NETACAD-DEVASC SKILLS-TEST
JAN 2024

Context - In this document you will experiment with skills demonstrating that you are able to solve tasks equivalent to the tasks in the hands-on labs and challenges of the DevNet Associate course.

There are practical tasks about the following topics:

1. Github **
2. Ansible ***
3. Docker ***
4. Jenkins *
5. Rest API & RESTCONF ***
6. Webex Teams API ***
7. Bash **
8. Unit testing -
9. pyATS -
10. Filtering DNAC Response Data ***

Evaluation scoring

****** 5p*
***** 4p*
**** 3p*
*** 2p*
** 1p*
- no score

Required score: 14/20

Create a **README.md** document in which you concisely document ***the tasks*** using the following structure for each task:

- Task name => name of the task
- Task preparation => what preparation is necessary to perform the task? (short)
- Task implementation => what is the way you have implemented the task?
- Task troubleshooting => what were the problems encountered?
- Task verification => proof of the quality of the result

The details of the particular tasks are explained in the following pages.

Task 1 -- GitHub Skills Test **

>Task name: GitHub

>Task Description
Manage GitHub scripts and documents

>Task Execution

1. **Preparation**
   Create a folder "Devasc_Skills" in your DEVASC virtual machine and start a git repository
2. **Scripts**
   The scripts you create in every task will be copied into the repository "Devasc_Skills"
3. **Repository**
   Make sure that your local repository is connected to an online repository on github (Devasc_Skills_YourInitals)
4. **Upload**
   After every step make sure that the local files are present on GitHub with a tag indicating the task name
5. **Screenshoots**
   Take **screenshots** indicating the success of your actions
6. **README**
   At the end finalize the README file indicating the list of tasks and your remarks

Create a **README.md** document in which you concisely document ***the tasks*** using the following structure for each task:

- Task name => name of the task
- Task preparation => what preparation is necessary to perform the task? (short)
- Task implementation => what is the way you have implemented this?
- Task troubleshooting => what were the problems encountered?
- Task verification => proof of the quality of the result

Task 2 -- Ansible Skills Test ***

>Task name: Ansible

>Task Description
Manage Ansible scripts

>Task Execution

1. Connect to your DEVASC virtual machine and start your virtual router
2. Create an Ansible playbook for your virtual router:
    a. Collect information about IOS version
    b. Collect information about L3 interfaces
    c. Configure the logging buffer with a size of 5000 bytes
3. Adapt the Ansible scripts described in the urls shown under **task source file** in such a way that it can be executed as a playbook. It may be necessary to remove **"cisco.ios"** in the module reference of the task. Also, it does not matter if some commands are deprecated. This is related to the specific Ansible version in the virtual machine.
4. Name the playbook ios_facts
5. Run the playbook in your DEVASC virtual machine
6. Take **screenshots** indicating the success of your actions, save and upload your script files

>Task source files
URL2a: Run commands on remote devices running Cisco IOS
URL2b: Collect facts from remote devices running Cisco IOS
URL2c: Logging resource module

Update the **README.md** document using the following structure for this task:

- Task name => name of the task
- Task preparation => what preparation is necessary to perform the task?
- Task implementation => what is the way you have implemented this?
- Task troubleshooting => what were the problems encountered?
- Task verification => proof of the quality of the result

Task 3 -- Docker ***

>Task name: Docker

>Task Description

Create a Docker microservice

This task tests your skill to interpret an Ansible playbook and to convert the playbook into a different deployment method, i.e a **Dockerfile**

>Task Execution

1. **(**)**
   a. Create a docker image based on the Ansible playbook shown under **Task source files below.** Only interpret the Ansible playbook and create a Dockerfile performing a similar configuration implementing the same type of service.
   b. Write a bash script to create the Apache image and to start the Apache container with the correct parameters.
2. **(***)**
   Create a **default home page** for the Apache webserver showing the following elements in a style according to your preference:
   a. Title **DevNet Associate Skills Test:Your Name**
   b. Header **DevNet Associate Skills Test**
   c. Created by: Your Name
   d. Date: Current Date and Time

   Tip: root folder for Apache2 in Docker may be different than in the playbook below
3. Take **screenshots** indicating the success of your actions and save script files and related docs

