télécom saint-étienne
école d'ingénieurs / nouvelles technologies

Technical Documentation

Scripting System:

Archive Manager

Student :
Youssra Rahmouni

Supervised by :
Prof. Pierre Yves Fraisse

Academic Year : 2021-2022

Contents

List of Figures

Abstract

The project in hand is an archive manager system that enables the user to download an SQL file from a given link and place a tgz archive format of the file in a remote server. Upon launching the setup file all dependencies needed for the project are installed and a crontab file is created to automate the execution of the project's script. The system also allows the user to personalise its configuration through a configuration file, thus allowing the user to specify the link of the SQL file, duration of historization, and to enable or disable receiving notifications on a Mattermost server. This project is written in python, and ideally to be executed and used on a linux based OS.

Part 1: Technical choices

In order to meet the specifications of the project multiple technical choices were made. This part presents justifications for technical and functional choices.

Transfer method : Webdav

Security	WebDAV is different from other transfer protocols (SMB....) as it uses HTTP for transporting files and is designed from the ground up to be secure on the internet. WebDAV can also use HTTPS, secure and encrypted HTTP, allowing no one to eavesdrop on your file transfer.
Speed	WebDAV file system is faster than Samba / CIFS file system both in write tests and read tests. WebDAV doesn't need to establish a fresh connection to transfer each file and so the session establishment overhead is reduced.
Easy to implement	davfs2 allows mounting WebDAV shares and can easily be configured with a remote Server address (i.g a cloud hosting server).

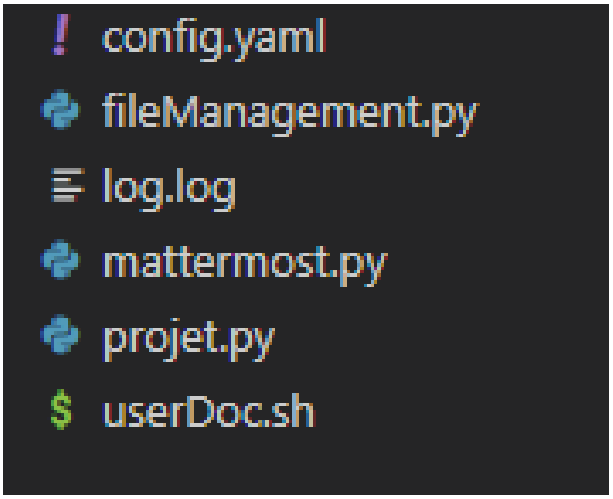
Configuration file: yaml format

Easy to read and edit	Yaml is text-based and structured in such a way that is easy to understand. Even non-developers should be able to read. Support nested structures and natively encodes some data types.
Allow comments	Writing comments is essential and allows us to explain to the user how to change each configuration variable, thus making the config file more expressive. Other formats like json do not support comments.
Easy to deploy	Accepted by all the operating systems and environments.

Part 2: Project structure

The project contains six files in total:

- **Config.yml**: configuration file, has a list of variables used by the script.
- **userDoc.sh**: a bash set up file that needs to be executed before launching the script in order to set up the environment for the user.
- **Project.py**: reads the configuration file and calls all the functions.
- **FileManagement.py**: contains functions that manages the sql file download, extraction and upload, and also historization of the file on the server.
- **Mattermost.py**: contains functions that send notification to the mattermost channel of the webhook specified in the configuration file.
- **Log.log**: the log of the project's execution.



Config.yml

In this file a list of configuration variables is defined with default values that the user is allowed to change and custom.

```
---
#Link of zip file to be archived
LINK: https://opensource.telecomste.fr/inforx/test_export.sql.zip
#After this duration the archive will be deleted, insert an int
DURATION: 1
#You can specify duration date type: minutes, hours, days, weeks
DURATIONTYPE: days
#URL of matermost webhook
WEBHOOK: https://chat.telecomste.fr/hooks/13ta3nw3e787tnxxqkyxfh5xnr
#Specify whether or not to notify mattermost. Accepted values are true, false
NOTIFY: true
```

Figure 1: Config.yml

The user can specify the link to the SQL file zip. The user can also specify the duration of which the archive will remain on the remote server as well as the type in minutes, hours,

days or weeks. Finally the user can specify whether or not he wants to receive notifications on a Mattermost channel.

userDoc.sh

This file sets up the configuration for the user by installing python 3 and davfs2.

```
echo "Installing python"
sudo apt-get install -i python3
echo "Installing davfs2"
sudo apt-get install -i davfs2
```

