📖 Qlearsite DevOps Challenge

Introduction

The purpose of this DevOps coding challenge is to build a script for configuration driven instance management. This is to test your ability to script infrastructure automation, for when tooling is insufficient.

We have provided an example YAML config file `instances.yaml` containing a list of instances and configuration settings which should be applied to these instances. A config file of this shape should be the only input to your script, and can be passed in as a filename argument, piped in through `stdin`, or however you like. The output of your script (`stdout`) will be `json`.

You can use any programming language, and any libraries, to solve this challenge. Our preference is Python (v2 or v3) or Groovy, but the priority is a working solution.

Challenge parts

This challenge is split into three parts. So as not to take too much of your time, you are free to complete as many parts as you would like, but you should complete at least the first part. Each part will add functionality, and extend the `json` output of the script.

1. The first operation your script should do is to read in the list of instances from a given YAML file (with the same format as `instances.yaml`), and output a JSON list of dictionaries, one for each instance, giving the instance name and hostname. This output should be "pretty-printed".

   For example, given:

   ```yaml
   ---
   - name: Test 1
     hostname: "server1.test.vpn.domain.dom"

   - name: Live 1
     hostname: "server1.live.vpn.domain.com"

   - name: Live 2
     hostname: "server2.live.vpn.domain.com"
   ```

   Your script should output:

   ```json
   [
     {
       "hostname": "server1.test.vpn.domain.com",
       "name": "Test 1"
     },
     {
       "hostname": "server1.live.vpn.domain.com",
       "name": "Live 1"
     },
     {
       "hostname": "server2.live.vpn.domain.com",
       "name": "Live 2"
     }
   ]
   ```

2. The second operation your script should do is check that the current user is able to `ssh` into these machines as the user given in `ssh_user`, and elevate themselves to the user given as `remote_user` from the provided `yaml` file. If this is possible for an entry in the `yaml` file then add `'available': true` to the `json` output, otherwise add `'available': false`. You may need to create some virtual machines to test this script.

   For example, if `ssh` access to `centos@server1.test.vpn.domain.com` and `centos@server1.live.vpn.domain.com` succeeds in the current shell, and `ubuntu@server2.live.vpn.domain.com` fails, *and* elevating to `root` succeeds on `server1.test.vpn.domain.com` and elevating to `liveusr` succeeds on `server1.live.vpn.domain.com`, then this input:

```yaml
- name: Test 1
  hostname: "server1.test.vpn.domain.com"
  ssh_user: centos
  remote_user: root

- name: Live 1
  hostname: "server1.live.vpn.domain.com"
  ssh_user: centos
  remote_user: liveusr

- name: Live 2
  hostname: "server2.live.vpn.domain.com"
  ssh_user: ubuntu
```

Should give this output:

```json
[
  {
    "available": true,
    "hostname": "server1.test.vpn.domain.com",
    "name": "Test 1"
  },
  {
    "available": true,
    "hostname": "server1.live.vpn.domain.com",
    "name": "Live 1"
  },
  {
    "available": false,
    "hostname": "server2.live.vpn.domain.com",
    "name": "Live 2"
  }
]
```

3. The third operation your script should do is create any paths and apply any file permissions given in the input YAML. All paths that have been created *or* had their permissions changed should be given in the `json` output as a list of paths.

   For example, given:

```yaml
- name: Test 1
  hostname: "server1.test.vpn.domain.com"
  ssh_user: centos
  remote_user: root
  paths:
  - path: /var/www
    owner: apache
    group: apache
    mode: u=rwx,g=rxw,o=

- name: Live 1
  hostname: "server1.live.vpn.domain.com"
  ssh_user: centos
  remote_user: root
  paths:
  - path: /etc/service1
    owner: liveusr
    group: liveusr
    mode: u=rwx,g=,o=
  - path: /etc/service2
    owner: liveusr
    group: liveusr
    mode: u=rwx,g=,o=

- name: Live 2
  hostname: "server2.live.vpn.domain.com"
  ssh_user: ubuntu
```

   This should create/update a directory at `/var/www` on `server1.test.vpn.domain.com` with the given owner, group, and permissions. It should also create/update directories at `/etc/service1` and `/etc/service2` on `server1.live.vpn.domain.com` with the given owner, group, and permissions.

Assuming that `/var/www` had its permissions changed, and `/etc/service1` was created, and nothing was changed for `/etc/service2` the script should output:

```
[
  {
    "hostname": "server1.test.vpn.domain.com",
    "name": "Test 1",
    "available": true,
    "changed": [
      "/var/www"
    ]
  },
  {
    "hostname": "server1.live.vpn.domain.com",
    "name": "Live 1",
    "available": true,
    "changed": [
      "/etc/service1"
    ]
  },
  {
    "hostname": "server2.live.vpn.domain.com",
    "name": "Live 2",
    "available": false
  }
]
```

If run a second time the script should then output:

```
[
  {
    "hostname": "server1.test.vpn.domain.com",
    "name": "Test 1",
    "available": true,
    "changed": []
  },
  {
    "hostname": "server1.live.vpn.domain.com",
    "name": "Live 1",
    "available": true,
    "changed": []
  },
  {
    "hostname": "server2.live.vpn.domain.com",
    "name": "Live 2",
    "available": false
  }
]
```

Submission instructions

Please submit an archive with your script source code, and any other resources you used to solve this problem.