

Understanding YAML and JSON

Guide to Comparing and Using These Data
Formats

Introduction to YAML and JSON

YAML (Yet Another Markup Language) and JSON (JavaScript Object Notation) are both **human-readable data serialization formats**. They are widely used in **configuration files, data exchange between APIs, and infrastructure management**.

| Feature | YAML | JSON |
|--------------------|--|---|
| Syntax | Uses indentation for structure | Uses brackets <code>{}</code> and <code>[]</code> |
| Readability | More human-friendly | More machine-readable |
| Data Types | Supports strings, numbers, lists, maps | Supports strings, numbers, lists, objects |
| Comments | ✅ Supports comments (<code>#</code>) | ❌ No native comment support |
| Used In | Docker Compose, Kubernetes, Ansible | APIs, Web Applications, Configuration |

YAML vs. JSON: Syntax Comparison

Basic Data Structure (Key-Value Pairs)

YAML:

```
name: John Doe
age: 30
married: true
address:
  city: New York
  zip: "10001"
```

JSON:

```
{
  "name": "John Doe",
  "age": 30,
  "married": true,
  "address": {
    "city": "New York",
    "zip": "10001"
  }
}
```

✓ **YAML uses indentation**, while **JSON uses nested braces** `{}`.

Lists (Arrays)

YAML:

```
fruits:  
- Apple  
- Banana  
- Orange
```

JSON:

```
{  
  "fruits": ["Apple", "Banana", "Orange"]  
}
```

✅ **YAML** uses dashes `-`, while **JSON** uses `[]` brackets.

Multi-line Strings

YAML (Block Scalars):

```
description: |  
  This is a multi-line string.  
  YAML makes it easy to read.
```

JSON (No Native Multi-Line Support):

```
{  
  "description": "This is a multi-line string.\nJSON requires escape sequences."  
}
```

✅ **YAML supports clean multi-line text** without needing escape characters.

Booleans and Null Values

YAML:

```
is_admin: yes
logged_in: false
data: null
```

JSON:

```
{
  "is_admin": true,
  "logged_in": false,
  "data": null
}
```

✓ **YAML allows multiple Boolean notations (`yes/no`, `true/false`), while JSON only supports `true/false`.**

Nested Objects

YAML:

```
person:
  name: Alice
  contact:
    email: alice@example.com
    phone: "123-456-7890"
```

JSON:

```
{
  "person": {
    "name": "Alice",
    "contact": {
      "email": "alice@example.com",
      "phone": "123-456-7890"
    }
  }
}
```

✓ **YAML keeps indentation clean, while JSON requires extra `{}` brackets.**

Referencing Values (YAML Anchors)

YAML (Using Anchors `&` and Aliases `*`):

```
default: &config
  retries: 3
  timeout: 30s

service1:
  <<: *config
  name: API Service

service2:
  <<: *config
  name: Database Service
```

JSON (No Native Equivalent, Requires Duplication):

```
{
  "service1": {
    "retries": 3,
    "timeout": "30s",
    "name": "API Service"
  },
  "service2": {
    "retries": 3,
    "timeout": "30s",
    "name": "Database Service"
  }
}
```

✓ **YAML supports reusing configurations, while JSON requires duplication.**

Introduction to `jq` – The JSON Query Tool

`jq` is a powerful **command-line tool** for **parsing, filtering, and transforming JSON data**.

- ✓ **Lightweight and fast**
- ✓ **Works with JSON APIs**
- ✓ **Can extract specific data, format JSON output**

Installing **jq**

Linux (Ubuntu/Debian)

```
sudo apt install jq
```

Mac (Homebrew)

```
brew install jq
```

Windows

Install via [Chocolatey](#):

```
choco install jq
```

Using `jq` to Parse JSON

Basic JSON Example

File: `data.json`

```
{  
  "name": "John Doe",  
  "age": 30,  
  "city": "New York"  
}
```

Extract Specific Key

```
cat data.json | jq '.name'
```

Output:

```
"John Doe"
```

Using `jq` to Parse JSON

Filter Nested Data

```
{
  "users": [
    { "name": "Alice", "age": 25 },
    { "name": "Bob", "age": 30 }
  ]
}
```

Extract **Bob's age**:

```
cat data.json | jq '.users[] | select(.name=="Bob") | .age'
```

Output:

```
30
```

Formatting JSON Output

Pretty Print JSON

```
cat data.json | jq .
```

Minify JSON (Remove Spaces):

```
cat data.json | jq -c .
```

Working with JSON Arrays

Filter Only Names

```
cat data.json | jq '.users[].name'
```

Output:

```
"Alice"  
"Bob"
```

Count Elements in an Array

```
cat data.json | jq '.users | length'
```

Output:

```
2
```

Modifying JSON with `jq`

Adding New Fields

```
cat data.json | jq '. + { "country": "USA" }'
```

Output:

```
{  
  "name": "John Doe",  
  "age": 30,  
  "city": "New York",  
  "country": "USA"  
}
```

Updating Values

```
cat data.json | jq '.age = 35'
```

Removing a Key

```
cat data.json | jq 'del(.city)'
```

Comparison: YAML vs. JSON

| Feature | YAML Example | JSON Example |
|-----------------|--|---|
| Key-Value Pair | <code>name: John</code> | <code>{ "name": "John" }</code> |
| Lists (Arrays) | <code>- Apple</code> | <code>["Apple"]</code> |
| Booleans | <code>yes</code> , <code>no</code> | <code>true</code> , <code>false</code> |
| Nested Objects | <code>person:\n name: John</code> | <code>{ "person": { "name": "John" } }</code> |
| Multi-line Text | <code>`description:</code> | <code>\n This is text`</code> |
| Comments | <code># This is a comment</code> | ✗ (No support) |
| YAML Anchors | <code>&default\n retries: 3\n*default</code> | ✗ (No equivalent) |

Summary

- ✓ **YAML is easier to read** and is commonly used in **Docker, Kubernetes, Ansible**.
- ✓ **JSON is machine-friendly**, used in **APIs and data exchange**.
- ✓ **jq** is a **must-have tool** for handling JSON in the command line.