Intelligent System Supervision (VIMIA370)

Homework Assignment

Configuration Management

ATTENTION: Before getting started with the assignment, read the homework checklist on the website of the course. (Also do this while doing the finalizing touches on your solution.)

Task

Until now, the user accounts at our firm were not managed centrally, thus resulting various configurations with different users and groups on each machine. Before designing and setting up the new system, we have to evaluate the current status. Instead of visiting every office with pen and paper, let's develop a script that can collect and present the group and user descriptors from the powered-on, network-connected machines, in a useable form.

Let's create a ***Python 3*** script that collects the required information using ***CIM-XML***. During the development and testing we can assume that every remote machine runs **Linux**.

The name and the parameters of the script

```
get_users.py [-s SOURCE] [-o OUTPUT] [-v]
```

The script must use these named parameters. The order of the parameters should not be restricted. The parameters of the script:

- `-s SOURCE, --source SOURCE`
  `the path of the JSON source file`
- `-o OUTPUT, --output OUTPUT`
  `the path of the output directory`
- `-v, --verbose`
  `optional parameter; if set, the output files are more verbose about the users`

Example executions of the script:

```
python3 get_users.py -s machines.json -o ./results
python3 get_users.py -v -o /home/meres/Desktop/results --source net.json
```

Input file

The script consumes a JSON file containing a structure like below. It may contain more than one connection configuration. The script shall try to query the required information from every one.

```
{
  "Configurations": [
    {
      "Host": "localhost",
      "Username": "meres",
      "Password": "jelszo"
    },
    {
      "Host": "192.168.183.145",
      "Username": "meres",
      "Password": "jelszo"
    }
  ]
}
```

For the assignment, it is enough to only support the http protocol, https is not required.

(Storing the passwords in plain text files in production environments is not advised. It is now allowed, only to make the assignment easier. In production, the password should be secured with encryption or the connection should be performed with methods based on public-key cryptography.)

Output

The output consists of **multiple JSON** files in the provided output directory. For every configuration in the input file, a file identified by the **"Host"** value is to be created, e.g., "localhost.json". In these files the results of the queries are listed: for every group (**ordered** ascending by the name of the group), the users are listed (**ordered** ascending by the username of the user) in an array. The indentation of the JSON file should be **two** spaces.

By default, without the verbose flag, it only lists the names of the groups and the users in the groups. Using the verbose flag, the script produces more verbose output: instead of the username of the user, it returns a structure about the user: username, home directory location, user ID.

(Part of) the output without the verbose flag:

```
{
  "Groups": [
    {
      "Name": "meres",
      "Users": [
        "meres"
      ]
    },
    {
      "Name": "root",
      "Users": [
        "root",
        "sync"
      ]
    }
  ]
}
```

(Part of) the output with the verbose flag:

```
{
  "Groups": [
    {
      "Name": "meres",
      "Users": [
        {
          "Name": "meres",
          "Home": "/home/meres",
          "UserID": "1000"
        }
      ]
    },
    {
      "Name": "root",
      "Users": [
        {
          "Name": "root",
          "Home": "/root",
          "UserID": "0"
        },
        {
          "Name": "sync",
          "Home": "/sbin",
          "UserID": "1"
        }
      ]
    }
  ]
}
```

Further requirements

- Pay attention to error handling, e.g., always check, whether the current configuration input is readable or it throws an exception when parsed.
- Handle when the remote machine is not available. The script should not terminate when one of the remote machines cannot be reached. The other configurations should still be processed.
- Try to avoid superfluous queries, but make sure that every available information is acquired and saved in the output file.

Hints

- Use the VM provided:
    - Download from http://static.inf.mit.bme.hu/edu/irf/virtualmachines/IRF-2015-fedora-v4.7z
    - Read the readme
    - The lmiwbem Python library has already been installed on the VM. You can use the Eclipse with PyDev as for the 1st homework, the lmiwbem library can be used automatically in the PyDev (within Eclipse).
    - Start the openwsman daemon:

```
sudo systemctl start openwsmand
```

    - Start the pegasus daemon:

```
sudo systemctl start tog-pegasus.service
```

    - Add the `meres` user to the `pegasus` group:

```
sudo usermod -a -G pegasus meres
```

    - You have to do this only at the first time. If either openwsman or pegasus service is not running start them, as showed above.
- Use lmiwbem. http://phatina.github.io/python-lmiwbem/index.html Sample enumerate query:

```
import lmiwbem

hostname = 'http://localhost'
username = 'meres'
password = 'LaborImage'
cls = 'CIM_Processor'

#Connect to CIMOM.
conn = lmiwbem.WBEMConnection()
conn.connect(hostname, username, password)

# Enumerate Instances.
proc = conn.EnumerateInstances(
    cls,
    'root/cimv2',
    LocalOnly = False,
    DeepInheritance = True,
    IncludeQualifiers = True,
    IncludeClassOrigin = True,
    PropertyList = None,
)
print proc[0].classname
print proc[0].path

# Disconnect from CIMOM.
conn.disconnect()
```

- To learn more about the JSON format:
    - http://www.json.org/
    - http://en.wikipedia.org/wiki/JSON
    - http://www.w3schools.com/json/
    - https://docs.python.org/3.4/library/json.html
- Again: use Python 3.