# 项目部署上线指南

## 更新Python环境到3.x

```
yum -y install gcc zlib-devel bzip2-devel openssl-devel ncurses-
devel sqlite-devel readline-devel tk-devel gdbm-devel db4-devel
libpcap-devel xz-devel libffi-devel
wget https://www.python.org/ftp/python/3.7.0/Python-3.7.0.tar.xz
xz -d Python-3.7.0.tar.xz
tar -xvf Python-3.7.0.tar
cd Python-3.7.0
./configure --prefix=/usr/local/python3 --enable-optimizations
make && make install
cd ~
vim .bash_profile
export PATH=$PATH:/usr/local/python3/bin
ln -s /usr/local/python3/bin/python3 /usr/bin/python3
ln -s /usr/local/python3/bin/pip3 /usr/bin/pip3
```

## 项目目录结构

下面是项目的目录结构，四个文件夹conf、logs、src和venv分别用来保存项目的配置文件、日志文件、源代码和虚拟环境。conf目录下的子目录cert中保存了配置HTTPS需要使用的证书和密钥。

```
project
├── conf
│   ├── cert
│   │   ├── 214915882850706.key
│   │   └── 214915882850706.pem
│   ├── nginx.conf
│   └── uwsgi.ini
├── logs
│   ├── access.log
│   ├── error.log
│   └── uwsgi.log
├── requirements.txt
├── src
│   └── fang
│       ├── common
│       ├── fang
│       ├── forum
│       ├── manage.py
│       ├── README.md
│       ├── rent
```

```
│       ├── static
│       └── templates
│
└── venv
    ├── bin
    │   ├── activate
    │   ├── activate.csh
    │   ├── activate.fish
    │   ├── celery
    │   ├── celerybeat
    │   ├── celeryd
    │   ├── celeryd-multi
    │   ├── coverage
    │   ├── coverage3
    │   ├── coverage-3.7
    │   ├── django-admin
    │   ├── django-admin.py
    │   ├── easy_install
    │   ├── easy_install-3.7
    │   ├── pip
    │   ├── pip3
    │   ├── pip3.7
    │   ├── __pycache__
    │   ├── pyrsa-decrypt
    │   ├── pyrsa-decrypt-bigfile
    │   ├── pyrsa-encrypt
    │   ├── pyrsa-encrypt-bigfile
    │   ├── pyrsa-keygen
    │   ├── pyrsa-priv2pub
    │   ├── pyrsa-sign
    │   ├── pyrsa-verify
    │   ├── python -> python3
    │   ├── python3 -> /usr/bin/python3
    │   └── uwsgi
    ├── include
    ├── lib
    │   └── python3.7
    ├── lib64 -> lib
    ├── pip-selfcheck.json
    └── pyvenv.cfg
```

## uWSGI的配置

可以激活项目的虚拟环境并通过pip安装uWSGI。

```
pip install uwsgi
```

/root/project/conf/uwsgi.ini

```
[uwsgi]
# 配置前导路径
base=/root/project
# 配置项目名称
name=fang
# 守护进程
master=true
# 进程个数
processes=4
# 虚拟环境
pythonhome=%(base)/venv
# 项目地址
chdir=%(base)/src/%(name)
# 指定python解释器
pythonpath=%(pythonhome)/bin/python
# 指定uwsgi文件
module=%(name).wsgi
# 通信的地址和端口(自己服务器的IP地址和端口)
socket=172.18.61.250:8000
# 日志文件地址
logto = %(base)/logs/uwsgi.log
```

可以先将“通信的地址和端口”项等号前面改为http来进行测试，如果没有问题再改回成socket，然后通过Nginx来实现项目的“动静分离”（静态资源交给Nginx处理，动态内容交给uWSGI处理）。

```
uwsgi --ini uwsgi.ini &
```

## Nginx的配置

### 全局配置

/etc/nginx/nginx.conf

```
# 全局配置
# 用户(可以设置为)
user root;
# 工作进程数(建议跟CPU的核数量一致)
worker_processes auto;
# 错误日志
error_log /var/log/nginx/error.log;
# 进程文件
pid /run/nginx.pid;

# 包含其他的配置
include /usr/share/nginx/modules/*.conf;

# 工作模式和连接上限
```

```
events {
    use epoll;
    worker_connections 1024;
}

# HTTP服务器相关配置
http {
    # 日志格式
    log_format  main  '$remote_addr - $remote_user [$time_local]
"$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    # 访问日志
    access_log  /var/log/nginx/access.log  main;
    # 开启高效文件传输模式
    sendfile            on;
    # 用sendfile传输文件时有利于改善性能
    tcp_nopush          on;
    # 禁用Nagle来解决交互性问题
    tcp_nodelay         on;
    # 客户端保持连接时间
    keepalive_timeout   15;
    types_hash_max_size 2048;
    # 包含MIME类型的配置
    include             /etc/nginx/mime.types;
    # 默认使用二进制流格式
    default_type        application/octet-stream;
    # 包含其他配置文件
    include /etc/nginx/conf.d/*.conf;

    # 包含项目的Nginx配置文件
    include /root/project/conf/*.conf;
}
```

