Automation配置

## 配置升级python环境

### 说明

1. linux系统一般默认的python环境为2.6.x
2. 因平台的运行需要2.7.x中的某些特性，推荐python环境升级为2.7.8。
3. 因系统底层依赖python，盲目升级可能会有影响系统运行，所以此时需要在系统中安装多个python，即实现python的多版本共存。Pyenv就是这样一个python版本管理器。
4. 注意此方式升级只是针对当前用户升级，并非所有用户，例如此次我们就是安装在deploy用户中。

### 安装pyenv

什么是pyenv？pyenv就是一个可以在系统中安装多个python，但又不影响系统自带的python环境的Python版本管理器。

$ git clone https://github.com/yyuu/pyenv.git ~/.pyenv

### 定义pyenv环境变量

$ echo 'export PYENV\_ROOT="$HOME/.pyenv"' >> ~/.bashrc

$ echo 'export PATH="$PYENV\_ROOT/bin:$PATH"' >> ~/.bashrc

$ echo 'eval "$(pyenv init -)"' >> ~/.bashrc

$ exec $SHELL -l

### 通过pyenv安装python

#### 查看可安装的版本

$ pyenv install --list

Available versions:

2.7.10 # Python 2最新版本

3.4.3 # Python 3最新版本

anaconda-2.2.0 # 支持Python 2.6和2.7

anaconda3-2.2.0 # 支持Python 3.3和3.4

其中形如x.x.x这样的只有版本号的为Python官方版本，其他的形如xxxxx-x.x.x这种既有名称又有版本后的属于“衍生版”或发行版。Anaconda是一个和Canopy类似的科学计算环境，但用起来更加方便。自带的包管理器conda也很强大。

#### 安装Python的依赖包

$ sudo yum install readline readline-devel readline-static

$ sudo yum install openssl openssl-devel openssl-static

$ sudo yum install sqlite-devel

$ sudo yum install bzip2-devel bzip2-libs

#### 安装指定版本

$ pyenv install 2.7.8

该命令会从github上下载python的源代码，并解压到/tmp目录下，然后在/tmp中执行编译工作。若依赖包没有安装，则会出现编译错误，需要在安装依赖包后重新执行该命令。

#### 更新数据库

安装完成之后需要对数据库进行更新：

$ pyenv rehash

#### 查看当前已安装的python版本

$ pyenv versions

\* system (set by /home/seisman/.pyenv/version)

2.7.8

其中的星号表示当前正在使用的是系统自带的python。

#### 设置全局的python版本

$ pyenv global 2.7.8

$ pyenv versions

system

\* 2.7.8 (set by /home/seisman/.pyenv/version)

$ python -v

## 使用pip软件安装工具

$yum install python-pip

pip 是“A tool for installing and managing Python packages.”，也就是说pip是python的软件安装工具.

## 使用python虚拟环境

virtualenv 用来**创建隔**离的Python环境, 创建Python的虚拟环境可以使一个Python程序拥有独立的库library和解释器interpreter，而不用与其他Python程序共享统一个library和interpreter.通俗点就是：Virtualenv是一个python工具，它可以创建一个独立的python环境，这样做的好处是你的python程序运行在这个环境里，不会受到其他python library的版本问题影响。

### 安装virtualenv

$ pip install virtualenv

### 创建虚拟环境

$ virtualenv /data/Automation/

/data/Automation/是新创建的虚拟环境的名称。 同时会创建一个与虚拟环境名称相同的文件夹/data/Automation/, 里面存储了一个独立的Python执行环境。

### 进入虚拟环境

source /data/Automation/bin/activate

进入虚拟环境后，命令行的提示符会加入虚拟环境的名称，例如：(Automation)[deploy@test52 Automation]$

### 安装依赖软件

#### 首次安装环境

1. 确保当前处于虚拟环境的情况下
2. 安装程序包

pip install ansible Django django-filter djangorestframework gevent gunicorn Jinja2 MarkupSafe paramiko PyYAML sh MySQL-python

注意：程序对于的版本号ansible==1.9.0.1 Django==1.7 django-filter==0.9.2 djangorestframework==3.0.0 gevent==1.0.1 gunicorn==19.3.0 Jinja2==2.7.3 MarkupSafe==0.23 paramiko==1.15.2 PyYAML==3.11 sh==1.11，其中这些版本号可指定也可不指定

