PUBLISHING ANSIBLE ROLES & SCAP REFRESHER

Ansible Introduction and Usage of SCAP Workbench

INTRODUCTION

- Presenter
- Ansible Introduction Overview
- Turning a Playbook into an Ansible Role
- Publishing the Role on Ansible Galaxy
- SCAP Workbench Refresher for RHEL8
  - Customizing SCAP Content with Workbench
  - Generating Ansible Remediation Playbooks

PRESENTER INTRODUCTION

- **Speaker:** Travis Michette
- **Github Project:** https://github.com/tmichett/LUG
- **Presentation and Materials:** Located in the **Ansible_Roles** directory of the Github project.

ANSIBLE INTRODUCTION & OVERVIEW

- **Ansible** – An automation language leveraging modules to be used in one or more tasks on managed systems. Most Ansible automation leverages and Ansible playbook which is a YAML formatted file containing Ansible directives.

- **Ansible modules** – Components used by Ansible tasks and playbooks which are generally implemented and developed in Python. Ansible modules work with certain system utilities and are optimized to be leveraged as a declarative automation language and provide idempotency.

- **Ansible ad-hoc commands** – A way of executing a single Ansible task quickly that relies on a single Ansible module to perform the tests/changes of the task.

ANSIBLE COMPONENTS & COMMANDS

- **ansible.cfg** – Configuration file for running the **ansible** and **ansible-playbook** commands.
- **Inventory** – Inventory file used by the **ansible** and **ansible-playbook** commands identifying managed hosts/nodes.
- **ansible** – Command used to perform/execute Ansible ad-hoc commands against a managed node.
- **ansible-playbook** – Command used to execute and run Ansible playbooks
- **ansible-galaxy** – Command to create or utilize Ansible roles. Many of these roles are published on http://galaxy.ansible.com
- **Playbook** – Collection of Ansible tasks organized into one or more Ansible plays.
- **Task** – Collection/list of Ansible modules arranged into instructions. Each task utilizes an Ansible module to perform a given action.
- **Ansible Module** – Specific module (small program generally implemented in Python) which perform the commands and executes the program to get the desired state of a given task.

ANSIBLE.CFG FILE

- This file defines the configuration directives which apply directly to how the **ansible and ansible-playbook** command interact with the Ansible application and which configuration items are applied to a given Ansible session.
  - **./ansible.cfg** – When located in the current working directory (CWD) this file is the highest precedence.
  - **~/.ansible.cfg** – When located in the user's home directory, this file will have precedence if an *ansible.cfg* doesn't exist in the CWD.
  - **/etc/ansible/ansible.cfg** – This is the default configuration file and has the lowest precedence. This file is used when no other *ansible.cfg* file exists.

Important	It is also possible to define the **ansible.cfg** file with the environment variable **ANSIBLE_CONFIG**. If this variable is used, it will override all other configuration files.

INVENTORY FILE

- The **inventory** file can contain both Ansible managed hosts/nodes as well as inventory variables to be used for the managed nodes.
    - Inventory location is generally specified by the **ansible.cfg** file
        - **./inventory** – Common practice to leverage inventory files with playbooks and the **ansible.cfg** file in the current working directory
        - **/etc/ansible/hosts** – Default inventory file deployed with the Ansible package

PLAYBOOK STRUCTURE

```
---
- name: Ansiblize Managed Hosts
  hosts: localhost
  vars_files:
    - variables.yml

  tasks:
    - name: Create Ansible User
      debug:
          msg: This will use the USER  module

  handlers:
    - name: Restart SSHD
      debug:
          msg: This sill use the SERVICE or SYSTEMD
```

CREATING AN ANSIBLE ROLE

- Use the **ansible-galaxy init <RoleName>** command to create a Role
- Empty directories or unused directories can be deleted to clean up the Role
- Populate the various Role structures
  - Must have the following components (at minimum):
    - README.md
    - meta/main.yaml
    - tasks/main.yaml

```
[student@workstation ~]$ ansible-galaxy init My_Role
- My_Role was created successfully
```

ANSIBLE ROLE STRUCTURE

```
[student@workstation ROLES]$ tree My_Role/
My_Role/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
```

ANSIBLE ROLE STRUCTURE

Subdirectory	Function of Directory
defaults	The **main.yml** file contains default variable values that are used by the role. These have the lowest precedence and priority.
files	Contains files that are used by the tasks in the role.
handlers	The **main.yml** file contains handlers that are executed by the role.
meta	The **main.yml** file contains information about the role. At a minimum you should modify the ***author***, ***license***, ***platforms***, and ***dependencies***.
tasks	The **main.yml** file contains the tasks being used by the Role.
templates	Contains Jinja2 templates used by tasks in the role.
tests	Contains an inventory and **test.yml** playbook that can be used to test the role.
vars	The **main.yml** file contains role variables and values. These are high precedence and not intended to be changed when used in a playbook.

DEMO

CREATING AN ANSIBLE ROLE

PUBLISHING AN ANSIBLE ROLE

REQUIREMENTS

- **Github Repository** – Repository created for housing just the Ansible role. If you are publishing more than a single role, you should have a repository for each role.
- **Ansible Galaxy Account** – This account is required to be created and linked to your Github account. Once this is completed, you can select repositories to import to Ansible Galaxy as Ansible Roles.
- **Role** – A properly formatted role. Ideally one that was created and initialized with the **ansible-galaxy init <RoleName>** command.

DEMO

PUBLISHING AN ANSIBLE ROLE

SCAP WORKBENCH AND OPENSCAP

- **SCAP:** The Security Content Automation Protocol (SCAP) is a method for using specific standards to enable the automated vulnerability management, measurement, and policy compliance evaluation of systems deployed in an organization, including e.g., FISMA compliance. The National Vulnerability Database (NVD) is the U.S. government content repository for SCAP. An example of an implementation of SCAP is OpenSCAP
  - https://csrc.nist.gov/projects/security-content-automation-protocol

- **OpenSCAP:** An auditing tool that utilizes the Extensible Configuration Checklist Description Format (XCCDF). XCCDF is a standard way of expressing checklist content and defines security checklists
  - https://www.open-scap.org/

- **SCAP Workbench**: Graphical utility that allows an easy way to interact and perform common ***oscap*** tasks. It also provides an easy way to modify and tailor ***XCCDF*** profiles.

DEMO

USING SCAP WORKBENCH AND ANSIBLE