### 局部配置

/root/project/conf/nginx.conf

```
server {
    listen          80;
    server_name     _;
    access_log      /root/project/logs/access.log;
    error_log       /root/project/logs/error.log;
    location / {
        include uwsgi_params;
        uwsgi_pass 172.18.61.250:8000;
    }
    location /static/ {
        alias /root/project/src/fang/static/;
```

```
        expires 30d;
    }
}
```

到此为止，我们可以启动Nginx来访问我们的应用程序，HTTP和HTTPS都是没有问题的，如果Nginx已经运行，在修改配置文件后，我们可以用下面的命令重新启动Nginx。

```
nginx -s reload
```

### 负载均衡配置

下面的配置中我们使用Nginx为HTTP、HTTPS以及Redis配置负载均衡。

```
user root;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

stream {
    upstream redis.local {
        server 172.18.61.250:36379;
        server 172.18.61.250:46379;
    }

    server {
        listen      6379;
        proxy_pass  redis.local;
    }
}

http {
    upstream fang.com {
        server 172.18.61.250:801;
        server 172.18.61.250:802;
        server 172.18.61.250:803;
    }

    server {
        listen       80 default_server;
        listen       [::]:80 default_server;
        listen       443 ssl;
```

```
        listen       [::]:443 ssl;

       ssl on;
        access_log /root/project/logs/access.log;
        error_log /root/project/logs/error.log;
        ssl_certificate /root/project/conf/cert/214915882850706.pem;
        ssl_certificate_key
/root/project/conf/cert/214915882850706.key;
        ssl_session_timeout 5m;
        ssl_ciphers ECDHE-RSA-AES128-GCM-
SHA256:ECDHE:ECDH:AES:HIGH:!NULL:!aNULL:!MD5:!ADH:!RC4;
        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers on;

        location / {
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
            proxy_buffering off;
            proxy_pass http://fang.com;
        }
    }
}
```

> *说明：上面的配置文件中的Nginx服务器（3个节点）和Redis服务器（2个节点，每个节点是1个master和2个slave的配置）都是通过Docker来创建的，实际部署的时候无论是否使用Docker进行部署，这些主机应该都是独立的服务器。*

## Keepalived

当使用Nginx进行负载均衡配置时，要考虑负载均衡服务器宕机的情况。为此可以使用Keepalived来实现负载均衡主机和备机的热切换，从而保证系统的高可用性。Keepalived的配置还是比较复杂，通常由专门做运维的人进行配置，一个基本的配置可以参照《Keepalived的配置和使用》。

## Docker

事实上，项目上线中最为麻烦的事情就是配置软件运行环境，环境的差异会给软件的安装和部署带来诸多的麻烦，而Docker正好可以解决这个问题。关于Docker在之前的文档中我们已经介绍过了，接下来我们对Docker的知识做一些必要的补充。

1. 创建镜像文件。

   将容器保存成镜像：

```
docker commit -m "..." -a "..." <container-name>
jackfrued/<image-name>
```

使用Dockerfile构建镜像：

```
# 指定基础镜像文件
FROM centos:latest

# 指定维护者信息
MAINTAINER jackfrued

# 执行命令
RUN yum -y gcc
RUN cd ~
RUN mkdir -p project/src
RUN mkdir -p project/logs

# 拷贝文件
COPY ...

# 暴露端口
EXPOSE ...

# 在容器启动时执行命令
CMD ~/init.sh
```

```
docker build -t jackfrued/<image-name> .
```

2. 镜像的导入和导出。

```
docker save -o <file-name>.tar <image-name>:<version>
docker load -i <file-name>.tar
```

3. 推送到DockerHub服务器。

```
docker tag <image-name>:<version> jackfrued/<name>
docker login
docker push jackfrued/<name>
```

4. 容器之间的通信。

```
docker run --link <container-name>:<alias-name>
```

我们在Docker中完成项目的部署，并且将整个部署好的容器打包成镜像文件进行分发和安装，这样就可以解决项目在多个节点上进行部署时可能遇到的麻烦。