Příjemci podpory: ***Poskytovatel:***

Bezpečné dopravní systémy nové generace

(VB01000048)

VB01000048-V3
Metody pro pokročilou analýzu dopravy (software)

Typ výsledku dle UV č. 837/2017	Evidenční číslo (příjemce)	Rok vzniku
R- software	VB01000048-V3	2022
ISBN-ISSN	**Webový odkaz na výsledek**	**Kde a kdy publikováno**
N/A	https://www.fit.vut.cz/research/product/754/	Online

Anotace k výsledku:

Software pro analýzu dopravy v cloudu. Webová aplikace pro zpracovávání dopravních dat v cloudu nabízí rozhraní (REST API) pro zpracování úloh analýzy dopravy a vlastní implementace metod samotných. V rámci vytvořeného cloudového software je aktuálně podporovaná úloha re-Identifikace vozidel na základě vizuálních vlastností (vzhledu) vozidla v obraze, společně s vlastní implementací vyvinutých metod. Software má výraznou možnost rozšíření o další úlohy z oblasti analýzy dopravy.

Řešitelský tým:

Eliška Vlčková (od 1.4. 2022), Ivana Sekaninová (do 31.3.2022) – manažer projektu, Lukáš Maršík – hlavní řešitel, Vítězslav Beran – další řešitel, Miroslav Juhas, Jiří Košák, Aleš Křupka, Marek Langr, Radek Němec, Jiří Řehák, Pavel Valenta, Daniel Bambušek, David Bařina, Adam Herout, Martin Kreslík, Pavel Smrž, Jakub Špaňhel, Pavel Zemčík

Obsah

1 Úvod

Tento dokument obsahuje popis výsledku VB01000048-V3 – Metody pro pokročilou analýzu dopravy (software) projektu MV SECTECH – Bezpečné dopravní systémy nové generace (VB01000048).

Pod tímto výsledek vznikl software pro analýzu dopravy v cloudu, ve formě webové aplikace pro zpracovávání dopravních dat v cloudu nabízí rozhraní (REST API) pro zpracování úloh analýzy dopravy a vlastní implementace metod samotných. V rámci vytvořeného cloudového software je aktuálně podporovaná úloha re-identifikace vozidel na základě vizuálních vlastností (vzhledu) vozidla v obraze. Software má výraznou možnost rozšíření o další úlohy z oblasti analýzy dopravy v dalším roce řešení projektu.

Informace o výsledku jsou k dispozici na https://www.fit.vut.cz/research/product/754/. Kód aplikace včetně popisu a návodu k použití je dostupný z https://git.fit.vutbr.cz/SSTS-NG/TrafficAnalysis-cloud.

2 Technická specifikace software

Tento framework založený na technologiích *Python, Flask, RabbitMQ, Redis, Celery* při použití virtualizačního nástroje *Docker* nabízí možnosti distribuovaného zpracování dopravních dat, včetně výpočtů modelů konvolučních neuronových sítí pro extrakci identifikačních příznaků za použití ONNX runtime.

