Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16, Bratislava 4

Inštalačná príručka

Dokumentácia k tímovému projektu

Tímový projekt

Tím č. 21

Vedúci: Ing. Ivan Srba, PhD.

Členovia tímu:
- **Matej Groma**
- **Matej Horváth**
- **Peter Jurkáček**
- **Jozef Kamenský**
- **Adam Kňaze**
- **Kristína Macková**
- **Lenka Pejchalová**
- **Jakub Sedlář**

tim21.2018.fiit@gmail.com

Akademický rok: 2018/2019

Posledná zmena: 9. mája 2019

Obsah

1 Úvod

Tento dokument obsahuje postupy využívané v tíme Traffic Watch v rámci predmetu Tímový projekt týkajúce sa konfigurovania a inštalácie podporných prostriedkov a samotnej aplikácie. V nasledujúcich kapitolách sa detailne venuje jednotlivým postupom, od pomocných určených prevažne na interné použitie v rámci tímu po postupy, ktoré je potrebné aplikovať na kompletné sprevádzkovanie systému.

2 Nasadenie serverovej časti systému (BE+FE)

Najskôr naklonujeme repozitár deployment, ktorý zastrešuje záležitosti týkajúce sa nasadzovania. Presunieme sa do priečinku backend_and_frontend, v ktorom sa nachádzajú skripty pre nasadenie backendu a frontendu aplikácie na server.

V súbore hosts pod [servers] nakonfigurujeme FQDN servera, na ktorý chceme aplikáciu nasadiť (pre dané FQDN sa musí dať získať certifikát cez letsencrypt). Parametrom ansible_user špecifikujeme meno používateľa, prostredníctvom ktorého sa na server pripájame (root alebo používateľ so sudo oprávneniami). Pod [servers:vars] môžeme upraviť niektoré konfiguračné parametre servera (napr. porty, na ktorých BE/FE beží, alebo deployment_prefix, ktorý identifikuje konkrétnu inštanciu aplikácie v prípade, že beží viaceré na jednom systéme - v takom prípade zmeníme aj čísla portov a databázu). Pod [all:vars] upravujeme nastavenia, ktoré sa aplikujú lokálne (ako cesta k repozitáru, alebo vetva ktorú nasadiť). Následne skopírujeme súbor host_vars/example.com do host_vars/**FQDN** a definujeme používateľské mená a heslá ktoré chceme použiť pre MQTT (kvôli citlivosti obsahu takéto súbory nie sú verziované). MQTT server je zdielaný naprieč viacerými inštanciami (s prihlasovacími údajmi naposledy špecifikovanými v tomto súbore). Možné je zmeniť aj údaje k TURN serveru (čo však nemá vplyv na bezpečnosť, keďže sú poskytnuté koncovým zariadeniam).

Pre nasadením nainštalujeme Ansible podľa návodu na https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#installing-the-control-machine. Nasadenie spustíme cez ./play.sh (stiahne podporné roly a vykoná samotný playbook). Po vykonaní aplikácia beží na serveri bez potreby ďalšieho manuálneho zásahu. Nasadenie je možné opakovať pre aktualizáciu aplikácie v prípade zmien v nasadzovaných vetvách. Predvolene aplikácia beží s využitím https na portoch 8123 (backend) a 8124 (frontend). Aplikáciu je možné manuálne obsluhovať pomocou systemd služby springboot_backend_**deployment_prefix** (napr sudo systemctl start springboot_backend_staging). MQTT server je možné obsluhovať pomocou služby mosquitto, TURN server pomocou coturn, web server pomocou nginx a dastabázu pomocou postgresql.

Predpokladom pre beh backendu a frontendu sú spustené služby postgresql, nginx, coturn a mosquitto v ľubovoľnom poradí. V PostgreSQL musí byť nainštalované a aktivované rozšírenie TimescaleDB, podľa návodu na https://docs.timescale.com/v1.3/getting-started.

3 Nastavenie Jetson

Nastavenie Jetson zahŕňa inštaláciu systému, požadovaných balíkov na manuálnu kompiláciu opencv, jeho kompiláciu a ďalšie podporné konfiguračné záležitosti.

