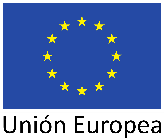




MANUAL DE EXPLOTACIÓN Y MANTENIMIENTO



Programa Operativo Fondo Europeo de Desarrollo Regional Aragón 2014-2020

*Construyendo Europa desde Aragón*

Contenido

[1. Creación del entorno 4](#_Toc58601404)

[1.1 Arquitectura del sistema 4](#_Toc58601405)

[1.2 Directorios del proyecto 4](#_Toc58601406)

[1.2.1 Servidores frontend 4](#_Toc58601407)

[1.2.2 Servidores backend 4](#_Toc58601408)

[1.3 Instalación de Python 3.6.9 5](#_Toc58601409)

[1.3.1 Lista de paquetes 5](#_Toc58601410)

[1.3.2 Descarga de paquetes 5](#_Toc58601411)

[1.3.3 Instalación de paquetes 8](#_Toc58601412)

[1.3.4 Resultado y verificación 8](#_Toc58601413)

[1.4 Configuración de PostgreSQL 9](#_Toc58601414)

[1.5 Entorno virtual servidores *frontend* 10](#_Toc58601415)

[1.6 Entorno virtual servidores *backend* 10](#_Toc58601416)

[1.6.1 Paquetes del entorno 10](#_Toc58601417)

[1.6.2 Creación del entorno *online* 11](#_Toc58601418)

[1.6.3 Configuración de Apache Airflow 11](#_Toc58601419)

[1.6.4 Despliegue en *backend* de desarrollo 12](#_Toc58601420)

[1.6.5 Traspaso a entorno *backend* de preproducción 13](#_Toc58601421)

[1.6.6 Traspaso a entorno *backend* de producción 14](#_Toc58601422)

[2. Explotación y mantenimiento 15](#_Toc58601423)

[2.1 Scripts de *bash* 15](#_Toc58601424)

[2.2 Gestión de logs 15](#_Toc58601425)

[2.2.1 Logs de Airflow 15](#_Toc58601426)

[2.2.2 Logs de código 15](#_Toc58601427)

[2.3 Invocaciones a scripts Python 16](#_Toc58601428)

[3. Copias de seguridad 17](#_Toc58601429)

[3.1 Recuperación de copias de seguridad 17](#_Toc58601430)

Control de versiones

|  |  |  |  |
| --- | --- | --- | --- |
| Versión | Fecha | Autor | Descripción |
| 1.0 |  | Hiberus | Primera versión (incompleta) del documento |

# Creación del entorno

En este capítulo se explica el proceso y los pasos que han sido requeridos para instalar las herramientas *open source* descritas en el apartado 1.1 del Manual Técnico. También se detalla como desplegarlos en los servidores de los entornos de desarrollo, preproducción y producción.

## Arquitectura del sistema

La arquitectura de los servidores en desarrollo, preproducción y producción es idéntica. El sistema consta de dos servidores distintos, uno ejerciendo de *frontend* y otro de *backend*. Sus características principales son las siguientes:

* Servidor *frontend*: sistema Linux basado en Red Hat (RHLN) y arquitectura x86\_64. Dispone de conexión a Internet y de la versión Python 3.6.5 ya instalada.
* Servidor *backend*: sistema Linux basado en Red Hat (RHLN) y arquitectura x86\_64. No dispone ni de conexión a Internet ni tampoco de ninguna versión de Python3 instalada inicialmente.

Aunque la aplicación desarrollada en este proyecto utiliza ambos servidores, la mayor parte de la ejecución se realiza en el servidor *backend*. El servidor *frontend* solo se utiliza para realizar la descarga de los Boletines Oficiales, mientras que el *backend* los recibe tras la descarga, los procesa, extrae toda la información requerida y los almacena. Todo este proceso se ejecuta diariamente y es orquestado por Apache Airflow, descrito en la sección 1.1.3 del Manual técnico.

## Directorios del proyecto

La estructura de directorios del proyecto es diferente para los servidores de producción y de preproducción. A continuación, se muestra la estructura por directorios. Para un mayor detalle sobre cada uno de los ficheros se puede consultar la sección 2.1.

### Servidores frontend

/data/

|---- apps/

|--- ingesta-BO/

|--------- env/

|--------- run.sh

|--------- pre/

|------ ficheros\_configuracion/

|------ ingesta/

|------ data/

### Servidores backend

/data/

|---- apps/

|--- ingesta-BO/

|--------- data/

|--------- pre/

|--- ingesta-BO-env/

|--- airflow/

|--------- airflow.cfg

|--------- dags/

|---- DAG-ingesta.py

|--------- logs/

|---- ingesta-BO/

## Instalación de Python 3.6.9

El primer paso para crear el entorno de ejecución es instalar una versión de Python3 junto con sus librerías principales en los servidores *backend*. Debido a que son servidores sin conexión a Internet (*offline*), los paquetes de instalación deben ser obtenidos previamente desde un terminal/servidor que sí disponga de conexión (*online*). Los paquetes de instalación son archivos .rpm (Red Hat Package Manager) que pueden obtenerse de manera fiable a través de repositorios oficiales. Una vez obtenidos, basta con copiarlos en el servidor *backend* y realizar un proceso de instalación estándar a través de la herramienta yum.

