R&Co demo

Demo scope

1. The capability of Ansible of using the same Playbook to create/manage a set of machines, as well as to be run in check-mode to show drifts. Leveraging the JSON callback mechanism and a script the check-mode run should create a visual report in HTML format. This should show the Initial configuration(Job template), a Scheduled DryRun, the remediation of the configuration drift based on DryRun result or exempt(considering the exemptions and changes done manually into the system).
2. Ansible Tower should run a scheduled Playbook containing a set of security checks. Exception handling for compliance should not be local to inventory or host or group vars. Show a process like the following:
    1. Users update exception in CMDB
    2. Ansible fetch the exception from CMDB
    3. Ansible Job
        1. Compliance management based on exceptions.
        2. For every host that fails one or more security checks or drifted from original configuration, a ticket in ServiceNow should be created containing which security check failed. eg: if all IIS web servers should be configured to only serve data via HTTPS. If there's some IIS servers that HAVE to use HTTP then they would be an exception and get slightly different config to "normal" IIS servers

Ansible Inventory

In this repository there is the following Ansible Inventory Plugin configuration

- *servicenow.yaml*
    - This is a YAML file containing the information needed by the ServiceNow Ansible Inventory Plugin to retrieve the inventory from ServiceNow

Ansible Modules

In this repository there are the following Ansible Module that are needed for the demo:

- *beautify_ansible_job_output.py*

  - This Ansible Module will take as input an Ansible Tower Job ID and output the steps that changed grouped by host
  - Input
    * `tower_base_url`: the Ansible Tower URL (ie: https://192.168.1.10)
    * `tower_username`: the Username of the designed user
    * `tower_password`: the Password of the designed user
    * `tower_job_id`: the Ansible Tower Job ID of the target Job
  - Output
    * `value`: a structure with the structure map[hostname][]changed_steps
  - This module requires an Ansible Tower User with enough capabilities to be able to read the target Ansible Tower Job output

- *tower_previous_workflow_job.py*
  - This Ansible Module will take as input an Ansible Tower Job ID that is part of an Ansible Tower Workflow and output the ID of the previous Ansible Tower Job in the same Ansible Tower Workflow
  - Input
    * `tower_base_url`: the Ansible Tower URL (ie: https://192.168.1.10)
    * `tower_username`: the Username of the designed user
    * `tower_password`: the Password of the designed user
    * `tower_job_id`: the Ansible Tower Job ID of the target Job
  - Output
    * `job_id`: the ID of the previous Ansible Tower Job in the same Ansible Tower Workflow
  - This module requires an Ansible Tower User with enough capabilities to be able to read the given Ansible Tower Job details as well as the details of the Ansible Tower Workflow

Ansible Playbooks

In this repository there are the following Ansible Playbooks that compose the demo:

- *win-playbook.yaml*
  - This Ansible Playbook can be run in "real-mode" or in "check-mode"
  - This is the only Ansible Playbooks that has customer-specific logic
  - This Ansible Playbook is composed by the following steps:
    1. Install IIS if it was not already installed
    2. Disable HTTP binding if the `iis_https_exception` is set to `false`
    3. Enables HTTPS binding if the `iis_https_exception` is set to `false`

4. Disables HTTPS binding if the `iis_https_exception` is set to `true`
5. Enables HTTP binding if the `iis_https_exception` is set to `true`

- The reason why we have both steps 2,3 and 4,5 is because we want Ansible to be able to apply and rollback the exception. If the requirement is that Ansible should only be able to enforce the standard state and not to enforce the exception, step 4 and 5 could be omitted.