#### 迁移环境

##### 为什么迁移

之前安装部署过此项目环境，所以可采取迁移环境来对应操作

##### 如何进行迁移

1. 进入原虚拟环境dj\_release中
2. 执行 $ pip freeze > requirements.txt 将包依赖信息保存至requirements.txt中。
3. 进入当前虚拟环境中，执行 $ pip install –r requirements.txt，pip就会自动从网上下载并安装所有包。
4. 由于我们平台的数据库使用的是mysql，需安装mysql插件 $ pip install MySQL-python

##### 迁移中注意哪些

若在执行pip install –r requirements.txt时出现版本相关的错误可打开此文件将程序包对于的版本号做对应更改。

获取版本方法一：

$ pip install django-filter==0.9.212 #==后随意跟数字

Collecting django-filter==0.9.2888

Could not find a version that satisfies the requirement django-filter==0.9.2888 (from versions: 0.1.0, 0.2.0, 0.5.0, 0.5.1, 0.5.2, 0.5.3, 0.5.4, 0.6a1, 0.6, 0.7, 0.8, 0.9.0, 0.9.1, 0.9.2, 0.10.0)

No matching distribution found for django-filter==0.9.2888

获取版本方法二：

$ pip install django-filter=> 0.9.2 #将==更改成=>，程序会自动寻找更高级版本

### 退出虚拟环境方式

deactivate

## 部署程序文件

cd /data/Automation

tar xf dj.tar.gz

## 创建数据库

这里我们的数据库选择的是mysql

### 创建库

create database dj\_pro DEFAULT CHARACTER SET utf8 COLLATE utf8\_general\_ci;

### 授权用户

grant all privileges on dj\_pro.\* to djroot@'127.0.0.1' identified by 'huilian123';

flush privileges;

### 更改数据库相关文件

编辑local\_settings.py，更改DATABASES中的default下的键值，以匹配数据库连接设置

vi /data/Automation/dj/dj\_pro/local\_settings.py

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql', # Add 'postgresql\_psycopg2', 'postgresql', 'mysql', 'sqlite3' or 'oracle'. #更新为自己熟悉的数据库

'NAME': 'dj\_pro', # Or path to database file if using sqlite3. #数据库名 默认使用dj\_pro

'USER': 'djroot', # Not used with sqlite3. #更新为自己的授权账号

'PASSWORD': 'huilian123', # Not used with sqlite3. #更新为自己的授权密码

'HOST': '127.0.0.1', # Set to empty string for localhost. Not used with sqlite3. #更新为自己的数据库地址

'PORT': '3306', # Set to empty string for default. Not used with sqlite3. #更新为选择的数据库的端口

},

}

### 生成数据库程序文件

切换至项目目录下(cd /data/Automation/dj)，运行下面的命令：

./manage.py makemigrations release\_pro

其中：

manage.py: 一种命令行工具，允许你以多种方式与该DJango项目进行交互。键入python manage.py help 看一下它能做什么。

makemigrations: 基于当前的model创建新的迁移策略文件，值得注意的是，这个命令是Django 1.7中新增加的命令。它的使用非常简单，就是根据应用程序下的models.py中定义的各种模型的字符类型射生成数据库结构文件并将文件保存在项目程序文件的migrations中，比如：/data/Automation/dj/release\_pro/migrations/。若变更了模型，比如添加了一列或删除了一列，然后运行makemigrations，它就会扫描新的models.py文件并和当前版本生成的migrations文件中的模型进行比较，然后生成一系列新的文件。值得注意的是，这种方案并非真正完美，对于复杂的变更它可能做不到如你期望的那样检测出来，建议在做复杂变更时进入migrations中进行手动更改相应文件即可。

release\_pro：项目程序文件，此处必须使用“相对路径”。

### 应用到数据库

切换至项目目录下(cd /data/Automation/dj)，运行下面的命令：

./manage.py migrate

其中：

migrate：用于执行迁移更新动作的，这个动作就是将makemigrations生成的新模型应用到数据库并确保他们如期望般工作。

## 运行Django程序

切换至项目目录下(cd /data/Automation/dj)，运行下面的命令：

$./manage.py runserver

*Performing system checks...*

*System check identified no issues (0 silenced).*

*June 05, 2015 - 03:06:43*

*Django version 1.7, using settings 'dj\_pro.settings'*

*Starting development server at http://127.0.0.1:8000/*

*Quit the server with CONTROL-C.*

*Error: That port is already in use.*

这将会在端口8000上启动一个本地服务器(Django默认服务端口为8000), 并且只能***从你的这台电脑(是本机)连接和访问***。 既然服务器已经运行起来了，现在用网页浏览器访问 http://127.0.0.1:8000/ 。

