Scripting System Project

Archiving System

Student :

EL HAIMER Fatima Ezzahraa

OUADID Hamza

ZIANI Zakaria

Supervised by :
Prof. RAWAT A PRIYANK

Major :
Computer Science - Network

Academic year 2023/2024

Table des matières

Abstract **1**

1 First Approach : From WEB Server to FTP **2**
1.1 Technical Stack . . . 2
1.2 Project Structure . . . 3
1.2.1 userDoc.sh . . . 3
1.2.2 .env . . . 4
1.2.3 docker-compose.yaml . . . 5
1.2.4 FTP_server.py . . . 5
1.2.5 config.yaml . . . 6
1.2.6 backup.py . . . 7
1.2.7 BackupUtilities.py . . . 9
1.2.8 emailling.py . . . 11
1.2.9 log.log . . . 11
1.2.10 utilz.py . . . 12
1.3 Project Execution . . . 14

2 Second Approach : From WEB Server to WebDAV **15**
2.1 Technical Stack . . . 15
2.2 Project Structure . . . 16
2.2.1 userDoc.sh . . . 17
2.2.2 userDocConfig.sh . . . 17
2.2.3 readme.txt . . . 18
2.2.4 config.json . . . 18
2.2.5 webdav_configuration.txt . . . 18
2.2.6 backup.py . . . 19
2.2.7 backupUtilities.py . . . 20
2.3 backup_exe.log . . . 24
2.4 Project Execution . . . 24
2.4.1 No archived files in the WebDAV Server . . . 24
2.4.2 WebDAV Server contains at least one archived file . . . 25

Abstract

The primary objective of this project is to automate the process of archiving an SQL file, which can be downloaded by users from a web server. Subsequently, the file is compressed in tgz format and then securely stored on a separate remote server.

We adopted two distinct approaches to accomplish this task. One approach leverages Docker to configure the remote server, while the other relies on WebDAV technology. In both approaches, a carefully crafted shell script is utilized to specify and manage all essential dependencies. Automation is facilitated through crontab, ensuring regular execution of the backup script. Furthermore, there exists a configuration file that defines various variables essential for script execution. This level of customization empowers users to modify parameters, such as the expiration date and email notification preferences.

This comprehensive report offers an in-depth exploration of each approach, providing insights into the technical stack, project structure, and the execution workflow. It's important to note that all the scripts developed for this project are scripted using the Python programming language.

Chapitre 1

First Approach : From WEB Server to FTP

The goal of this approach is to create an automated system for backing up files from a remote web server and archiving them on an FTP server hosted within a Docker container. This approach is designed to offer an efficient archiving solution, particularly beneficial for users seeking a user-friendly, detachable archiving system.

1.1 Technical Stack

In carrying out our project, we relied on a set of technical choices that can be justified by the table below :

Technical choice	*justification*
Docker	- The execution of installed programs does not depend on the operating system. - Facilitates the handling of microservices. - Teamwork management. - Streamlines coordination of behaviors between containers.
Apache	- It provides a stable platform for applications - Apache's configuration files are typically human-readable and well-documented, making it relatively straightforward to set up and manage.
FTPS	- Facile à implémenter - Permet de lire ce qui est dedans.
Netfily	- We used netfily deployment services just to make our server open to the internet and to facilitate teamwork
Configuration file .YML	- Easy to handle (read, comment, modify...) - Compatible with Docker - Easy to add a validation layer
Email	- On a choisi un serveur SMTPS google afin de pouvoir communiquer avec toutes les boîtes possibles ce qui n'était pas réalisable avec notre serveur local.

FIGURE 1.1 – First Approach's Technical Choices

1.2 Project Structure

The project includes the following files and folders :

— **userDoc.sh** : A bash script managing dependencies installation and automation.
— **.env** : Contains environment variables for the FTP server.
— **docker-compose.yaml** : This Docker Compose file is used to set up an FTP server.
— **FTP_server.py** : Contains primary functions for managing files on the remote server.
— **config.yaml** : A configuration file enabling users to customize email settings, archive retention duration, and file links.
— **backup.py** : Executes the entire logic of the archiving management system.
— **BackupUtilities.py** : Houses various methods essential to the system, such as getFileLink, extractFile, archiveFile, and manageFile functions.
— **emailling.py** : Defines the mailing process using SMTP.
— **log.log** : A log file for tracking scripts execution.
— **utilz.py** : This script manages archiving operations.
— **temp** : A temporary folder for saving extracted SQL zip files.
— **storage** : A folder for storing backup tgz files from the remote server.
— **last_backup** : A folder for storing the most recent backup file.
— **__pycache__** : A directory generated by the Python interpreter when importing modules.

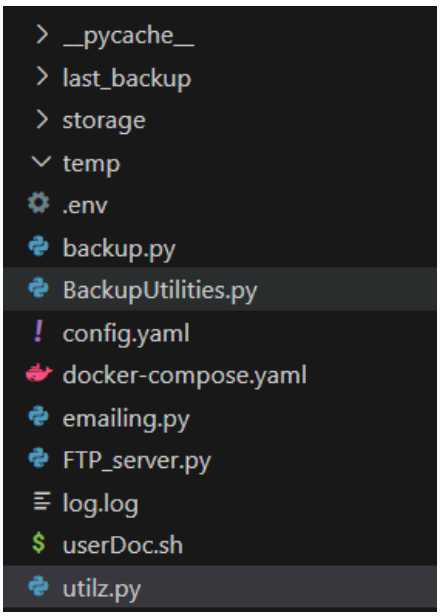


FIGURE 1.2 – The project stracture.

1.2.1 userDoc.sh

This Bash script sets up the environment for launching a Python script and using an FTP server for automated file archiving.

— Installs Python 3.
— Installs Docker and its dependencies, including adding Docker's GPG key and repository.
— Checks the Docker version.
— Runs a Docker container for the FTP server.

```
echo "Installing python"
sudo apt-get install -i python3
echo "Installing docker"
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Add the repository to Apt sources:
echo \
  "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
#checking if docker version
docker --version
#runing the FTP server container
docker compose up
```

FIGURE 1.3 – Dependencies insallation.

The script then proceeds to automate script execution using crontab with the following schedule :
- The script runs every day at 1 am. creates a crontab entry for scheduled script execution. Finally, it checks if the crontab file was created successfully and starts the cron service.

```
# Add the crontab entry with the provided values
(crontab -l ; echo "0 1 * * * /usr/bin/python3 $backup_dir/backup.py") | crontab -u $username -

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

FIGURE 1.4 – Cron configuration.

It prompts the user for the backup directory path and username.

```
# Prompt user for backup directory and username
read -p "Enter the backup directory path: " backup_dir
read -p "Enter the username: " username
```

FIGURE 1.5 – Shell prompt for username and backup directory.

In essence, this script prepares the environment for automated file archiving using Python, Docker, and crontab scheduling while also handling Docker installation and FTP server setup.

1.2.2 .env

In this file we defined the user and the passowrd for the FTP remote server.

```
.env
1   FTP_USER=admin
2   FTP_PASSWORD=fghj1234
```

FIGURE 1.6 – Environment variables.

1.2.3 docker-compose.yaml

This file will create and run a Docker container based on the "delfer/alpine-ftp-server" image, which sets up an FTP server. Users can connect to this server on port 21, and data transfers can happen on ports 21000 to 21010. The server's data is stored in the "./storage" directory on the host machine.

```yaml
docker-compose.yaml
version: "3.3"
services:
  ftp:
    image: delfer/alpine-ftp-server
    restart: always
    ports:
      - 21:21
      - 21000-21010:21000-21010
    volumes:
      - ./storage:/home/${FTP_USER}
    environment:
      - USERS=${FTP_USER}|${FTP_PASSWORD}|/home/${FTP_USER}
```

FIGURE 1.7 – Docker-compose configuration.

1.2.4 FTP_server.py

In this script we provide functions to interact with an FTP server.

add_ftp(file_name) This function uploads a file specified by file_name to an FTP server running locally (on 'localhost') using the FTP protocol. It establishes a connection, logs in with a username and password, and stores the file on the server.

```python
def add_ftp(file_name):
    username = 'admin'
    password = 'fghj1234'
    server_address = 'localhost'
    port = 21
    session = ftplib.FTP()
    session.connect(server_address, port)
    session.login(user=username, passwd=password)
    with open(file_name, 'rb') as file:
        session.storbinary('STOR {}'.format(file.name), file)
    session.quit()
```

FIGURE 1.8 – add_ftp method.

retrieve_ftp(file_name) This function retrieves a file specified by file_name from the FTP server and saves it locally. It connects to the FTP server, logs in, and uses the RETR command to retrieve the file and save it.