Figure 2: userDoc.sh

Davfs2 is used to connect to the remote server, for testing I used a cloud service Box, any service can be used. The user only needs to specify a valid username and password and a mount share will be created on the user's system.

```
sudo mkdir -p /mnt/dav
echo "Enter cloud service URL"
read URL
echo "Enter username for cloud account"
read username
echo "Enter password"
read password
echo "Beginning configuration"
fstabLine="$URL  /mnt/dav davfs _netdev,noauto,user,uid=$uid,gid=$gid 0 0"
echo "$fstabLine" >> /etc/fstab
davSecrets="$URL $username $password"
echo "$davSecrets" >> /etc/davfs2/secrets
sudo usermod -a -G davfs2 $user
echo "Starting mount"
sudo mount /mnt/dav
echo "Mount successful"
```

Figure 3: userDoc.sh

In this file I also set the rights on the files of the projects as some may only have the reading rights and others need the execution rights and cannot be modified by the user.

```
#Setting rights on all the project files
#The scripts files should be executable
sudo find . -type f -iname "*.py" -exec chmod 511 {} \;
# Config file has all rights
sudo find . -type f -iname "config.yaml" -exec chmod 777 {} \;
# Log file has read only rights, won't need to be executed or written in
sudo find . -type f -iname "log.log" -exec chmod 444 {} \;
```

Figure 4: userDoc.sh

For the project automation I used CRON.

```
# Automate script execution
# Configure crontab to call projet.py and launch the archive of the file
# Each day at 1 am, every day, every month, every day of the week
# execute projet.py
{ crontab -l -u $user; echo "0 1 * * * cd $directory; python3 projet.py"; } | crontab -u $user -

# Check crontab was created.
if sudo test -f "/var/spool/cron/crontabs/$user"
then
    echo "Crontab file created successfully"
    # Launch cron service
    sudo service cron start
else
    echo "Error: Cron file was not created."
fi
```

Figure 5: userDoc.sh

Projet.py

This file reads from the configuration file and sets the basic configuration variables in the script that will be passed to the functions afterwards.

```
#defining syntax for log file
logging.basicConfig(filename='log.log', filemode='w', level=logging.INFO, format='%(levelname)s:%(asctime)s:%(message)s')
```

Figure 6: Projet.py

This line sets the configuration and format of the logging file, using the python logging library. The lines of the logging file are constructed as such:

Levelname: type of the information; an error, a warning or a simple information.

Asctime: time when the log was raised to make reading through the file easier for the user.

Message: specific messages describing successful or unsuccessful actions.

Yaml.safe_load() function is used to read the yaml file and retrieve configuration variables.

```
1  # Readinfg the yaml configuration file
2  def read_yaml(file_path):
3      with open(file_path,"r") as f:
4          return yaml.safe_load(f)
5  CONFIG=read_yaml("./config.yaml")
6  #Getting all variables from the yaml file
7  #Link stores the zip file link
8  LINK=CONFIG['LINK']
9  if LINK is None:
10     logging.error(' No link was specified in the configuration file')
11     exit()
12 #Duration of conservation of the file on the server
13 DURATION=CONFIG['DURATION']
```

```
14 if DURATION is None:
15     logging.info(' No duration was specified, default duration is used : 1')
16     DURATION=1
17 #Duration type of conservation of the file on the server
18 DURATIONTYPE=CONFIG['DURATIONTYPE']
19 if DURATIONTYPE is None:
20     logging.info(' No duration type was specified, default duration is used :
           days')
21     DURATIONTYPE="days"
22 NOTIFY=CONFIG['NOTIFY']
```

After the file load an INFO of the success is added to the log file.
Afterwards four main functions of the script are called:

```
fileLink=getFileLink(LINK,WEBHOOK,NOTIFY)
SQL_FILE=extractFile(fileLink,WEBHOOK,NOTIFY)
archiveFile(SQL_FILE,WEBHOOK,NOTIFY)
manageFile(DURATION,DURATIONTYPE,WEBHOOK,NOTIFY)
```

Figure 7: Projet.py

All the four functions are called from the FileManagement.py script.

FileManagement.py

As the name suggests this script regroups all functions that handle the file management.

`getFileLinkThis` function sends an http request to retrieve the zip file from the link provided by the user. Success actions are logged into the log file. If an exception occurs it is also added to the log file with the level ERROR.

```
1  def getFileLink(LINK,webHook,NOTIFY):
2      try:
3          logging.info(' Trying to connect...')
4          fileLink=requests.get(LINK)
5          logging.info(' Connection successful')
6          if NOTIFY:
7              payload=buildPayload('File retrieval','Done')
8              notifyMattermost(payload,webHook)
9          return fileLink
10     except requests.exceptions.RequestException as e:
11         logging.error(e.strerror)
12         if NOTIFY:
13             payload=buildPayload('File retrieval','ERROR, review log')
14             notifyMattermost(payload,webHook)
```

If the NOTIFY flag is set to True the user will receive a notification of each step of the process.