而这里出现Error表示8000端口已被占用，那么想要更改服务器端口的话可将端口作为命令行参数传入：

$./manage.py runserver 12345

也行你会有这样的需求，Django程序部署在机房服务器上并非本地，该如何进行配置访问呢？

通过指定一个IP地址，你可以告诉服务器允许非本机连接访问。如果你想和其他人员共同享用同一个开发站点的话，该功能特别有用。“0.0.0.0”这个IP地址，告诉服务器去侦听任意的网络接口。

$./manage.py runserver 0.0.0.0:12345

完成这些设置，能访问此计算机的其他计算机就可以在浏览器中访问你的IP地址了，比如：<http://192.168.1.52:12345>

## 多进程运行Django程序

gunicorn -k gevent dj\_pro.wsgi:application --bind 0.0.0.0:12345

这里我们要介绍的主角就是Gunicorn，对于它我们这里做个简单的介绍：

Gunicorn是一个开源的Python WSGI HTTP服务器，移植于Ruby的Unicorn项目的采用pre-fork模式的服务器。Gunicorn服务器可与各种Web框架，包括django、flask、pyramid等。只要简单配置执行，轻量级的资源消耗，而且相当迅速。与各个Web结合紧密，部署很方便。缺点不支持HTTP 1.1,并发访问性能也不高。

其中：

-c CONFIG --config= CONFIG 指定的路径配置文件或Python模块。

-b BIND –bind= BIND 指定绑定服务器地址及端口

-w WORKERS --workers=WORKERS 指定工作进程。这个数量根据服务器的核数进行设置。

-k WORKERCLASS, --worker-class=WORKERCLASS 指定工作进程的运行类型。可以将其设置为sync, eventlet, gevent, or tornado, gthread, gaiohttp.而其中sync是默认的类型

dj\_pro.wsgi:application 其中dj\_pro.wsgi指定是dj\_pro/wsgi.py application就是那个wsgifunc的名字。

## 后台运行Django程序

### 利用nohup运行

nohup 命令运行由 Command参数指定的命令，忽略所有挂断（SIGHUP）信号。在注销后使用 nohup 命令运行后台中的程序。要运行后台中的 nohup 命令，添加& （ 表示“and”的符号）到命令的尾部。

如果不将 nohup 命令的输出重定向，输出将附加到当前目录的 nohup.out 文件中。如果当前目录的 nohup.out 文件不可写，输出重定向到 $HOME/nohup.out 文件中。

运行命令如下：

nohup gunicorn -k gevent dj\_pro.wsgi:application --bind 0.0.0.0:12345 &

### 利用screen运行

Screen是一个可以在多个进程之间多路复用一个物理终端的窗口管理器。Screen中有会话的概念，用户可以在一个screen会话中创建多个screen窗口，在每一个screen窗口中就像操作一个真实的telnet/SSH连接窗口那样。

运行方式如下：

直接在命令行键入screen命令

$ screen –S web

screen将创建一个执行shell的窗口。你可以执行任意shell程序，就像在ssh窗口中那样。其中-S参数表示创建一个名字为web的会话窗口如：

$ gunicorn -k gevent dj\_pro.wsgi:application --bind 0.0.0.0:12345

在该窗口中键入exit退出该窗口，如果这是该screen会话的唯一窗口，该screen会话退出，否则screen自动切换到前一个窗口。

显然exit直接退出违背了我们的初衷，我们的想法是暂时离开而已，方法如下：

在screen窗口下键入Ctrl+a 输入d 即可，Screen会给出detached提示；这样就隐藏起来了

假如我们想要切换到刚才的screen窗口进行重新连接会话：

$ screen –r web

此处的web就是刚才创建的名称为web的窗口

那么假如我们由于搁置的时间过长忘记之前窗口的名称了，没关系，可这样做：

$ screen –ls

$ screen –r 20981

看看出现什么了，太棒了，一切都在。找到自己的窗口。继续干吧。

问题一、*出现Cannot open your terminal '/dev/pts/1' - please check.*

*解决方案：$script /dev/null*

## 人性化输出信息插件沿用

首先确保输出信息插件已添加成功

$ll /usr/share/ansible\_plugins/callback\_plugins/ 检查是否存在log\_release.py脚本

存在脚本，将插件进行平台集成

vi /data/Automation/lib/python2.7/site-packages/ansible/callbacks.py

callback\_plugin.task = task *#搜索这一行在其后面输入以下两句*

log\_add = []

error\_add = []

log\_unflock(runner) *#搜索这一行在其后面输入以下5句*

try:

log\_add.append(msg2)

except:

pass

return msg2

不存在脚本，请参考ansible日志插件文章进行插件添加。