Informe técnico final

Anexo 13

Manual de instalación del sistema para la estimación de saldos netos migratorios

1. Manual de instalación del sistema para la estimación de saldos netos migratorios

1.1 Introducción

Este documento tiene el objetivo de proporcionar la información necesaria para la instalación del sistema para la estimación de saldos netos migratorios en un servidor de producción.

1.2 Requerimientos

La instalación puede realizarse en cualquier computadora o sistema operativo que soporte Python 2.7. Sin embargo, para evitar complicaciones en el seguimiento de este manual se recomienda fuertemente un equipo de computo con lo siguiente:

- 512mb de memoria RAM como mínimo
- Sistema operativo Ubuntu Server 12.04 LTS instalado
- Python 2.7

Dependiendo de las características del equipo de computo, el sistema puede tomar más o menos tiempo de ejecución.

1.3 Convenciones

El código y líneas de comando mostrados en este manual se presentan en cuadros de texto con la tipografía llamada `Courier`. Las líneas que empiecen con el signo numeral ("#") representan comentarios y descripciones de la funcionalidad de las líneas que siguen. Las líneas de comando en este manual inician con el signo de pesos ("$"), los nombres de variables se encierran en los signos mayor-que y menor-que ("<…>") y deberán ser remplazadas por los valores que el lector de este manual considere adecuados.

1.4 Preparación de Ubuntu Server para Django en producción

Una vez instalado Ubuntu Server 12.04 LTS e iniciada la sesión, se prepara el sistema con los siguientes pasos.

1.4.1 Actualizar los paquetes de Ubuntu

```
# Actualizar el índice local de paquetes
        $ sudo apt-get update

# Actualizar todos los paquetes que puedan ser actualizados
        $ sudo apt-get dist-upgrade

# Remover los paquetes que no sean necesarios
        $ sudo apt-get autoremove

# Reiniciar la maquina (solo necesario para algunas actualizaciones)
        $ sudo reboot
```

1.4.2 Instalar *build-essential* y *python-dev*

- *build-essential* es el paquete que provee todas las herramientas de compilación estándar de C.
- *python-dev* provee los archivos necesarios para compilar módulos Python/C.

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

1.4.3 Instalar *distribute* y *pip*,

distribute y *pip* son herramientas para instalar paquetes de Python.

```
# Descargar distribute

      $ curl -O http://python-distribute.org/distribute_setup.py

# Instalar distribute

      $ sudo python distribute_setup.py

# Remover archivos de instalación

      $ rm distribute*

# Usar distribute para instalar pip

      $ sudo easy_install pip
```

1.4.4 Instalar *virtualenv* y *virtualenvwrapper*

- *virtualenv* se usa para aislar en ambientes virtuales diferentes los paquetes que se van a instalar.
- *virtualenvwrapper* se usa para agilizar *virtualenv*.

```
# Instalar virtualenv y virtualenvwrapper

      $ sudo pip install virtualenv virtualenvwrapper

# Editar el archivo .bashrc con ayuda de vim

      $ vim .bashrc

# Agregar la siguiente linea al final del archivo para habilitar el
# virtualenvwrapper

      $ source /usr/local/bin/virtualenvwrapper.sh

# Guardar y cerrar el editor

# Salir y volver a acceder

      $ exit
```

1.4.5 Crear un ambiente virtual

```
# Crear un ambiente virtual

      $ mkvirtualenv <NOMBRE_DEL_AMBIENTE_VIRTUAL>

# Algunos comandos útiles para el ambiente virtual:

# Desactivar  ambiente  virtual

      $  deactivate

# Activar  ambiente  virtual  o  cambiar  a  otro

      $  workon  <NOMBRE_DEL_AMBIENTE_VIRTUAL>

#  Mostrar  lo  paquetes  instalados  en  un  ambiente  virtual

      $  workon  <NOMBRE_DEL_AMBIENTE_VIRTUAL>
      $  pip  freeze
```