En los siguientes apartados se enumeran y detallan los paquetes que se han instalado, los comandos para descargarlos e instalarlos, el resultado de la instalación y cómo verificar que el proceso se ha realizado de manera correcta.

**NOTA**: los paquetes descritos se instalaron de manera satisfactoria en los servidores *backend* de los tres entornos (desarrollo, preproducción y producción), habilitando la versión 3.6.9 de Python. En todos los casos, la instalación de los .rpm la llevó a cabo un administrador del sistema, ya que se requieren permisos de superusuario. La descarga y copia de los .rpm la realizaron los desarrolladores del proyecto.

### Lista de paquetes

Los paquetes .rpm requeridos pueden separarse en dos tipos, los asociados propiamente a Python3 y sus librerías y aquellos que Python3 requiere, es decir, sus dependencias principales:

**Python y librerías principales**

* rh-python36-python-libs
* rh-python36
* rh-python36-python
* rh-python36-python-devel
* rh-python36-python-setuptools
* rh-python36-python-pip
* rh-python36-runtime
* rh-python36-python-virtualenv

**Dependencias principales**

* libc.so.6
* glibc
* scl-utils-build
* dwz
* perl-srpm-macros
* python-srpm-macros
* redhat-rpm-config

### Descarga de paquetes

El proceso de obtención se ha llevado a cabo por los desarrolladores en un sistema Centos7 con conexión a Internet y a través de la utilidad yum, propia de sistemas Linux basados en Red Hat y similares (como Centos7). Los comandos realizados fueron los siguientes:

$ yum install yum-utils centos-release-scl epel-release

$ mkdir ./packages

$ yumdownloader --destdir=./packages/ --resolve \

libc.so.6 \

glibc \

scl-utils-build \

dwz \

perl-srpm-macros \

python-srpm-macros \

redhat-rpm-config \

rh-python36-python-libs \

rh-python36 \

rh-python36-python \

rh-python36-python-devel \

rh-python36-python-setuptools \

rh-python36-python-pip \

rh-python36-runtime \

rh-python36-python-virtualenv

El resultado de ejecución es una carpeta packages que contiene los archivos .rpm mencionados previamente en el apartado 1.3.1. El detalle de dichos paquetes (posteriormente instalados en los servidores *backend*) se resume en la siguiente tabla:

|  |  |  |  |
| --- | --- | --- | --- |
| Paquete | Arquitectura | Versión | Repositorio |
| nss-softokn-freebl | x86\_64 | 3.44.0-8.el7\_7 | base |
| glibc | x86\_64 | 2.17-307.el7.1 | base |
| scl-utils-build | x86\_64 | 20130529-19.el7 | base |
| dwz | x86\_64 | 0.11-3.el7 | base |
| perl-srpm-macros | noarch | 1-8.el7 | base |
| python-srpm-macros | noarch | 3-32.el7 | base |
| redhat-rpm-config | noarch | 9.1.0-88.el7.centos | base |
| rh-python36 | x86\_64 | 2.0-1-el7 | centos-sclo-rh |
| rh-python36-python | x86\_64 | 3.6.9-2.el7 | centos-sclo-rh |
| rh-python36-python-libs | x86\_64 | 3.6.9-2.el7 | centos-sclo-rh |
| rh-python36-python-devel | x86\_64 | 3.6.9-2.el7 | centos-sclo-rh |
| rh-python36-python-setuptools | noarch | 36.5.0-1.el7 | centos-sclo-rh |
| rh-python36-python-virtualenv | noarch | 15.1.0-2.el7 | centos-sclo-rh |
| rh-python36-python-pip | noarch | 9.0.1-2.el7 | centos-sclo-rh |
| rh-python36-runtime | x86\_64 | 2.0-1.el7 | centos-sclo-rh |

#### Descarga desde los repositorios

Aunque el proceso de descarga se llevó a cabo a través de los comandos anteriores, no solo pueden descargarse los archivos .rpm a través de la herramienta yum. Si no se dispone de un sistema Centos como el utilizado, se puede acceder a los repositorios directamente a través de la web, utilizando cualquier navegador (Firefox, Google Chrome, Opera, etc.).