`extractFile`: This function extracts the SQL file from the zip and names it under a specific format AAAADDMM. It takes as arguments the file link the webhook link and also the notify flag to check whether or not to send this function's specific notification to Mattermost.

```
def extractFile(fileLink,webHook,NOTIFY):
    logging.info(" FUNCTION FOR FILE EXTRACTION RUNNING")
    z=zipfile.ZipFile(io.BytesIO(fileLink.content))
    now=datetime.datetime.now()
    name=now.strftime("%Y%d%m")
    pathFolder="./"+name+"/"
    if os.path.exists(pathFolder):
        print(" AN ARCHIVE HAS ALREADY BEEN CREATED")
        logging.info(' AN ARCHIVED FOR THIS FILE ALREADY EXISTS, NO FILE
            CREATED')
        SQL_FILE=None
    else:
        z.extractall(path=pathFolder)
        print(z.infolist()[0].filename)
        SQL_FILE=z.infolist()[0].filename
        logging.info(' EXTRACTION SUCCESSFUL'+'filename'+SQL_FILE)
        if NOTIFY:
            payload=buildPayload('File extraction','Done')
            notifyMattermost(payload,webHook)
    return SQL_FILE
```

`archiveFile`: This function takes the returned sql file from the previous function and creates a tgz archive. Then places the archive in the /mnt/dav directory connected with the remote server.

```
def archiveFile(SQL_FILE,webHook,NOTIFY):
    if SQL_FILE is None:
        print("Empty")
    else:
        logging.info(' ENTERED ARCHIVING FILE')
        print("ARCHIVING FILE ")
        now=datetime.datetime.now()
        name=now.strftime("%Y%d%m")
        print(name)
        archiveName="/mnt/dav/"+name+".tgz"
        file_obj=tarfile.open(archiveName,"w")
        file_obj.add("./"+name+"/"+SQL_FILE)
        file_obj.close()
        shutil.rmtree("./"+name, ignore_errors=True)
        logging.info( 'ARCHIVE SUCCESSFULL')
        if NOTIFY:
```

```
            payload=buildPayload('File archive','Done')
            notifyMattermost(payload,webHook)
```

If an archive has already been done and the duration of historisation is not exceeded yet the zip file is updated on the server and not deleted since WEBDAV supports version control.
Eventually the user has the latest version of the zip file of each day. And depending on the duration he specifies on the configuration file these versions lifecycle will be managed on the remote server.

`manageFile`: This function manages the files historization on the remote server, depending on the duration specified by the user, files that exceed this duration are removed from the remote server.

```
def manageFile(Duration,durationType,webHook,NOTIFY):
    print("looking for files")
    files=os.listdir("/mnt/dav/")
    for f in files:
        if f.endswith('.tgz'):
            print(f)
            try:
                dateCreation=datetime.datetime.strptime(time.ctime(os.path
                    .getctime("/mnt/dav/"+f)),"%c")
                now=datetime.datetime.now()
                if dateCreation+datetime.timedelta(**{durationType:
                    Duration})<now:
                    print("should be deleted")
                    print("-----------------")
                    try:
                        os.remove("/mnt/dav/"+f)
                        logging.info( 'FILES EXCEEDING DURATION DELETED')
                        if NOTIFY:
                            payload=buildPayload('Files management','Done')
                            notifyMattermost(payload,webHook)
                    except OSError as e:
                        logging.error(e.strerror)
                        if NOTIFY:
                            payload=buildPayload('Files management','ERROR,
                                review log')
                            notifyMattermost(payload,webHook)
            except OSError:
                print("Path does not exist")
                if NOTIFY:
                    payload=buildPayload('File management','ERROR, review log')
                    notifyMattermost(payload,webHook)
```

mattermost.py

In this file I defined buildPayload that builds the message to be sent to the user over the Mattermost channel and then sends using a post request. The message specifies the task that was done by the script and its state in tabular format.

```
1  def buildPayload(task,state):
2      payload="|  Task    |   State    | \n"+"| ------------|---------------|
           \n"+"|   "+ task + "|    " + state +"|  \n"
3      formatedPayload={'text':payload}
4      return json.dumps(formatedPayload)
5  def notifyMattermost(message,webHook):
6      try:
7          headers={'Content-Type' : 'application/json'}
8          r=requests.post(webHook, data=message, headers=headers)
9          print(r.status_code)
10     except requests.exceptions.RequestException as e:
11         print(e.strerror)
```