2.1 Návrh cloudové aplikace

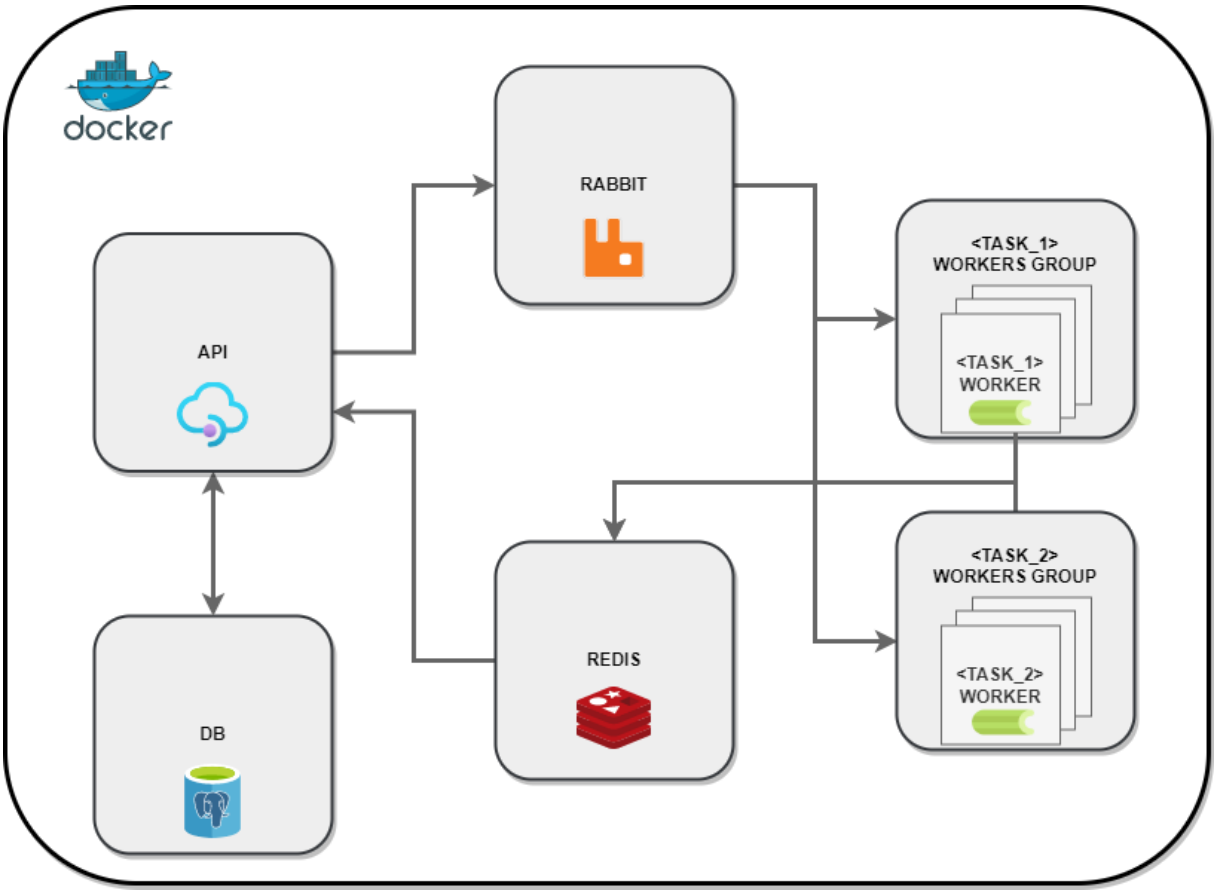
Software pro analýzu dopravy v cloudu je založená na *open-source* virtualizační technologii Docker[1], jehož cílem je poskytnout jednotné aplikační rozhraní pro izolaci a nasazování softwaru v balíčcích, zvané *kontejnery*. Tyto kontejnery obsahují vše podstatné pro spuštění vyvinutého software včetně knihoven, systémových nástrojů a dalších závislostí běh aplikace (*runtime*). Tato technologie umožňuje jednoduché nasazení a dobrou přenositelnost vytvořeného řešení.

Cloudová aplikace se skládá z několika modulů, které jsou reprezentovány Docker kontejnery. Schéma na *Obrázku 1* ukazuje jednotlivé moduly aplikace a jejich propojení. Celkem se jedná o 5 typů kontejnerů:

- **API** poskytuje rozhraní aplikace a je vstupním bodem pro komunikaci s aplikací
- **RABBIT** slouží jako *message broker* (více Sekce 2.1.1)
- **REDIS** je zde využit jako *message backend* (více Sekce 2.1.2)

[1] https://www.docker.com/

- **<TASK_xx> WORKER/WORKERS GROUP** je aplikačně specifický kontejner/skupina kontejnerů, který zpracovává přijímané požadavky (viz. Sekce 2.2) a zajišťuje tedy *task management* (více Sekce 2.1.3)



Obrázek 1: Schéma jednotlivých modulů cloudové aplikace pro analýzu dopravy a jejich propojení.

