边缘端部署文档

第一章 文档概述

1.1 文档目的

该文档是用来操作部署是对系统的安装部署以及运行过程中可能存在的问题进行原因分析，并针对问题进行相关解决

1.2 读者对象

本文档预期的读者对象包括系统技术小组人员、测试人员、维护人员。

1.3 软件清单

第二章 运行环境要求

2.1 硬件环境最低要求

我们同时提供了两套性能不同的解决方案，以便同时满足集中式服务器和开发板的实际部署需求。

服务器集中部署方式适合于同时并发处理多个摄像头的请求，并且可以应用识别率更高的算法。开发板部署方式适合于预算不高的场景，但仅能处理一个摄像头且算法识别率一般。

处理器

gpu：1060

内存 16G DDR3

存储 128G SSD

其他 千兆网卡

处理器 1.4GHz 4 核处理器

gpu：不要求，若有 gpu 更佳

内存: 2G DDR3

存储 64G

其他 无线网卡

2.1.1

2.2 软件环境要求

主要环境依赖	版本建议要求
ubuntu	16.04
python	3.5
pytorch	1.0
opencv	3.4.2.17
nginx	1.13.12
mysql	5.6
redis	不限制
docker	不限制
cuda	9
cudnn	7

对于英伟达驱动程序，请读者根据自己的 gpu 型号自行查阅相关资料

第三章 应用程序的安装、部署和配置

3.1 部署环境概要说明

文档主要描述服务器的环境部署方式，对于开发板的部署方式以及 docker k8s 的部署方式将在不久后提供。

3.2 安装、部署和配置步骤

3.2.1 nginx 的安装与配置

前提条件

安装 Nginx 及其他所需软件之前先安装一些前提软件。需要 pip 与 virtualenv：

sudo apt-get install python-setuptools

sudo easy_install pip

sudo pip install virtualenv

如果使用 apt-get 安装 Nginx，需要添加 Nginx 库到 apt-get source 中：

sudo add-apt-repository ppa:nginx/stable

如果“add-apt-repository”命令在你的 Ubuntu 版本中不存在的话，你需要安装“software-properties-common”包，使用命令：sudo apt-get software-properties-common

升级已有的包，确保系统上有 uWSGI 所需的编译器和工具：

sudo apt-get update && sudo apt-get upgrade

sudo apt-get install build-essential python python-dev

安装 Nginx

安装并运行 Nginx：

sudo apt-get install nginx

sudo /etc/init.d/nginx start

Nginx 是一个提供静态文件访问的 web 服务，它不能直接执行托管 Python 应用程序，而 uWSGI 解决了这个问题。先安装 uWSGI，稍候再配置 Nginx 和 uWSGI 之间的交互。

sudo pip install uwsgi

步骤 1

打开浏览器访问你的服务器，你应该能看到 Nginx 欢迎页：

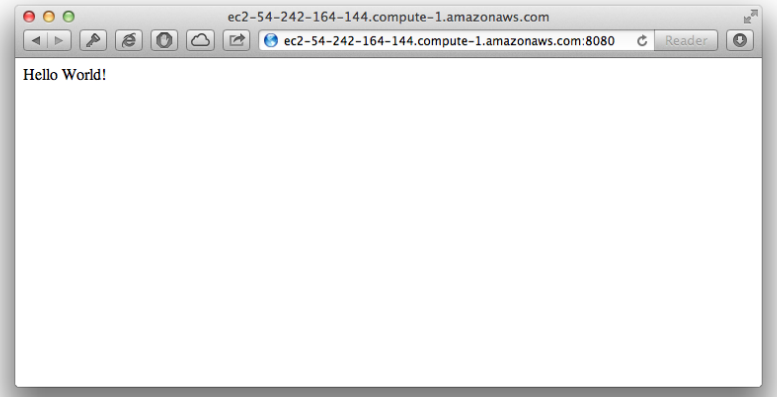
应用部署

将所有应用相关的文件存放在/var/www/projectname 文件夹中。下面创建这个文件夹并在其中初始化一个虚拟环境：

sudo mkdir /var/www

sudo mkdir /var/www/ projectname 使用 root 权限创建这个文件夹，它目前归 root 用户所有，

让我们更改它的所有权给你登录的用户（我的例子中是 ubuntu）
sudo chown -R ubuntu:ubuntu /var/www/ projectname /
创建并激活一个虚拟环境，在其中安装 Flask：
cd /var/www/ projectname
virtualenv venv
. venv/bin/activate
pip install flask
将项目移入/var/www/ projectname 文件夹中
步骤 2
执行 app.py：
python app.py
现在可以通过浏览器访问你服务器的 8080 端口