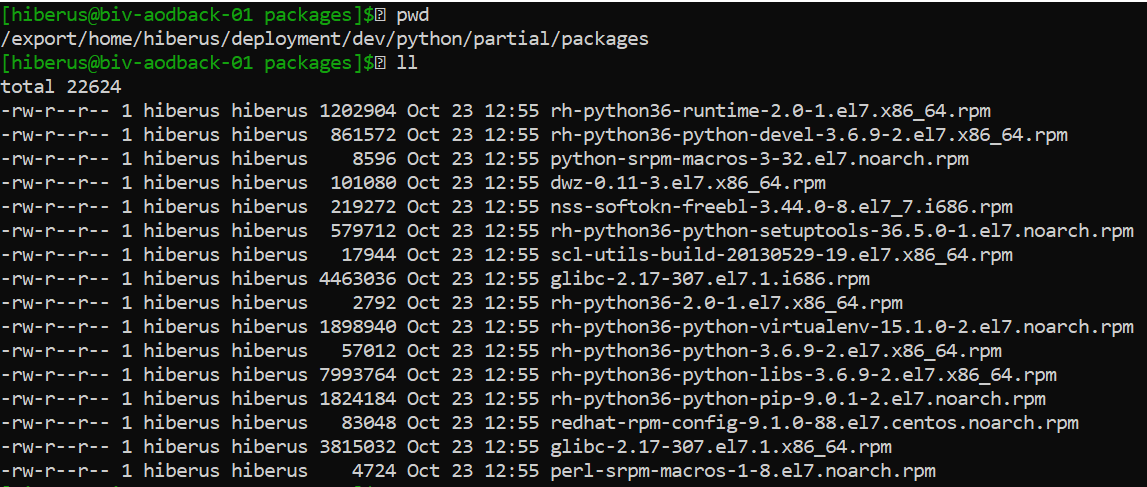
Ambas opciones, la descarga vía web y el uso de yum (que accede automáticamente a los repositorios), son válidas, sin embargo, es importante verificar siempre la identidad y la oficialidad, o no oficialidad, de los sitios de descarga.

Los repositorios a los que accedieron los desarrolladores utilizando yum son los siguientes:

|  |  |  |
| --- | --- | --- |
| Nombre | URL Base | Oficial |
| CentOS-7 – Base [base] | http://mirror.centos.org/centos/7/os/x86\_64/ | Si |
| Extra Packages for Enterprise  Linux 7 – x86\_64 [epel] | http://download.fedoraproject.org/pub/epel/7/x86\_64 | Si |
| CentOS-7 – SCLo sclo  [centos-sclo-sclo] | http://mirror.centos.org/centos/7/sclo/x86\_64/sclo | Si |
| CentOS-7 – SCLo rh  [centos-sclo-rh] | http://mirror.centos.org/centos/7/sclo/x86\_64/rh | Si |

#### Disponibilidad de los paquetes

Todos los archivos .rpm utilizados para instalar Python3 están disponibles en el servidor *backend* del entorno de desarrollo:



**PATH**: /export/home/hiberus/deployment/python/partial/packages

### Instalación de paquetes

Una vez se dispone de los archivos .rpm en los servidores *backend* (*offline*), para instalarlos solo es necesario que un usuario con permisos de administrador (grupo *sudoers*) ejecute el siguiente comando:

$ cd <carpeta con los rpm>

$ sudo yum –-nogpgcheck localinstall <archivo rpm>

También pueden instalarse todos los paquetes sustituyendo <archivo rpm> por \*. Sin embargo, se recomienda instalar primero las dependencias principales de Python (ver sección 1.3.1).

### Resultado y verificación

Si el proceso se ha ejecutado correctamente, en el servidor se dispondrá de un intérprete de Python 3.6.9 en la siguiente ruta:

$ /opt/rh/rh-python36/root/usr/bin/python

Junto a él estarán los ejecutables de pip y virtualenv, así como pyvenv y *aliases* de los ejecutables. Nótese que la ruta de instalación no es la común /usr/bin/. Esto es debido a que los sistemas basados en Red Hat y similares contienen programas básicos que requieren Python2 para funcionar correctamente (instalado por defecto en sistemas de estas características). Para evitar incompatibilidades y problemas en el sistema, se ha realizado la instalación de Python3 a través de las *Software Collections*. Para activar y utilizar Python3, basta con ejecutar:

$ scl enable rh-python36 bash

Para una mejor verificación, se pueden ejecutar los comandos que aparecen en la siguiente ilustración, en la que se demuestra cómo la versión y el ejecutable de Python cambia de Python2 a Python3 tras ejecutar el comando scl enable rh-python36 bash. Para desactivar Python3 tras ejecutar el comando scl, basta con lanzar el comando exit.