>Docker Task: Source Files

Below is a the Ansible playbook that provides the elements for creating an equivalent docker environment.

```
---
- hosts: webservers
  become: yes
  tasks:
    - name: INSTALL APACHE2
      apt:
        name: apache2
        update_cache: yes
        state: latest

    - name: ENABLED MOD_REWRITE
      apache2_module:
        name: rewrite
        state: present
      notify:
        - RESTART APACHE2

    - name: APACHE2 LISTEN ON PORT 8088
      lineinfile: dest=/etc/apache2/ports.conf regexp="^Listen 80" line="Listen 8088" state=present
      notify:
       - RESTART APACHE2

    - name: HOME PAGE
      copy:
        src: files/index.html
        dest: /var/www/html/index.html
      notify:
       - RESTART APACHE2

  handlers:
    - name: RESTART APACHE2
      service: name=apache2 state=restarted
```

Update the **README.md** document using the following structure for this task:

- Task name => name of the task
- Task preparation => what preparation is necessary to perform the task?
- Task implementation => what is the way you have implemented this?
- Task troubleshooting => what were the problems encountered?
- Task verification => proof of the quality of the result

Task 4 -- Jenkins *

>Task name: Jenkins

>Task Description
Create a CI/CD pipeline

>Task Execution

1. Create a Jenkins pipeline with three stages in which you download the necessary scripts and files from a GitHub repository and install the service from Task 3 -- Docker in a docker container. It is necessary to **test** the success of the deployment. Do *not* use a freestyle style jenkins project. Create an independent groovy script.
    a. Stage 1: Prepare
    b. Stage 2: Docker build, Docker run
    c. Stage 3: Test
2. Take **screenshots** indicating the success of your actions and save script files

Update the **README.md** document using the following structure for this task:

- Task name => name of the task
- Task preparation => what preparation is necessary to perform the task?
- Task implementation => what is the way you have implemented this?
- Task troubleshooting => what were the problems encountered?
- Task verification => proof of the quality of the result

Task 5 -- Rest API & Restconf ***

>Task name: restconf

>Task Description
Create a Python script based on curl command examples
Use your virtual DEVASC virtual machine as well as your virtual CSR1Kv router

>Task Execution

1. Test the curl instructions presented from page 8 to 10 in the webpage **RESTCONF Protocol** below
2. It is sufficient to transform three curl commands in to restconf.
3. Transform the curl commands into Python code. *In case of errors, use a quick fix to activate "logging monitor" using the cisco ios cli of your virtual router*
4. Take **screenshots** indicating the success of your actions
5. Save the Python code on Github

>Task source files
RESTCONF Protocol & curl
URL: RESTCONF Programmable Interface
More examples of curl and python scripting
RESTCONF Tutorial - Everything you need to know about RESTCONF
URL: Everything you need to know about RESTCONF

Update the **README.md** document using the following structure for this task:

- Task name => name of the task
- Task preparation => what preparation is necessary to perform the task?
- Task implementation => what is the way you have implemented this? (python code)
- Task troubleshooting => what were the problems encountered?
- Task verification => proof of the quality of the result

>Restconf: Task Curl Examples
RESTCONF Protocol & curl -> request & response

-1- OPTIONS

Command/ request

```
curl -i -k -X "OPTIONS" \
"https://192.168.56.107:443/restconf/data/Cisco-IOS-XE-native:native/logging/monitor/severity"
\
-H 'Accept: application/yang-data+json' \
-u 'cisco:cisco123!'
```

Output/ response

```
HTTP/1.1 200 OK
Server: nginx
Date: Sun, 22 Nov 2020 20:19:35 GMT
Content-Type: text/html
Content-Length: 0
Connection: keep-alive
Allow: DELETE, GET, HEAD, PATCH, POST, PUT, OPTIONS
Cache-Control: private, no-cache, must-revalidate, proxy-revalidate
Accept-Patch: application/yang-data+xml, application/yang-data+json
Pragma: no-cache
```

-2- POST

It may be necessary to activate logging monitor using the ios cli before issuing this curl command

Command/ request

```
curl -i -k -X "POST"
"https://192.168.56.107:443/restconf/data/Cisco-IOS-XE-native:native/logging/monitor" \
-H 'Content-Type: application/yang-data+json' \
-H 'Accept: application/yang-data+json' \
-u 'cisco:cisco123!' \
-d $'{
       "severity": "alerts"
}'
```

Output/ response

```
HTTP/1.1 201 Created
Server: nginx
Date: Sun, 22 Nov 2020 11:20:37 GMT
Content-Type: text/html
Content-Length: 0
Location: https://192.168.56.107/restconf/data/Cisco-IOS-XE-native:native/logging/monitor/severity
Connection: keep-alive
Last-Modified: Sun, 22 Nov 2020 11:20:36 GMT
Cache-Control: private, no-cache, must-revalidate, proxy-revalidate
Etag: 1606-44036-936119
Pragma: no-cache
```

