Policy Management with Ansible

Introduction

In the previous lab, we learned how to perform basic Gaia configurations and run the first time wizard. In this lab, we will configure the management API, add the remote gateway and create the security policy.

Exercise 1: Enable API Access

The Management API accept local connections only (from the mgmt. server itself). In this first exercise, we will create a playbook to allow access on all ports via Ansible. Once the mgmt. API is exposed and we have access, we can continue to add the gateway and create the policy.

1. Create a new playbook and name it configure_mgmt_api.yml. The first task is to use the mgmt_cli command to enable API access on all IP addresses.

```
---
- name: Configure the manageemnt API access
  hosts: checkpoint_mgmt
  gather_facts: no

  tasks:

    - name: Set API server to listen to any source
      shell: source /etc/bashrc ; mgmt_cli -r true -d "System Data" set api-settings accepted-api-calls-from "All IP addresses"
```

   - Note that we are using the module shell. It works almost as the command module but it runs the command through a shell.
   - For more details, refer to: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/shell_module.html

2. For the changes above to take effect, we need to restart the API using the command api restart. We will use the command module for this task. Make sure your playbook has the following steps:


```yaml
---
- name: Configure the manageemnt API access
  hosts: checkpoint_mgmt
  gather_facts: no

  tasks:

    - name: Set API server to listen to any source
      shell: source /etc/bashrc ; mgmt_cli -r true -d "System Data" set api-settings accepted-api-calls-from "All IP addresses"

    - name: restart the API
      command: api restart

    - name: Check the API status
      command: api status
      register: api_status

    - name: show API state
      debug:
        msg: '{{ api_status.stdout_lines }}'
...
```

3. Run the playbook and make sure the results shows access to all IP addresses.

```
TASK [show API state] ***********************************************************************************************
ok: [203.0.113.80] => {
    "msg": [
        "",
        "API Settings:",
        "--------------------",
        "Accessibility:                    Require all granted",
        "Automatic Start:                  Enabled",
        "",
        "Processes:",
        "",
        "Name      State     PID       More Information",
        "-----------------------------------------------",
        "API       Started   6232      ",
        "CPM       Started   6232      Check Point Security Management Server is running and ready",
        "FWM       Started   5111      ",
        "APACHE    Started   4505      ",
        "",
```

Exercise 2: Using the Check Point Mgmt. Collection

In Lab 3, we installed the Gaia collection from Ansible galaxy. In this exercise, we will install the Management collection, which allows us to perform management task exposed via the management API.

For more details about the Check Point management collection for ansible, refer to:

https://galaxy.ansible.com/check_point/mgmt

1. Us the following command to install the management collection: ansible-galaxy collection install check_point.mgmt

```
administrator@orchestrator:~/playbooks$ ansible-galaxy collection install check_point.mgmt
Process install dependency map
Starting collection install process
Installing 'check_point.mgmt:2.0.0' to '/home/administrator/.ansible/collections/ansible_collections/check_point/mgmt'
```

2. Review the inventory file. Note that the variable ansible_network_os can be written as: ansible_network_os=checkpoint or ansible_network_os= check_point.mgmt.checkpoint while we must use ansible_network_os= check_point.gaia.checkpoint for the Gaia collection and the use of ansible_network_os=checkpoint will be invalid.

```
[checkpoint:children]
checkpoint_mgmt
checkpoint_gw

[checkpoint_mgmt]
203.0.113.80 ansible_user=admin ansible_password=vpn123

[checkpoint_gw]
203.0.113.81 ansible_user=admin ansible_password=vpn123

[checkpoint_external_gw]
203.0.113.40 ansible_user=admin ansible_password=vpn123


[checkpoint:vars]
ansible_httpapi_use_ssl=True
ansible_httpapi_validate_certs=False
ansible_network_os=checkpoint
# ansible_python_interpreter=/opt/CPsuite-R81/fw1/Python/bin/python3.7
```

3. Create a playbook called get_simple_gateways.yml to collect details regarding the existing gateways. We are using the module cp_mgmt_simple_gateway_facts. Sicne we are using the API, make sure yoyu have the keywork connection: httpapi configured.

```
---
- name: get existing Gateways
  hosts: checkpoint_mgmt
  connection: httpapi
  gather_facts: no

  tasks:
    - name: get existing Gateway objects
      check_point.mgmt.cp_mgmt_simple_gateway_facts :
      register: existing_gateways

    - name: print gateways from the Mgmt
      debug:
        msg: '{{ existing_gateways }}'
...
```