```python
def retrieve_ftp( file_name):
    username = 'admin'
    password = 'afghj1234'
    server_address = 'localhost'
    port = 21
    session = ftplib.FTP()
    session.connect(server_address, port)
    session.login(user=username, passwd=password)
    retrieved_file_name = '_' + file_name
    with open(retrieved_file_name, 'wb') as retrieved_file:
        session.retrbinary('RETR ' + file_name, retrieved_file.write)
    session.quit()
    return retrieved_file
```

FIGURE 1.9 – retrieve_ftp method.

delete_ftp(file_name) This function deletes a file specified by file_name from the FTP server. It connects to the FTP server, logs in, and uses the DELE command to delete the specified file.

```python
def delete_ftp(file_name):
    username = 'admin'
    password = 'fghj1234'
    server_address = 'localhost'
    port = 21
    session = ftplib.FTP()
    session.connect(server_address, port)
    session.login(user=username, passwd=password)
    session.delete(file_name)
    session.quit()
```

FIGURE 1.10 – delete_ftp method.

1.2.5 config.yaml

In this yaml file, we configure variables for the file link to be archived, duration settings, SMTP server information for email notifications, and recipients. The script send mail notifications if the "NOTIFY" flag is set to "true".

config.yaml

```yaml
---
#Link of zip file to be archived
LINK: https://leafy-kringle-a61e00.netlify.app/sql-sample.zip

#After this duration the archive will be deleted, insert an int
DURATION: 1

# Information about smtp server
SMTP_PORT: 465 # smtp port
SMTP_SERVER: "smtp.gmail.com" # server smtp
SMTP_PASSWORD: "emsd bqio pkau yefn" # user password
SMTP_MAIL_SENDER: "ensiasteam@gmail.com" # mail sender
# List of all the emails that will receive the mail.
SMTP_MAIL_RECEIVERS:
  - "hamzaouadid@gmail.com"
  - "ezermathst81321@gmail.com"



#You can specify duration date type: minutes, hours, days, weeks
DURATIONTYPE: days

#Specify whether or not to notify mattermost. Accepted values are true, false
NOTIFY: 1
```

FIGURE 1.11 – Yaml configuration file.

1.2.6 backup.py

In this script we start by reading the config.yaml file to obtain various settings and parameters for the archiving process.

```
#Getting all variables from the yaml file
CONFIG=read_yaml("./config.yaml")
#Link stores the zip file link
LINK=CONFIG['LINK']
if LINK is None:
    logging.error(' No link was specified in the configuration file')
    exit()
#Duration of conservation of the file on the server
DURATION=CONFIG['DURATION']
if DURATION is None:
    logging.info(' No duration was specified, default duration is used : 1')
    DURATION=1
#Duration type of conservation of the file on the server
DURATIONTYPE=CONFIG['DURATIONTYPE']
if DURATIONTYPE is None:
    logging.info(' No duration type was specified, default duration is used : days')
    DURATIONTYPE="days"
NOTIFY=CONFIG['NOTIFY']
#check if there are specified receivers for the notification
SMTP_MAIL_RECEIVERS=CONFIG['SMTP_MAIL_RECEIVERS']
if SMTP_MAIL_RECEIVERS is None:
    logging.info(' No receivers were specified, notification flag is set to false')
    NOTIFY=False
logging.info(' Retrieved information successfully from configuration file')
MESSAGE = """\
Hello there,
You are receiving this email, to keep up to date with what is happening with your archiving management system.
You find attached the log file to track the execution of your scripts.\n
{}
ENSIAS TEAM.
"""
SUBJECT = 'Archive Management System Notification'
SMTP_MAIL_SENDER = CONFIG['SMTP_MAIL_SENDER']
SMTP_SERVER = CONFIG['SMTP_SERVER']
SMTP_PORT = CONFIG['SMTP_PORT']
SMTP_PASSWORD = CONFIG['SMTP_PASSWORD']

logging.info(' Retrieved information successfully from configuration file')
```

FIGURE 1.12 – Backup script parameters

We get URL link to retrieve the sql zip file.
Then we checks the configuration for the duration of file conservation on the server. If no duration is specified, we use a default value of 1.
After that we check the configuration for the type of duration and sets a default to "days" if not specified.
Then we look if there are specified receivers for email notifications. If none are provided, it sets the notification flag to false.
The script logs successful retrieval of information from the configuration file. And it constructs an email message with a log file attached to track the execution of the archiving process.
The script calls various functions from the imported modules to perform the following tasks :

- **manageFile** Manages files on the server, specifically deleting files that have exceeded the specified duration.
- **getFileLink** Retrieves a file from the specified URL link in the configuration.
- **extractFile** Extracts a zip file obtained from the URL.
- **archiveFile** Archives the extracted SQL file.
- **sendEmail** Sends an email notification with the log file attached.

```
manageFile(DURATION,DURATIONTYPE,NOTIFY)
fileLink=getFileLink(LINK,NOTIFY)

SQL_FILE=extractFile(fileLink,NOTIFY)

archiveFile(SQL_FILE,NOTIFY)
sendEmail(MESSAGE, SUBJECT, SMTP_MAIL_RECEIVERS, SMTP_MAIL_SENDER, SMTP_SERVER, SMTP_PORT, SMTP_PASSWORD)
manageFile(DURATION,DURATIONTYPE,NOTIFY)
```

FIGURE 1.13 – Called methods in backup script.

1.2.7 BackupUtilities.py

In this script we define all the main methods for our archiving management system.

The **getFileLink** function attempts to connect to a URL (specified in the configuration) to retrieve a file. If successful, it returns the file.

```
def getFileLink(LINK, NOTIFY):

    try:
        logging.info('Trying to connect...')
        fileLink = requests.get(LINK)
        logging.info('Connection successful')
        return fileLink
    except requests.exceptions.RequestException as e:
        logging.error(e.strerror)
        if NOTIFY:
            message = buildPayload('File retrieval', 'ERROR, review log')
            subject = 'File Retrieval Notification'
            sender_email = CONFIG['SMTP_MAIL_SENDER']
            recipient_email = CONFIG['SMTP_MAIL_RECEIVERS']
            smtp_server = CONFIG['SMTP_SERVER']
            smtp_port = CONFIG['SMTP_PORT']
            smtp_password = CONFIG['SMTP_PASSWORD']
            sendEmail(message, subject, recipient_email, sender_email, smtp_server, smtp_port, smtp_password)
```

FIGURE 1.14 – getFileLink method.

The **extractFile** function takes a file obtained from the URL and extracts it to a specified target folder. It logs the extraction process.

```
def extractFile(fileLink, NOTIFY):
    path= 'test100.sql'
    target_folder = './temp'
    with zipfile.ZipFile(io.BytesIO(fileLink.content)) as archive:
        archive.extractall(target_folder)
        logging.info('EXTRACTION SUCCESSFUL: filename : test100.sql' )

    return path
```

FIGURE 1.15 – extractFile method.

The **archiveFile** function archives the SQL file extracted in the previous step. It creates a tar.gz (tgz) archive with a unique name based on the current date. It also checks for existing archives and handles differences, replacing old archives with new ones if necessary.

```python
def archiveFile(SQL_FILE, NOTIFY):
    path= 'temp/' + SQL_FILE
    temp = 'temp'
    last_backup = 'last_backup'
    temp_tgz = 'temp_tgz'

    logging.info('ARCHIVING FILE')
    if not os.path.exists(temp):
        print("Empty")
    else:
        logging.info('ARCHIVING FILE')
        now = datetime.datetime.now()
        name = now.strftime("%Y%d%m")
        logging.info("The archive name" + name)
        archiveName =  name + ".tgz"

        try:
         # Create the tar.gz archive
         with tarfile.open(archiveName, 'w:gz') as archive:
            # Add the SQL file to the archive
            archive.add(temp, arcname=os.path.basename(SQL_FILE))
        finally:
         # Clean up temporary directory
         empty(temp)

        # checking if there's a prior backup to compare with
        files = os.listdir(last_backup)

        # Filter out directories
        files = [f for f in files if os.path.isfile(os.path.join(last_backup, f))]

        # Check if there's only one file in the directory
        if len(files) == 1:
            file_name = files[0]
            compare_path = os.path.join(last_backup, file_name)

            # Comparing the hash of the latest backup file and the new one
            if diffrent_hashes(compare_path, archiveName, SQL_FILE)== True:
                logging.info('Hashes do not match. Proceeding with archiving.')
                add_ftp(archiveName)
```

FIGURE 1.16 – archiveFile method.

The **manageFile** function is responsible for managing files on the system, specifically deleting files that have exceeded a specified duration. The script initializes a logging configuration to record events in a log file.