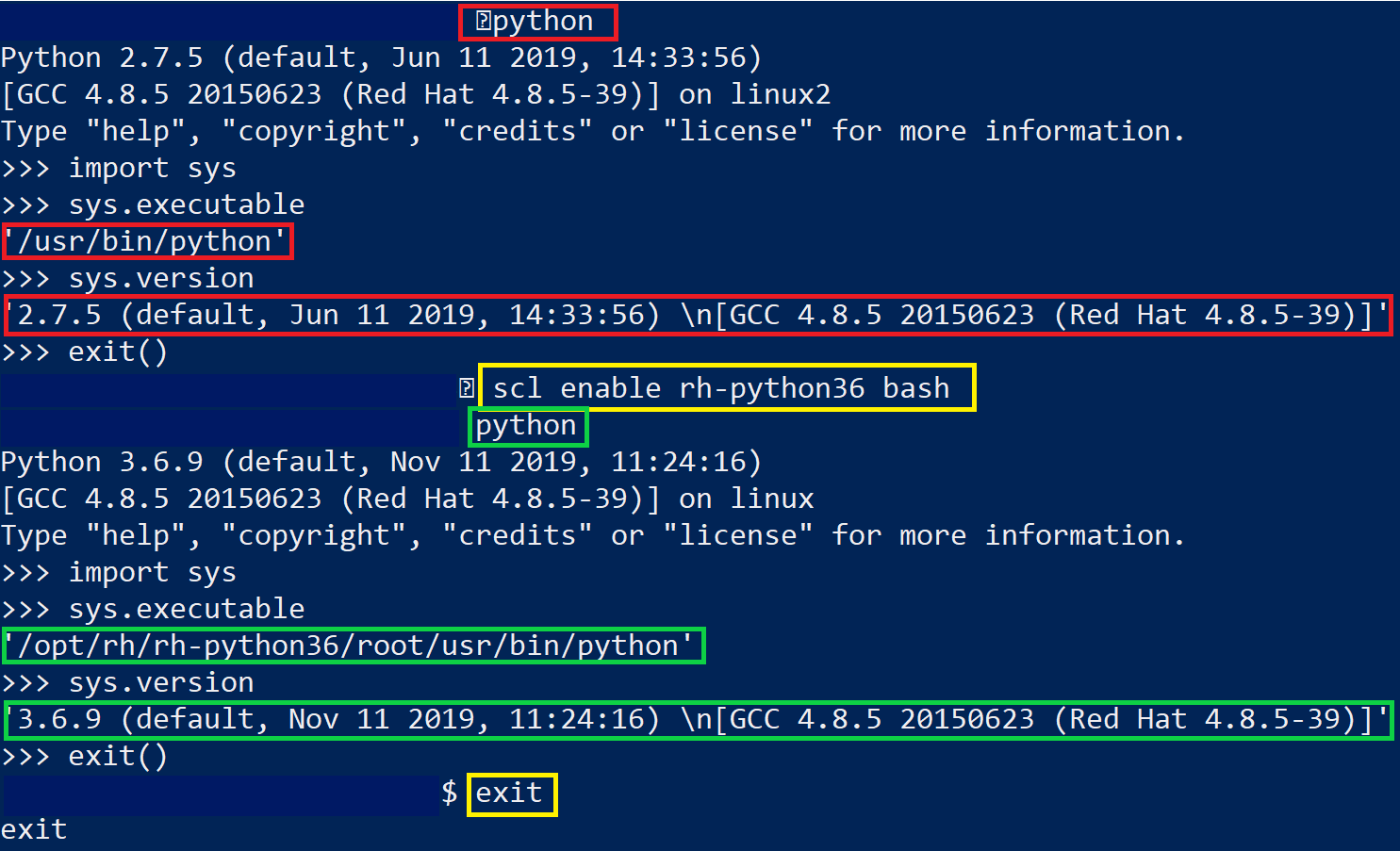


Ilustración 1: ejecución de Python3 con scl

## Configuración de PostgreSQL

PostgreSQL no solo es utilizado para almacenar los resultados del procesamiento de Boletines Oficiales, sino también como *metadatabase* por Apache Airflow. Airflow requiere de un sistema de tablas, almacenadas en una BBDD, para poder guardar registros asociados a las diferentes tareas, conexiones, resultados o fechas y horas de ejecución.

Por tanto, se debe crear una base de datos y un usuario en los servidores *backend* de los distintos entornos (desarrollo, preproducción y producción). Para ello, se pueden utilizar los siguientes comandos:

$ su - postgres

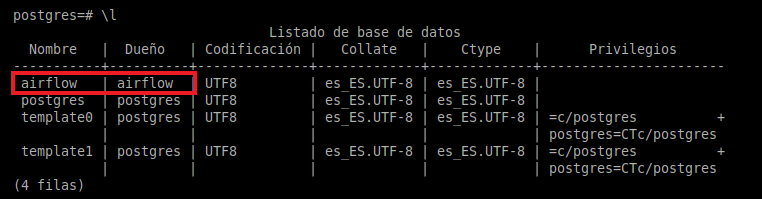
$ createuser --interactive --pwprompt

$ createdb -O airflow airflow

De esta forma se crea un usuario airflow y una base de datos llamada airflow, cuyo dueño es el usuario airflow recién creado. Para evitar incompatibilidades, se recomienda que la contraseña establecida sea la misma para todos los servidores *backend* y que el usuario airflow solo tenga acceso a su propia base de datos.

**NOTA**: PostgreSQL ya se encontraba operativo inicialmente, por lo que no se ha requerido instalarlo. La versión disponible es la 9.6. El proceso de creación de la BBDD y el usuario debe realizarlo el administrador del sistema o un usuario con privilegios en el sistema PostgreSQL.

Una vez completado el proceso, al listar las bases de datos existentes en PostgreSQL se debería obtener una salida semejante a la mostrada en la siguiente imagen:



El detalle de las tablas requeridas para almacenar los Boletines Oficiales se encuentra en el capítulo 3 del Manual Técnico.