4. Run the task and notice that we have a single gateway named “Internal-GW”.

```
TASK [get existing Gateway objects] ************************************************
ok: [203.0.113.80]

TASK [print gateways from the Mgmt] ************************************************
ok: [203.0.113.80] => {
    "msg": {
        "ansible_facts": {
            "discovered_interpreter_python": "/usr/bin/python3",
            "simple-gateways": {
                "from": 1,
                "objects": [
                    {
                        "domain": {
                            "domain-type": "domain",
                            "name": "SMC User",
                            "uid": "41e821a0-3720-11e3-aa6e-0800200c9fde"
                        },
                        "name": "Internal-GW",
                        "type": "simple-gateway",
                        "uid": "073805ec-7b92-4392-994e-79eee4639ae0"
                    }
                ],
                "to": 1,
                "total": 1
```

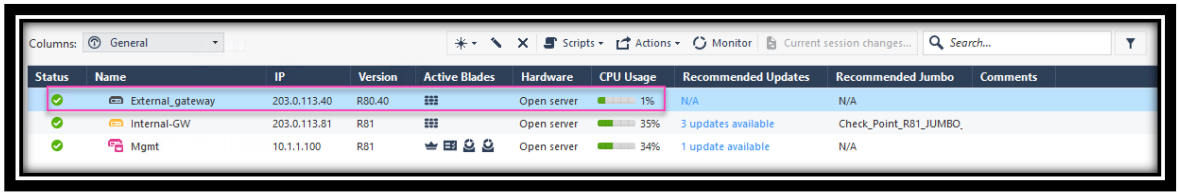
5. Create a new playbook called create_simple_gateway and add the following details to create the Gateway object for the external gateway and establish SIC.

```yaml
---
- name: Creating a simple Gateway
  connection: httpapi
  hosts: checkpoint_mgmt
  gather_facts: no
  name: "Create simple Gateway"

  tasks:

    - name: "create simple Gateway"
      check_point.mgmt.cp_mgmt_simple_gateway:
        firewall: true
        gateway_version: R80.40
        ip_address: "203.0.113.40"
        name: External_gateway
        one_time_password: "vpn123"
        state: present
        auto_publish_session: yes
      register: gateway_created

    - name: show GW creation results
      debug:
        msg: "{{ gateway_created }}"
...
```

6. Run the task and review the results.

```
TASK [show GW creation results] ****************************************************
ok: [203.0.113.80] => {
    "msg": {
        "add-simple-gateway": {
            "anti-bot": false,
            "anti-virus": false,
            "application-control": false,
            "color": "black",
            "comments": "",
            "content-awareness": false,
            "domain": {
                "domain-type": "domain",
                "name": "SMC User",
                "uid": "41e821a0-3720-11e3-aa6e-0800200c9fde"
```


7. Login to SmartConsole and verify that the new gateway is present and configured correctly.



- For more details on the variable and options, Refer to https://docs.ansible.com/ansible/latest/collections/check_point/mgmt/cp_mgmt_simple_gateway_module.html

Exercise 3: Managing the Security Policy with Ansible

In this exercise, we will create a new policy package and create simple rules for the new gateway we just added.

1. Create a new playbook called create_policy_package.yml and add the steps to create a policy package.

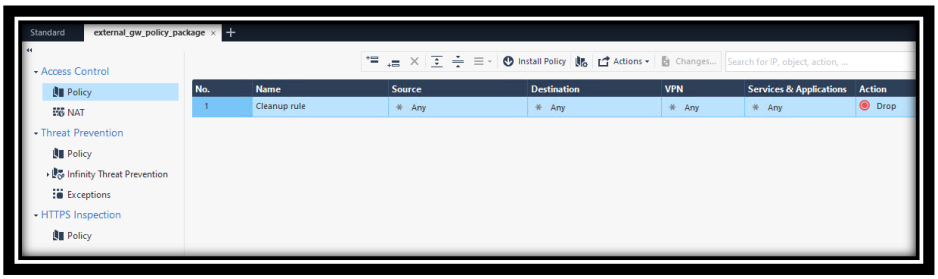
```yaml
---
- name: Creating a policy package
  connection: httpapi
  hosts: checkpoint_mgmt
  gather_facts: no

  tasks:

    - name: "create a policy package"
      check_point.mgmt.cp_mgmt_package:
        name: "external_gw_policy_package"
        color: "cyan"
        access: "true"
        threat_prevention: "true"
        state: present
        auto_publish_session: yes
      register: policy_package_results

    - name: showpolicy package results
      debug:
        msg: "{{ policy_package_results }}"
...
```