```python
def manageFile(Duration, durationType, 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()
                # Compare the date of creation of the archive with the duration from configuration file
                if dateCreation + datetime.timedelta(**{durationType: Duration}) < now:
                    print("should be deleted")
                    print("-----------------")
                    try:
                        delete_ftp(f)
                        logging.info('FILES EXCEEDING DURATION DELETED')
                    except OSError as e:
                        logging.error(e.strerror)
                        if NOTIFY:
                            message = buildPayload('Files management', 'ERROR, review log')
                            subject = 'Files Management Notification'
                            sender_email = CONFIG['sender_email']
                            recipient_email = CONFIG['recipient_email']
                            smtp_server = CONFIG['smtp_server']
                            smtp_port = CONFIG['smtp_port']
                            smtp_password = CONFIG['smtp_password']
                            sendEmail(message, subject, recipient_email, sender_email, smtp_server, smtp_port, smtp_password)

            except OSError:
                logging.error("Path does not exist")
                if NOTIFY:
                    message = buildPayload('File management', 'ERROR, review log')
                    subject = 'Files Management Notification'
                    sender_email = CONFIG['sender_email']
                    recipient_email = CONFIG['recipient_email']
                    smtp_server = CONFIG['smtp_server']
                    smtp_port = CONFIG['smtp_port']
                    smtp_password = CONFIG['smtp_password']
                    sendEmail(message, subject, recipient_email, sender_email, smtp_server, smtp_port, smtp_password)
```

FIGURE 1.17 – manageFile

1.2.8 emailling.py

This script is responsible for sending email notifications. It constructs the email payload with a specified task and its state, attaches a log file, and sends the email using SMTP over SSL. It also logs in to the sender's email account to dispatch the notifications to the recipient's email addresses. If any error occurs during this process, it captures and prints an error message.

emailing.py

```python
import smtplib, ssl
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.base import MIMEBase
from email import encoders


def buildPayload(task, state):
    payload = f"| Task | State |\n|------|-------|\n| {task} | {state} |\n"
    return payload

def sendEmail(message, subject, recipient_email, sender_email, smtp_server, smtp_port, smtp_password):
    try:
        # Create a MIMEText object to represent the email content
        msg = MIMEMultipart()
        msg['From'] = sender_email
        msg['To'] = ", ".join(recipient_email)
        msg['Subject'] = subject

        # Attach the message as plain text
        with open('log.log', 'r') as file:
            data = file.read()
        message = message.format(data)
        msg.attach(MIMEText(message, 'plain'))
        filename = "log.log"
        attachment = open("log.log", "rb")
        part = MIMEBase('application', 'octet-stream')
        part.set_payload((attachment).read())
        encoders.encode_base64(part)
        part.add_header('Content-Disposition', "attachment; filename= %s" % filename)
        msg.attach(part)
        context = ssl.create_default_context()
        with smtplib.SMTP_SSL(smtp_server,smtp_port,context=context) as server:
            server.login(sender_email,smtp_password)
            print("log in to ur email was successfull ;).")
            server.sendmail(sender_email,recipient_email,msg.as_string())
    except Exception as e:
        print(f"Error: {e}")

```

FIGURE 1.18 – emailling script.

1.2.9 log.log

The log file displays a series of information messages related to the execution of a our scripts.

log.log

```
INFO:2023-11-05 20:06:28,153: Retrieved information successfully from configuration file
INFO:2023-11-05 20:06:28,153: Retrieved information successfully from configuration file
INFO:2023-11-05 20:06:28,154:Trying to connect...
INFO:2023-11-05 20:06:28,421:Connection successful
INFO:2023-11-05 20:06:28,425:EXTRACTION SUCCESSFUL: filename : test100.sql
INFO:2023-11-05 20:06:28,426:ARCHIVING FILE
INFO:2023-11-05 20:06:28,426:ARCHIVING FILE
INFO:2023-11-05 20:06:28,427:The archive name20230511
INFO:2023-11-05 20:06:28,431:No prior version found. Archiving directly without hash check.
INFO:2023-11-05 20:06:28,489:ARCHIVE SUCCESSFUL
```

FIGURE 1.19 – Logs output.

1.2.10 utilz.py

This script performs several file and hash-related operations :
The **diffrent_hashes** function compares the hashes of two SQL files contained in different tar.gz archives and returns **True** if they are different, indicating the need for archiving.

```python
def diffrent_hashes(file1_path, file2_path, sql_file_name):
    archive = False

    # Extract files from the first tgz file
    with tarfile.open(file1_path, "r:gz") as tar1:
        tar1.extractall(path= 'temp/TAR1')
        extracted_dir1 = f"temp/TAR1/{tar1.getnames()[0]}"

    # Extract files from the second tgz file
    with tarfile.open(file2_path, "r:gz") as tar2:
        tar2.extractall(path= 'temp/TAR2')
        extracted_dir2 = f"temp/TAR2/{tar2.getnames()[0]}"
    # Calculate the hash of the SQL file in the first extracted directory
    with open(f"{extracted_dir1}/{sql_file_name}", "rb") as sql_file1:
        hash1 = hashlib.sha256(sql_file1.read()).hexdigest()

    # Calculate the hash of the SQL file in the second extracted directory
    with open(f"{extracted_dir2}/{sql_file_name}", "rb") as sql_file2:
        hash2 = hashlib.sha256(sql_file2.read()).hexdigest()
    # Clean up extracted files
    shutil.rmtree(extracted_dir1)
    shutil.rmtree(extracted_dir2)
    if hash1 == hash2:
        archive = False
    else:
        archive = True

    return archive
```

FIGURE 1.20 – diffrent_hashes method.

The **replace_tgz** function replaces the contents of a directory with a new tar.gz file.

```python
def replace_tgz(original_path, new_tgz_path, target_directory):
    # Remove the original tgz file
    try:
        shutil.rmtree(target_directory)
    except FileNotFoundError:
        pass

    # Copy the new tgz file to the target directory
    shutil.copy(new_tgz_path, target_directory)



def add_tgz(new_tgz_path, target_directory):
    # Copy the new tgz file to the target directory
    shutil.copy(new_tgz_path, target_directory)
```

FIGURE 1.21 – replace_tgz method.

The **add_tgz** function adds a new tar.gz file to a target directory.

```python
def add_tgz(new_tgz_path, target_directory):
    # Copy the new tgz file to the target directory
    shutil.copy(new_tgz_path, target_directory)
```

FIGURE 1.22 – add_tgz method.

The **empty** function deletes all files and directories within a specified directory.

```python
def empty(directory_path):
    # Check if the directory exists
    if not os.path.exists(directory_path):
        print(f"Error: Directory '{directory_path}' does not exist.")
        return

    # Get a list of all files and directories in the target directory
    for item in os.listdir(directory_path):
        item_path = os.path.join(directory_path, item)
        if os.path.isfile(item_path):
            os.remove(item_path)  # Remove files
        elif os.path.isdir(item_path):
            shutil.rmtree(item_path)  # Remove directories and their contents
```

FIGURE 1.23 – empty method.

The **delete_file** function deletes a specific file.

```python
def delete_file(file_path):
    try:
        os.remove(file_path)
        print(f"{file_path} has been deleted successfully")
    except OSError as e:
        print(f"Error: {e.filename} - {e.strerror}")
```

FIGURE 1.24 – delete_file method.

These functions are used to manage and manipulate tar.gz archives and ensure that files are correctly archived and replaced when necessary.

1.3 Project Execution

A containerized FTP server instance is actively running, providing secure file transfer services within an isolated environment and we have as well a successful upload of the archived file

