ARCHIVING TOOL: TECHNICAL GUIDE

LEANG Denis

2021-2022

FISE 2

Telecom Saint-Etienne

Table des matières

I – About the code **2**

- **Libraries** 2
- **Architecture** 3

II – About the files **4**

- **Configuration file** 4
- **Non classes files** 5

1. **server.py** **5**
2. **log.py** **5**

- **Classes files** 6

1. **archive.py** **6**
2. **mail.py** **7**
3. **smb_client.py** **8**

- **Batch file** 9

III – How it works **10**

IV – Conclusion **11**

I – About the code

This is a technical guide for my Python archiving tool.

Libraries

I used many built in libraries and two external ones. Here is the complete list:

- **Built in libraries**
  - time/datetime: Allowed me to create variables with precious data on date and time
  - json: Allowed me to used ***config.json*** file and manipulate json objects
  - os: Allowed me to easily manipulate files and paths
  - zipfile: Allowed me to manipulate zipfiles (extract)
  - tarfile: Allowed me to manipulate tarfiles (compress)
  - shutil: Allowed me to delete files
  - email/smtplib/ssl: Allowed me to send E-mails
  - http.server/socketserver: Allowed me to establish an HTTP server

- **External libraries**
  - pysmb: Allowed me to establish connection with SMB server and do operations
  - requests: Allowed me to send/receive HTTP requests

I also used pylint to help me write the code more conveniently. It forced me to be careful about how I wrote my functions, and I spent a lot of time improving them until I reached 10/10 on every single .**py** files.

- ## Architecture

I tried to be as simple as possible when building my code architecture. Here are the complete folder and sub directories:

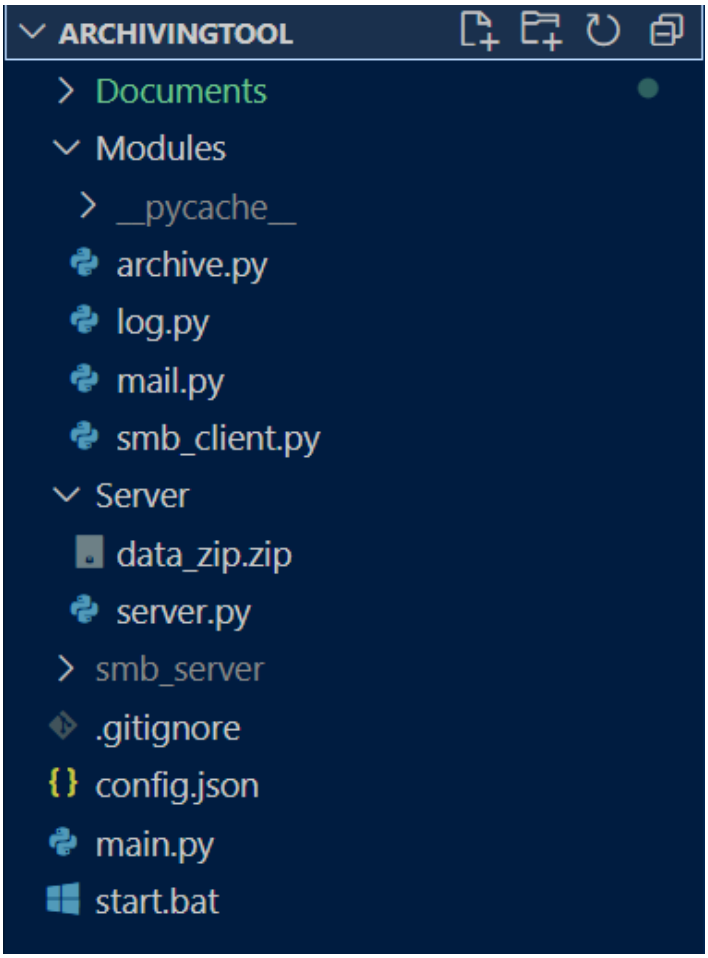

Figure 1: Code architecture

The core of my archiving tool is inside the ***main.py*** file. Therefore I decided to put it at the root of the folder alongside the ***config.json***, which is also very important (configuration file), and ***start.bat***, which basically runs the whole archiving tool.

I separated the rest of the code into two folders:

- **Server**: contains everything related to Web server and its launching. It also contains the zip file one wishes to download
- **Modules**: contains classes and methods we will need to do our archiving tool

II – About the files

Here I will explain the choices I made and how my code is written for each file.