2. Run the playbook and validate the results.

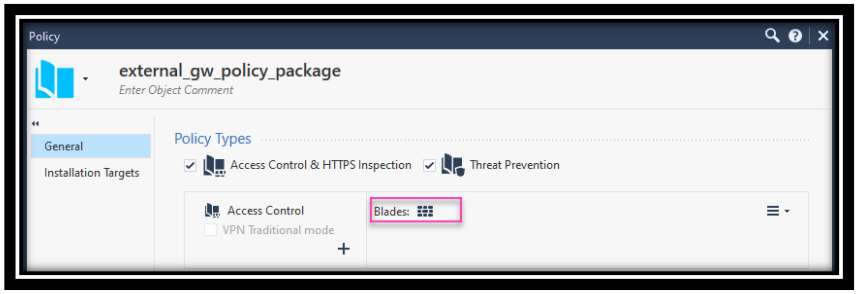
```
"name": "external_gw_policy_package Network",
"type": "access-layer",
"uid": "ebcdb1ee-f90d-4f29-8c85-5c252348c4ec"
```


3. Open SmartConsole and validate that the new policy package exists.

4. Open the policy package, review the existing layers and default rules.



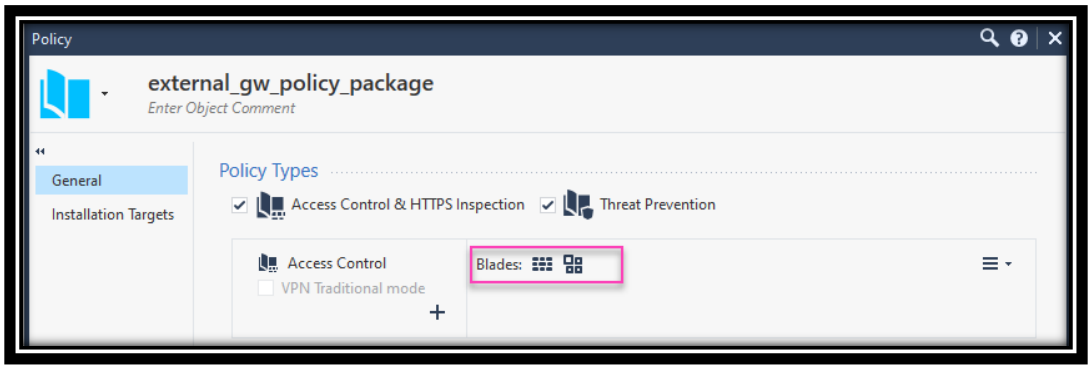
5. Review the access layer and notice that only the firewall blade is enabled by default.



6. Create a playbook called “policy_layer.yml” to enable application control and URLF on the existing layer.

```
---
- name: Rule base playbook
  connection: httpapi
  hosts: checkpoint_mgmt
  gather_facts: no

  tasks:

    - name: Set Access Layer for URLF and APPC
      cp_mgmt_access_layer:
        applications_and_url_filtering: true
        firewall: true
        name: "external_gw_policy_package Network"
        state: present
        auto_publish_session: yes
...
```

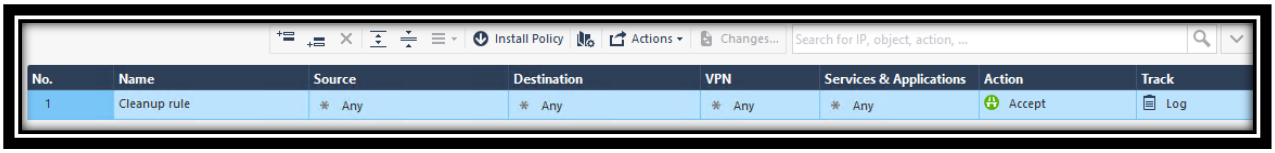

7. Run the playbook and verify that the layer was configured correctly.



8. We will now edit the default Cleanup rule. Create a new playbook called rulebase.yml and create a rule representing the existing Cleanup rule but change the action to accept and the track type to log.

```yaml
---
- name: Rule base playbook
  connection: httpapi
  hosts: checkpoint_mgmt
  gather_facts: no

  tasks:

    - name: edit the clean up rule.
      check_point.mgmt.cp_mgmt_access_rule:
        layer: "external_gw_policy_package Network"
        name: "Cleanup rule"
        position: "1"
        source:
          - "Any"
        service:
          - Any
        track:
          type: "Log"
        action: Accept
        auto_publish_session: yes
...
```

