YAML Tutorial

What is YAML ?

- YAML originally stands for “Yet Another Markup Language”
- However later they have found that It’s not marking up various elements of text document like xml
- Instead it acts as Serialization language (textual representation of cyclical data graphs)
- Hence later they renamed it as “YAML Ain’t Markup Language” means YAML is not a markup language)

Common use cases of YAML

- **Configuration management (CM)** – Ansible uses yaml files to describe all CM configurations (playbooks, roles, etc.).
- **Infrastructure as code (IaC)** – OpenTofu, for example, can read yaml files and use them as input for different resources, data sources, and even outputs.
- **CI/CD** – Many CI/CD products rely on yaml to describe their pipelines (GitHub Actions, GitLab CI/CD, Azure DevOps, CircleCI)
- **Container orchestration (CO)** – K8s and Docker Compose rely heavily on yaml files to describe the infrastructure resources.
- **Data serialization** – YAML can be used to describe complex data types such as lists, maps, and objects.
- **APIs** – YAML can be used in defining API contracts and specifications (e.g. OpenAPI

How YAML works ?

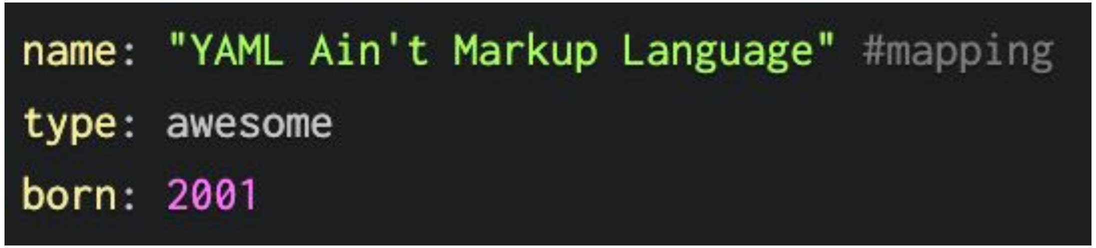
- YAML file relies on **whitespace** and **indentation**
- **TAB** characters cannot be used for indentation in YAML files
- Only **WHITE SPACES** can be used as **INDENTATION**
- Nested hierarchy of YAML components can be defined by **Indentation**

Components of YAML file

- YAML Dictionaries (Key-value pairs)
- YAML Lists
- Literals (Strings)
- Comments
- Nested Components
- Documents
- Anchors and Aliases (How to Override)
- vim configs for yaml

YAML Dictionaries (Key: Value pairs)

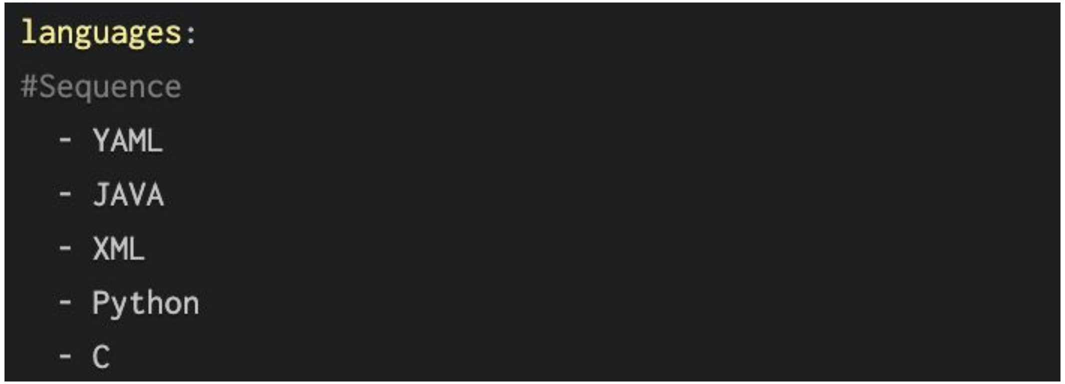
- Used to define key/value pairs
- Represented by key: value
- It is unordered



```yaml
name: "YAML Ain't Markup Language" #mapping
type: awesome
born: 2001
```