2.1.1 Message broker

RabbitMQ[2] je nejvíce rozšířený open-source message broker (systém pro zasílání zpráv).

Message broker funguje jako prostředník pro různé služby (např. webové aplikace). Lze jej použít ke snížení zátěže a doby doručení serverů webových aplikací tím, že úkoly, které by normálně zabraly mnoho času nebo zdrojů, deleguje na třetí stranu, která nemá jinou práci. [1]

[2] https://www.rabbitmq.com/

Obrázek 2: Základní použití RabbitMQ jako message brokera. Producent (Producer) posílá požadavky zprostředkovateli (Broker), konzument (Consumer) je zpracovává a posílá zprostředkovateli zpět výsledky. Zdroj: [1]

Základní architektura fronty zpráv je jednoduchá – existují klientské aplikace zvané producenti, které vytvářejí zprávy a doručují je zprostředkovateli (frontě zpráv). Ostatní aplikace, nazývané konzumenti, se připojují k frontě a přihlašují se k odběru zpráv, které mají být zpracovány. Schématické znázornění procesu je na *Obrázku 2*. Software může vystupovat jako producent nebo konzument nebo jako konzument i producent zpráv. Zprávy umístěné do fronty jsou uloženy, dokud si je spotřebitel nevyzvedne. [1]

2.1.2 Message backend

Redis[3] je open-source datové úložiště v paměti, které vývojáři používají jako databázi, mezipaměť, streamovací stroj a zprostředkovatele zpráv.

Redis je často označován jako server datových struktur. To znamená, že Redis poskytuje přístup k měnitelným datovým strukturám prostřednictvím sady příkazů, které jsou odesílány pomocí modelu server-klient se sockety TCP a jednoduchým protokolem. Různé procesy se tak mohou dotazovat a upravovat stejné datové struktury sdíleným způsobem.

Datové struktury implementované do služby Redis mají několik speciálních vlastností:

- Redis se stará o jejich ukládání na disk, i když jsou vždy obsluhovány a modifikovány do paměti serveru. To znamená, že Redis je rychlý, ale že je také nevolatilní.
- Implementace datových struktur klade důraz na paměťovou efektivitu, takže datové struktury uvnitř Redisu budou pravděpodobně využívat méně paměti ve srovnání se stejnou datovou strukturou modelovanou pomocí vysokoúrovňového programovacího jazyka.
- Redis nabízí několik funkcí, které je přirozené najít v databázi, jako je replikace, laditelná úroveň trvanlivosti, clusterování a vysoká dostupnost.