```
[+] Running 2/2
 ✔ Network p_ss_default  Created                                          0.6s
 ✔ Container p_ss-ftp-1  Created                                          0.3s
Attaching to p_ss-ftp-1
p_ss-ftp-1  | Changing password for admin
p_ss-ftp-1  | New password:
p_ss-ftp-1  | Bad password: too weak
p_ss-ftp-1  | Retype password:
p_ss-ftp-1  | passwd: password for admin changed by root
p_ss-ftp-1  | process has died, quitting
p_ss-ftp-1 exited with code 0
p_ss-ftp-1  | adduser: user 'admin' in use
p_ss-ftp-1  | Mon Nov  6 14:28:28 2023 [pid 36] CONNECT: Client "172.18.0.1"
p_ss-ftp-1  | Mon Nov  6 14:28:28 2023 [pid 35] [admin] OK LOGIN: Client "172.18.0.1"
p_ss-ftp-1  | Mon Nov  6 14:28:28 2023 [pid 37] [admin] OK UPLOAD: Client "172.18.0.1", "/home/adm
in/20230611.tgz", 3888 bytes, 3096.96Kbyte/sec
```

FIGURE 1.25 – ftp server container runnig.

Here we have an example of the notification that the user receives to keep him up to date whith the state of his files

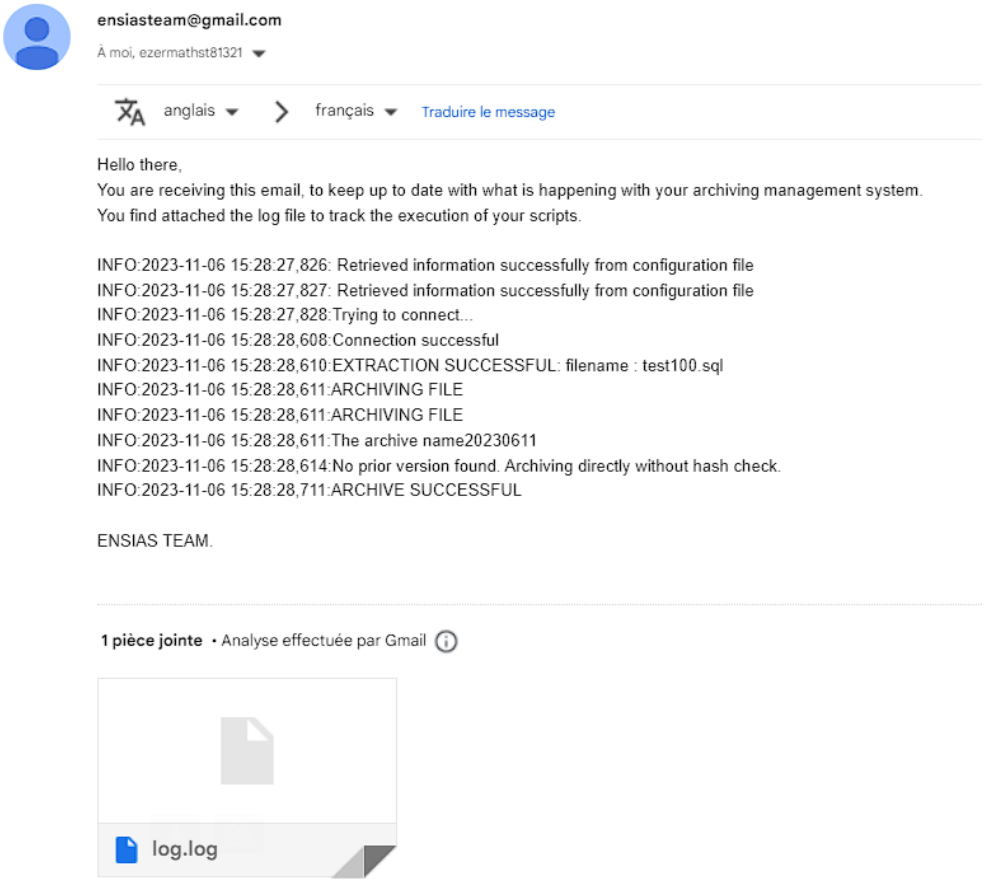


FIGURE 1.26 – exemple of the email notification.

Chapitre 2

Second Approach : From WEB Server to WebDAV

In this second approach, our aim is to optimize an archiving system that prioritizes the secure storage of SQL file backups acquired from web servers. We achieve this by leveraging WebDAV as a remote server to guarantee data integrity and facilitate long-term preservation. This approach is intended to deliver an effective and dependable archiving solution, particularly catering to users seeking a locally accessible and user-friendly archiving system.

2.1 Technical Stack

In carrying out our second approach of the project, we relied on a set of technical choices that can be justified by the table below :

Choix technique	*justification*
WebDAV	- Ease of Setup + It can be deployed quickly. - Ease of Access via standard protocols like HTTP/HTTPS - Cross-Platform Compatibility : supported by most operating systems. - Local Control : by using a local WebDAV server
Apache	- It provides a stable platform for applications - Apache's configuration file are human-readable and well-documented, making it relatively straightforward to set up and manage.
FTPS	- Easy to implement - Easy to read and understand.
Configuration file .json	- Easy to handle (read, import, write) - Compatible with our script - Well structured
Email	- Wide Compatibility: We chose Google's SMTPS server to ensure compatibility with a wide range of email providers and services. This decision allows us to communicate with a diverse set of email platforms, which was not feasible with our local server.

FIGURE 2.1 – Second Approach's Technical Choices

2.2 Project Structure

— **userDoc.sh** : A Bash script designed for launching the automation process. It simplifies the setup and execution of the archiving system.
— **userDocConfig.sh** : A Bash script that manages the installation of dependencies required for the archiving system to function properly.
— **readme.txt** : This text file serves as a user guide, providing instructions on how to use and run the archiving management system.
— **config.json** : This configuration file contains various settings, including email settings, remote server configurations, and the link to the SQL file.
— **webdav_configuration.txt** : This script provides instructions on configuring the WebDAV remote server, an integral part of the archiving system.
— **backup.py** : This script is responsible for executing the entire logic of the archive management system. It orchestrates the archiving process.
— **backupUtilities.py** : This script acts as a central repository for methods used in the archiving system. It also handles error detection and sends notifications to the user.
— **backup_exe.log** : This file compiles all the logs generated during the execution of the archiving system. It provides a record of system activities.
— **temp** : This is a directory used for temporary storage during the execution of the archiving process. It may be used for file downloads or other temporary data.
— **extracted_last_archive** : This directory is used to store the last archived file obtained from the remote server. It keeps track of the most recent backup.
— **extractedSQLFolder** : This directory hosts the SQL zip file downloaded from the web server. It's a temporary location for storing the data that needs to be archived.
— **__pycache__** : A directory generated by the Python interpreter when importing modules.

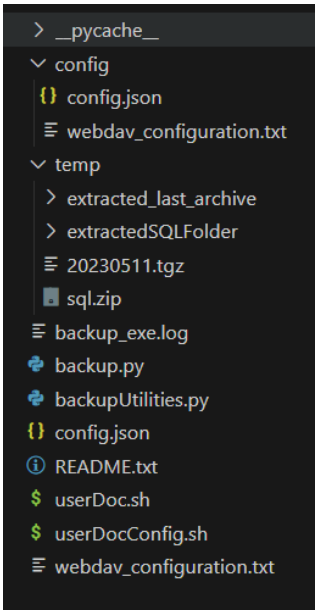


FIGURE 2.2 – Project structure.

2.2.1 userDoc.sh

This script automates the execution of another script (backup.py) by configuring a cron job. It prompts the user for a username, adds a crontab entry that schedules the execution of backup.py every day at 1 am, and starts the cron service. This ensures that the archiving process defined in backup.py runs automatically at the specified time.

```
# Prompt user for backup crontab username
read -p "Enter the username: " username

# Add the crontab entry with the provided values
(crontab -l ; echo "0 1 * * * /usr/bin/python3 ./backup.py") | crontab -u $username -



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

FIGURE 2.3 – userDoc.sh script.

2.2.2 userDocConfig.sh

This script prepares the environment for an automated archiving system. It installs Python 3, configures a WebDAV server, and sets up user authentication for WebDAV access. Additionally, it enables necessary Apache modules and restarts the Apache service to apply the new configurations. This script is a crucial pre-setup step for launching the automated archiving system.

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




#webdav configuration
echo "configuration WebDAV"
apt -y install apache2-utils

mkdir /home/webdav
chown www-data. /home/webdav
chmod 770 /home/webdav
cat ./config/webdav_configuration.txt >> /etc/apache2/sites-available/webdav.conf

# Set the password directly in the script
password="testeur123"

# The username you want to set the password for
username="ubuntu"

# Specify the path to the htpasswd file
htpasswd_file="/etc/apache2/.htpasswd"

# Use the htpasswd command with the -b option to set the password
htpasswd -b -c "$htpasswd_file" "$username" "$password"

# Check for any errors
if [ $? -eq 0 ]; then
  echo "Password has been set successfully for $username"
else
  echo "Failed to set the password"
fi

