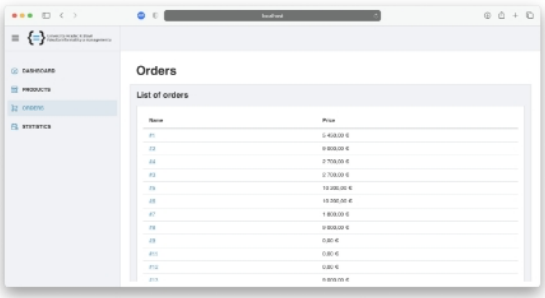


Zoznam produktov

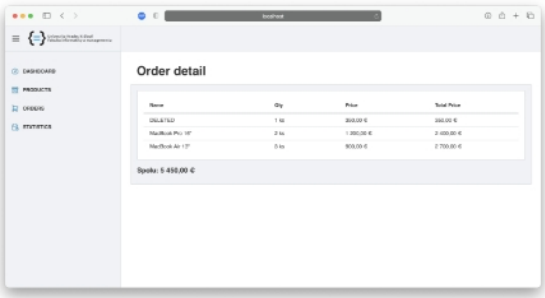


Úprava/vytvorenie produktu

Štatistiky



Zoznam objednávok



Detail objednávky

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Tomáš Šepeľa**

Studium: I1900648

Studijní program: B0688A140001 Informační management

Studijní obor: Informační management

Název bakalářské práce: **Cloudové riešenie založené na microservice architektúre**

Název bakalářské práce AJ: Cloud solution based on microservice architecture

Cíl, metody, literatura, předpoklady:

Cieľom projektu je popísať problematiku Cloud riešení založených na microservice architektúre v Springu.
Skúmané problémy: security, inter-service communication, messaging, gateway, logging, testing, monitoring, CI/CD.

MELL, Peter a Timothy GRANCE. *The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology* [online]. 2012, , 2-3 Dostupné z: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

What are microservices? Microservice Architecture [online]. Dostupné z: https://microservices.io/

What is the cloud? | Cloud definition: The cloud is made up of servers in data centers all over the world. Moving to the cloud can save companies money and add convenience for users. [online]. Dostupné z: https://www.cloudflare.com/learning/cloud/what-is-the-cloud/

Garantující pracoviště: Katedra informačních technologií, Fakulta informatiky a managementu

Vedoucí práce: Mgr. Hana Rohrová

Datum zadání závěrečné práce: 15.10.2021