1.4.6 Instalar Django 1.4

Asegurarse de que se tiene activado el ambiente virtual en donde se instalará <NOMBRE_DEL_AMBIENTE_VIRTUAL>.

```
# instalar django 1.4

      $ pip install Django==1.4


# Instalar docutils, utilizado para el administrador de Django

      $ pip install docutils


# Si se quiere probar Django, hacer un proyecto de prueba

      $ django-admin.py startproject <NOMBRE_DE_LA_APP>

      $ cd <NOMBRE_DE_LA_APP>


# Darle permisos de ejecución a manage.py

      $ chmod +x manage.py

# correr el servidor

      $ ./manage.py runserver 0.0.0.0:8000
```

1.4.7 Instalar la librería de imágenes *Pillow*

```
# Instalar librerías

      $ sudo apt-get install libjpeg8-dev libfreetype6-dev zlib1gdev

# Instalar pillow

      $ pip install pillow
```

1.4.8 Instalar PostgreSQL

```
      $ sudo apt-get install postgresql libpq-dev

      $ pip install psycopg2
```

1.4.9 Instalar *memcached*

- *memcached* incrementa el rendimiento y minimiza las consultas a la base de datos.

```
# Instalar el librerías y servidor con memcach

      $ sudo apt-get install memcached libmemcached-dev


# Instalar pylibmc

      $ pip install pylibmc


# Editar en settings.py los ajustes de CACHES


$ vim <TU_APP>/settings.py


      CACHES = {

            'default': {

                    'BACKEND':
                  'django.core.cache.backends.memcached.PyLibMCCache',

                  'LOCATION': '127.0.0.1:11211',
            }
      }
```

1.4.10 Instalar RabbitMQ

Para realizar la ejecución asíncrona de tareas se requiere de un servidor que administre las tareas que las diferentes tareas de *celery* (paquetería utilizada para la ejecución asíncrona de tareas) van a estar realizando. Para esto se utilizara RabbitMQ.

```
$ sudo apt-get install rabbitmq-server

$ sudo rabbitmqctl add_user <USER_RABBIT> <CONTRASEÑA>

$ sudo rabbitmqctl add_vhost <RABBIT_VHOST>

$ sudo rabbitmqctl set_permissions -p <RABBIT_VHOST> <USER_RABBIT> ".*"
".*" ".*"
```

1.4.11 Instalar Celery

Celery es la paquetería que nos permite ejecutar tareas asíncronas.

```
# Instalar celery

      $ pip install django-celery

# Editar algunas configuraciones en settings.py

      $ vim <TU_APP>/settings.py

# Agregar djcelery a  INSTALLED_APPS

            INSTALLED_APPS =
                      ...

                      'djcelery',
                      ...

# Agregar estas configuraciones(seguimos settings.py)

      BROKER_URL =
"amqp://<USUARIO_RABBIT>:<CONRASEÑA_RABBIT>@localhost:5672/<RABBIT_VHOST>"

      CELERY_RESULT_BACKEND = "database"

# Elegir la configuración que concuerde con la configuración de tu base de
datos.

      CELERY_RESULT_DBURI = "postgresql://<USUARIO_DE_LA_BD>:
<CONTRASEÑA_DE_LA_BD>@localhost/<NOMBRE_DE_LA_BD>"

# Poner al final del settings.py las siguientes lineas

      import djcelery
      djcelery.setup_loader()

# Comandos útiles para celeryd

      #  Iniciar  celeryd  con  Beat  and  Event

            $  ./manage.py  celeryd  -B  -E

      #  Iniciar  celeryd
```

1.4.12 Instalar *gunicorn*

gunicorn es el WSGI (*web server gateway interface*) que se va a utilizar. Un WSGI es la manera en la que nuestra aplicación se va a comunicar con el servidor web.

```
# Instalar gunicorn

      $ pip install gunicorn

#Instalar gevent

      $ sudo apt-get install libevent-dev
      $ pip install gevent

# Agregar gunicorn dentro de INSTALLED_APPS(settings.py)

      $ vim <YOUR_APP>/settings.py

            INSTALLED_APPS =

                  ...
                     'gunicorn',
                  ...
# Probar gunicorn


$ ./manage.py run_gunicorn -w 4 -k gevent
```