a2enmod dav*
a2ensite webdav
systemctl restart apache2
```

FIGURE 2.4 – userDocConfig.sh script.

2.2.3 readme.txt

This script is a guide provided to the user before they start using the archiving automated system. It provides important instructions and reminders for proper system usage.It highlights the location for configuring mail receivers and archive retention periods in the "./config/config.json" file and encourages users to contact "ensiasteam@gmail.com" for assistance or reporting any issues.

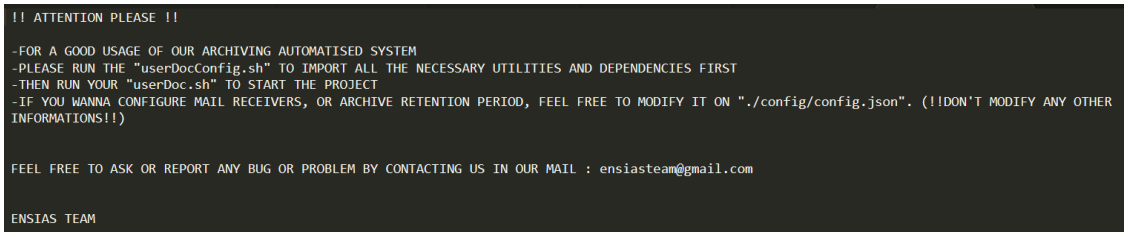


FIGURE 2.5 – readme usage guide.

2.2.4 config.json

The provided JSON script is a configuration file for an archiving automated system. It specifies settings such as file locations, server details, email notifications, and retention periods for the system to operate effectively.

FIGURE 2.6 – config.json file.

2.2.5 webdav_configuration.txt

This script is the configuration for an Apache virtual host where we specifie settings for serving WebDAV content over the HTTP protocol on a local server.

FIGURE 2.7 – webdav_configuration.txt

2.2.6 backup.py

This script imports a configuration from a JSON file and proceeds to perform the following tasks :

loading a configuration from a JSON file. This configuration likely contains various settings and parameters needed for the archiving process.

Then it checks if all the required configuration keys are present in the loaded configuration. If any required keys are missing, it logs an error message to alert the user.

FIGURE 2.8 – Loading configurations.

After that it cleans up old archives on a WebDAV server. It determines which archives have exceeded the specified retention period and deletes them. A summary of this action is added to the summary message.

Then it extracts the latest archive from the WebDAV server to a local directory. This archive is then used for comparison with the newly downloaded file.

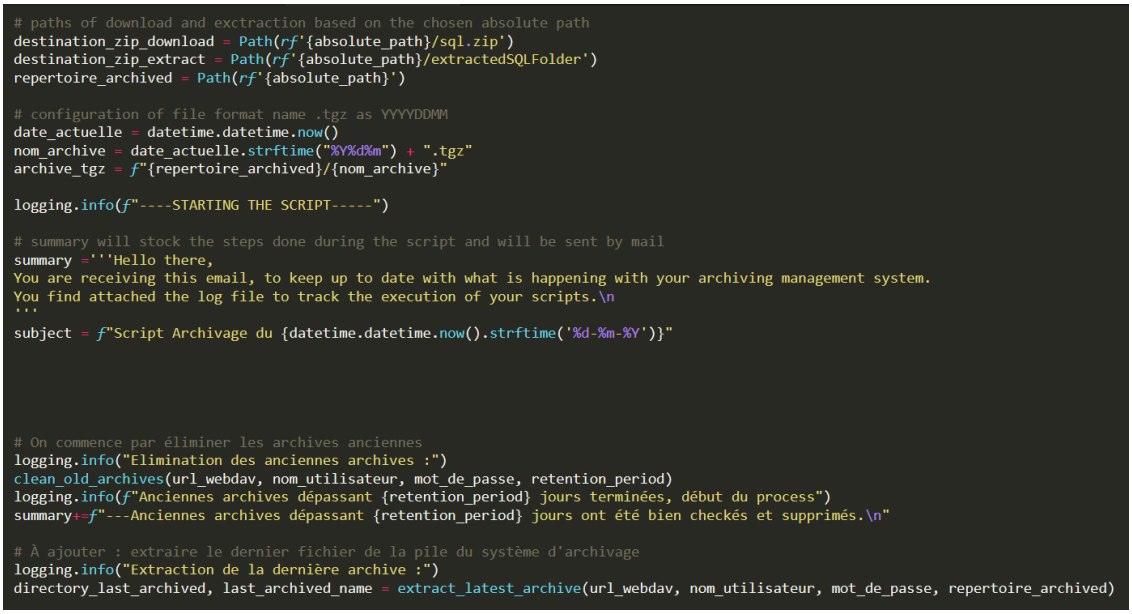


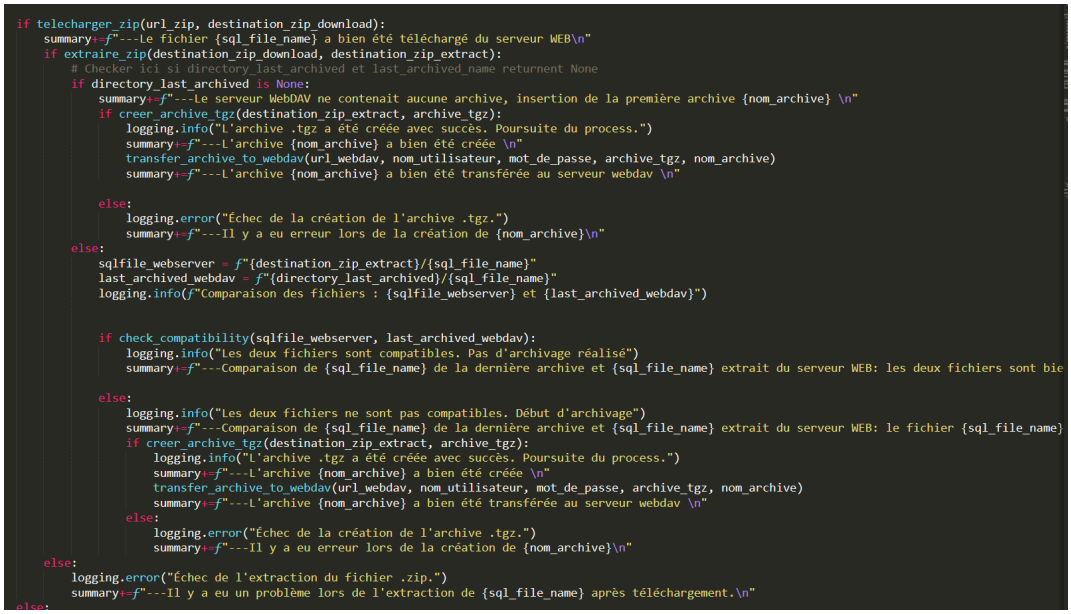
FIGURE 2.9 – Manage archives on webdav remote server.

Then it downloads a ZIP file from the specified URL and extracts its contents to a local directory. The success or failure of this process is logged, and the results are added to the summary message.

The script also compares the newly downloaded file with the last archived version. If the

files are identical, no further archiving is done. If they differ, the script proceeds to create a new archive of the downloaded file.

If a new archive needs to be created, the script packages the downloaded file into a TAR.GZ archive. The created archive is then transferred to the WebDAV server. Success or failure of these actions is logged, and the results are included in the summary.



```
if telecharger_zip(url_zip, destination_zip_download):
    summary+=f"---Le fichier {sql_file_name} a bien été téléchargé du serveur WEB\n"
    if extraire_zip(destination_zip_download, destination_zip_extract):
        # Checker ici si directory_last_archived et last_archived_name returnent None
        if directory_last_archived is None:
            summary+=f"---Le serveur WebDAV ne contenait aucune archive, insertion de la première archive {nom_archive} \n"
            if creer_archive_tgz(destination_zip_extract, archive_tgz):
                logging.info("L'archive .tgz a été créée avec succès. Poursuite du process.")
                summary+=f"---L'archive {nom_archive} a bien été créée \n"
                transfer_archive_to_webdav(url_webdav, nom_utilisateur, mot_de_passe, archive_tgz, nom_archive)
                summary+=f"---L'archive {nom_archive} a bien été transférée au serveur webdav \n"

            else:
                logging.error("Échec de la création de l'archive .tgz.")
                summary+=f"---Il y a eu erreur lors de la création de {nom_archive}\n"
        else:
            sqlfile_webserver = f"{destination_zip_extract}/{sql_file_name}"
            last_archived_webdav = f"{directory_last_archived}/{sql_file_name}"
            logging.info(f"Comparaison des fichiers : {sqlfile_webserver} et {last_archived_webdav}")


            if check_compatibility(sqlfile_webserver, last_archived_webdav):
                logging.info("Les deux fichiers sont compatibles. Pas d'archivage réalisé")
                summary+=f"---Comparaison de {sql_file_name} de la dernière archive et {sql_file_name} extrait du serveur WEB: les deux fichiers sont bie