YAML Lists

- Represented by prefix with hyphen and space
- It is Ordered
- Can be embedded inside a map using indentation as shown below



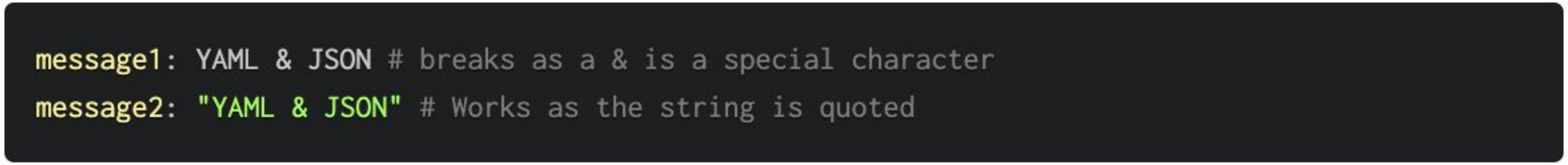
```yaml
languages:
#Sequence
  - YAML
  - JAVA
  - XML
  - Python
  - C
```

Note: Order matters with sequences but not with mappings.

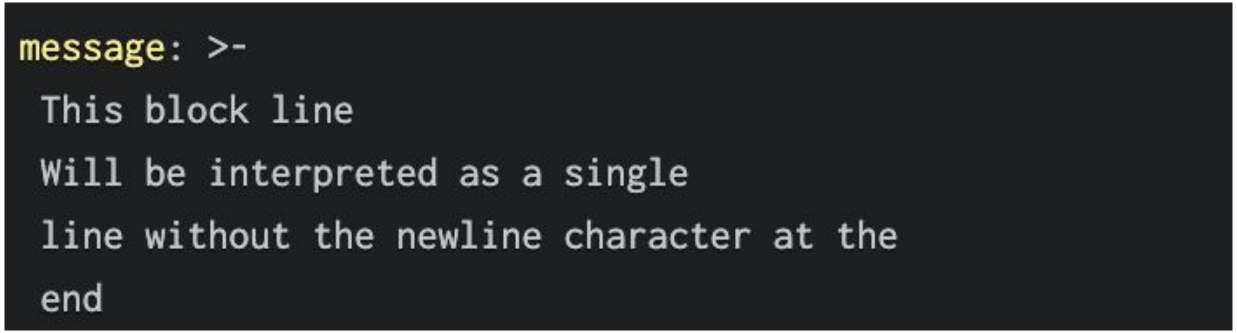
Literals (Strings)

String literals in YAML do not need to be quoted. It is only important to quote them when they contain a value that can be mistaken for a special character.

Here is an example where the string has to be quoted as & is a special character.



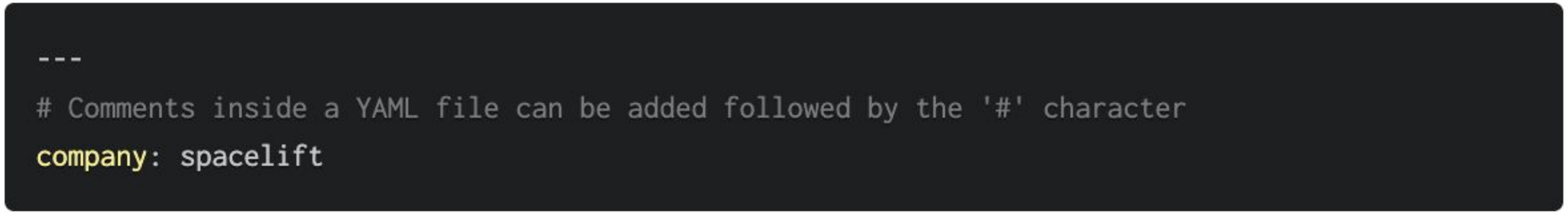
```yaml
message1: YAML & JSON # breaks as a & is a special character
message2: "YAML & JSON" # Works as the string is quoted
```



```yaml
message: >-
 This block line
 Will be interpreted as a single
 line without the newline character at the
 end
```