## Entorno virtual servidores *frontend*

Tal y como se ha explicado previamente, en los servidores *frontend* se realiza la descarga de los boletines, ya que dichos servidores tienen conectividad con Internet. Además, ya se dispone de Python3 instalado, concretamente la versión 3.6.5.

Para los entornos de preproducción y producción, se debe crear un entorno virtual con las dependencias necesarias y requeridas por la aplicación en la ruta /data/apps/ingesta-BO. Para ello, ejecutar los siguientes comandos:

$ mkdir /data/apps/ingesta-BO

$ cd /data/apps/ingesta-BO

$ python3.6 -m venv env

$ source env/bin/activate

$ pip install requests

$ pip install PyPDF2

$ deactivate

Tras la ejecución, existirá un entorno virtual denominado env con los paquetes necesarios ya instalados. Dicho entorno será activado y utilizado posteriormente por la aplicación.

## Entorno virtual servidores *backend*

En los servidores *backend* también es necesario desplegar un entorno virtual, pero el proceso es diferente al seguido en la sección 1.5, ya que estos servidores no pueden descargarse e instalar las dependencias. Los entornos virtuales en backend se componen de:

* Paquetes requeridos por la aplicación y las componentes software desarrolladas.
* Paquetes para instalar Apache Airflow.

En este punto, ya se dispone de Python 3.6.9 instalado en los servidores *backend* (ver sección 1.3). Gracias a ello, se pueden crear entornos virtuales en dichos servidores, pero no instalarles las dependencias requeridas. Esto se soluciona obteniendo las dependencias en sistemas *online*, para después habilitarlas en los servidores de *backend*.

### Paquetes del entorno

Los paquetes que se han instalado en el entorno virtual de los servidores *backend* son los siguientes:

* Apache Airflow, versión 1.10.12, junto con las extensiones de ssh y postgresql.
* Las dependencias de Apache Airflow 1.10.12 ejecutado sobre la versión 3.6 de Python. El fichero requerido se puede obtener del siguiente enlace:

[*https://raw.githubusercontent.com/apache/airflow/constraints-1.10.12/constraints-3.6.txt*](https://raw.githubusercontent.com/apache/airflow/constraints-1.10.12/constraints-3.6.txt)

* Los siguientes paquetes disponibles en PyPI (repositorio oficial de paquetes de Python):
  + pdfplumber
  + textwrap3
  + spacy
  + modelo es\_core\_news\_md de spacy
  + workalendar
  + spa2num
  + pypdf2
  + beautifulsoup
  + psycopg2

### Creación del entorno *online*

Para realizar el despliegue en desarrollo se creó un entorno virtual en un sistema CentOS en el que se encontraba Python3 instalado tal y como se describe en la sección 1.3. Nótese que el sistema CentOS es ajeno a los servidores, y que es utilizado como medio de obtención de las dependencias necesarias. Los comandos que se ejecutaron en el sistema CentOS fueron:

$ scl enable rh-python36 bash

$ python -m venv ingesta-BO-env

$ source ./ingesta-BO-env/bin/activate

$ (ingesta-BO-env) $ wget -O constraint.txt \

<https://raw.githubusercontent.com/apache/airflow/constraints->1.10.12/constraints-3.6.txt

$ (ingesta-BO-env) pip install apache-airflow[‘ssh’,’postgres’]==1.10.12

--constraint constraint.txt

Una vez instalados los paquetes de Apache Airflow, se instalan los paquetes enumerados en el apartado 1.6.1. Esto puede realizarse preparando un fichero requirements.txt enumerándolos, y después ejecutar:

$ (ingesta-BO-env) pip install -r requirements.txt

Y, finalmente, añadir el modelo de es\_core\_news\_md de spacy:

$ (ingesta-BO-env) python -m spacy download es\_core\_news\_md

### Configuración de Apache Airflow

Apache Airflow se configura a través del archivo airflow.cfg, almacenado en la ruta $AIRFLOW\_HOME. Este fichero se ha creado en el sistema CentOS, para poder exportarlo posteriormente al servidor de desarrollo directamente. Para crear una plantilla del fichero de configuración, basta con ejecutar el comando airflow initdb tras realizar los pasos descritos en 1.6.2, y después modificar el archivo situado en la ruta /home/<user>/airflow/airflow.cfg:

# modificar el executor

executor = LocalExecutor

# modificar el string de conexión a la BBDD ‘airflow’ situada en PostgreSQL

sql\_alchemy\_conn = postgresql+psycopg2://airflow:<passwd>@localhost/airflow

# no cargar los DAG de ejemplo

load\_Examples = False

# configurar los límites y los recursos de ejecución

parallelism = 6

dag\_concurrency = 4

max\_active\_runs\_per\_dag = 4

worker\_refresh\_interval = 300

workers = 2

scheduler\_heartbeat\_sec = 60

min\_file\_process\_interval = 60

Una vez creado el entorno y el fichero de configuración (1.6.2, 1.6.3), se procedió a incluirlos en el servidor backend de desarrollo.

