Learning Objectives: YAML Files and Syntax

Learners will be able to...

- **Define Ansible use-cases**
- **Apply the basic components of YAML syntax**

What is Ansible?

Ansible Automation

Ansible is an open-source IT Automation Platform written in Python which automates cloud provisioning, configuration management, software deployment, orchestration, and many other manual IT processes.

Ansible's simple, flexible framework allows users to solve many administrative and automation tasks.

Infrastructure Provisioning

Ansible provides reliable server configuration functionality and can be used to build a "golden image" for your servers: an AMI image that can be used to spin up ready instance almost instantly, or configure and setup hosts on creation, for example.

It can also be used for Docker Build to reduce the number of layers and simplify configuration of complex Docker images.

Configuration Management

Ansible provides a human-readable description of your infrastructure, aiding the transfer of configuration updates from testing environment to production.

Ansible also has the ability to maintain different environments in sync; it can be used in conjunction with tools like `vagrant` to provision local development infrastructure, making sure everybody on the team has the same configuration that's synced to testing and production environment.

Application Deployment

Lastly, Ansible can be used as deployment tool to automate work with artifacts, service reload, restart, and redistribution of multi-tier applications by using simple tasks.

Ansible language - YAML

Offical YAML Logo

YAML Ain’t Markup Language™

Ansible refers to its configuration management scripts as **playbooks**. Playbooks are built on top of YAML to take advantage of its easy-to-read syntax.

Therefore, mastering Ansible automation requires knowledge of YAML syntax. The following pages will focus on the aspects of YAML that are most applicable to writing playbooks. In particular you will learn how to:
- Organize lists (arrays)
- Write dictionaries
- Format strings
- Declare variables

YAML Syntax: Lists

Lists

Almost all YAML files start with a list (array). Let's explore the two ways of formatting lists in YAML.

Basic List Syntax

Lists begin with the name of the list followed by a colon. Below the list name, members are listed on independent lines with the same indentation level. Each member is preceded by a hyphen and a space as shown in the example below:

```
list:
  - one
  - two
  - three
```

Abbreviated List Syntax

YAML lists can be formatted on a single line as well. The list naming convention remains the same but its members are written inside a pair of brackets `[ ]` and separated by commas, as shown below:

```
list: [one, two, three]
```

Using Quotations

Quotation usage is optional when formatting strings in YAML, even if there are spaces. For example `my string`, `'my string'`, and `"my string"` are interpreted the same.

However, there are a few occasions when quotations are consequential:
* A numeric value, such as `5446`, will be returned as a string if it is written as `'5446'` or `"5446"`.
* If you are using any of the reserved words for a boolean value (`Yes`, `yes`, `No`, `no`, `true`, or `false`) in a string, double quotes should be used.
* For YAML to parse escape characters, like `\n` for a newline, they must be

placed inside double quotes: `"\n"`.
* Generally, if special characters (`: - { } [ ] ! # | > & % @`) are included in a string, quotes should be used.

▼ **YAML and special characters example**

In YAML, `:` (colon followed by a space) is an indicator of a mapping.

The following code is going to result in syntax error:

```
drive: c:
```

... but the following will work:

```
drive: c:\path
```

This is why for the first case we should use single or double quotes

```
drive: 'c:'
```

YAML Syntax: Dictionaries

Dictionaries

A list that contains **key-value** pairs is called a dictionary and is represented in a simple `key: value` form (the colon should be followed by a space):

```
- toots:
    name: Frederick Hibbert
    job: frontman
    number: '5446'
```

Dictionaries also can be written in abbreviate form, inside `{ }`:

```
- toots: { name: Frederick Hibbert, job: frontman, number:
'5446' }
```

Complex data structures are common in Ansible playbooks and can be created by combining lists and dictionaries:

```
- toots:
    name: Frederick Hibbert
    job: frontman
    number: '5446'
    roles:
      - singer
      - guitarist
      - organist
```

In the example above, the `roles` list is nested within the `toots` list.

YAML Syntax: Variables

Variables in Ansible

An Ansible playbook should be designed to manage multiple different systems, though we don't have to write unique commands to manage each one. Variables are used to manage the variations between systems so that tasks can be executed across multiple systems with a single command.

Variables are created using the standard YAML syntax; lists are used to define a variable with multiple values while dictionaries define single key-value pairs. They can be defined just about anywhere, but typically we create them in the playbook, inventory, roles, or passed at the command line.

We've already seen several variables defined in the previous examples; any dictionary or list is a variable, with a few restrictions:

- variables may only contain letters, numbers, underscores
- variables can't begin with a number, nor can they be Python or playbook keywords
- variables *can* begin with an underscore

Referencing Variables

While variables are written in YAML, we reference them using Jinja2 syntax. Let's take look at how to reference list and dictionary variables.

Example: dictionary reference

```
number: '5446'
```

Ansible allows variable substitution through the use of double quotes and braces `{ }`. For the following expression:

```
num: "{{ number }}"
```

... the value will be `5446`

Example: list reference

```
savage:
 - classy
 - bougie
 - ratchet
```

For the following expression:

```
iam: "{{ savage[1] }}"
```

... the value will be `bougie`

YAML Syntax: Strings

Multiline Strings

Scalars refers to strings, numbers, and booleans in YAML. For the purposes of writing Ansible playbooks we will be focusing on *Block Scalar* formatting.

Within *Block Scalar* formatting, YAML supports two formatting options for multiline strings: literal and folded.

Literal Block Scalar

The `|` character denotes **Literal Block Scalar** which keeps the newline characters; it maintains multiline formatting.

```
include_newlines: |
            This poetic string
            prefaced by a literal
            remains a haiku.
```

It will result in:

```
This poetic string
prefaced by a literal
remains a haiku.
```

As you can see in the example above, `|` preserves all newlines in the resulting text.

Folded Block Scalar

Alternatively, `>` denotes **Folded Block Scalar** which 'folds' new lines into spaces. This allows us to include longer strings while maintaining the file's readability.

```
fold_newlines: >
          I have conducted
          a poll confirming haikus
          are overrated.

          Notice how empty lines

          force new lines in the result.
```

will result in:

```
I have conducted a poll confirming haikus are overrated.
Notice how empty lines
force new lines in the result.
```

YAML Syntax: Complex Data Structures

Nesting

You may have noticed from previous examples that indentation is important; it indicates hierarchy, a parent-child relationship.

Consider the example from the previous reading question:

```
- nephew:
    name: D'Angelo Barksdale
    aka: "Dee"
    role: lieutenant
    location: 'The Pit'
    pups:
      - 'Bodie'
      - 'Poot'
      - 'Wallace'
```

As we can see above, 'Bodie', 'Poot', and 'Wallace' are listed directly below `pups` with identical indentation. This means several things:
- 'Bodie', 'Poot', and 'Wallace' are a part of the same task.
- They share a **parent-child** relationship:
- `pups` is the parent item of 'Bodie', 'Poot', and 'Wallace'.
- 'Bodie', 'Poot', and 'Wallace' are children of `pups`

Booleans

The YAML Boolean types that we will be using in our playbooks are `True` and `False`.

Other YAML Syntax

- Comments are made using a single # mark and include all text to the end of that line.
- A valid variable name can include any combination of letters, numbers, and underscores, but cannot begin with a number.
- Though not a requirement, YAML files usually:
  - start with `---`
  - end with `...`