- ## Configuration file

I decided to use a ***config.json*** file. It is a quite simple and convenient way to allow a user to modify the script's behavior without having to dive into the ***.py*** files. The user must only change the values in the different json objects. Here is the whole ***config.json*** file:

```
{
        "archive": {
                "url": "http://localhost:8000/Server/",
                "zip_file": "data_zip.zip",
                "file": "data.sql",
                "log_file": "log.txt",
                "expiring_time": 86400
        },

        "smb_client": {
                "username": "username",
                "password": "password",
                "machine_name": "name",
                "server_name": "pc-name",
                "ip_adress": "ip_address",
                "smb_share": "smb_server_name",
                "save_duration": 86400
        },

        "mail": {
                "sender_adress": "sender@gmail.com",
                "password": "password",
                "port": "465",
                "smtp_server": "smtp.gmail.com",
                "receiver_adress": "receiverg@gmail.com",
                "send_mode": "on",
                "attach_mode": "on"
        }
}
```

Figure 2: config.json file

For more details about what each object does, just refer to the "User guide" document. For better readability, I also decided to name these json objects very similarly to the classes I will talk about. It is way simpler for any foreigner to the code to identify in which file the json object will be most likely used.

- **Non classes files**

1. server.py

This python file is self-explanatory and very simple. It just launches a web server in the specified code. Here is the whole code:

```
"""
    Script creating HTTPS server
"""

import http.server
import socketserver

PORT = 8000
HANDLER = http.server.SimpleHTTPRequestHandler

def launch_server():
    """
    Function launching the HTTPS server
    """
    print("Server started at localhost:" + str(PORT))
    with socketserver.TCPServer(("", PORT), HANDLER) as httpd:
        httpd.serve_forever()

launch_server()

```

Figure 3: server.py file

2. log.py

This python file is also simple. This is where I stored the methods, I needed in the other classes for logs management. Here are the different methods:

- **add_to_log**(file, line): Appends a line to a text file starting from a new line
- **clear_log**(file): Clears a text file by erasing its whole content
- **delete_log**(file): Deletes a file
- **read_log**(file): Returns the content of a text file inside a variable
- **read_log_first_line**(file): Returns "Operation: Successful" or "Operation: Failure" depending on the first line of the text file. It is mainly used for the Emails' title to identify if the operation failed or succeded

It will be mainly used in the Archive class we will see now.

- ## Classes files

1. archive.py

This is the biggest class of the three. This class contains methods allowing the user to do every single of the operations on this archiving tool except for the emails. Here are the attributes:

```
def __init__(self, config_file):
    """
    Attributes :
    --> url : Web server's url (localhost)
    --> zip_file : Zip file name
    --> file : File name
    --> tgz_file : Tgz file name
    --> log_file : log file name
    --> expiring_time : Duration (in seconds) for which files are
    considered outdated
    """
    with open(config_file, encoding="utf8") as conf:
        data = json.load(conf)
        archive = data["archive"]
    self.url = archive["url"]
    self.zip_file = archive["zip_file"]
    self.file = archive["file"]
    self.tgz_file = DATE + ".tgz"
    self.log_file = archive["log_file"]
    self.expiring_time = archive["expiring_time"]
```

Figure 4: archive.py file

The methods names are explicit and immediately explain what they are used for:

- **download_file**(self): Downloads a file from web server
- **extract_zip**(self): Extracts file from zip
- **check_file**(self): Checks is file is up to date compared to the expiring time entered in ***config.json***
- **create_tgz**(self): Creates a tgz with a file
- **send_tgz**(self, client): Sends a tgz file to the SMB server
- **clean_zip** (self), **clean_file**(self) **clean_tgz**(self): Deletes these files from current folder
- **clean_server**(self, client): Deletes outdated files in SMB server
- **clear_log_file**(self) **delete_log_file**(self) **read_log_file**(self) **read_log_file_first_line**(self): Use the methods from ***log.py*** to perform the actions explained above on self.log_file

As you can see, class Archive handles a lot of the functionalities of the archiving tool.

- **Main operations**: The operations are all handled by the methods with the same name
- **Logs**: Every time one of these methods is called, the **add_to_log** method from log will be called to append the state of the operation to a text file
- **Save duration**: The clean_server method comes in handy with this functionality. It enables old versions archiving controlling by date of modification.

2. mail.py

It is a rather short class which contains only two methods. This class handles all the emails operations. Here are the attributes which are also linked to the **config.json** values:

```
def __init__(self, config_file):
    """
    Attributes :
    --> username : Current session username
    --> password : Current session password
    --> machine_name : Arbitrary string
    --> server_name : Name of the computer
    --> ip : IP
    --> log_file : log file name
    """
    with open(config_file, encoding="utf8") as conf:
        data = json.load(conf)
        mail = data["mail"]
        archive = data["archive"]
    self.sender_adress = mail["sender_adress"]
    self.password = mail["password"]
    self.port = mail["port"]
    self.smtp_server = mail["smtp_server"]
    self.receiver_adress = mail["receiver_adress"]
    self.log_file = archive["log_file"]
```

Figure 5: mail.py file

As explained above, the methods do these actions:

- **send_email_ssl**(self, body, subject): Sends an email
- **send_email_ssl_with_log**(self, body, subject): Sends an email with log attached

I used the SSL method as it is quite safe and easy to use. It does the job fast and efficiently.

The mailing functionality is fully handled with this class:

- **Emails**: The are called depending on the send_mode and attach_mode values in ***config.json***.