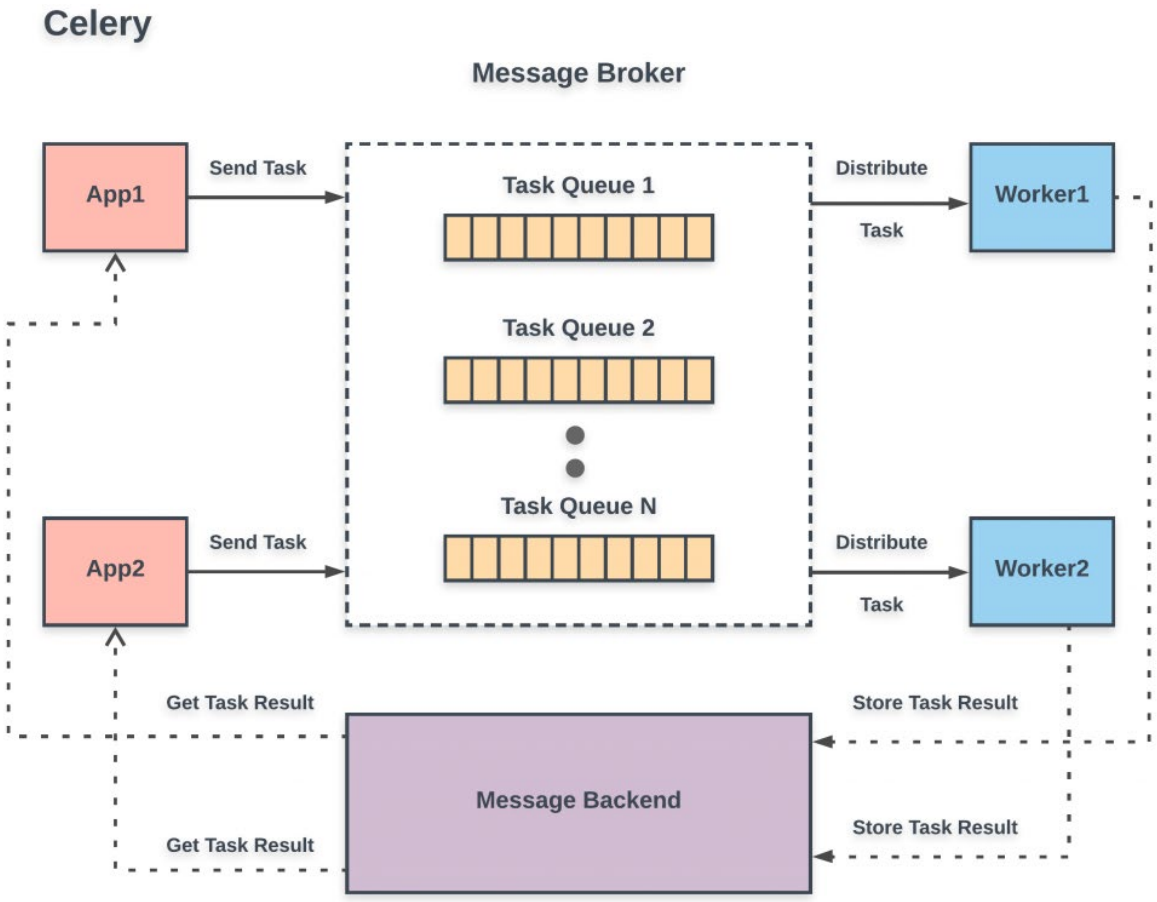
Dalším dobrým příkladem je představit si Redis jako komplexnější verzi pamětové cache (memcached), kde nejsou jen SET a GET operace, ale pracující se složitými datovými typy, jako jsou seznamy, množiny, uspořádané datové struktury atd. [2]

[3] https://redis.io/

2.1.3 Task management

Celery je asynchronní fronta úloh/úkolů založená na distribuovaném předávání zpráv. Je zaměřen na práci v reálném čase, ale podporuje i plánování. V našem návrhu aplikace Celery funguje jako správce úloh. Celery je zodpovědné za správnou distribuci úloh jednotlivým *worker* nodům přes zprostředkovatele zpráv (*broker*) a také za získávání výsledků z *backendu* zpráv. [3]

V našem případě slouží RabbitMQ jako zprostředkovatel zpráv a Redis jako *backend* zpráv. Zprostředkovatel zpráv i backend v Celery může být sdílen pro více úloh. Každá úloha je určena samostatnou frontou pod zprostředkovatelem zpráv. *Workeři* jsou pak na základě poskytnuté konfigurace připojeny ke konkrétním frontám úloh a zpracovávají výsledky distribuované Celery. Po dokončení výsledků *worker* uloží výsledek úlohy do *message backendu*. Schéma tohoto principu je uvedeno na *Obrázku 3*.



Obrázek 3: Schéma komunikace mezi aplikací a workerem při použití framework Celery. Zdroj: https://blog.imaginea.com/scale-part-i-task-queues/

2.2 Implementace worker uzlů pro zpracování požadavků

Worker uzly z Obrázku 1 představují v rámci aplikace účelově specifické a samostatně běžící kontejnery, které je možné nasazovat vícenásobně (ve více replikách). Všechny vytvořené kontejnery se připojují k *message brokeru*/*message backendu* a zajišťují vlastní

zpracování požadavků. Z pohledu systému jsou workeři rozlišování podle typu jednotlivých úloh analýzy dopravy, které je potřeba zpracovávat (např. re-identifikace vozidel, určení typu vozidel, odhad rozměrů vozidla, atd.). V rámci jedné úlohy pak mohou vykonávat více metod (např. extrakce příznaků z obrazu, verifikace podobnosti dvou snímků, vyhledání podobných příznaků v databázi, atd.).