9. Run the playbook and review the changes in SmartConsole.



10. Create a rule on top to allow traffic from the InternalZone to access the ExternalZone using HTTP and HTTPS services and set the log type to “Extended Log”.


```yaml
---
- name: Rule base playbook
  connection: httpapi
  hosts: checkpoint_mgmt
  gather_facts: no

  tasks:


    - name: internet rule.
      check_point.mgmt.cp_mgmt_access_rule:
        layer: "external_gw_policy_package Network"
        name: "Internet Access"
        position: "1"
        source:
          - "InternalZone"
        destination:
          - "ExternalZone"
        service:
          - http
          - https
        track:
          type: "Extended Log"
        action: Accept
        auto_publish_session: yes

    - name: edit the clean up rule.
      check_point.mgmt.cp_mgmt_access_rule:
        layer: "external_gw_policy_package Network"
        name: "Cleanup rule"
        position: "2"
        source:
          - "Any"
        service:
          - Any
        track:
          type: "Log"
        action: Accept
        auto_publish_session: yes
...
```

11. Run the playbook and validate that the rule was created as expected.

No.	Name	Source	Destination	VPN	Services & Applications	Action	Track
1	Internet Access	InternalZone	ExternalZone	Any	http https	Accept	Extended Log Accounting
2	Cleanup rule	Any	Any	Any	Any	Accept	Log

12. Notice that we are still using preexisting objects. Create a new playbook called network_objects.yml and add a host object representing the public DNS server (8.8.8.8) and the external subnet (203.0.113.0/24).

```yaml
---
- name: Network Objects playbook
  connection: httpapi
  hosts: checkpoint_mgmt
  gather_facts: no

  tasks:
    - name:  Create External Network
      cp_mgmt_network:
        name: "External_Net"
        subnet: "203.0.113.0"
        mask_length: "24"
        state: present
        color: blue
        auto_publish_session: yes

    - name:  Create public DNS object
      cp_mgmt_host:
        name: "External_DNS"
        ip_address: 8.8.8.8
        color: red
        auto_publish_session: yes
...
```


13. Run the playbook and confirm the two objects were created as expected.

14. Edit the rulebase.yml playbook and add a new rule to allow access from the external network to the public DNS server. Change the rule position accordingly.

```yaml
- name: DNS rule.
  check_point.mgmt.cp_mgmt_access_rule:
    layer: "external_gw_policy_package Network"
    name: "DNS rule"
    position: 2
    source:
      - "External_Net"
    destination:
      - "External_DNS"
    service:
      - dns
    action: Accept
    auto_publish_session: yes
```

15. Run the rulebase.yml playbook and review the rule base.

No.	Name	Source	Destination	VPN	Services & Applications	Action	Track
1	Internet Access	InternalZone	ExternalZone	Any	http https	Accept	Extended Log Accounting
2	DNS rule	External_Net	External_DNS	Any	dns	Accept	None
3	Cleanup rule	Any	Any	Any	Any	Accept	Log

- Note that if the track option is not specified, the default is None.

Exercise 3: Organizing playbooks

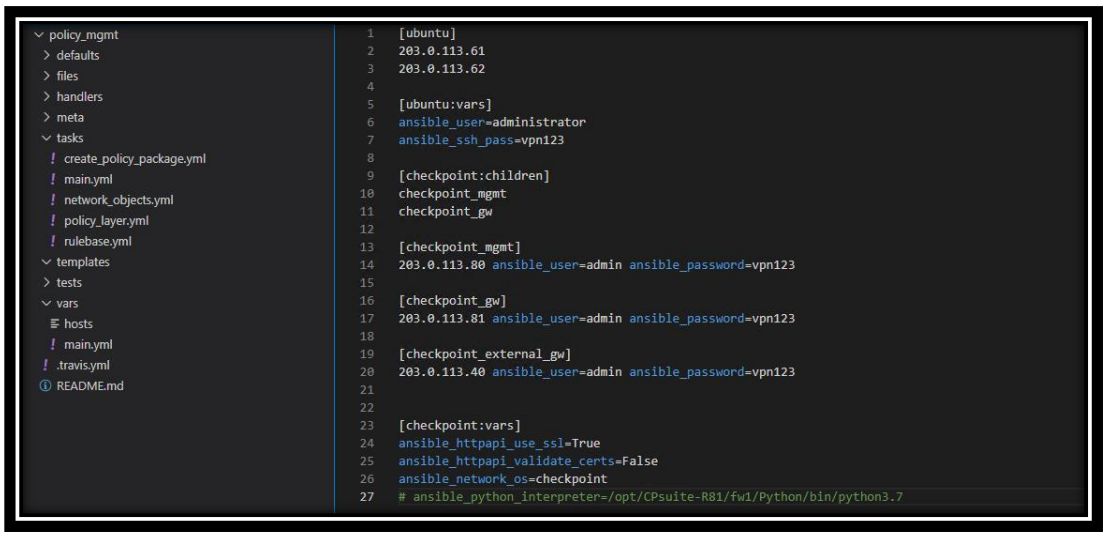
In the previous exercises, we organized the playbook based on the function they performs such as creating rule base or adding rules. We can organize in Roles which enables the reuse and sharing of Ansible playbooks.