注意：因为 80 端口已被 Nginx 使用，这里使用 8080 端口。
配置 Nginx
首先删除掉 Nginx 的默认配置文件：
sudo rm /etc/nginx/sites-enabled/default
注意：如果安装了其他版本的 Nginx，默认配置文件可能在/etc/nginx/conf.d 文件夹下
创建新配置文件
/var/www/demoapp/projectname_nginx.conf：

```
server {
    listen          80;
    server_name localhost;
    charset         utf-8;
    client_max_body_size 75M;

    location / { try_files $uri @yourapplication; }
    location @yourapplication {
        include uwsgi_params;
        uwsgi_pass unix:/var/www/projectname/projectname_uwsgi.sock;
    }
}
```

将刚建立的配置文件使用符号链接到 Nginx 配置文件文件夹中，重启 Nginx：

sudo ln -s /var/www/ projectname / projectname_nginx.conf/etc/nginx/conf.d/
sudo /etc/init.d/nginx restart
步骤 3
到 uWSGI 的链接在 Nginx 配置文件的第 10 行定义：
uwsgi_pass unix:/var/www/projectname/projectname_uwsgi.sock;
这代表 Nginx 和 uWSGI 之间的链接是通过一个 socket 文件，这个文件位于/var/www/ projectname / projectname _uwsgi.sock。因为还没有配置 uWSGI，所以这个文件还不存在，因此此时 Nginx 返回“bad gateway”错误，
配置 uWSGI
创建一个新的 uWSGI 配置文件/var/www/ projectname / projectname _uwsgi.ini：

```
[uwsgi]
#application's base folder
base = /var/www/ projectname

#python module to import
app = hello
module = %(app)

home = %(base)/venv
pythonpath = %(base)

#socket file's location
socket = /var/www/ projectname /%n.sock

#permissions for the socket file
chmod-socket    = 666

#the variable that holds a flask application inside the module imported at line #6
callable = app

#location of log files
logto = /var/log/uwsgi/%n.log
```

创建一个新文件夹存放 uWSGI 日志，更改文件夹的所有权：

```
sudo mkdir -p /var/log/uwsgi
sudo chown -R ubuntu:ubuntu /var/log/uwsgi
```

步骤 4
执行 uWSGI，用新创建的配置文件作为参数：

```
uwsgi --ini /var/www/demoapp/ projectname _uwsgi.ini
```

访问服务器，现在 Nginx 可以连接到 uWSGI 进程了：

uWSGI Emperor

uWSGI Emperor 负责读取配置文件并且生成 uWSGI 进程来执行它们

创建一个初始配置来运行 emperor - /etc/init/uwsgi.conf：

```
description "uWSGI"
start on runlevel [2345]
stop on runlevel [06]
respawn

env UWSGI=/usr/local/bin/uwsgi
env LOGTO=/var/log/uwsgi/emperor.log

exec $UWSGI --master --emperor /etc/uwsgi/vassals --die-on-term --uid www-data --gid www-data --logto $LOGTO
```

最后一行运行 uWSGI 守护进程并让它到/etc/uwsgi/vassals 文件夹查找配置文件。创建这个文件夹，在其中建立一个到链到刚创建配置文件的符号链接。

```
sudo mkdir /etc/uwsgi && sudo mkdir /etc/uwsgi/vassals
sudo ln -s /var/www/ projectname / projectname _uwsgi.ini /etc/uwsgi/vassals
```

同时，最后一行说明用来运行守护进程的用户是 www-data。为简单起见，将这个用户设置成应用和日志文件夹的所有者。

```
sudo chown -R www-data:www-data /var/www/ projectname /
sudo chown -R www-data:www-data /var/log/uwsgi/
```

注意：我们先前安装的 Nginx 版本使用“www-data”这个用户来运行 Nginx，其他 Nginx 版本的可能使用“Nginx”这个替代用户。

由于 Nginx 和 uWSGI 都由同一个用户运行，我们可以在 uWSGI 配置中添加一个安全提升项。

打开 uWSGI 配置文件，将 chmod-socket 值由 666 更改为 644：

```
...
#permissions for the socket file
chmod-socket        = 644
```

现在我们可以运行 uWSGI 了：

```
sudo start uwsgi
```

最后，Nginx 和 uWSGI 被配置成启动后立即对外提供我们的应用服务。

出错解决解决

如果出现错误的话，第一个检查的地方是日志文件。Nginx 默认将错误信息写到 /var/log/nginx/errors.log 文件。

我们已经配置了 uWSGI emperor 将日志写到/var/log/uwsgi/emperor.log。这个文件夹还包含着每个配置应用的单独日志。我们的例子是 - /var/log/uwsgi/demoapp_uwsgi.log。
静态文件
如果应用提供静态文件的话，将下面的规则添加到 projectname _nginx.conf 文件：

```
location /static {
    root /var/www/ projectname /;
}
```