Comments

- YAML file supports comments which starts with #



```yaml
---
# Comments inside a YAML file can be added followed by the '#' character
company: spacelift
```

Nested Components

Mapping 1:

- Sequence 1
- Sequence 2

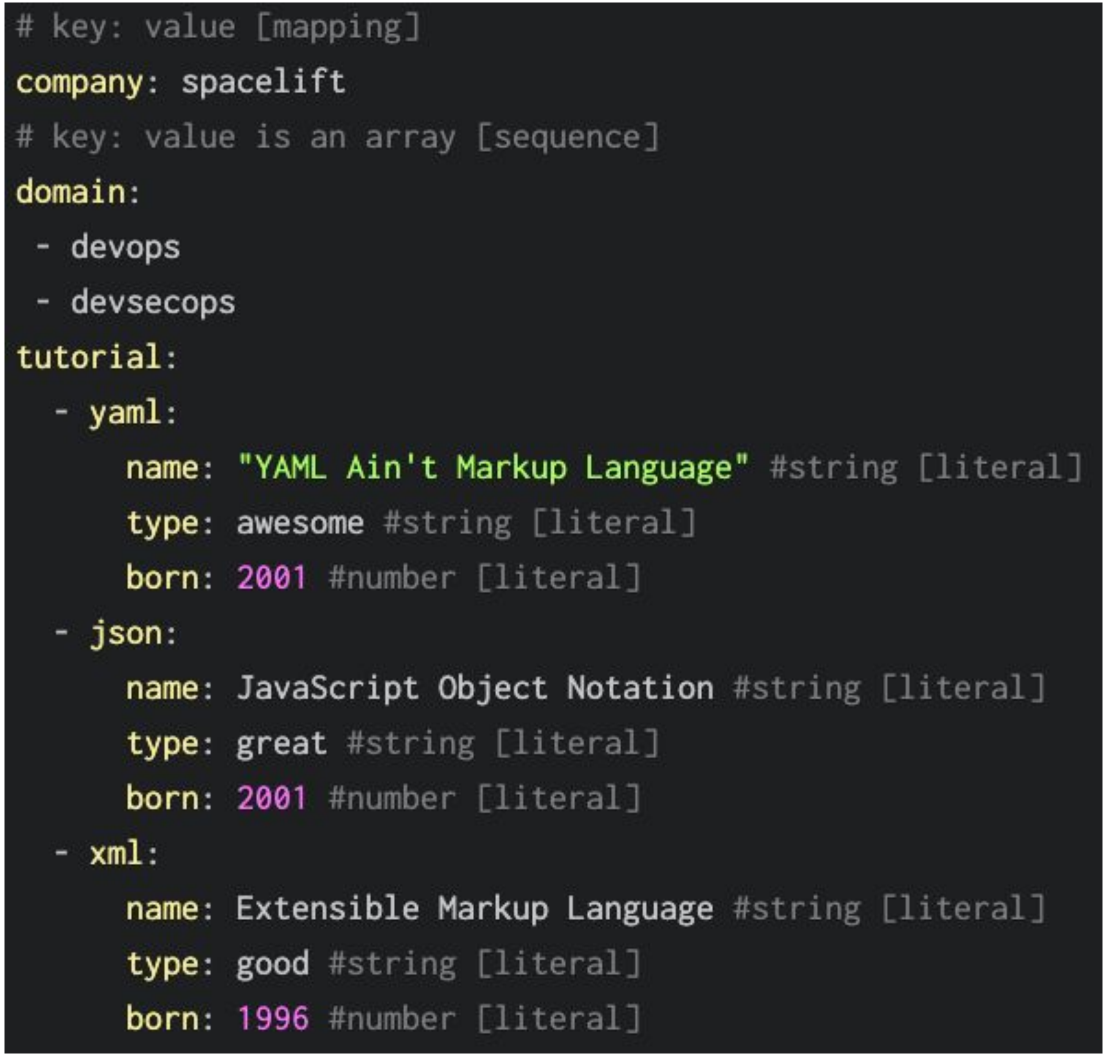
Mapping 2:

Mapping 2-1:

