Linux Administration with Ansible: Writing Ansible Playbooks

Writing in YAML

Andrew Mallett
Linux Author and Trainer

@theurbanpenguin   www.theurbanpenguin.com

Overview

In this course you will learn to create repeatably correct configurations in Ansible using Playbooks

Course Overview

- Understanding YAML
- Playbooks vs Scripts
- Linting YAML
- Common Playbook Solutions

Module Overview

- Configuring Editors for YAML
- Using VS Code
- Creating and Linting Simple Playbooks

Lab Systems

Three Virtual Systems

- VirtualBox / Vagrant
- RHEL 8, Ubuntu 20.04, Centos Stream
- Using multiple Linux distributions throughout the course allows us to see the power of Ansible at its agnostic best
- Lab setup in Getting Started with Ansible

YAML

Ain't

Markup

Language.

YAML Represents Data Structures in Text Files

Online YAML Parser

https://yaml-online-parser.appspot.com/
https://jsonformatter.org/yaml-parser

Demo

For our first demonstration we will:

- connect to the online YAML parser

```
$ # cat file file1

user:
        name: bob
        dept: sales

user:
        name: bob
        dept: sales
```

Significant White Space

In YAML, leading whitespace is significant, meaning we do not need the array of brackets seen in JSON. Indent related keys with two spaces, convert tabs to spaces to make life easier

```
# cat -vet file file1

user:$
^Iname: bob$
^Idept: sales$

user:$
        name: bob$
        dept: sales$
```

These Files Look Similar

But they are not, one uses 8 spaces as an indent level the other uses a tab

Configuring Nano

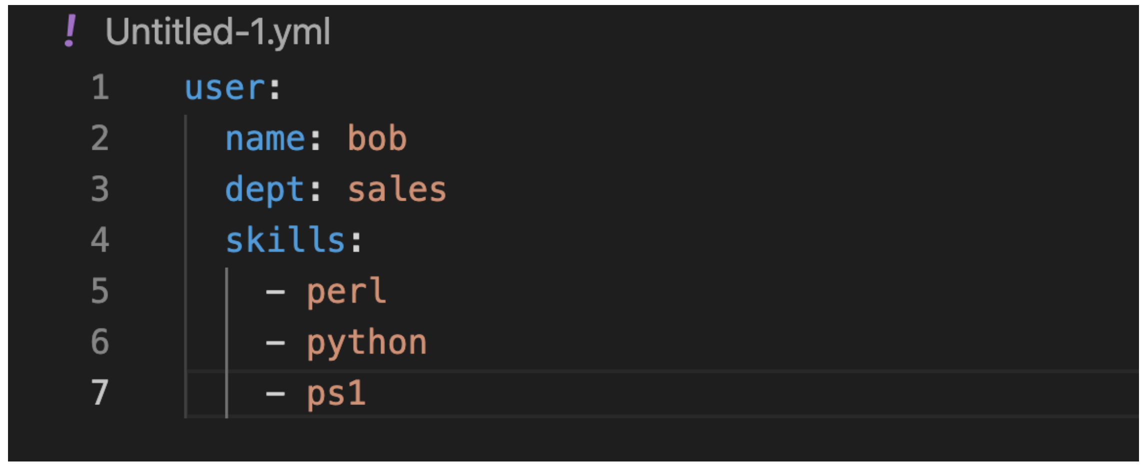
~/.nanorc

```
set autoindent
set tabsize 2
set tabstospaces
```

Configuring Vim

~/.vimrc

```
syntax on
set bg=dark
autocmd FileType yaml setlocal ai et ts=2 sw=2 cuc cul
```


Untitled-1.yml

```yaml
user:
  name: bob
  dept: sales
  skills:
    - perl
    - python
    - ps1
```

Microsoft Visual Studio Code (community edition) is ready go with YAML files and is free

Demo

In this demonstration we identify the problems that tabs can cause in YAML files

Demo

Creating a ~/.nanorc file for those using Nano as a text editor

Demo

Creating a ~/.vimrc file for those using Vim as a text editor

Demo

Using Visual Studio Code as an IDE for YAML

```
$ mkdir -p ~/ansible/simple

$ cd ~/ansible/simple

$ vim FirstPlay.yaml

- name: My First Play
  hosts: all
  become: true
  tasks:
    - name: My First Task
      package:
        name: tree
        state: present
```

Creating Playbooks

Using Playbooks, we can reference the same configuration and inventory used with ad-hoc commands. A Playbook will contain at least one Play and one of more Tasks within the Play

```
$ pip3 install ansible-lint --user
$ ansible-lint -v FirstPlay.yaml
Failed to guess project directory using git:
WARNING  Overriding detected file kind 'yaml' with 'playbook' for given positional
argument: FirstPlay.yaml
INFO     Executing syntax check on FirstPlay.yaml (0.33s)
```

Linting YAML

We can install the ansible-lint Python modules that will check you Playbook against style guidelines. Here there are no errors.

```
$ ansible-playbook FirstPlay.yaml --syntax-check

playbook: FirstPlay.yaml
```

Syntax Check

We can also check the correct syntax is employed in the playbook

```
$ ansible-playbook -C -v FirstPlay.yaml
Using /home/vagrant/.ansible.cfg as config file
...
changed: [192.168.33.11] => {"changed": true, "msg": "Check mode: No changes made, but
would have if not in check mode", "rc": 0, "results": ["Installed: tree-1.7.0-
15.el8.x86_64"]}
...
```

Implementing a No-Operation Check

Going beyond syntax checking we can try the option -C to check the operation without implementing change . This works best with the verbose option -v

Demo

Writing Playbooks:

- Writing a YAML Playbook
- Linting Playbooks
- Checking Playbook
- Implementing Playbooks

Default Task

The default task collects facts about the operating system. If these facts are not needed, we can turn it off in the Play

```yaml
- name: My First Play
  hosts: all
  become: true
  tasks:
    - name: My First Task
      package:
        name: tree
        state: present

    - name: Print Progress
      debug:
        msg: "This is {{ ansible_os_family }}"
```

Debug and Show Progress

In many languages, being able to print variables and the progress is very useful in ensuring the correct operation. We can use the gathered facts as variables in the Playbook.

Demo

Extending and Debugging Playbooks:

- Understand gather_facts
- Use variables and debug messages

```
$ find /usr/lib -name 'debug.py'

$ find /usr/lib -name 'package.py'
```

Locating Modules

Tasks execute Python modules; these are files in the file system

Demo

Locating Ansible Modules

Summary

In this module we have introduced the course highlights to you.

Course Highlights

- Compare scripts with Playbooks
- Writing Playbooks
- Common Tasks

In this Module

- Learned YAML
- Configured Editor
- Created a Playbook

Scripting Linux Administration