上面配置的结果就是所有在/var/www/ projectname /static 文件夹中的文件将由提供 Nginx 对外服务

3.2.2 opencv 运行环境的安装与配置

openCV 不建议使用 pip install opencv-python 的方式安装，否则会在高分辨率视频流下有卡顿现象。
安装依赖项

```
sudo apt-get install libv4l-dev
sudo apt-get install libv4l-dev
```

下载 opencv 源码
opencv 不建议
卸载之前的 opencv

```
pip uninstall opencv-contrib-python
pip uninstall opencv-contrib-python
```

编译 opencv
进入 opencv 目录

```
mkdir build
cd build
cmake -D WITH_LIBV4L=ON ../
make -j8
make install
ldconfig

cp lib/cv2.so /home/bruce/miniconda2/lib/python2.7/site-packages # 注意，这里是当前使用的python 环境

mkdir build
cd build
cmake -D WITH_LIBV4L=ON ../
make -j8
make install
ldconfig

cp lib/cv2.so /home/bruce/miniconda2/lib/python2.7/site-packages # 注意，这里是当前使用的
```

python 环境

验证
这次使用的是录制视频并输出视频的代码

```
def simple_video(minutes=5):
    start = int(time.time())
    cap = cv2.VideoCapture(1)

    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
    cap.set(cv2.CAP_PROP_FPS, 25)
    # cap.set(cv2.CV_CAP_PROP_BUFFERSIZE, 20)

    # Define the codec and create VideoWriter objectXVID
    # fourcc = cv2.VideoWriter_fourcc(*'XVID')
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter('{}.avi'.format(str(start)), fourcc, 25.0, (1920, 1080))

    while (cap.isOpened()):
        step = int(time.time())

        ret, frame = cap.read()
        if ret == True:
            frame = cv2.flip(frame, 0)

            # img = cv2.resize(frame,(1920,1080))
            # write the flipped frame
            out.write(frame)
            cv2.imshow('frame', frame)
            if (cv2.waitKey(1) & 0xFF == ord('q')) or ((step - start) >= minutes * 60):
                print("enter to the time used")
                break
        else:
            break

    # Release everything if job is finished
    cap.release()
    out.release()
    cv2.destroyAllWindows()


if __name__ == '__main__':
    simple_video(1)
```

```
def simple_video(minutes=5):
    start = int(time.time())
    cap = cv2.VideoCapture(1)

    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
    cap.set(cv2.CAP_PROP_FPS, 25)
    # cap.set(cv2.CV_CAP_PROP_BUFFERSIZE, 20)

    # Define the codec and create VideoWriter objectXVID
    # fourcc = cv2.VideoWriter_fourcc(*'XVID')
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter('{}.avi'.format(str(start)), fourcc, 25.0, (1920, 1080))

    while (cap.isOpened()):
        step = int(time.time())

        ret, frame = cap.read()
        if ret == True:
            frame = cv2.flip(frame, 0)

            # img = cv2.resize(frame,(1920,1080))
            # write the flipped frame
            out.write(frame)
            cv2.imshow('frame', frame)
            if (cv2.waitKey(1) & 0xFF == ord('q')) or ((step - start) >= minutes * 60):
                print("enter to the time used")
                break
        else:
            break

    # Release everything if job is finished
    cap.release()
    out.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    simple_video(1)
```

3.2.3 mysql 及 redis 数据库的配置

redis 安装好后在终端使用 redis-server 开启服务

Mysql 数据库表字段

字段名	数据类型	是否 NULL	说明
deviceID	varchar(40)	否	设备 ID（主键）
deviceUrl	varchar(300)	否	设备 url
deviceName	varchar(20)	否	设备别名或地点名

分别在 flask/myconfig.py 和 traffic/config.py 修改为自己的配置项。

四 附录

cuda 与英伟达驱动对应表

Table 1. CUDA Toolkit and Compatible Driver Versions

CUDA Toolkit	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 10.1.105	>= 418.39	>= 418.96
CUDA 10.0.130	>= 410.48	>= 411.31
CUDA 9.2 (9.2.148 Update 1)	>= 396.37	>= 398.26
CUDA 9.2 (9.2.88)	>= 396.26	>= 397.44
CUDA 9.1 (9.1.85)	>= 390.46	>= 391.29
CUDA 9.0 (9.0.76)	>= 384.81	>= 385.54
CUDA 8.0 (8.0.61 GA2)	>= 375.26	>= 376.51
CUDA 8.0 (8.0.44)	>= 367.48	>= 369.30
CUDA 7.5 (7.5.16)	>= 352.31	>= 353.66
CUDA 7.0 (7.0.28)	>= 346.46	>= 347.62