YAML: YAML Ain't Markup Language or Yet Another Markup Language

What It Is YAML:

YAML is a human-friendly data serialization language for all programming languages.

YAML is a data serialization language for storing information in a human-readable form.

It is similar to XML and JSON files but uses a more minimalist syntax **YAML is commonly used** to create **configuration files**. Widely used in development & **In DevOps tools like Ansible, Docker, Kubernetes, Gitlab, Azure DevOps** etc.

Advantages of YAML

- Lightweight
- It is very easy and simple for represent complex mapping
- Human friendly readable and writable
- Simple to modify with any text editor
- Suitable for configuration settings
- Support for major programming languages

Readable syntax

YAML files use an indentation to show the structure of your data. You're required to **use spaces** to create **indentation** rather than **tabs** to avoid confusion **or errors.**

No executable commands

As a data-representation format, YAML does not contain executables. It's therefore very safe to exchange YAML files with external parties.

YAML must be integrated or use with other languages, like Python, Go or Java, to add executables.

Built-in commenting

YAML allows you to add comments to files using the hash symbol (#)

YAML Syntax

YAML has a few basic concepts that make up(represent or define) the majority of data.

Key-value pairs

```
<key>: <value>
```

In general, most things in a YAML file are a form of key-value pair where the key represents the pair's name and the value represents the data linked to that name. Key-value pairs are the basis for all other YAML constructions.

Scalars and mapping

Scalars represent a single stored value. Scalars are assigned to key names using mapping. You define a mapping with a name, colon, and space, then a value for it to hold.

YAML supports common types like integer and floating-point numeric values, as well as non-numeric types Boolean and String.

```
name: "Mithun"        # Sting data
age: 35               # Integer data
salary: 152000.65     # Float values
isWorking:  true.     # Boolean values true or false yes or no
```

Numbers

YAML supports integers, floating numbers, and exponential floating numbers.

```
integer: 123
float: 123.123
```

Booleans

In YAML, we can represent the boolean value **True** and **False** in three different ways. Look at them.

- The values **True, On,** and **Yes** are considered as **True** in YAML.
- The values **False, Off,** and **No** are considered as **False** in YAML.

```
employed: true
contractor: false
```

String

Strings are a collection of characters that represent a sentence or phrase. You either use | to print each string as a new line or > to print it as a paragraph.

Strings in YAML can be represented with or without quotes. Both are similar.. But, if we need to use the escape sequences, then we must use double-quotes

```
str: Hello World
data: |
  These
  Newlines
  Are broken up
data: >
  This text is
  wrapped and is a
  single paragraph
```

Sequence(lists or arrays)

Sequences are data structures similar to a **list or array that hold multiple values under the same key.** They're defined using a block or inline flow style.
Block style uses spaces to structure the document. It's easier to read but is less compact compared to flow style.
In Block Style Array elements in separate lines preceded hyphen (-)

```
---
# Shopping List Sequence in Block Style
shopping:
- milk
- eggs
- juice
```

Flow style allows you to write sequences inline using square brackets, similar to an array declaration in a programming language like Python or JavaScript. Flow style is more compact but harder to read at a glance.

```
---
# Shopping List Sequence in Flow Style
shopping: [milk, eggs, juice]
```

Dictionaries

Dictionaries are collections of key-value pairs. We can have any valid data type as a value to the key. YAML even supports the nested dictionaries.

```
# An employee record
employee:
  name: Mithun
  job:  Architect
  team: DevOps
```

Dictionaries can contain more complex structures as well, such as sequences(Array or List of Dictionaries). Nesting sequences is a good trick to represent complex relational data.

```
# List of employee(dictionaries) records with complex data.
employess:
- name: Bhaskar
  job: Developer
  skills:
  - python
  - perl
  - ruby
- name: Balaji
  job: Developer
  skills:
  - java
  - go
```

Multi-document support

You can have multiple YAML documents in a single YAML file to make file organization or data parsing easier.

The separation between each document is marked by three dashes (`---`)

```
---
player: playerOne
action: attack (miss)
---
player: playerTwo
action: attack (hit)
```

Sample yaml files in ansible, docker & Kubernetes

Ansible Playbook

```
- hosts: webservers
  become: yes
  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: ensure apache is running
    service:
      name: httpd
      state: started
```

Docker Compose file

```yaml
version: '3.1'

services:
  springboot:
    image: dockerhandson/spring-boot-mongo:latest
    environment:
    - MONGO_DB_HOSTNAME=mongo
    - MONGO_DB_USERNAME=devdb
    - MONGO_DB_PASSWORD=devdb123
    ports:
    - 8080:8080
    depends_on:
      - mongo
    networks:
    - springappnetwork
  mongo:
    image: mongo
    environment:
    - MONGO_INITDB_ROOT_USERNAME=devdb
    - MONGO_INITDB_ROOT_PASSWORD=devdb123
    volumes:
      - mongodb:/data/db
    restart: always
    networks:
    - springappnetwork

volumes:
  mongodb:
    external: true

networks:
  springappnetwork:
```

Kubernetes Manifest file

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    name: javawebappdeployment
  name: javawebapp
spec:
  replicas: 2
  selector:
    matchLabels:
      app: javawebapp
  template:
    metadata:
      labels:
        name: javawebapp
    spec:
      containers:
      - image: dockerhandson/java-web-app:1
        name: javawebapp
        ports:
        - containerPort: 8080
---
# Node Port Service
apiVersion: v1
kind: Service
metadata:
  labels:
    name: javawebapp
  name: javawebapp
spec:
  type: NodePort
  ports:
    - port: 8080
      targetPort: 8080
  selector:
    name: javawebapp
```

YAML vs JSON vs XML

YAML (.yml or .yaml):

- Human-readable code
- Minimalist syntax
- Solely designed for data
- Similar inline style to JSON (is a superset of JSON)
- Allows comments
- Strings without quotation marks
- Considered the “cleaner” JSON
- Advanced features (extensible data types, relational anchors, and mapping types preserving key order)

Use Case: YAML is best for data-heavy apps that use DevOps pipelines or configurations. It’s also helpful for when other developers on your team will work with this data often and therefore need it to be more readable.

JSON(.json)

- Harder to read
- Explicit, strict syntax requirements
- Similar inline style to YAML (some YAML parsers can read JSON files)
- No comments
- Strings require double quotes

Use Case: JSON is favoured in web development as it’s best for serialization formats and transmitting data over HTTP connections.

XML(.xml)

- Harder to read
- More verbose
- Acts as a markup language, while YAML is for data formatting
- Contains more features than YAML, like tag attributes
- More rigidly defined document schema

Use Case: XML is best for complex projects that requiree fine control over validation, schema, and namespace. XML is not human-readable and requires more bandwidth and storage capacity, but offers unparalleled control.