- Sequence 1

Note:

Nested components are identified by INDENTATION (white spaces)

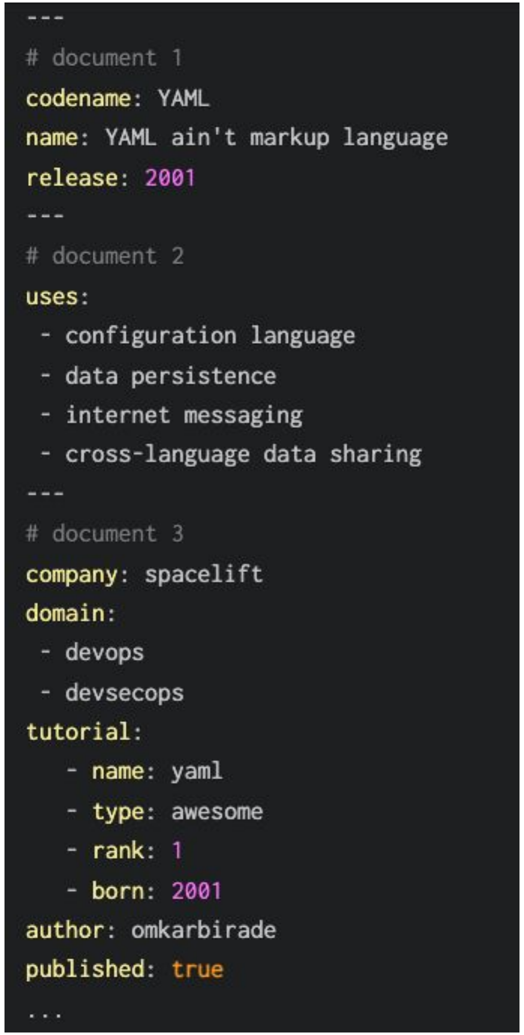


```yaml
# key: value [mapping]
company: spacelift
# key: value is an array [sequence]
domain:
 - devops
 - devsecops
tutorial:
  - yaml:
      name: "YAML Ain't Markup Language" #string [literal]
      type: awesome #string [literal]
      born: 2001 #number [literal]
  - json:
      name: JavaScript Object Notation #string [literal]
      type: great #string [literal]
      born: 2001 #number [literal]
  - xml:
      name: Extensible Markup Language #string [literal]
      type: good #string [literal]
      born: 1996 #number [literal]
```

Document (Multiple YAML files)

- Single YAML file can have more than one document
- Beginning of a document is represented by Three hyphens (---)
- Ending of a document is represented by

  Triple dots (...)



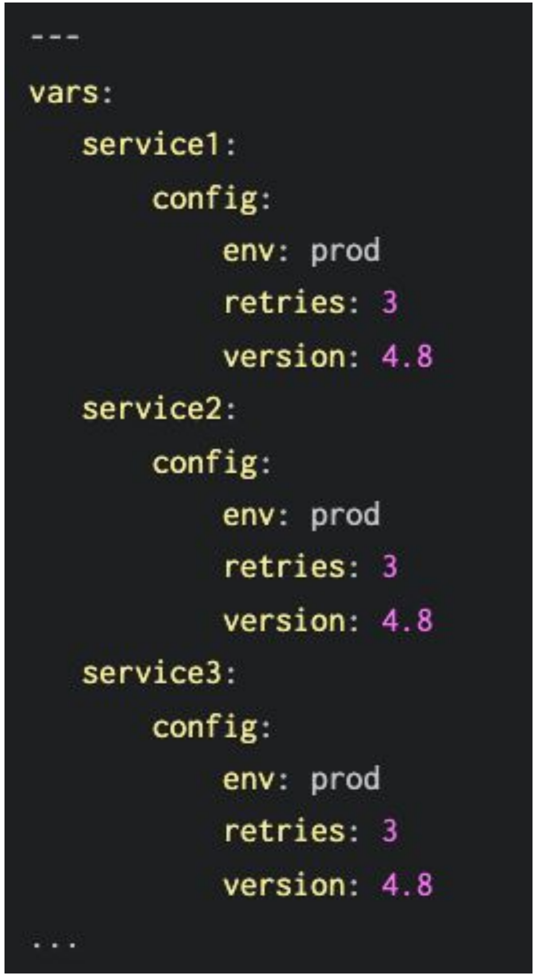
```yaml
---
# document 1
codename: YAML
name: YAML ain't markup language
release: 2001
---
# document 2
uses:
 - configuration language
 - data persistence
 - internet messaging
 - cross-language data sharing
---
# document 3
company: spacelift
domain:
 - devops
 - devsecops
tutorial:
   - name: yaml
   - type: awesome
   - rank: 1
   - born: 2001
author: omkarbirade
published: true
...
```