1.4.13 Instalacion de paquetes que son requerimientos de la aplicación.

```
$pip install -r requirements.txt
```

1.4.14 Instalar supervisor

Supervisor va a administrar los servicios de *celery*, *celerycam*, y *gunicorn*. Para esto se deben de crear los archivos de configuración *celeryd.conf*, *celerycam.conf* y *gunicorn.conf*.

```
# Instalar supervisor

      $ sudo apt-get install supervisor
```

```
# Para la configuración de celeryd

# En /etc/supervisor/conf.d/celeryd.conf escribir

# Nombre del servicio
[program:celeryd]

# Comando para iniciar celery
# Nota: Para hacer referencia a otra ubicación del settings se utiliza
# --settings=<>
# Ejemplo: python manage.py celeryd -B -E --settings=directorio.settings

command = /home/<USERNAME>/.virtualenvs/
<NOMBRE_DE_VIRTUALENV>/bin/python
/home/<USERNAME>/<NOMBRE_DE_LA_APP>/manage.py celeryd -B -E --
settings=<UBICACION>

# Directorio en el que vamos a estar al correr command

directory = /home/<USERNAME>/<NOMBRE_DE_LA_APP>

# Nombre del usuario con el que vamos a correr el comando

user = <USERNAME>

# Correr el comando cuando se inicie o reiniciar si falla

autostart = true
autorestart = true

# Quitar stdout y stderr de celery y escribir estos archivos de log

stdout_logfile = /var/log/supervisor/celeryd.log
```

```
# Contenido de  /etc/supervisor/conf.d/celerycam.conf

[program:celerycam]
# Nota: Para hacer referencia a otra ubicación del settings se utiliza
# --settings=<>
#Ejemplo: python manage.py celeryd -B -E --settings=directorio.settings
command = /home/<USERNAME>/.virtualenvs/<VIRTUALENV_NAME>/bin/python
/home/<USERNAME>/<APP_NAME>/manage.py celerycam --settings=<UBICACION>
directory = /home/<USERNAME>/<APP_NAME>
user = <USERNAME>
autostart = true
autorestart = true
stdout_logfile = /var/log/supervisor/celerycam.log
stderr_logfile = /var/log/supervisor/celerycam_err.log
```

```
# Contenido de /etc/supervisor/conf.d/gunicorn.conf

[program:gunicorn]
#Nota: Para hacer referencia a otra ubicación del settings se utiliza --
settings=<>
#Ejemplo: python manage.py celeryd -B -E --settings=directorio.settings

command = /home/<USERNAME>/.virtualenvs/<VIRTUALENV_NAME>/bin/python
/home/<USERNAME>/<APP_NAME>/manage.py run_gunicorn -w 4 -k gevent --
settings=<UBICACION>
directory = /home/<USERNAME>/<APP_NAME>
user = <USERNAME>
autostart = true
autorestart = true
```

1.4.15 Reiniciar *supervisor*

```
$ sudo service supervisor restart

# Comandos útiles de supervisor
# Reiniciar/detener/comenzar  todos  los  servicios administrados
# por supervisor

$  sudo  supervisorctl  restart  all
$  sudo  supervisorctl  stop  all
$  sudo  supervisorctl  start  all

#  Reiniciar  únicamente  celeryd

$  sudo  supervisorctl  restart  celeryd

#  Comenzar  únicamente  gunicorn

$  sudo  supervisorctl  start  gunicorn
```

