RO Interview for SDN Developers

Background

As part of the SDN developer team, you were given the task to automate configurations on different networking devices. Most of the time, you will have to interact and gather information from other services and ensure that the workflows are correctly executed.

For this task, you were assigned to fetch data from an API endpoint, validate the content and ensure that the code is properly tested.

Data structures

Summary here (to be completed if necessary)

1. **Cluster**
   To be completed if necessary
2. **Datacenter**
   To be completed if necessary
3. **Entry**
   To be completed if necessary
4. **Network collection**
   To be completed if necessary

Project Structure

The folder structure for your assignment is described below.

```
ro_interview_assignment/
 data_structures ---> Complete assignments from section II and section III here.
    cluster.py
    datacenter.py
    entry.py
    __init__.py
    network_collection.py
 main.py ---> Complete assignments from section I here (and test your code if needed)
 response.json ---> Same response from Mock API, for better visibility
 tests ---> Your unit tests will go here (section IV)
```

Note: This structure is just an example of how the code can be structured. Feel free to make any changes to it.

TASKS

I. Data Gathering

Given the following URL: http://www.mocky.io/v2/5e539b332e00007c002dacbe complete the **get_data** function in main.py that makes a request to the given URL and returns the response as a JSON object.

Add a "retry" mechanism to this function so that if the request fails, it keeps retrying for 5 times.

II. Data Manipulation

For the next step, we want to organize the response in some data structures.

Complete the **__init__** methods for the following classes: **Entry, NetworkCollection, Cluster, Datacenter (recommended order).** The docstrings contain guidelines for what fields/data type each class contains.

III. Data Validation

We want to ensure that we only keep valid information from the response.

1. Complete the **remove_invalid_clusters** method in the **Datacenter** class. The method must remove invalid records from the **clusters** list. A cluster is valid if its name follows the given naming convention:

    - The cluster starts with the first three uppercase letters of the parent **Datacenter**, followed by a dash, followed by a number of at least one and maximum three digits.
      Use a regular expression for the requirements described above.
    - **Example:** for the 'Berlin' datacenter, the following cluster names: "BER-1", "BER-203" are considered valid, while "BER-4000" and "TEST-1" are considered invalid.
2. Complete the **remove_invalid_records** method in the **NetworkCollection** class. The method must remove invalid records from the **entries** list. An entry is considered valid if:
    - The **address** field is a valid IPv4 address
    - The **address** belongs to the IPv4 network of the parent **NetworkCollection** object.
    - **Example:** for IPv4 network "192.168.0.0/24", the entries "192.168.0.1", "192.168.0.2", "192.168.0.3", "192.168.0.4" are valid, while the rest of the entries are either invalid IPv4 addresses
      or do not belong to the IPv4 network.
3. We would like to keep our data organized for future operations. Complete the **Entry** class with the needed code to make the **sort_records** method defined in **NetworkCollection** work.
    - The **Entry** list is considered to be correctly sorted if the objects are sorted in ascending order by the **address** property.

IV. Code Testing

Write a set of unit tests for the code assignments from section III.

V. Evaluation

Your solution is going to be evaluated from different perspectives. In our review we will focus on:

- accuracy - is the result correct?
- quality - clean, tested implementation, which follows best practices (e.g: PEP8)
- performance - how does the implementation behave with a much bigger dataset?