Part 3: Project execution

As a first step the user needs to run the dependenscies.sh script as the root user. This will create a crontab file that will run the script automatically, but it can also be done manually using python3 projet.py
Beneath are screens for the project execution:

```
root@youssra:/mnt/dav# ls
lost+found  new
root@youssra:/mnt/dav# cd /projet/scriptSystem/
root@youssra:/projet/scriptSystem# ls
config.yaml  dependencies.sh  fileManagement.py  mattermost.py  projet.py  script.log
root@youssra:/projet/scriptSystem# python3 projet.py
200
test_export.sql
200
ARCHIVING FILE
20220901
200
looking for files
20220901.tgz
root@youssra:/projet/scriptSystem# cat script.log
INFO:2022-01-09 01:31:16,821: Retrieved information successfully from configuration file
INFO:2022-01-09 01:31:16,822: Trying to connect...
INFO:2022-01-09 01:31:17,169: Connection successful
INFO:2022-01-09 01:31:17,525: FUNCTION FOR FILE EXTRACTION RUNNING
INFO:2022-01-09 01:31:17,594: EXTRACTION SUCCESSFULfilenametest_export.sql
INFO:2022-01-09 01:31:17,829: ENTERED ARCHIVING FILE
INFO:2022-01-09 01:31:20,439:ARCHIVE SUCCESSFULL
root@youssra:/projet/scriptSystem# cd /mnt/dav
root@youssra:/mnt/dav# ls
20220901.tgz  lost+found  new
root@youssra:/mnt/dav#
```

Figure 8: Project execution

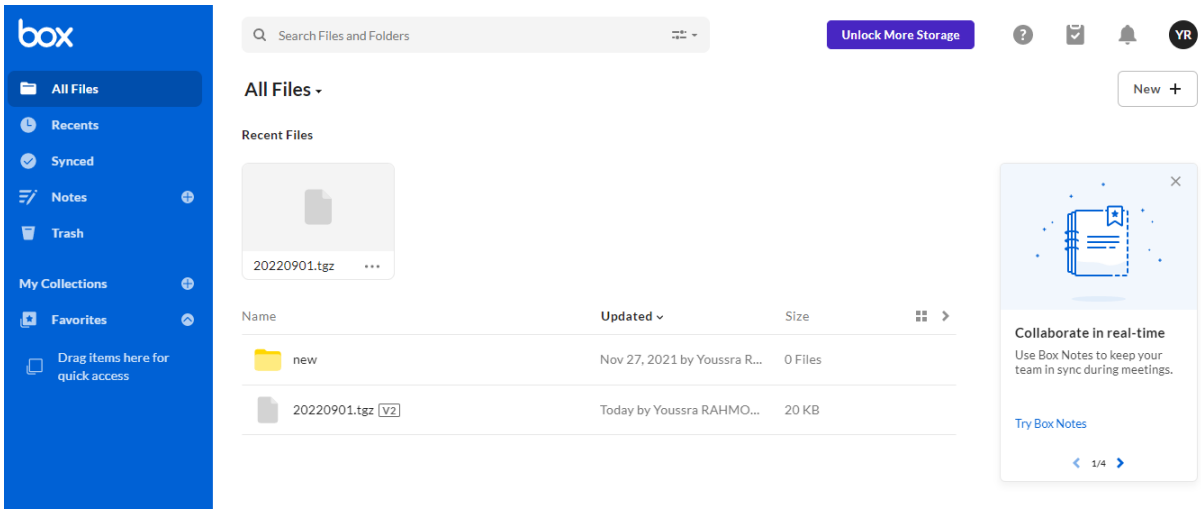


Figure 9: Remote server: BOX

```
log.log
1  INFO:2022-01-09 01:31:16,821: Retrieved information successfully from configuration file
2  INFO:2022-01-09 01:31:16,822: Trying to connect...
3  INFO:2022-01-09 01:31:17,169: Connection successful
4  INFO:2022-01-09 01:31:17,525: FUNCTION FOR FILE EXTRACTION RUNNING
5  INFO:2022-01-09 01:31:17,594: EXTRACTION SUCCESSFUL filename test_export.sql
6  INFO:2022-01-09 01:31:17,829: ENTERED ARCHIVING FILE
7  INFO:2022-01-09 01:31:20,439: ARCHIVE SUCCESSFULL
```

Figure 10: Log file