            else:
                logging.info("Les deux fichiers ne sont pas compatibles. Début d'archivage")
                summary+=f"---Comparaison de {sql_file_name} de la dernière archive et {sql_file_name} extrait du serveur WEB: le fichier {sql_file_name}
                if creer_archive_tgz(destination_zip_extract, archive_tgz):
                    logging.info("L'archive .tgz a été créée avec succès. Poursuite du process.")
                    summary+=f"---L'archive {nom_archive} a bien été créée \n"
                    transfer_archive_to_webdav(url_webdav, nom_utilisateur, mot_de_passe, archive_tgz, nom_archive)
                    summary+=f"---L'archive {nom_archive} a bien été transférée au serveur webdav \n"
                else:
                    logging.error("Échec de la création de l'archive .tgz.")
                    summary+=f"---Il y a eu erreur lors de la création de {nom_archive}\n"
    else:
        logging.error("Échec de l'extraction du fichier .zip.")
        summary+=f"---Il y a eu un problème lors de l'extraction de {sql_file_name} après téléchargement.\n"
else:
```

FIGURE 2.10 – Backup archive, log and mailling notification.

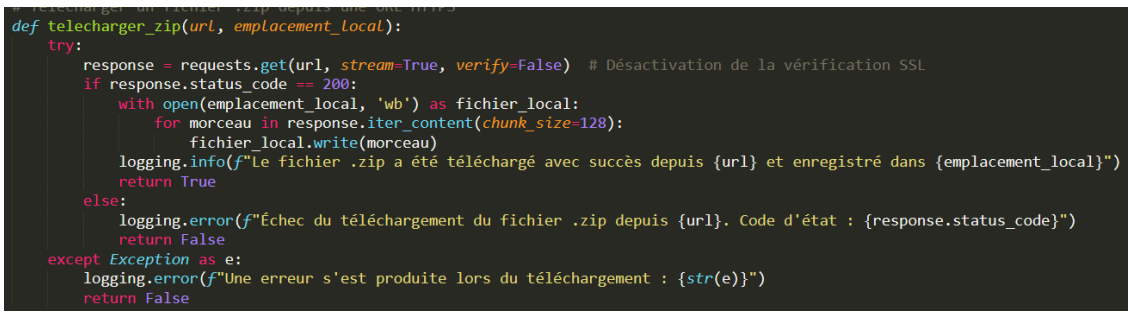
Throughout the script, various actions, successes, and errors are logged. These log entries are appended to a summary message. After completing the archiving process, the script sends an email with the summary message, including the log details, to specified recipients.

The script is designed to automate archiving tasks, provide notifications, and ensure that archives are transferred to a WebDAV server with proper management of retention periods.

2.2.7 backupUtilities.py

This script is a collection of functions for managing our archiving system.

telecharger_zip This function downloads a ZIP file from a specified URL and saves it locally. It makes an HTTP request and retrieve the file. If successful, it returns True, otherwise, it returns False.



```
def telecharger_zip(url, emplacement_Local):
    try:
        response = requests.get(url, stream=True, verify=False)  # Désactivation de la vérification SSL
        if response.status_code == 200:
            with open(emplacement_local, 'wb') as fichier_local:
                for morceau in response.iter_content(chunk_size=128):
                    fichier_local.write(morceau)
            logging.info(f"Le fichier .zip a été téléchargé avec succès depuis {url} et enregistré dans {emplacement_local}")
            return True
        else:
            logging.error(f"Échec du téléchargement du fichier .zip depuis {url}. Code d'état : {response.status_code}")
            return False
    except Exception as e:
        logging.error(f"Une erreur s'est produite lors du téléchargement : {str(e)}")
        return False
```

FIGURE 2.11 – telecharger_zip method.

extraire_zip : This function extracts a ZIP archive to a specified destination directory. If the extraction. was successful, it returns True, otherwise, it returns False.

```python
#Extraire un fichier .zip
def extraire_zip(chemin_zip, repertoire_destination):
    try:
        with zipfile.ZipFile(chemin_zip, 'r') as zip_ref:
            zip_ref.extractall(repertoire_destination)
        logging.info(f"Le fichier .zip a été extrait avec succès dans {repertoire_destination}")
        return True
    except Exception as e:
        logging.error(f"Une erreur s'est produite lors de l'extraction du fichier .zip : {str(e)}")
        return False
```

FIGURE 2.12 – extraire_zip method.

creer_archive_tgz : This function creates a TAR.GZ archive from files in a specified directory. If the archiving was successful, it returns True, otherwise, it returns False.

```python
def creer_archive_tgz(repertoire, archive_tgz):
    try:
        with tarfile.open(archive_tgz, "w:gz") as tar:
            for root, _, files in os.walk(repertoire):
                for file in files:
                    file_path = os.path.join(root, file)
                    arcname = os.path.relpath(file_path, repertoire)
                    tar.add(file_path, arcname=arcname)
            # Ajoutez un message de journalisation pour indiquer que l'archive a été créée avec succès
            logging.info(f"L'archive .tgz a été créée avec succès : {archive_tgz}")
            return True
    except Exception as e:
        # Ajoutez un message de journalisation en cas d'erreur
        logging.error(f"Une erreur s'est produite lors de la création de l'archive .tgz : {str(e)}")
        return False
```

FIGURE 2.13 – creer_archive_tgz method.

check_webdav_connection : This function checks the connection to a WebDAV server by sending an HTTP request with the provided credentials. It returns True if the connection is successful, otherwise, it returns False.

```python
#Checker l'état de la connection webdav
def check_webdav_connection(url_webdav, nom_utilisateur, mot_de_passe):
    try:
        response = requests.head(url_webdav, auth=(nom_utilisateur, mot_de_passe))
        if response.status_code == 200:
            # Ajoutez un message de journalisation pour indiquer une connexion réussie
            logging.info("Connexion au serveur WebDAV réussie.")
            return True
        else:
            # Ajoutez un message de journalisation en cas d'échec de connexion
            logging.error(f"Échec de la connexion au serveur WebDAV. Code d'état : {response.status_code}")
    except Exception as e:
        # Ajoutez un message de journalisation en cas d'erreur
        logging.error(f"Une erreur s'est produite lors de la connexion : {str(e)}")
    return False
```

FIGURE 2.14 – check_webdav_connection method.

transfer_archive_to_webdav : This function transfers a TAR.GZ archive to a WebDAV server. It first checks the WebDAV connection, and if successful, it uploads the archive to the server. It logs the success or failure of the transfer.

```python
#Transférer l'archive .tgz généré à notre serveur webdav local
def transfer_archive_to_webdav(url_webdav, nom_utilisateur, mot_de_passe, archive_tgz, nom_fichier):
    try:
        if check_webdav_connection(url_webdav, nom_utilisateur, mot_de_passe):
            url_destination = f"{url_webdav}/{nom_fichier}"

            # Ensure that the archive file is accessible
            if os.path.exists(archive_tgz):
                with open(archive_tgz, 'rb') as fichier:
                    response = requests.put(url_destination, data=fichier, auth=(nom_utilisateur, mot_de_passe))
                if response.status_code == 201 or response.status_code==204:
                    logging.info(f"L'archive .tgz a été transférée avec succès sur {url_destination}")
                else:
                    logging.error(f"Échec du transfert de l'archive .tgz. Code d'état : {response.status_code}")
            else:
                logging.error(f"Archive file '{archive_tgz}' does not exist or is inaccessible. Check local permissions.")
    except Exception as e:
        logging.error(f"Une erreur s'est produite lors du transfert : {str(e)}")
```

FIGURE 2.15 – transfer_archive_to_webdav method.

clean_old_archives : This function cleans old archives from the WebDAV server based on a specified retention period. It calculates the age of each archive and deletes those that exceed the retention period.

```python
# Eliminer les archives ayant dépassé "retention_period" jours
def clean_old_archives(url_webdav, nom_utilisateur, mot_de_passe, retention_period):
    try:
        # Calculate the current date
        current_date = datetime.datetime.now()

        # Access the root of the WebDAV server
        url_path = url_webdav
        response = requests.request("PROPFIND", url_path, auth=(nom_utilisateur, mot_de_passe))

        if response.status_code == 207:
            # Parse the XML response
            root = ET.fromstring(response.text)

            # Iterate through the 'response' elements
            for response_elem in root.findall('.//{DAV:}response'):
                href_elem = response_elem.find('{DAV:}href')
                if href_elem is not None:
                    file_name = href_elem.text.strip('/')  # Extract file name from the 'href' element

                    # Try to parse the date from the file name
                    try:
                        year = int(file_name[:4])
                        month = int(file_name[4:6])
                        day = int(file_name[6:8])
                        last_modified = datetime.datetime(year, day, month)
                        age = current_date - last_modified


                        if age > timedelta(days=retention_period):
                            # If the file is older than the retention period, delete it
                            url_file = f"{url_path}/{file_name}"
                            response = requests.delete(url_file, auth=(nom_utilisateur, mot_de_passe))
                            if response.status_code == 204:
                                logging.info(f"Archived file '{file_name}' deleted as it exceeds the retention period.")
                            else:
                                logging.error(f"Failed to delete archived file '{file_name}' with status code: {response.status_code}")
                    except ValueError:
                        logging.warning(f"Skipping file '{file_name}' - invalid date format")

            else:
```

FIGURE 2.16 – clean_old_archives method.

extract_latest_archive : This function extracts the latest TAR.GZ archive from the WebDAV server to a local directory. It checks the WebDAV connection, lists available archives, and downloads the latest one. It returns the path to the extraction directory and the name of the latest archive.