Pokročilé úlohy zpracování dopravních dat pro analýzu dopravy mohou vyžadovat *inferenci* (spočítání dopředného průchodu) nějaké připraveného modelu strojového učení – neuronové sítě. Pro tyto účely mohou být součástí workerů použity technologie ONNX a ONNXruntime.

2.2.1 ONNX

ONNX[4] (Open Neural Network Exchange) je otevřený formát vytvořený pro reprezentaci modelů strojového učení. ONNX definuje společnou sadu operátorů (stavebních bloků modelů strojového učení a hlubokého učení) a společný formát souborů, aby umožnil vývojářům umělé inteligence používat modely s různými frameworky, nástroji, runtimy a kompilátory. [4]

ONNX lze přirovnat k programovacímu jazyku specializovanému na matematické funkce. Definuje všechny potřebné operace, které model strojového učení potřebuje k implementaci své inferenční funkce pomocí tohoto jazyka.

Cílem ONNX je poskytnout společný jazyk, který může každý framework strojového učení používat k popisu svých modelů. Prvním scénářem je usnadnit nasazení modelu strojového učení do výroby. Interpret ONNX (neboli runtime) může být pro tento úkol speciálně implementován a optimalizován v prostředí, kde je nasazen. Pomocí ONNX je možné vytvořit jedinečný proces pro nasazení modelu v produkci, nezávislý na frameworku strojového učení použitého k vytvoření modelu.

2.2.2 ONNX Runtime

ONNX Runtime[5] je multiplatformní akcelerátor modelů strojového učení s flexibilním rozhraním pro integraci hardwarově specifických knihoven. ONNX Runtime lze používat s modely z PyTorch, Tensorflow/Keras, TFLite, scikit-learn a dalších frameworků strojové učení.

Použití ONNX Runtime je přímočaré:

[4] https://onnxr.ai

[5] https://onnxruntime.ai

1. Pořiďte si model. Ten lze natrénovat z jakéhokoli framework strojového učení, který podporuje export/konverzi do formátu ONNX.
2. Načtěte a spusťte model pomocí ONNX Runtime.
3. (Volitelné) Vyladťe výkon pomocí různých konfigurací runtime nebo hardwarových akcelerátorů.

I bez kroku 3 poskytne ONNX Runtime často zlepšení výkonu ve srovnání s původním frameworkem.

ONNX Runtime aplikuje na graf modelu řadu optimalizací a poté jej rozdělí na podgrafy na základě dostupných hardwarových akcelerátorů. Optimalizovaná výpočetní jádra v základu ONNX Runtime zajišťují zlepšení výkonu a přiřazené podgrafy využívají další akcelerace od různých *Execution providerů* (CPU, GPU, TPU,…). [5]

2.3 Ukládání zpracovaných dat

Výsledky vytvořené workery je nutné perzistentně ukládat, aby byla naplněna samotná podstata re-identifikace vozidla – tedy jeho vyhledání v databázi doposud viděných vozidel a ztotožnění.

Za tímto účelem byla vytvořena jednoduchá databázová struktura za použití PostgreSQL.

2.3.1 PostgreSQL

PostgreSQL[6] je výkonný objektově-relační databázový systém s otevřeným zdrojovým kódem, který využívá a rozšiřuje jazyk SQL v kombinaci s mnoha funkcemi, které bezpečně ukládají a škálují nejsložitější datové úlohy. Počátky PostgreSQL sahají do roku 1986 jako součást projektu POSTGRES na Kalifornské univerzitě v Berkeley a má za sebou více než 35 let aktivního vývoje základní platformy.