1. From the current directory, use the command ansible-galaxy init policy_mgmt to create a role to organize all the playbooks we created for the security policy tasks.

```
administrator@orchestrator:~/playbooks$ ansible-galaxy init policy_mgmt
- Role policy_mgmt was created successfully
administrator@orchestrator:~/playbooks$
```


2. Notice that the command created a structured unit that can be used to organize the playbooks we created.

3. Move the hosts file under vars and the rest of the playbook under tasks. You can delete the main.yml file under the vars directory

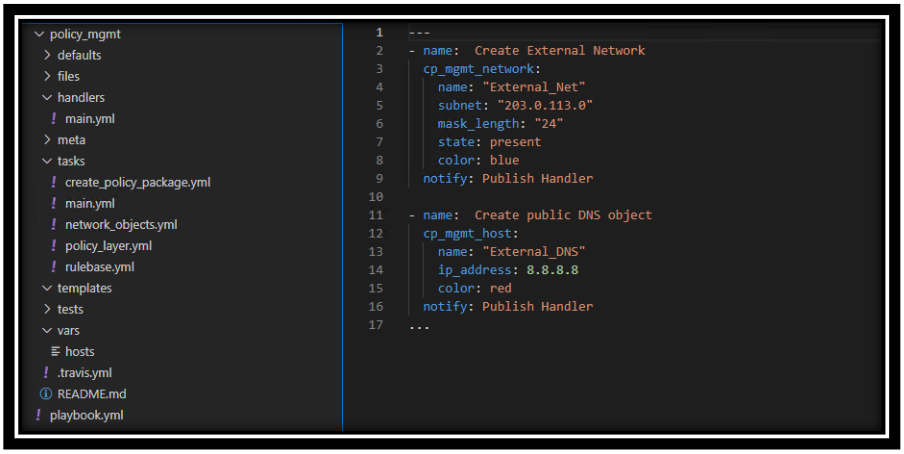


4. Edit the main.yml file inside the handlers directory and add the publish handler which we will notify to publish the session.

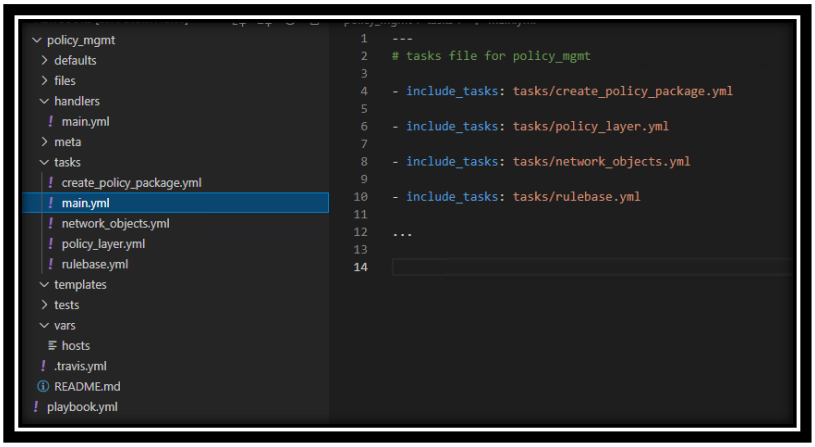
5. Edit our playbooks and remove all the details except the tasks (no indentations). Make sure to remove the auto_publish_session and replace it with the notify call to our handler (indentations matter!).

```yaml
---
- name: "create a policy package"
  check_point.mgmt.cp_mgmt_package:
    name: "external_gw_policy_package"
    color: "cyan"
    access: "true"
    threat_prevention: "true"
    state: present
  register: policy_package_results
  notify: Publish Handler

- name: showpolicy package results
  debug:
    msg: "{{ policy_package_results }}"
...
```