- *html-report.yaml*
  - This Ansible Playbook fetches the output of an Ansible Tower Job (usually run in check-mode), transforms it in HTML (making it more easily readable), and sends it via email
  - This Ansible Playbook is composed by the following steps:
    1. Get the previous job in the current workflow
    2. Change the provided output to make it easily parsable by Jinja2
    3. Prepare the report leveraging the *html-report.j2* Jinja2 template
    4. Send the report via email
  - Ansible Tower is not designed to host HTML reports, so saving it locally and displaying it using a local web server is not a suggested way
  - In case the email is not the best way to share the HTML document, this might be managed in many different ways by changing the last step of the Ansible Playbook. A non-exhaustive list of possible alternative ways to manage it are:
    * SSH
    * FTP
    * HTTP(S)

- *snow-ticket.yaml*
  - This Ansible Playbook fetches the output of an Ansible Tower Job (usually run in check-mode) and opens a ServiceNow Incident for each node that was not in the correct state
  - This Ansible Playbook is composed by the following steps:
    1. Get the previous job in the current workflow
    2. Change the provided output to make it easily parsable by Jinja2
    3. Create one ServiceNow incident per affected host
  - Additional data can be passed to ServiceNow by leveraging the `data` field that the Ansible module encodes to JSON and sends to ServiceNow

Ansible Tower configurations

Credential Types

- service_now
  - name: Service Now
  - input_configuration:

```
fields:
  - id: SN_USERNAME
      type: string
      label: Username
  - id: SN_PASSWORD
      type: string
      label: Password
      secret: true
  - id: SN_INSTANCE
      type: string
      label: Snow Instance
required:
  - SN_USERNAME
  - SN_PASSWORD
  - SN_INSTANCE
```

  - injector_configuration:

```
env:
  SN_INSTANCE: '{{ SN_INSTANCE }}'
  SN_PASSWORD: '{{ SN_PASSWORD }}'
  SN_USERNAME: '{{ SN_USERNAME }}'
```

- ansible_tower
  - name: Ansible Tower (custom type)
  - input_configuration:

```
fields:
  - id: TOWER_BASE_URL
      type: string
      label: Tower base URL
  - id: TOWER_USERNAME
      type: string
      label: Tower username
      secret: true
  - id: TOWER_PASSWORD
      type: string
      label: Tower password
```

```
required:
  - TOWER_BASE_URL
  - TOWER_USERNAME
  - TOWER_PASSWORD
```

  - injector_configuration:

```
env:
  tower_base_url: '{{ TOWER_BASE_URL }}'
  tower_username: '{{ TOWER_USERNAME }}'
  tower_password: '{{ TOWER_PASSWORD }}'
```

- smtp
  - name: SMTP
  - input_configuration:

```
fields:
  - id: SMTP_HOST
      type: string
      label: SMTP hostname
  - id: SMTP_PORT
      type: integer
      label: SMTP port
      secret: true
  - id: SMTP_USERNAME
      type: string
      label: SMTP username
  - id: SMTP_PASSWORD
      type: string
      label: SMTP password
required:
  - SMTP_HOST
  - SMTP_PORT
  - SMTP_USERNAME
  - SMTP_PASSWORD
```

  - injector_configuration:

```
env:
  smtp_host: '{{ SMTP_HOST }}'
  smtp_port: '{{ SMTP_PORT }}'
  smtp_username: '{{ SMTP_USERNAME }}'
  smtp_password: '{{ SMTP_PASSWORD }}'
```

Credentials

For this demo to work, the following credentials need to be prepared on Ansible Tower:

- github
  - name: github
  - credential type: Github Personal Access Token
  - token: 7cc0d82072d8ba4dee46c0d4941d3698c4aa5f62
- winrm/ssh credentials to access the Windows Machines
  - name: winMachine
  - type: machine
- service_now:
  - name: servicenow
  - type: servicenow
  - username: *[[set the right one]]*
  - password: *[[set the right one]]*
  - instance: rcodev.service-now.com
- AnsibleTowerCustom
  - name: ansible_tower (custom type)
  - type: Ansible Tower
  - Tower base URL: *[[set the right one]]*
  - Tower username: *[[set the right one]]*
  - Tower password: *[[set the right one]]*
- smtp
  - name: smtp
  - type: smtp
  - smtp host: *[[set the right one]]*
  - smtp port: *[[set the right one]]*
  - smtp username: *[[set the right one]]*
  - smtp password: *[[set the right one]]*