3. smb_client.py

Lastly, this class is also quite small. It handles connection with SMB server and operations related to it. Here are the attributes with the description:

```
def __init__(self, config_file):
    """
    Attributes :
    --> username : Current session username
    --> password : Current session password
    --> machine_name : Arbitrary string
    --> server_name : Name of the computer
    --> ip : IP
    --> smb_share : Name of the samba folder share
    --> save_duration : Duration (in seconds) for which files are stored
    """
    with open(config_file, encoding="utf8") as conf:
        data = json.load(conf)
        smb_client = data["smb_client"]
    self.username = smb_client["username"]
    self.password = smb_client["password"]
    self.machine_name = smb_client["machine_name"]
    self.server_name = smb_client["server_name"]
    self.ip_adress = smb_client["ip_adress"]
    self.smb_share = smb_client["smb_share"]
    self.save_duration = smb_client["save_duration"]
```

Figure 6: smb_client.py file

It contains two self-explanatory methods:

- **upload**(self, file_to_store, data): Stores a file inside the smb share
- **delete**(self): Deletes all files in the SMB server when they're considered outdated according to the value entered in ***config.json*** file for save_duration

SmbClient and Archive are therefore closely related to each other.

- ## Batch file

To make the launching of the tool automatic, I decided to use a batch file called ***start.bat***. I found it very simple to use for a user who may not know a lot about coding. This file looks like it acts the same as an ***.exe*** file which boots immediately after double clicking it. It also only took me two lines of code to create the automatic launching feature. With the steps provided in the User Guide, there simply is no way the user may not be able to use my archiving tool. Here is the code:

```
start py C:\Path\To\Script\Server\server.py
schtasks /create /sc daily /st 00:00 /tn "Archival Tool" /tr C:\Path\To\Script\main.py
```

Figure 7: start.bat file

To provide more information about how I wrote the batch file, the ***schtasks*** are quite the equivalent of ***cron*** for UNIX devices. Since I and 90% of users in the world use Windows OS on the daily basis, I found it more relevant to favor Windows users than UNIX ones.

III – How it works

Now that we have reviewed every single piece of code, I will explain in this section the general pattern of my archiving tool.

To begin with two files are launched in this order:

- ***server.py***: Launches the web server with the zip inside
- ***main.py***: Launches every single operation

Launching it in this order is important else the downloading of the zip will fail so will the rest of the operations. There are 6 important statuses in my tool: [DOWNLOAD], [EXTRACT], [CHECK], [COMPRESS], [UPLOAD], [CLEAN]. Setting up these statuses are very useful when reading the log file as it directly indicates which step failed in the process. For instance, here is a completely successful log:

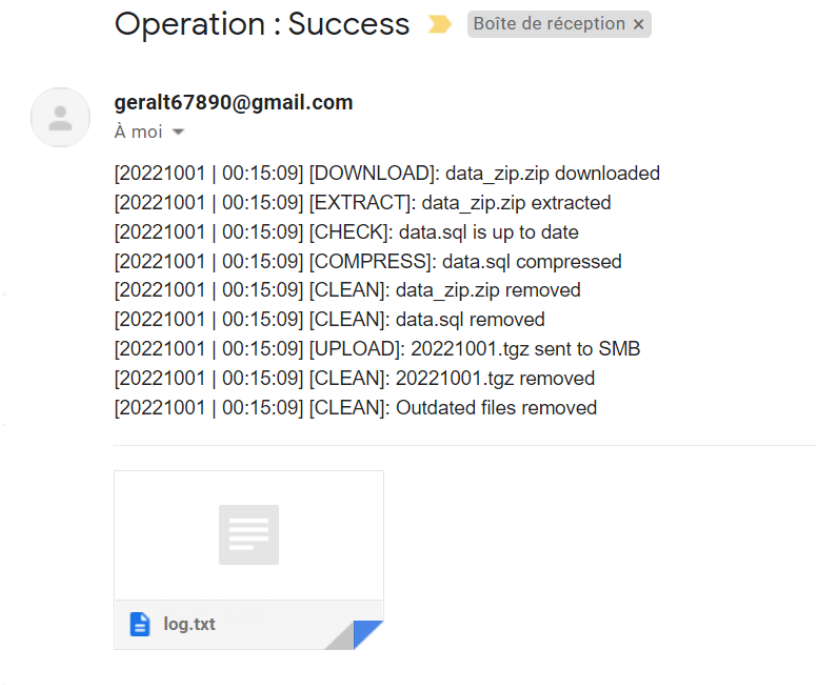

Figure 8: Successful launch

That order can easily be seen in main.py thank to the class made architecture of my code:

Figure 9: Logical order of the operations

The next lines only launch if send_mode and attach_mode are set to “on” which will decide if the mail will be sent with attachment or no.

IV – Conclusion

This is all for the technical guide. If you need help to setup the archiving tool, please refer to the User guide.

Contact: denis.leang@gmail.com