V prvom rade nainštalujeme systém z hostiteľského počítača Ubuntu 16.04.6 LTS použitím aplikácie NVIDIA SDK MANAGER na adrese https://developer.nvidia.com/embedded/dlc/nv-sdk-manager.

Na zariadení spustíme nasledujúce príkazy:

```
cat <<EOF|sudo tee /etc/systemd/system/maximize-performance.service >/dev/
    null
[Service]
Type=oneshot
```

```
ExecStart=/usr/sbin/nvpmodel --mode 0
ExecStart=/usr/bin/jetson_clocks

[Unit]
After=multi-user.target

[Install]
WantedBy=multi-user.target
EOF
sudo systemctl daemon-reload
sudo systemctl enable --now maximize-performance.service

sudo apt purge python3-opencv python-opencv
sudo apt install autossh build-essential cmake cmake-curses-gui g++
   gfortran git hdf5-tools libatlas-base-dev libavcodec-dev libavformat-
   dev libavutil-dev libboost-all-dev libdc1394-22-dev libeigen3-dev
   libffi-dev libgflags-dev libglew-dev libgoogle-glog-dev libgstreamer-
   plugins-base1.0-dev libgstreamer1.0-dev libgtk2.0-dev libgtk-3-dev
   libhdf5-dev libhdf5-serial-dev libjpeg-dev libjpeg-turbo8-dev libjpeg8-
   dev liblapack-dev liblapacke-dev libleveldb-dev liblmdb-dev libopenblas
   -dev libpng-dev libpostproc-dev libprotobuf-dev libsnappy-dev libsrtp2-
   dev libssl-dev libswscale-dev libtbb-dev libtbb2 libtiff-dev libtiff5-
   dev libv4l-dev libxine2-dev libxml2-dev libxvidcore-dev libx264-dev
   make pkg-config protobuf-compiler python-dev python-numpy python-pip
   python-py python-pytest python-tk python3-dev python3-numpy python3-pip
    python3-py python3-pytest python3-tk python3.6-dev qt5-default zlib1g-
   dev
sudo pip2 install numpy
sudo pip2 install numpy --upgrade
sudo pip3 install numpy matplotlib
sudo pip3 install numpy --upgrade

git clone git@github.com:team21-18/deployment.git
sudo cp -f deployment/camera_files/cuda_gl_interop.h /usr/local/cuda/
   include/cuda_gl_interop.h

git clone https://github.com/opencv/opencv.git
cd opencv
git checkout 7442100caaa9e6dafce320aeb0afcf86e8aea11c
cd ..
git clone https://github.com/opencv/opencv_contrib.git
cd opencv_contrib
git checkout c8ed08d0a245656c75617dfc93a9f5b6b84aaf8f
cd ..
mkdir opencv/build
cd opencv/build
cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/usr/local -
```

```
    DWITH_CUDA:BOOL="1" -DWITH_GTK:BOOL="0" -DWITH_QT:BOOL="1" -
    DWITH_OPENGL:BOOL="1" -DCUDA_ARCH_BIN:STRING="6.2" -
    DENABLE_PRECOMPILED_HEADERS:BOOL="0" -DBUILD_JAVA:BOOL="0" -
    DBUILD_TESTS:BOOL="0" -DOPENCV_ENABLE_NONFREE:BOOL="1" -
    DBUILD_PERF_TESTS:BOOL="0" -DOPENCV_GENERATE_PKGCONFIG:BOOL="1" -
    DOPENCV_EXTRA_MODULES_PATH:PATH="../../opencv_contrib/modules" -
    DBUILD_opencv_python3=yes ..
make -j6
sudo make install
```

4 Inštalácia softvéru kamery

Inštalácia kamery je popísaná nižšie. Prvým krokom je naklonovanie zdrojových súborov z git repozitára príkazom:

```
git clone https://github.com/team21-18/camera.git
```

Základ TrafficWatch kamera modulu je v jazyku Python, preň sa teda najprv odporúča vytvoriť samostané virtualenv:

```
cd /pracovny priecinok s kamerou/
virtualenv .env
source .env/bin/activate
```

Teraz môžeme inštalovať potrebné knižnice a Python balíčky.

```
sudo apt install libavdevice-dev libavfilter-dev libopus-dev libvpx-dev pkg-config

# pokiaľ nebol kompilovaný a inštalovaný opencv
pip install opencv-python

pip install -r requirements.txt
```

Projekt využíva knižnicu OpenCV pre počítačové videnie. V základnej verzií bez akcelerácie na grafickej karte postačí opencv nainštalované cez pip, inak je potrebný build zo zdroja aby bolo OpenCV použiteľné aj z C++ a aby podporovalo CUDA funkcie.