```python
def extract_latest_archive(url_webdav, nom_utilisateur, mot_de_passe, dossier_destination):
    try:
        options = {
            'webdav_hostname': url_webdav,
            'webdav_login': nom_utilisateur,
            'webdav_password': mot_de_passe
        }
        client = Client(options)

        # List files in the WebDAV directory
        file_list = client.list()

        # Filter the list to keep only .tgz files
        tgz_files = [f for f in file_list if f.endswith('.tgz')]

        if not tgz_files:
            logging.warning("Aucun fichier .tgz trouvé sur le serveur WebDAV.")
            return None, None

        # Find the latest .tgz file based on the file names (assuming they follow a YYYYMMDD.tgz naming conventi
        latest_tgz = max(tgz_files)
        logging.info(f"Fichier le plus récent '{latest_tgz}' trouvé sur le serveur WebDAV.")

        # Define the local file path for the downloaded archive
        local_archive_path = f"{dossier_destination}/{latest_tgz}"

        # Download the latest archive
        client.download_sync(latest_tgz, local_archive_path)

        logging.info(f"Fichier le plus récent '{latest_tgz}' téléchargé avec succès dans {local_archive_path}")

        # Define the local directory path for extraction
        extraction_directory = f"{dossier_destination}/extracted_last_archive"

        # Extract the archive to the local directory
        with tarfile.open(local_archive_path, "r:gz") as tar:
            tar.extractall(path=extraction_directory)

        logging.info(f"Archive extraite dans le dossier local '{extraction_directory}'")

        # Return the path of the extraction directory and the name of the latest .tgz file
        return extraction_directory, latest_tgz
```

FIGURE 2.17 – extract_latest_archive method.

sendEmail : This function sends an email with a specified message and log file as an attachment to the specified recipients. It uses the smtplib library for email sending. It logs success or failure in sending emails.

```python
#Send log file and a message resuming the state of our running script
def sendEmail(message, subject, recipients, sender_email, smtp_server, smtp_port, smtp_password):
    for to_email in recipients:
        try:
            # Create a MIMEText object to represent the email content
            msg = MIMEMultipart()
            msg['From'] = sender_email
            msg['To'] = to_email
            msg['Subject'] = subject

            # Attach the message as plain text + add log file
            msg.attach(MIMEText(message, 'plain'))

            filename = "backup_exe.log"
            attachment = open(filename, "rb")
            part = MIMEBase('application', 'octet-stream')
            part.set_payload((attachment).read())
            encoders.encode_base64(part)
            part.add_header('Content-Disposition', "attachment; filename= %s" % filename)
            msg.attach(part)
            context = ssl.create_default_context()
            with smtplib.SMTP_SSL(smtp_server, smtp_port, context=context) as server:
                server.login(sender_email, smtp_password)
                logging.info("Logged in to your email successfully.")
                server.sendmail(sender_email, to_email, msg.as_string())
                server.quit()
                logging.info(f"Email sent to {to_email} successfully.")
        except Exception as e:
            logging.error(f"Error sending email to {to_email}: {str(e)}")
```

FIGURE 2.18 – sendEmail method.

These functions work together to automate the archiving process, transfer archives to a WebDAV server, manage old archives, and notify users of the system's progress and status.

2.3 backup_exe.log

The log file displays a series of information messages related to the execution of a our scripts.