-3- PUT

Command/ request

```
curl -i -k -X "PUT"
"https://192.168.56.107:443/restconf/data/Cisco-IOS-XE-native:native/logging/monitor/severity"
\
-H 'Content-Type: application/yang-data+json' \
-H 'Accept: application/yang-data+json' \
-u 'cisco:cisco123!' \
-d $'{
       "severity": "warnings"
}'
```

Output/ response

```
HTTP/1.1 204 No Content
Server: nginx
Date: Sun, 22 Nov 2020 11:21:29 GMT
Content-Type: text/html
Content-Length: 0
Connection: keep-alive
Last-Modified: Sun, 22 Nov 2020 11:21:28 GMT
Cache-Control: private, no-cache, must-revalidate, proxy-revalidate
Etag: 1606-44088-916333
Pragma: no-cache
```

-4- PATCH

Command/ request

```
curl -i -k -X "PATCH" "https://192.168.56.107:443/restconf/data/Cisco-IOS-XE-native:native" \
-H 'Content-Type: application/yang-data+json' \
-H 'Accept: application/yang-data+json' \
-u 'cisco:cisco123!' \
-d $'{
"native": {
"logging": {
"monitor": {
"severity": "informational"
}
}
}
}'
```

Output/ response

```
HTTP/1.1 204 No Content
Server: nginx
Date: Sun, 22 Nov 2020 20:05:15 GMT
Content-Type: text/html
Content-Length: 0
Connection: keep-alive
Last-Modified: Sun, 22 Nov 2020 20:05:15 GMT
Cache-Control: private, no-cache, must-revalidate, proxy-revalidate
Etag: 1606-75515-266750
Pragma: no-cache
```

-5- OTHER GET EXAMPLE

command/ request

```
curl -H "Accept: application/yang-data+json" -k -X "GET"
"https://192.168.56.107/restconf/data/Cisco-IOS-XE-native:native/logging/monitor/severity" -u
'cisco:cisco123!'
```

output/ response

```
{
  "Cisco-IOS-XE-native:severity": "alerts"
}
```

-6- DELETE EXAMPLE

command/ request

```
curl -H "Accept: application/yang-data+json" -k -X "DELETE"
"https://192.168.56.107/restconf/data/Cisco-IOS-XE-native:native/logging/monitor/severity" -u
'cisco:cisco123!'
```

Task 6 -- Webex Teams API ***

>Task name: Webex

>Task Description
Execute Webex Teams API calls using a Python script

>Task Execution

1. Create or adapt an existing Python script to **create a Webex Teams space** with the name "devasc_skills_your_initials" with yourself and <yvan.rooseleer@biasc.be> as the members
2. Publish the url of your **github remote repository** (task 1 of the skills exam) in this new Webex Teams space
3. Create (or adapt) an existing Python script that **sends a message** "Here are my **screenshots** of devasc skills-based exam" to the new space
4. Upload one **screenshot** of every task to the Webex Teams Space

Update the **README.md** document using the following structure for this task:

- Task name => name of the task
- Task preparation => what preparation is necessary to perform the task?
- Task implementation => what is the way you have implemented this? (python code)
- Task troubleshooting => what were the problems encountered?
- Task verification => proof of the quality of the result

Task 7 -- Bash **

>Task name: Bash
>Task Desciption
Translate the given Python script into an equivalent bash script.
URL of the input file: restconf_put_get_interface.py
Tip: For the REST API call it is possible to use curl.

>Task Execution

1. **Preparation**
   Explore the given Python script and run the Python script to observe the output.
   Prepare a Linux environment in which you can create a bash script that is able to connect to the given host.

2. **Bash script characteristics**
   a. Name of the bash script: **restconf_api.sh**
   b. *Make the script executable*
   c. The bash script needs make two API calls and output the requested information
   d. After creating and testing the bash script, copy and upload the bash script to your **GitHub** repo, also copy and upload the output file to your GitHub repo
3. **Bash script output**
   Output needs to be redirected to a file named **check_restconf_api.txt**
   The bash script needs to create the following output
   a. First line of output: today's date
   b. Next line of output: 'START REST API CALL'
   c. ============
   d. Next line of output: 'FIRST API CALL'
   e. ============
   f. Status Code: (realtime output)
   g. ============
   h. Next line of output: 'SECOND API CALL'
   i. ============
   j. Status Code: (realtime output)
   k. Interfaces: (realtime output)
   l. Last line of output: 'END REST API CALL'
4. **Screenshot**
   Take **screenshots** indicating the success of your actions, add information to the **README**.md, save and **upload** your resulting script file to GitHub. Upload a screenshot of a relevant part of the bash script output to the Webex Teams Space