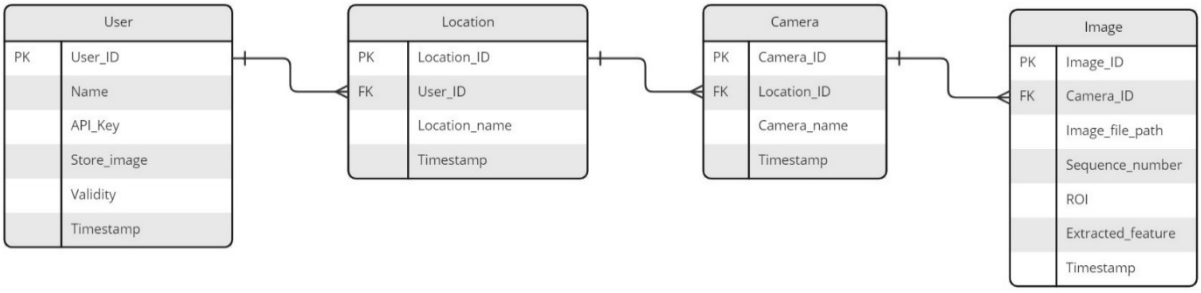
PostgreSQL si získal dobrou pověst díky své osvědčené architektuře, spolehlivosti, integritě dat, robustní sadě funkcí, rozšiřitelnosti a odhodlání open source komunity, která za tímto softwarem stojí, trvale poskytovat výkonná a inovativní řešení. PostgreSQL běží na všech hlavních operačních systémech, od roku 2001 je kompatibilní s ACID a má výkonné doplňky, například populární rozšíření geoprostorové databáze PostGIS. Není divu, že se PostgreSQL stala pro mnoho lidí a organizací oblíbenou relační databází s otevřeným zdrojovým kódem. [6]

2.3.2 Schéma databáze

Základní schéma databáze je velmi jednoduché a skladá se celkem ze čtyř tabulek. Tabulky *User*, která uchovává veškerá uživatelská data. Tabulek *Location* a *Camera*, které

[6] https://www.postgresql.org/

uchovávají všechny informace o lokalitě a kamerách, které se na ní vyskytují. Data příchozí na API a zpracované výsledky se následně ukládají do tabulky *Image*. Schéma databáze je možné vidět na *Obrázku 4*. Typy jednotlivých sloupců společně s popisem jejich významu lze nalézt v tabulkách níže.



Obrázek 4: Databázové schema cloudové části software pro analýzu dopravy v kamerových systémech nové generace.

Tabulka User

Název sloupce	Typ	Popis
id	int, PK	ID uživatele, primární klíč
name	string[1024]	Identifikátor uživatele API
api_key	string[128]	Vygenerovaný API klíč uživatele
store_image	bool	Rozhoduje, zda se pro uživatele ukládájí příchozí snímky
validity	bool	Platnost záznamu
timestamp	datetime	Poslední změna záznamu

Tabulka Location

Název sloupce	Typ	Popis
id	int, PK	ID lokality, primární klíč
user_id	int, FK	ID uživatele, cizí klíč
location_name	string[1024]	Identifikátor lokality / název
timestamp	datetime	Poslední změna záznamu

Tabulka Camera

Název sloupce	Typ	Popis
id	int, PK	ID kamery, primární klíč
location_id	int, FK	ID lokality, cizí klíč
camera_name	string[1024]	Identifikátor kamery / název
timestamp	datetime	Poslední změna záznamu

Tabulka Image

Název sloupce	Typ	Popis
id	int, PK	ID lokality, primární klíč
camera_id	int, FK	ID kamery, cizí klíč
image_file_path	string[4096]	Cesta ke snímku ve filesystému
sequence_number	int	Sekvenční číslo snímku z kamery
roi	text	Oblast zájmu v obraze (json)

extracted_feature	ARRAY(float)	Výsledek inference re-identifikačního modelu
timestamp	datetime	Poslední změna záznamu