### Despliegue en *backend* de desarrollo

El despliegue sobre el servidor backend de desarrollo fue llevado a cabo por los desarrolladores. Para ello, se realizó lo siguiente sobre el servidor backend, con el usuario hiberus:

$ cd ~

$ mkdir airflow

$ scl enable rh-python36 bash

$ python -m venv ingesta-BO-env

Después de crear el entorno ingesta-BO-env y la carpeta airflow, se copiaron los siguientes directorios:

* Se copió sin sobreescribir la carpeta ingesta-BO-env/bin proveniente del entorno virtual definido en el apartado 1.6.2 sobre la ruta /export/home/hiberus/ingesta-BO-env/.
* Se sustituyó la carpeta /export/home/hiberus/ingesta-BO-env/lib por la carpeta ingesta-BO-env/lib proveniente del entorno virtual definido en el apartado 1.6.2.
* Se copió el archivo airflow.cfg definido en 1.6.3 en la carpeta /export/home/hiberus/airflow/.

Para iniciar el servicio de Apache Airflow, se ejecutaron los siguientes comandos:

$ source ingesta-BO-env/bin/activate

$ (ingesta-BO-env) airflow initdb

$ (ingesta-BO-env) airflow scheduler -D

Una vez activado el servicio de orquestación (Airflow), se creó una clave rsa para que el servidor de *backend* pudiese autenticarse en el servidor de *frontend*, para así obtener los Boletines Oficiales tras el proceso de descarga. La autenticación fue habilitada para el usuario hiberus. En los entornos de preproducción y producción el proceso realizado es similar, pero el usuario autenticado varía en cada uno de ellos.

$ (ingesta-BO-env) ssh-keygen -t rsa

$ (ingesta-BO-env) ssh-copy-id -i ~/.ssh/id\_rsa.pub hiberus@<servidor\_front>

Después de habilitar la autenticación por clave, se registra la conexión en el la base de datos de Airflow:

$ (ingesta-BO-env) airflow connections --add --conn\_id ‘ssh\_front’ --conn\_uri ‘ssh://hiberus@<servidor\_front>:22/?key\_file=/export/home/hiberus/.ssh/id\_rsa’

Finalmente, se activa el DAG de ingesta de boletines oficiales.

$ (ingesta-BO-env) airflow unpause ingesta-BO

### Traspaso a entorno *backend* de preproducción

Los pasos necesarios para traspasar el entorno de ejecución al servidor backend de preproducción son bastante similares a los descritos en el apartado anterior. La diferencia fundamental es el directorio sobre el que se despliega la instalación. Tanto en preproducción como en producción los directorios base son:

* /data/apps/airflow como directorio principal de airflow.
* /data/apps/ingesta-BO-env como entorno virtual.

Los comandos requeridos son (en orden):

$ cd /data/apps

$ mkdir airflow

$ scl enable rh-python36 bash

$ python -m venv ingesta-BO-env

* Copiar sin sobreescribir la carpeta /export/home/hiberus/ingesta-BO-env/bin proveniente del entorno virtual del servidor *backend* de desarrollo sobre la ruta /data/apps/ingesta-BO-env/bin.
* Sustituir la carpeta /data/apps/ingesta-BO-env/lib por la carpeta /export/home/hiberus/ingesta-BO-env/bin proveniente del servidor *backend* de desarrollo.
* Copiar el archivo airflow.cfg proveniente del servidor *backend* de desarrollo (/export/home/hiberus/airflow/airflow.cfg).

A continuación, ejecutar:

$ cd /data/apps/ingesta-BO-env

$ echo “export AIRFLOW\_HOME=/data/apps” >> ./bin/activate

$ grep -ilr “/export/home/hiberus” . | xargs -d ‘\n’ sed -i ‘s/\/export\/home\/hiberus/\/data\/apps\//g’

$ cd /data/apps/airflow

$ sed -i ‘s/\/export\/home\/hiberus/\/data\/apps\//g’ ./airflow.cfg

Finalmente, añadir la autenticación por clave y lanzar los servicios:

$ cd /data/apps/ingesta-BO-env

$ source ingesta-BO-env/bin/activate

$ (ingesta-BO-env) airflow initdb

$ (ingesta-BO-env) airflow scheduler -D

$ (ingesta-BO-env) ssh-keygen -t rsa

$ (ingesta-BO-env) ssh-copy-id -i ~/.ssh/id\_rsa.pub deloitte@<servidor\_front>

$ (ingesta-BO-env) airflow connections --add --conn\_id ‘ssh\_front’ --conn\_uri ‘ssh://deloitte@<servidor\_front>:22/?key\_file=/<home\_deloitte>/.ssh/id\_rsa’

$ (ingesta-BO-env) airflow unpause ingesta-BO

**NOTA**: el usuario utilizado para autenticarse sobre el servidor frontend de preproducción es deloitte.

### Traspaso a entorno *backend* de producción

Los servidores backend de preproducción y producción son idénticos, por lo que el proceso de instalación se simplifica. En el servidor backend de producción se debe ejecutar lo siguiente:

$ cd /data/apps

$ mkdir airflow

$ scl enable rh-python36 bash

$ python -m venv ingesta-BO-env

* Copiar sin sobreescribir la carpeta /data/apps/ingesta-BO-env/bin proveniente del entorno virtual del servidor *backend* de preproducción sobre la ruta /data/apps/ingesta-BO-env.
* Sustituir la carpeta /data/apps/ingesta-BO-env/lib por la carpeta análoga proveniente del servidor *backend* de preproducción.
* Copiar el archivo airflow.cfg proveniente del servidor *backend* de preproducción (/data/apps/airflow/airflow.cfg).