Anchors and alias

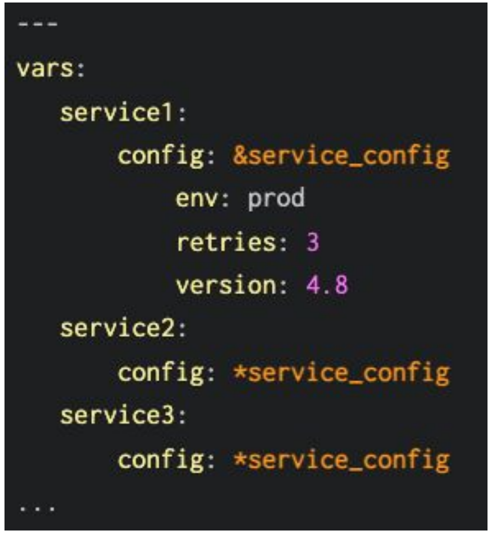
Anchors and aliases here helped us cut down the repeated configuration.

Anchors (&) are used to define a chunk of configuration

aliases (*) refer to that chunk at a different part of the configuration.



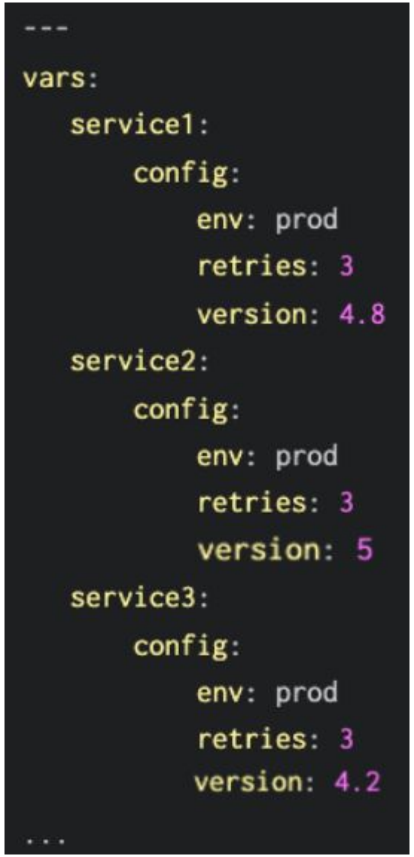
```yaml
---
vars:
  service1:
    config:
      env: prod
      retries: 3
      version: 4.8
  service2:
    config:
      env: prod
      retries: 3
      version: 4.8
  service3:
    config:
      env: prod
      retries: 3
      version: 4.8
...
```



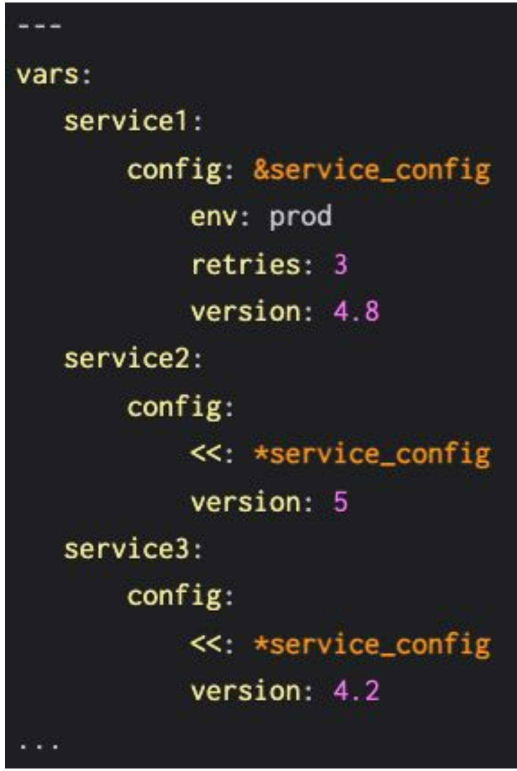
```yaml
---
vars:
  service1:
    config: &service_config
      env: prod
      retries: 3
      version: 4.8
  service2:
    config: *service_config
  service3:
    config: *service_config
...
```