V základnej verzií stačí už len nastaviť user config súbor - možné je skopírovať default_configuration.json na configuration.json a tento ďalej upravovať, postačuje však vytvorenie podľa nasledujúceho vzoru obsahujúceho základné parametre.

```
{
  "video_source": {{ video_source }},
  "mqtt_username": "{{ mqtt_username }}",
  "mqtt_password": "{{ mqtt_password }}",
  "mqtt_host": "{{ mqtt_host }}",
  "mqtt_port": {{ mqtt_port }},
  "camera_id": "{{ camera_id }}",
```

```
  "area_id": "{{ area_id }}",
  "mqtt_prefix": "{{ deployment_prefix }}",
  "headless": true,
  "CUDA": "pbcvt"
}
```

Jednotlivé parametre konfiguračného súboru sú popísané v dokumentácií k modulom systému. Program sa spúšťa jednoduchým príkazom:

```
sudo python3 trafficwatch.py
```

Na démonizáciu kamery je možné napríklad vytvoriť službu.

```
cat <<EOF|sudo tee /etc/systemd/system/camera.service >/dev/null
[Service]
Type=simple
ExecStart=/bin/sh -c 'cd /pracovny priecinok s kamerou/ && source .env/bin
   /activate && python3 trafficwatch.py'
Restart=always
User=root
Group=root

[Unit]
StartLimitIntervalSec=60
StartLimitBurst=2
Wants=network-online.target
After=network-online.target

[Install]
WantedBy=multi-user.target

EOF
sudo systemctl daemon-reload
sudo systemctl enable --now camera.service
```

Použitie kamery Na použitie vstupu z kamery namiesto videosúboru stačí v konfiguračnom JSONe hodnotu "video_source"zmeniť na id nahrávacieho zariadenia (číslo). Pre defaultné zariadenie nastav 0. Ak chceš zmeniť rozlíšenie/tvar/veľkosť videa nastav konfiguračný parameter "video_resize"na [<sirka>,<vyska>] a na každú snímku sa zavolá funkcia cv2.resize. Hodnota null nespúšťa resize.

Podpora Windows Väčšina funkcionality beží normálne aj na Windowse. Nefunguje livestream (používame knižnicu aiortc). Ak napriek tomu chcete za každú cenu používať livestream aj na Windowse, potrebujete WSL (Windows Subsytem for Linux). V ňom vám bude fungovať všetko (aj livestream), pre zmenu však nejde gui. Na to si nainštalujte XMing server a pred spustením trafficwatch.py zadajte príkaz 'export DISPLAY=:0'

4.1 Modul CudaWatch a akcelerácia na grafickej karte

Pre akceleráciu OpenCV na grafickej karte pomocou CUDA je potrebné volať OpenCV funkcie z C++ (Python bindingy ešte nie sú oficiálne dostupné). Modul CudaWatch používa wrapper pbcvt na volanie C++ kódu z Pythonu. Niektoré funkcie sú zrýchlené portovaním do C++ aj bez využitia grafickej karty.

Pre používanie akcelerovaných funkcií z CudaWatch je potrebné mať na systéme funkčnú verziu OpenCV 4.X s CUDA funkcionalitou. Wrapper pbcvt používa knižnicu Boost.Python pre komunikáciu Pythonu a C++.

Kód momentálne obsahuje tri možnosti akcelerácie kódu na grafickej karte. V konfiguračnom súbore parameter CUDA nastavený na null znamená žiadna akcelerácia, všetko beží v Pythone. Na tento režim stačí OpenCV naištalované cez pip a malo by to bežať všade.

