YAML for beginners

YAML is an easy, expressive, data-oriented language that distinguishes itself from document markup languages.

Posted: October 26, 2020 | Seth Kenlon (Red Hat)

Image by inspireus from Pixabay

YAML Ain't a Markup Language (YAML), and as configuration formats go, it's easy on the eyes. It has an intuitive visual structure, and its logic is pretty simple: indented bullet points inherit properties of parent bullet points.

But this apparent simplicity can be deceptive.

- Getting started with DevSecOps

It's easy (and misleading) to think of YAML as just a list of related values, no more complex than a shopping list. There is a heading and some items beneath it. The items below the heading relate directly to it, right? Well, you can test this theory by writing a little bit of valid YAML.

Open a text editor and enter this text, retaining the dashes at the top of the file and the leading spaces for the last two items:

```
---
Store: Bakery
  Sourdough loaf
  Bagels
```

Save the file as example.yaml (or similar).

If you don't already have `yamllint` installed, install it:

```
$ sudo dnf install -y yamllint
```

A *linter* is an application that verifies the syntax of a file. The `yamllint` command is a great way to ensure your YAML is valid before you hand it over to whatever application you're writing YAML for (Ansible, for instance).

Use `yamllint` to validate your YAML file:

```
$ yamllint --strict shop.yaml || echo “Fail”
$
```

But when converted to JSON with a simple converter script, the data structure of this simple YAML becomes clearer:

```
$ ~/bin/json2yaml.py shop.yaml
{“Store”: “Bakery Sourdough loaf Bagels”}
```

Parsed without the visual context of line breaks and indentation, the actual scope of your data looks a lot different. The data is mostly flat, almost devoid of hierarchy. There's no indication that the sourdough loaf and bagels are children of the name of the store.

[Readers also liked: Ansible: IT automation for everybody]

How data is stored in YAML

YAML can contain different kinds of data blocks:

- Sequence: values listed in a specific order. A sequence starts with a dash and a space (`-`). You can think of a sequence as a Python list or an array in Bash or Perl.
- Mapping: key and value pairs. Each key must be unique, and the order doesn't matter. Think of a Python dictionary or a variable assignment in a Bash script.

There's a third type called `scalar`, which is arbitrary data (encoded in Unicode) such as strings, integers, dates, and so on. In practice, these are the words and numbers you type when building mapping and sequence blocks, so you won't think about these any more than you ponder the words of your native tongue.

When constructing YAML, it might help to think of YAML as either a sequence of sequences or a map of maps, but not both.

YAML mapping blocks

When you start a YAML file with a mapping statement, YAML expects a series of mappings. A mapping block in YAML doesn't close until it's resolved, and a new mapping block is explicitly created. A new block can only be created *either*by increasing the indentation level (in which case, the new block exists inside the previous block) or by resolving the previous mapping and starting an adjacent mapping block.

The reason the original YAML example in this article fails to produce data with a hierarchy is that it's actually only one data block: the key `Store` has a single value of `Bakery Sourdough loaf Bagels`. YAML ignores the whitespace because no new mapping block has been started.

Is it possible to fix the example YAML by prepending each sequence item with a dash and space?

```
---
Store: Bakery
  - Sourdough loaf
  - Bagels
```

Again, this is valid YAML, but it's still pretty flat:

```
$ ~/bin/json2yaml.py shop.yaml
{"Store": "Bakery - Sourdough loaf - Bagels"}
```

The problem is that this YAML file opens a mapping block and never closes it. To close the `Store` block and open a new one, you must start a new mapping. The *value* of the mapping can be a sequence, but you need a *key* first.

Here's the correct (and expanded) resolution:

```
---
Store:
  Bakery:
    - 'Sourdough loaf'
    - 'Bagels'
  Cheesemonger:
    - 'Blue cheese'
    - 'Feta'
```

In JSON, this resolves to:

```
{"Store": {"Bakery": ["Sourdough loaf", "Bagels"],
"Cheesemonger": ["Blue cheese", "Feta"]}}
```

As you can see, this YAML directive contains one mapping (`Store`) to two child values (`Bakery` and `Cheesemonger`), each of which is mapped to a child sequence.

YAML sequence blocks

The same principles hold true should you start a YAML directive as a sequence. For instance, this YAML directive is valid:

```
Flour
Water
Salt
```

Each item is distinct when viewed as JSON:

```
[“Flour”, “Water”, “Salt”]
```

But this YAML file is *not* valid because it attempts to start a **mapping block** at an adjacent level to a **sequence block**:

```
---
- Flour
- Water
- Salt
Sugar: caster
```

It can be repaired by moving the mapping block *into* the sequence:

```
---
- Flour
- Water
- Salt
- Sugar: caster
```

You can, as always, embed a sequence into your mapping item:

```
---
- Flour
- Water
- Salt
- Sugar:
    - caster
    - granulated
    - icing
```

Viewed through the lens of explicit JSON scoping, that YAML snippet reads like this:

```
[“Flour”, “Salt”, “Water”, {“Sugar”: [“caster”, “granulated”, “icing”]}]
```

[A free guide from Red Hat: 5 steps to automate your business.]

YAML syntax

If you want to comfortably write YAML, it's vital to be aware of its data structure. As you can tell, there's not much you have to remember. You know about **mapping** and **sequence** blocks, so you know everything you need have to work with. All that's left is to remember how they do and do not interact with one another. Happy coding!