1.4.16 Crear directorios Static y Media

Ninguna de estos directorios contendrá el código de la funcionalidad de la aplicación, el contenido de la carpeta Static son generalmente archivos tipo css, js e imagines (se puede pasar todo el contenido o solo realizar un enlace simbólico de los directorios fuente). El contenido del directorio Media contendrá archivos que suba un usuario.

```
sudo mkdir /var/www
sudo mkdir /var/www/static
sudo mkdir /var/www/media

#Otorgar permisos sobre estos directorios
sudo chown -R <USERNAME>:www-data /var/www


$ vim <YOUR_APP>/settings.py

# Cambiar rutas de direcotorios
MEDIA_ROOT = '/var/www/media/'
MEDIA_URL = '/media/'
STATIC_ROOT = '/var/www/static/'
STATIC_URL = '/static/'

#Ejecutar cuando los archivos estáticos son cambiados

$ ./manage.py collectstatic
```

1.4.17 Instalar Nginx

```
# Instalar nginx

$ sudo apt-get install nginx

# Crear y editar los archivos de configuración para nginx

# Remover el enlace simbólico que está por default

$ sudo rm /etc/nginx/sites-enabled/default


# Crear un nuevo archivo de configuración y hacerle un enlace simbólico

$ sudo touch /etc/nginx/sites-available/<TU_APP>

$ cd /etc/nginx/sites-enabled

$ sudo ln -s ../sites-available/<TU_APP>
```

```
# Editar el archivo de configuración

$ vim /etc/nginx/sites-available/<TU_APP>

# Definir un servidor upstream llamado gunicorn en el puerto 8000

upstream gunicorn {
    server localhost:8000;
}

# Hacer un servido nginx

server {

    # Escuchar en el puerto 80

    listen 80;

    # Para solicitudes a estos dominios

    server_name 192.168.6.149;

    # Buscar en este directorio los archivos a servir

    root /var/www/;

    # Guardar los logs en los siguientes archivos

    access_log /var/log/nginx/<TU_APP>.access.log;
    error_log /var/log/nginx/<TU_APP>.error.log;
    client_max_body_size 0;
    try_files $uri @gunicorn;

    location @gunicorn {

        client_max_body_size 0;

        proxy_pass http://gunicorn;

        proxy_redirect off;

        # Definir el tiempo de respuesta necesario

        proxy_read_timeout 5m;

        proxy_set_header Host            $host;

       proxy_set_header X-Real-IP       $remote_addr;

       proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

1.4.18 Reiniciar nginx

```
$ sudo service nginx restart
```

1.5 Configuración de PostgreSQL

En esta sección se presentan los pasos para la configuración de la base de datos en PostgreSQL y sus permiso a usuarios.

1.5.1 Asignar permisos sobre el directorio

```
$ sudo chown -R poblacion: <direccion>/esne/
```

1.5.2 Creación de la base de datos y el usuario

```
$ sudo su postgres -c psql
      # Comandos SQL
      CREATE database colef_poblacion_db owner postgres;
      CREATE user poblacion with password '12345';
      ALTER database colef_poblacion owner poblacion;
```

1.5.3 Cambiar contraseña al usuario postgres

```
$ sudo su postgres -c psql

      # Comandos SQL
      ALTER USER postgres WITH PASSWORD 'password';
```

1.5.4 Cargar archivo .sql

```
$ sudo bash
$ su postgres
$ psql -U postgres -d colef_poblacion_db -f archivo.sql
```

1.5.5 Permisos y usuario para postgres

```
$sudo su postgres -c psql
\c colef_poblacion_db
GRANT SELECT ON ALL TABLES IN SCHEMA public to poblacion
GRANT SELECT ON ALL SEQUENCES IN SCHEMA public to poblacion
GRANT SELECT ON ALL TABLES IN SCHEMA public to poblacion
GRANT SELECT ON ALL SEQUENCES IN SCHEMA public to poblaci
```

Ejemplo:

```
postgres=# GRANT SELECT ON ALL TABLES IN SCHEMA public TO asuario;
postgres=# GRANT SELECT ON ALL SEQUENCES IN SCHEMA public TO asuario;
postgres=# GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO asuario;
```