Finalmente, añadir la autenticación por clave y lanzar los servicios:

$ cd /data/apps/ingesta-BO-env

$ source ingesta-BO-env/bin/activate

$ (ingesta-BO-env) airflow initdb

$ (ingesta-BO-env) airflow scheduler -D

$ (ingesta-BO-env) ssh-keygen -t rsa

$ (ingesta-BO-env) ssh-copy-id -i ~/.ssh/id\_rsa.pub opendata@<servidor\_front>

$ (ingesta-BO-env) airflow connections --add --conn\_id ‘ssh\_front’ --conn\_uri ‘ssh://opendata@<servidor\_front>:22/?key\_file=/<home\_opendata>/.ssh/id\_rsa’

$ (ingesta-BO-env) airflow unpause ingesta-BO

**NOTA**: el usuario utilizado para autenticarse sobre el servidor frontend de preproducción es opendata.

# Explotación y mantenimiento

En este apartado se van a detallar algunos aspectos de la explotación y el mantenimiento de los desarrollos explicados en los otros manuales.

## Scripts de *bash*

En esta sección se detalla el funcionamiento de los scripts de *bash* (de formato .sh), que lanzan los scripts de Python con los desarrollos realizados. Estos ficheros se ubican en ‘/data/apps/airflow/dags’ en el *backend*, mientras que en el *frontend* el fichero a lanzar (que lanza la ingesta) es ‘/data/apps/ingesta-BO/run.sh’.

* **run.sh:** Es el único fichero aparecido en el *frontend*, y tiene día (formato ‘aaaammdd’) y directorio base existente como parámetros. Realiza la ingesta de los artículos de todos los boletines indicados.
* **copia\_ingesta.sh:** Realiza una copia de los artículos descargados en el *frontend*, al *backend*. El día como parámetro.
* **conversion.sh:** Ejecuta la conversión de los artículos descargados a ficheros de texto plano. El día como parámetro.
* **extraccion.sh:** Lanza la extracción de la información de los artículos tratados. El día como parámetro.
* **almacenamiento.sh:** Inserta la información obtenida de los diferentes artículos. El día como parámetro.
* **cierres.sh:** Cierra las ofertas indicadas en los artículos de cierre. El día como parámetro.
* **pg\_backup.sh:** Realiza la copia de seguridad en la ruta ‘/data/apps/ingesta-BO/backup.sql’.

## Gestión de logs

Este apartado detalla los diferentes logs que se han ido creando y comprobando para asegurarse del correcto funcionamiento durante los desarrollos, tanto en desarrollo como preproducción y producción.

### Logs de Airflow

Se guarda una serie de logs para encontrar posibles problemas tanto en la ejecución del código desarrollado como en el propio Airflow. Hay una carpeta principal que se gestiona con Airflow (/data/apps/airflow/logs) donde aparecen tres subdirectorios:

* **ingesta-BO:** Dentro de este directorio se crea automáticamente un subdirectorio por cada tarea del DAG, donde se almacenarán los logs de dicha tarea. Dentro de estos subdirectorios aparece otro subdirectorio con la fecha y hora de la ejecución. Dentro de estos, se crea un fichero llamado ‘1.log’ en el que aparecen los errores acontecidos (si está vacío, no los ha habido).
* **scheduler:** Dentro de este directorio se crean automáticamente subdirectorios con la fecha de ejecución. Dentro de estos solo se registran aquellas situaciones en las que se produce algún fallo en el proceso airflow-scheduler.

Estos logs tienen un ciclo de vida de diez días. Cada día (día X), al ejecutarse el DAG, se borran los logs del día X-10, en caso de existir.

### Logs de código

En el código se guardan una serie de logs para poder comprobar los artículos que no han pasado diferentes condiciones:

* Se guarda un log con los nombres de los ficheros de cierres que no cierran ofertas (en muchas ocasiones ocurre porque sus correspondientes aperturas no existen en la base de datos, por ser anteriores al inicio de la base de datos). Se llama artículos\_sin\_cierres.log.
* Se guarda otro log con los artículos que se habían identificado como ofertas de empleo inicialmente, pero que finalmente no han pasado el umbral de número de campos requerido para su inserción en la base de datos. Este se llama artículos\_no\_insertados.log. Además, se ha creado una tabla en la base de datos que guarda el nombre del artículo, el enlace al mismo, el número de campos que ha detectado y la fecha del mismo. Esta tabla se llama ARTICULOS\_NO\_INSERTADOS.
* Además, se guarda un log para posibles errores que pudieran aparecer en el código. Bastantes de estos errores son realmente *warnings*: por ejemplo, cuando no se detecta un XML Sumario para alguna de las ingestas, el día aparece en este log, siendo que muy probablemente en ese día simplemente no hay artículos (por ejemplo, los domingos). Se llama artículos\_erroneos.log.