```
2023-11-06 09:50:22 - INFO - Configuration values were imported correctly.
2023-11-06 09:50:22 - INFO - ----STARTING THE SCRIPT-----
2023-11-06 09:50:22 - INFO - Elimination des anciennes archives :
2023-11-06 09:50:22 - WARNING - Skipping file '' - invalid date format
2023-11-06 09:50:22 - INFO - Anciennes archives dépassant 7 jours terminées, début du process
2023-11-06 09:50:22 - INFO - Extraction de la dernière archive :
2023-11-06 09:50:22 - INFO - Fichier le plus récent '20230511.tgz' trouvé sur le serveur WebDAV.
2023-11-06 09:50:22 - INFO - Fichier le plus récent '20230511.tgz' téléchargé avec succès dans temp/20230511.tgz
2023-11-06 09:50:22 - INFO - Archive extraite dans le dossier local 'temp/extracted_last_archive'
2023-11-06 09:50:22 - INFO - Le fichier .zip a été téléchargé avec succès depuis https://leafy-kringle-a61e00.netlify.app/sql-sample.Zip et enregistré
dans temp/sql.zip
2023-11-06 09:50:22 - INFO - Le fichier .zip a été extrait avec succès dans temp/extractedSQLFolder
2023-11-06 09:50:22 - INFO - Comparaison des fichiers : temp/extractedSQLFolder/test100.sql et temp/extracted_last_archive/test100.sql
2023-11-06 09:50:22 - INFO - Les deux fichiers sont compatibles. Pas d'archivage réalisé
2023-11-06 09:50:22 - INFO - ----ENDING THE SCRIPT-----
2023-11-06 09:50:23 - INFO - Logged in to your email successfully.
2023-11-06 09:50:24 - INFO - Email sent to lesvideodezack@gmail.com successfully.
```

FIGURE 2.19 – Logs output.

2.4 Project Execution

Let's study our two main cases :

2.4.1 No archived files in the WebDAV Server

As there's no archived files in the WebDAV Server :

FIGURE 2.20 – Empty WebDAV Server

the downloaded sql file from the web server will be directly archived and uploaded in our WebDAV Server Here are the mail received by the user and the content of backup_exe.log attached file :

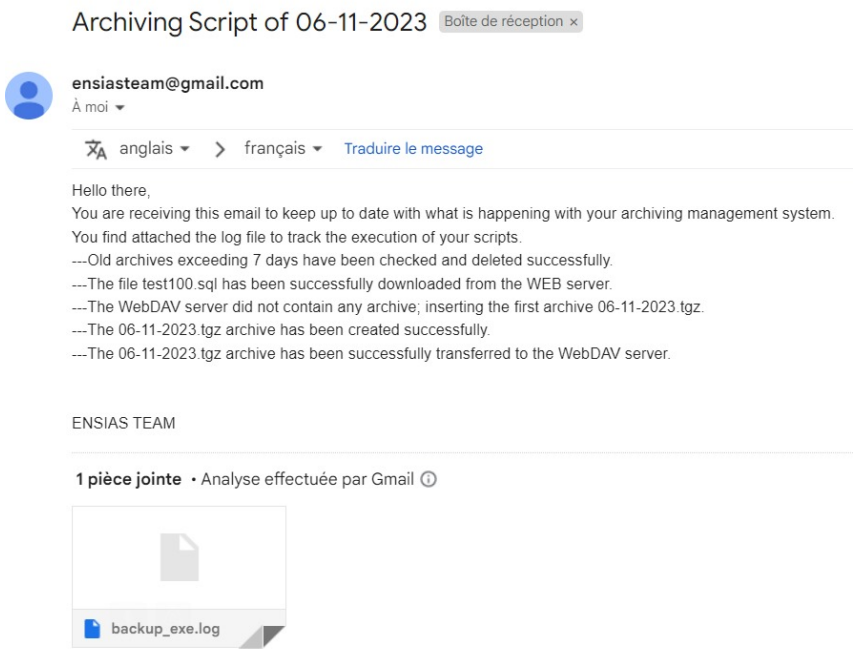


FIGURE 2.21 – Case 1 : Received Email

2023-11-06 19:36:28 - INFO - Configuration values were imported correctly.
2023-11-06 19:36:28 - INFO - ----STARTING THE SCRIPT-----
2023-11-06 19:36:28 - INFO - Cleaning old archives:
2023-11-06 19:36:28 - WARNING - Skipping file " - invalid date format
2023-11-06 19:36:28 - INFO - Old archives exceeding 7 days have been checked and deleted successfully. The process begins.
2023-11-06 19:36:28 - INFO - Extracting the latest archive:
2023-11-06 19:36:28 - WARNING - Aucun fichier .tgz trouvé sur le serveur WebDAV.
2023-11-06 19:36:28 - INFO - The .zip file was successfully downloaded from https://leafy-kringle-a61e00.netlify.app/sql-sample.Zip and saved to temp/sql.zip
2023-11-06 19:36:28 - INFO - The .zip file has been successfully extracted to temp/extractedSQLFolder
2023-11-06 19:36:28 - INFO - L'archive .tgz a été créée avec succès : temp/20230611.tgz
2023-11-06 19:36:28 - INFO - The .tgz archive has been created successfully. The process continues.
2023-11-06 19:36:28 - INFO - Connexion au serveur WebDAV réussie.
2023-11-06 19:36:28 - INFO - L'archive .tgz a été transférée avec succès sur http://127.0.1.1//20230611.tgz
2023-11-06 19:36:28 - INFO - ----ENDING THE SCRIPT-----

FIGURE 2.22 – Case 1 : Received Logs

2.4.2 WebDAV Server contains at least one archived file

For example, we gonna use the WebDAV Server with one archived file :

FIGURE 2.23 – WebDAV Server with an archived file

Here we have two sub cases :

SQL WebServer file not modified

Here, our script will end in comparing the two files (webserver one and the last archived in webdav). Here are the mail received by the user and the content of backup_exe.log attached file :

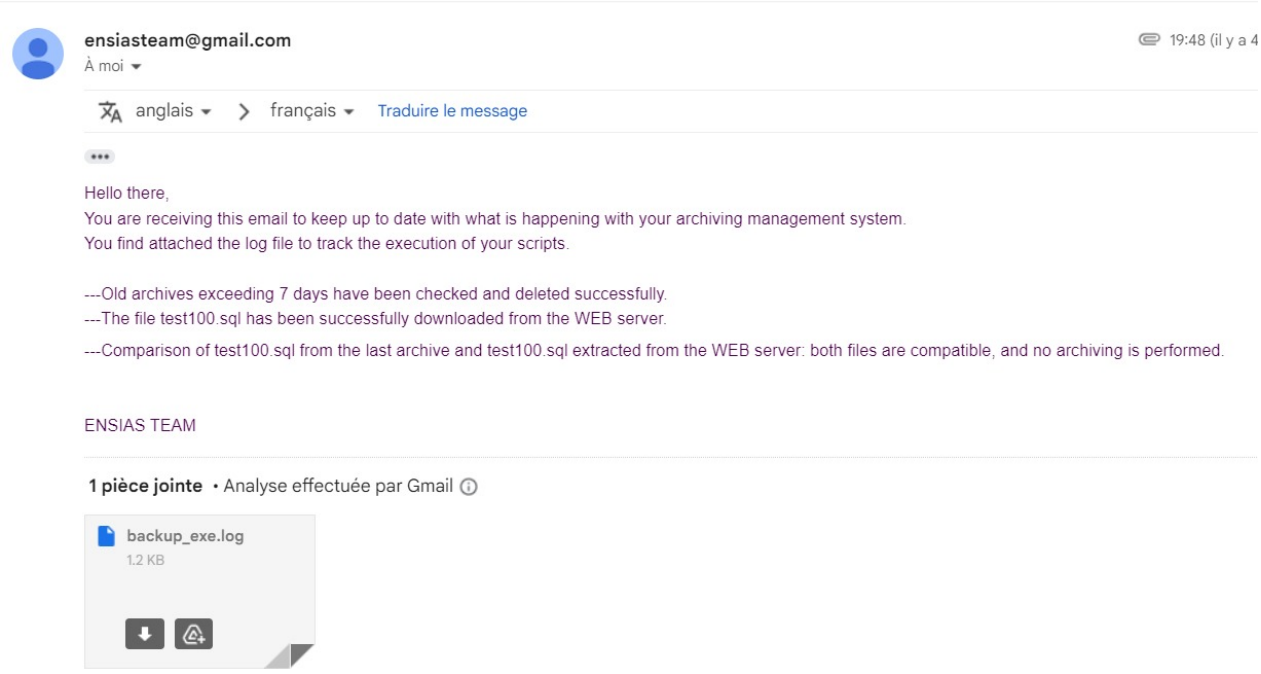


FIGURE 2.24 – Case 2 (No Modification) : Received Email

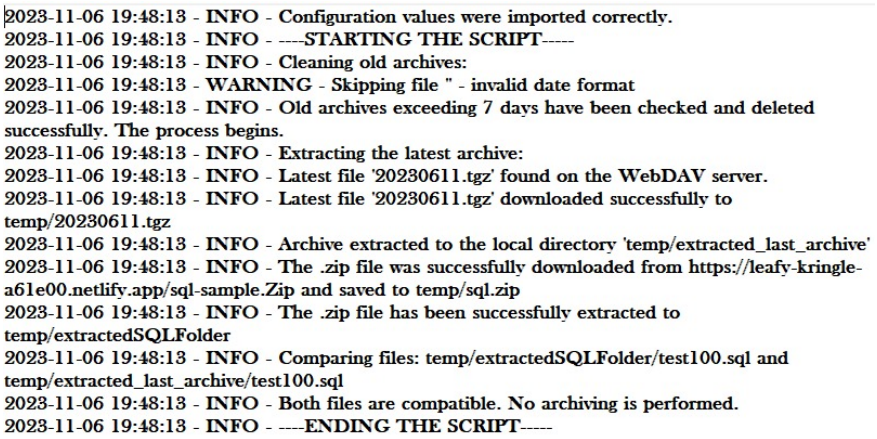


FIGURE 2.25 – Case 2 (No Modification) : Received Logs

SQL WebServer file modified

Here, after comparing the two files (webserver one and the last archived in webdav) and finding that there are different, a new archive will be created under this specified nomination "YYYYDDMM.tgz" and will be transfered to our local WebDAV Server Here are the mail received by the user and the content of "backup_exe.log" attached file :

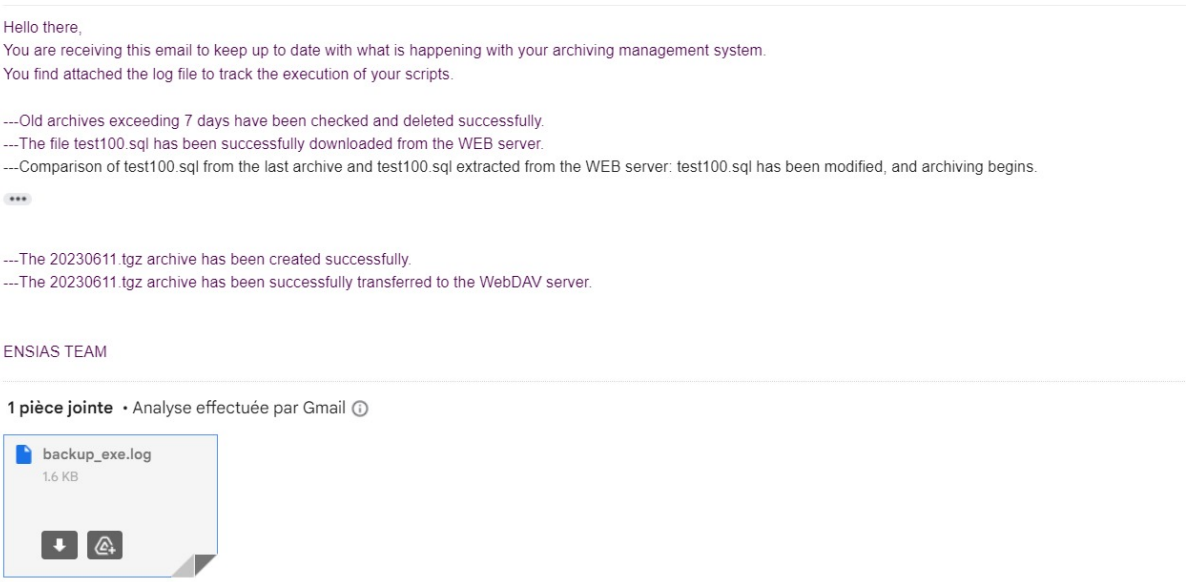


FIGURE 2.26 – Case 2 (File Modified) : Received Email

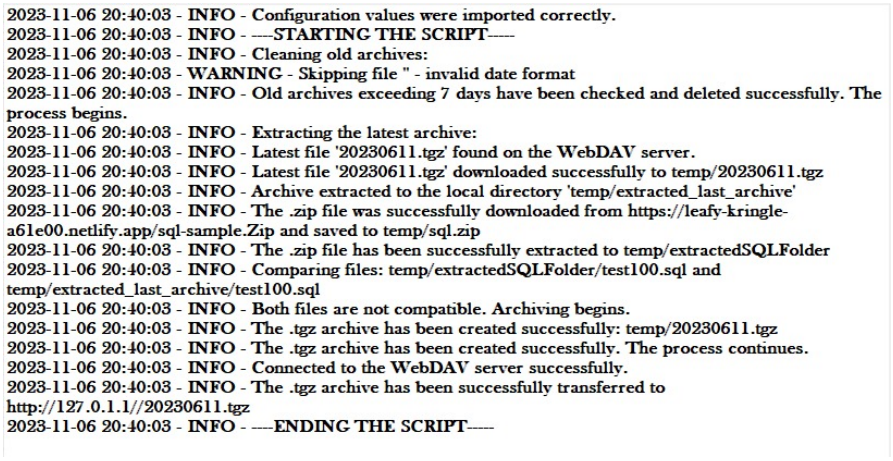


FIGURE 2.27 – Case 2 (File Modified) : Received Logs

Conclusion

The archiving system using WebDAV stands out as a more secure and user-friendly solution, ensuring data integrity and long-term preservation. It simplifies the archiving process and is compatible with different platforms. However, it may require some initial configuration.

In contrast, the FTP in Docker approach is more suitable for users with a strong technical background and a specific preference for Docker containers. It offers automation and portability but may pose security risks.

Ultimately, the choice between the two approaches depends on user preferences, technical expertise, and the specific requirements of the archiving system.