Overriding Anchors and Aliases

We can override the specific values while using Anchors and Aliases



```yaml
---
vars:
  service1:
    config:
      env: prod
      retries: 3
      version: 4.8
  service2:
    config:
      env: prod
      retries: 3
      version: 5
  service3:
    config:
      env: prod
      retries: 3
      version: 4.2
...
```



```yaml
---
vars:
  service1:
    config: &service_config
      env: prod
      retries: 3
      version: 4.8
  service2:
    config:
      <<: *service_config
      version: 5
  service3:
    config:
      <<: *service_config
      version: 4.2
...
```

Privilege Escalation Attributes

`become: true` => Enabling run with privilege

`become_method: sudo` => we can use either sudo or su

`become_user: privileged_user` => either root or any user with sudo permissions

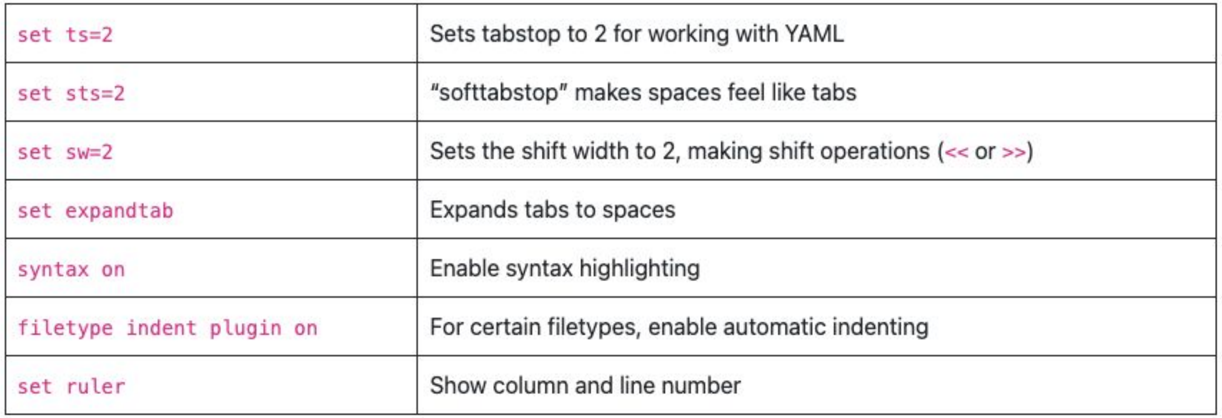
Example:

```
- name: /etc/hosts is up-to-date
  hosts: datacenter-west
  remote_user: automation
  become: true

  tasks:
    - name: server.example.com in /etc/hosts
      ansible.builtin.lineinfile:
        path: /etc/hosts
        line: '192.0.2.42 server.example.com server'
        state: present
```

vim configs for YAML

Edit these in your ~/.vimrc

```
set ts=2

set sts=2

set sw=2

set expandtab

syntax on

filetype indent plugin on

set ruler

set cursorcolumn

set nu
```



`set ts=2`	Sets tabstop to 2 for working with YAML
`set sts=2`	"softtabstop" makes spaces feel like tabs
`set sw=2`	Sets the shift width to 2, making shift operations (`<<` or `>>`)
`set expandtab`	Expands tabs to spaces
`syntax on`	Enable syntax highlighting
`filetype indent plugin on`	For certain filetypes, enable automatic indenting
`set ruler`	Show column and line number