## Invocaciones a scripts Python

En este apartado se van a enumerar las invocaciones a todos los scripts de Python. Cada uno de ellos cuenta con su propio main(), aunque también están preparados para ser usados por otros scripts dentro de sus mismos directorios. Estas invocaciones aparecen al principio de cada fichero, así como metadatos del mismo, descripción e incluso una invocación de ejemplo, pero se prefieren enumerar las invocaciones aquí para hacerlas más rápidamente accesibles. Los ficheros y sus invocaciones estándar son:

* **ingesta.py:** python ingesta.py dia\_AAAAMMDD directorio\_base
* **ingesta\_aragon.py:** python ingesta\_aragon.py dia\_AAAAMMDD directorio\_base
* **ingesta\_boe.py:** python ingesta\_boe.py dia\_AAAAMMDD directorio\_base
* **ingesta\_extra.py:** python ingesta\_extra.py dia\_AAAAMMDD directorio\_base tipo\_boletin
* **conversion\_a\_texto.py:** python conversion\_a\_texto.py directorio\_base ruta\_auxiliar bool\_legible
* **html\_a\_txt.py:** python html\_a\_txt.py ruta\_html ruta\_txt tipo\_boletin bool\_legible
* **pdf\_a\_txt.py:** python pdf\_a\_txt.py folder\_pdf folder\_txt
* **xml\_a\_txt.py:** python xml\_a\_txt.py ruta\_xml ruta\_txt tipo\_boletin bool\_legible
* **extraccion.py:** python extraccion.py dia\_aaaammdd directorio\_base ruta\_modelo\_NER ruta\_regex ruta\_auxiliar
* **extraccion\_reglas.py:** python extraccion\_reglas.py dia\_aaaammdd ruta\_info ruta\_texto ruta\_regex
* **extraccion\_ner.py:** python extraccion\_ner.py ruta\_texto ruta\_modelo
* **puestos\_tablas\_anexos.py:** python puestos\_tablas\_anexos.py C:\prueba\20201001\pdf\BOE\_20201001\_2\_rotado.pdf (si se pone otro parámetro, numérico, es para obtener solo los puestos de la tabla de la página siguiente al número indicado)
* **extraccion\_tablas.py:** python extraccion\_tablas.py ruta\_pdf ruta\_auxiliar
* **almacenamiento.py:** python almacenamiento.py dia\_aaaammdd directorio\_base ruta\_auxiliar PSQL\_DB PSQL\_HOST PSQL\_PORT PSQL\_USER PSQL\_PASS
* **cierres\_convocatorias.py:** python cierres\_convocatorias directorio\_base dia\_aaaammdd ruta\_regex ruta\_auxiliar PSQL\_DB PSQL\_HOST PSQL\_PORT PSQL\_USER PSQL\_PASS

# Copias de seguridad

La política seguida para las copias de seguridad es realizar el *backup,* de tipo completo, de la base de datos (no de los artículos en diferentes formatos ni demás ficheros utilizados durante el flujo, por cuestiones de memoria especialmente). La operación de *backup* se realiza en el propio DAG de Airflow donde se realizan las demás operaciones, tras todas las demás. Esta operación reemplaza la copia de seguridad del día anterior.

La instrucción utilizada para realizar la copia de seguridad es:

pg\_dump --dbname=postgresql://$DB\_USR:$DB\_PASS@$DB\_HOST:$DB\_PORT/$DB\_NAME > ruta

, siendo $DB\_USR el usuario de la base de datos, $DB\_PASS la contraseña de la base de datos, $DB\_HOST el hostname de la base de datos, $DB\_PORT el puerto, $DB\_NAME el nombre de la base de datos y ruta la ruta donde se va a guardar el fichero .sql con la copia de seguridad.

Esta instrucción sigue la directiva [pg\_dump](https://www.postgresql.org/docs/9.6/app-pgdump.html) de PostgreSQL.

## Recuperación de copias de seguridad

En caso de querer recuperar la información del *backup*, los pasos a realizar son:

1. Eliminar la base de datos (DROP DATABASE nombre)
2. Crear la base de datos (CREATE DATABASE nombre).
3. Hacer el restore del backup.

psql $DB\_NAME -U $DB\_USR -h $DB\_HOST < ruta

, siendo $DB\_NAME el nombre de la base de datos, $DB\_USR el usuario de la base de datos, $DB\_HOST el hostname de la base de datos y ruta la ruta del fichero .sql con la copia de seguridad.

Con esto se consigue que la base de datos quede idéntica a la de la copia de seguridad, tanto en estructura como en lo referido a datos.