3 Použití software v praxi

Cloudovou aplikace je možné jednoduše nasadit na zařízení běžící na platformě Windows, Linux i MacOS s použitím nástroje Docker a jeho rozšíření *docker compose*. Repozitář s aplikaci obsahuje konfigurační YAML soubor, který definuje všechny potřebné parametry pro spuštění jednotlivých modulů ve formě Docker kontejnerů a samotné zprovoznění aplikace. Ukázka konfigurace souboru *docker-compose.yaml* je zobrazena na Fragmentu kódu 1. Aktuální verze konfigurace je dostupná na adrese: https://git.fit.vutbr.cz/SSTS-NG/TrafficAnalysis-cloud/src/branch/master/docker-compose.yaml

Samotné spuštění aplikace na systému Linux se provede následující posloupností příkazu:

- `docker compose build`
- `docker compose up &`

První příkaz vytvoří Docker kontejnery specifikované souboury Dockerfile. Druhý příkaz zajistí stažení kontejnerů, které není potřeba překládát a poté je na pozadí spustí. Alternativně lze použít také příkaz *docker-compose* v závislosti na způsobu instalace doplňku.

Aplikační rozhraní cloudové aplikace je potom dostupné na adrese hostovského počítače a portu 5000. Konfigurace portu lze změnit v *docker-compose.yaml*.

3.1 Popis aplikačního rozhraní (API)

Aplikační rozhraní cloudové aplikace je založené na architektuře REST s použitím frameworku Flask, převážně s použitím HTTP metod GET a POST. Cílem bylo standardizovat aplikační rozhraní tak, aby bylo použitelné pro různé typy úloh a bylo verzovatelné, tedy aby si umělo zachovávat také zpětnou kompatibilitu.

Standardizovaný formát cesty (*endpointuú* v rámci aplikačního rozhraní je následující:

- `/api/v1/{TASK}/{JOB}`

kde *TASK* specifikuje konkrétní úlohu analýzy dopravy a *JOB* specifikuje konkrétní činnost. Požadavky jsou přijímány v podobě JSON stringů, jejich obsah se liší v závislosti na hodnotě *TASK* a *JOB*.

Aktuálně aplikační rozhraní podporuje několik činností pro úlohu re-identifikace vozidel na *endpointech* specifikovaných v tabulce XX. Aktuální verze rozhraní včetně specifikace obsahu požadavků je k dispozici na adrese https://git.fit.vutbr.cz/SSTS-NG/TrafficAnalysis-cloud/src/branch/master/README.md

Endpoint	Popis funkcionality
/api/v1/reid/extract	Provede extrakci příznakového vektoru z příchozího snímku detekovaného vozidla. Na základě API_KEY uloží snímek do filesystému.
/api/v1/reid/verify	Provede verifikaci podobnosti dvou příchozích snímků. Z obou snímků jsou vyextrahovany příznakové vektory, které jsou následně porovnány a je vyhodnocena jejich podobnost.
/api/v1/reid/find_by_feature	Podle příchozího příznakového vektoru provede vyhledání nejvíce podobných vektorů v databázi dříve zpracovaných snímků.
/api/v1/reid/find_by_image	Vyhledá nejpodobnější snímky v databází na podle příchozího snímku. Vyhledávání je implementováno formou porovnávání příznakových vektorů, jako v ostatních případech.

Fragment kódu 1: Ukázková konfigurace souboru *docker-compose.yaml*, zajišťující spuštění všech komponent cloudové aplikace.

```yaml
version: '3.8'

volumes:
  redis_data:
  pg_db_data:

services:

  db:
    image: postgres:14.1-alpine
    restart: always
    environment:
      POSTGRES_USER: <db_user>
      POSTGRES_PASSWORD: <db_pass>
      POSTGRES_DB: <db_name>
    ports:
      - '5432:5432'
    expose:
      - 5432
    volumes:
      - pg_db_data:/var/lib/postgresql/data

  redis:
    image: redis:latest
    hostname: redis
    volumes:
      - redis_data:/data

  rabbit:
    hostname: rabbit
    image: rabbitmq:latest
    environment:
      - RABBITMQ_DEFAULT_USER=<user>
      - RABBITMQ_DEFAULT_PASS=<pass>
```

```yaml
api:
    build:
      context: .
      dockerfile: Dockerfile
    hostname: api
    command: ./scripts/run_api.sh
    volumes:
      - .:/app/
    ports:
      - "5000:5000"
    depends_on:
      - db
    links:
      - rabbit
      - redis
      - db
    environment:
     - RABBITMQ_DEFAULT_USER=<user>
     - RABBITMQ_DEFAULT_PASS=<pass>

  worker:
    build:
      context: .
      dockerfile: Dockerfile
    command: ./scripts/run_celery.sh
    volumes:
      - .:/app/
    links:
      - rabbit
      - redis
      - db
    depends_on:
      - rabbit
      - db
    environment:
      - RABBITMQ_DEFAULT_USER=<user>
      - RABBITMQ_DEFAULT_PASS=<pass>
```