Ak je parameter CUDA v konfiguračnom súbore nastavený na "native"použijú sa native Python bindigy. Tie ale ešte nie sú oficiálne vydané, neexistuje k nim dokumentácia a implementovali sme s nimi len background substracion. Netreba ale buildiť zo zdroja modul Cudawatch.

Ten sa použije pri nastavení CUDA na "pbcvt". Momentálne je v ňom implementovaný celý detector (viď. súbor detector_cuda.py). Toto je podporovaný a ďalej vyvíjaný detector modul.

4.1.1 Inštalácia

1. Predispozícia je správne nainštalované OpenCV 4
2. Nainštaluj knižnicu Boost.Python, na Ubuntu by malo stačiť “sudo apt-get install libboost-python-dev”. Alternatíva je buildiť zo zdroja, ale celkom bolí.
3. Nainštaluj CMake a CMake-gui (http://www.cmake.org/download/, “sudo apt-get install cmake cmake-gui” na Ubuntu)
4. Spusti CMake-gui, ako zdroj nastav priečinok cudawatch a build priečinok nastav “cudawatch/build”. Klikni Configure a vyber generátor (default by mal byť fajn). CMake zbehne pričom sa pokúsi nájsť všetky závislosti ktoré potrebuje (Boost, OpenCV, Python, CUDA). V okne vyskočí kopa premenných ktoré chceme skontrolovať (napr. na Jetsone dosť veľa vecí nedal a bolo ich treba prepísať):
   - Ungrouped Entries - Skontrolovať OpenCV_DIR a nastaviť PYTHON_DESIRED_VERSION na 3.X (pre použitie Pythonu 3)
   - Boost - Basically, všade kde je napísané python2 (v nejakej forme) chceme to prepísať na python3 alternatívu. Napríklad premenná s názvom Boost_PYTHON27_LIBRARY_DEBUG bola na Jetsone po konfigurácií nastavená na “/usr/lib/aarch64-linux-gnu/libboost_python-py27.so”, tak ju prepíšeme na “/usr/lib/aarch64-linux-gnu/libboost_python-py36.so” (názov premennej neriešime)
   - CMAKE aj CUDA boli na Jetsone správne.
   - Chceme použiť Python 3, CMake ale v kategórií PYTHON_3 nastavil všade python2 prostriedky. Takže ich znovu treba zaradom prejsť a opraviť na python3 varianty. Posledná premenná PYTHON3_PACKAGES_PATH určuje kam bude neskôr vygenerovaný modul uložený po zavolaní “sudo make install”.
5. Znova klikni Configure a dúfaj, že tam nebudú žiadne errory, inak sa budeš musieť špŕtať v CMake dlhšie (na Jetsone bol error No header defined for python-py27; skipping header check, ale keďže je to python2 error dá sa ignorovať a bude to fungovať). Teraz klikni Generate a priečinok build je pripravený na buildenie samotného modulu.
6. Zbuildi modul, ak si použil defaultné unix makefiles pusti v priečinku build príkaz “make”. Znova nechceš žiadne errory :D. Hopefully ti vznikol nejaký takýto súbor: “cudawatch.cpython-36m-aarch64-linux-gnu.so”, ktorý vieš neskôr importovať z Pythonu. Nainštaluješ ho príkazom “sudo make install” (skopírujš do priečinku s packages ktorý si nastavil vyššie).

Tento návod celkom počíta s Ubuntu ako OS. Wrapper funguje aj na Widowse, návod k nemu sa dá nájsť na gite v repe wrappera samotného. Zároveň, asi netreba pripomínať, že je to bolestivejšie.

4.1.2 Vývoj

Kód ktorý nás zaujíma je v súbore cudawatch/src/python_module.cpp. Momentálne sú tam všetky používané funkcie. Je to normálny čistý C++ kód. Funkcie sa odkrývajú Pythonu naspodku súboru cez príkazy def, wrapper umožňuje posielať medzi Pythonom a C++ matice. Konkrétne, na strane Pythonu sú to Numpy polia a na strane C++ OpenCV matice cv::Mat. Posielať sa dá aj všetko ostatné čo Boost.Python unesie (detaily prenechám na čitateľa). Ako fungujú ostatné veci sa dá zistiť z už existujúceho kódu, a tie čo sa nedajú, tie neviem ani ja. Have fun :)