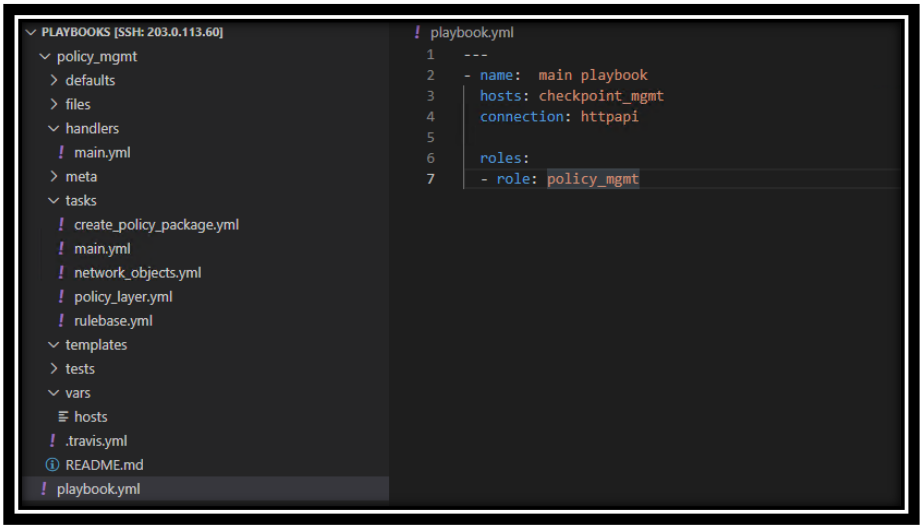
6. Move all the playbooks into the tasks folder.



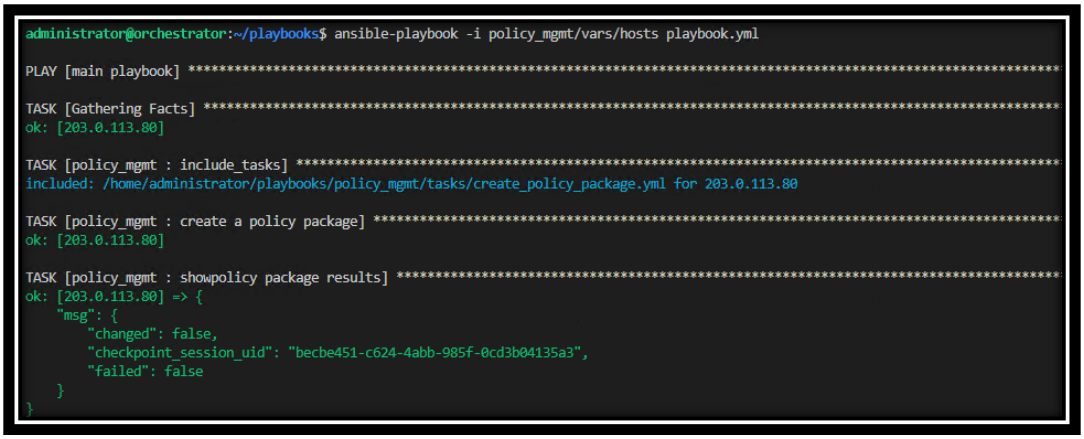
7. Now that we have everything organized, edit the main.yml file and import the other tasks.



8. Create a new playbook in the parent directory /home/administrator/playbooks and call the role we just created “policy_mgmt”.



9. Run the playbook and provide the relative location for the inventory file.



- To summarize, we have a list of tasks in separate plays inside the tasks folder under the new ‘policy_mgmt’ Role we created.
- The tasks are imported into the main.yml play inside the same folder. The main.yml will be called by default when we call the Role from playbook we created in the parent directory.
- The plays will call the publish handler we created under the handlers fiolder inside the Role.
- When we use the role in the new playbook, all the required details already exist in the Role itself and will be performed as expected.

10. Add another handler to install the access policy, this handler will be called by the publish handler after we publish the session.

- Note that we are adding the Gateway name as a variable to be provided and we also provided a condition to install policy only if the variable No_Policy_Install is not defined.

11. Create a third handler to install the Threat Prevention policy after the access policy is installed (notify the new handler by the handler installing the access policy).

End of Lab4