Reference

[1] RabbitMQ for begginers – What is RabbitMQ?. *CloudAMQP - RabbitMQ as a Service* [online]. Dostupné z: https://www.cloudamqp.com/blog/2015-05-18-part1-rabbitmq-for-beginners-what-is-rabbitmq.html

[2] GitHub – redis/redis: Readme. *GitHub: Where the world builds software · GitHub* [online]. Dostupné z: https://github.com/redis/redis/README.md

[3] GitHub - celery/celery: Distributed Task Queue (development branch). *GitHub: Where the world builds software · GitHub* [online]. Copyright © 2022 GitHub, Inc. [cit. 24.06.2022]. URL: https://github.com/celery/celery

[4] ONNX | Home [online]. URL: https://onnx.ai/

[5] ONNX Runtime – documentation [online]. URL: https://onnxruntime.ai/docs/

[6] PostgreSQL: The World's Most Advanced Open Source Relational Database [online]. URL: https://www.postgresql.org/

Příloha A

Výpis software na webových stránkách dalšího účastníka – Fakulty informačních technologií, Vysoké učení technické v Brně. https://www.fit.vut.cz/research/product/754/

DETAIL PRODUKTU

Methods for advanced traffic analysis in cloud

Vznik: **2022**

NÁZEV ČESKY	Metody pro pokročilou analýzu dopravy v cloudu
TYP	software
LICENCE	vyžadována - licenční poplatek
AUTOŘI	**Špaňhel Jakub, Ing.** (UPGM FIT VUT) **Herout Adam, prof. Ing., Ph.D.** (UPGM FIT VUT)
POPIS	Software pro analýzu dopravy v cloudu, vytvořený v rámci projektu MV SECTECH - Bezpečné dopravní systémy nové generace (VB01000048). Webová aplikace pro zpracovávání dopravních dat v cloudu nabízí rozhraní (REST API) pro problematiku re-identifikace vozidel na základě vizuálních vlastností (vzhledu) vozidla v obraze, společné s vlastní implementací vyvinutých metod. Tento framework založený na techologiích Flask, RabbitMQ, Redis, Celery nabízí možnosti distribuovaného zpracování dopravních dat, včetně výpočtů modelů konvolučních neuronových sítí pro extrakci identifikačních příznaků za použití ONNX runtime. Vyvinutá webová aplikace nabízí možnosti rozšíření na další úlohy z oblasti analýzy dopravy pro další rozvoj projektu.
UMÍSTĚNÍ	**https://git.fit.vutbr.cz/SSTS-NG/TrafficAnalysis-cloud**
LICENCE	Pro výzkumné účel formou open-source, pro komerční použití pod licencí. Pro informace o licenčních podmínkách prosím kontaktujte: Výzkumné centrum informačních technologií, Fakulta informačních technologií VUT v Brně, Božetěchova 2, 612 66 Brno, +420 541 141 472
PROJEKTY	**Bezpečné dopravní systémy nové generace** (VB01000048)
VÝZKUMNÉ SKUPINY	**Výzkumná skupina počítačové grafiky** (VZ GRAPH)
PRACOVIŠTĚ	**Ústav počítačové grafiky a multimédií FIT VUT v Brně** (UPGM FIT VUT)