Update the **README.md** document using the following structure for this task:

- Task name => name of the task
- Task preparation => what preparation is necessary to perform the task?
- Task implementation => what is the way you have implemented this? (bash script)
- Task troubleshooting => what were the problems encountered?
- Task verification => proof of the quality of the result

Task 8 -- unit testing

>Task name:
Unit testing

>Task description
Create a unittest script in Python that asserts the output of all the functions in a given Python library

>Task execution

=> You do not have to complete a task about unit testing for this skills based assessment.

Task 9 -- pyATS

>Task name:
pyATS/Genie

>Task description

1. Use an automated tool to gather facts about hostnames, ip addresses, model and serial number of the infrastructure devices.
2. Upload the results to the GitHub repository

>Task execution

=> You do not have to complete a task about pyATS for this skills based assessment.

Task 10 -- FILTERING DNAC RESPONSE DATA ***

>Task name: DNAC

>Task Description

Adapt the Python code in the script on the next page in order to obtain a working application that is able to provide the output shown in the output example OUTPUT TASK 10.

>Task Output Example

Adapt the given Python script in such a wat that it produces the following output. *Not all records are shown in the output.*

DNAC OUTPUT TASK 10

```
Current date and time:
2021-10-29 09:42:23.613764
===
Hostname: c3504.abc.inc
Family: Wireless Controller
MAC: ac:4a:56:6c:7c:00
IP: 10.10.20.51
Software version: 8.5.140.0
Reachability: Reachable
===
Hostname: leaf1.abc.inc
Family: Switches and Hubs
MAC: 84:8a:8d:05:76:00
IP: 10.10.20.81
Software version: 17.3.3
Reachability: Reachable
```

>Task Execution

1. Use your virtual DEVASC virtual machine using a connection to the internet.
2. Copy the sample script on the next page to your Python execution environment. *Mind the indentations.*
3. Adapt the script in such a way that the above **OUTPUT TASK 10** will be produced.
4. Replace all elements showing **XXXXXXXX** with suitable parameters, variables, keys, names or code.
5. Take **screenshots** indicating the success of your actions, add information to the **README**.md, save and **upload** your resulting script file to GitHub.

DNAC SAMPLE SCRIPT -- WITH 10 MISSING ELEMENTS

```
import XXXXAXXXX
import datetime
import json
requests.packages.XXXXBXXXX.disable_warnings()

DNAC_scheme = 'https://'
DNAC_authority='sandboxdnac.cisco.com'
DNAC_port=':443'
DNAC_path_token='/dna/system/api/v1/auth/token'
DNAC_path='/dna/intent/api/v1/network-device'
DNAC_user = 'devnetuser'
DNAC_psw = 'Cisco123!'

# DATE AND TIME
print ("Current date and time: ")
print(datetime.XXXXCXXXX)

# TOKEN REQUEST
token_req_url = DNAC_scheme+DNAC_authority+DNAC_path_token
auth = (DNAC_user, DNAC_psw)
req = requests.request('XXXXDXXXX', token_req_url, auth=auth, verify=False)
token = req.json()['Token']

# INVENTORY REQUEST
req_url = DNAC_scheme+DNAC_authority+DNAC_port+DNAC_path
headers = {'X-AUTH-TOKEN': XXXXEXXXX}
resp_devices = requests.request('GET', req_url, headers=XXXXFXXXX, verify=False)
resp_devices_json = resp_devices.json()

# FILTER RESPONSE DATA
for device in resp_devices_json['response']:
    if device['type'] != None:
        print('===')
        print('Hostname: ' + device['hostname'])
        print('Family  : ' + device['XXXXGXXXX'])
        print('MAC: '+ device['XXXXHXXXX'])
        print('IP: ' + device['XXXXIXXXX'])
        print('Software version: ' + device['softwareVersion'])
        print('Reachability: ' + device['XXXXJXXXX'])
```

Update the **README.md** document using the following structure for this task:

- Task name => name of the task
- Task preparation => what preparation is necessary to perform the task?
- Task implementation => what is the way you have implemented this?
- Task troubleshooting => what were the problems encountered?
- Task verification => proof of the quality of the result