Projects

For this demo to work, the following project needs to be prepared on Ansible Tower:

- randco
  - name: randco
  - scm type: git
  - scm url: https://github.com/Fale/ansible-hcl-randco-demo.git
  - scm credentials: github
  - scm branch: master

Inventories

For this demo to work, the following inventory needs to be prepared on Ansible Tower:

- randco ("normal" Inventory, not smart)
  - name: randco
  - sources
    - servicenow
      - name: servicenow
      - source: sourced from a project
      - credential: servicenow
      - project: randco
      - inventory file: servicenow.yml
      - overwrite: true
      - overwrite variables: true
      - update on project update: true

Templates

The following job templates need to be created:

- windows playbook (check mode)
  - name: win-playbook.yaml checkmode
  - job_type: check
  - inventory: randco
  - project: randco
  - playbook: win-playbook.yaml
  - credentials:
    - winMachine
- windows playbook
  - name: win-playbook.yaml
  - job_type: run
  - inventory: randco
  - project: randco
  - playbook: win-playbook.yaml
  - credentials:
    - winMachine
- html report
  - name: html-report.yaml
  - job_type: run

  - inventory: randco
  - project: randco
  - playbook: html-report.html
  - credentials
    * Ansible Tower
    * smtp
  - extra_variables:
    * email_destination: *[[set the right one]]*

- snow ticket
  - name: snow-ticket.yaml
  - job_type: run
  - inventory: randco
  - project: randco
  - playbook: snow-ticket.yaml
  - credentials
    * Ansible Tower
    * service_now

Workflows

An Ansible Tower Workflow is a sequence of Ansible Tower Templates, each matching an Ansible Playbook, that run sequentially after the Ansible Tower Workflow is triggered. There are 3 Ansible Tower Workflows for this demo:

- Ansible Tower Workflow 1
  - The goal of this Workflow is to show how the customer is able to receive HTML reports based on the output of a check-mode run
  - configuration
    * name: HTML report
    * inventory: randco
    * workflow visualizer
      · step 1: *win-playbook.yaml check-mode*
      · step 2: *html-report.yaml*
- Ansible Tower Workflow 2
  - The goal of this Workflow is to show how the customer is able to have ServiceNow Incidents opened based on the output of a check-mode run
  - configuration
    * name: SNow ticket
    * inventory: randco
    * workflow visualizer

            - step 1: *win-playbook.yaml check-mode*
            - step 2: *snow-tickets.yaml*

- Ansible Tower Workflow 3
  - The goal of this Workflow is to show that the same *win-playbook.yaml* Playbook previously used can also be used to do align the machines to the expected state
  - This can be a simple Template instead of a Workflow, since it's a single step
  - configuration
    - name: Real execution
    - inventory: randco
    - workflow visualizer
      - step 1: *win-playbook.yaml*

Additional resources

Used modules

- Ansible template module
- Ansible mail module
- Ansible snow_record module
- Ansible win_feature module
- Ansible win_iis_webbinding module
- Ansible snow inventory plugin

Dynamic inventories

- Working with dynamic inventories
- ServiceNow inventory

ServiceNow

- ServiceNow on Galaxy
- ServiceNow/Ansible demos
- ServiceNow/Ansible Tower example
- ServiceNow inventory
- ServiceNow create tickets:
  - Part 1
  - Part 2

Windows

- Connecting to a Windows Host
- Setting up a Windows Host
- Windows install IIS

Writing modules

- Ansible module development: getting started
- Building a sinmple module
- Developing modules

Misc

- Tower get job output in workflow
- Ansible Tower Custom credentials