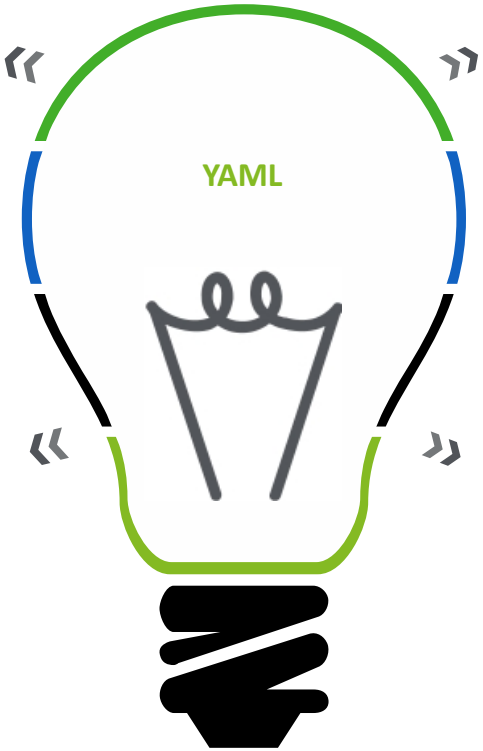
YAML

YAML ZERO TO MASTER

WHAT WE COVER IN THIS COURSE

Introduction to YAML

- *What is YAML*
- *XML vs JSON vs YAML*
- *YAML Use cases*
- *Writing Sample YAML file*



YAML Advance concepts

- *Anchors, Alias, Overriding*
- *Multi documents support*
- *Writing Complex keys*
- *Yamllint to validate YAML files*

YAML Basic concepts

- *Scalars, Strings, Sequences, Dictionaries*
- *Comments, Explicit Typing*

YAML Real examples

- *Exploring sample YAML configurations from AWS, Docker & Kubernetes*

WHAT IS YAML?

YAML is a light-weight, human-readable data-serialization language. It is primarily designed to make the format easy to read while including advanced features.

YAML stands for “YAML Ain't Markup Language”.

It is similar to XML and JSON files but uses a more minimalist syntax even while maintaining similar capabilities.

YAML files are created with extensions “.yaml” or “.yml” . You can use any IDE or text editor to open/create YAML files.

YAML is similar inline style to JSON (is a superset of JSON).

It is very easy and simple for represent complex mapping. Due to which it is heavily used in configuration settings.

YAML Official Website - https://yaml.org/

XML vs JSON vs YAML

Evolution of XML, JSON & YAML

XML JSON YAML

YAML is a super set of JSON which means all the features of JSON can be found in YAML. For example if you compare with JavaScript and TypeScript, TypeScript will be YAML and JavaScript will be JSON.

XML vs JSON vs YAML

```xml
<applications>
  <application>
      <name>Accounts</name>
      <technology>Python</technology>
   </application>
   <application>
       <name>Cards</name>
       <technology>Java</technology>
    </application>
    <application>
        <name>Loans</name>
        <technology>Ruby</technology>
     </application>
</applications>
```

5

XML

eXtensible Markup Language

```json
{
    "applications": [
      {
        "name": "Accounts",
        "technology": "Python"
      },
      {
        "name": "Cards",
        "technology": "Java"
      },
      {
        "name": "Loans",
        "technology": "Ruby"
      }
    ]
}
```

JSON

JavaScript Object Notation

```yaml
applications:
- name: Accounts
  technology: Python
- name: Cards
  technology: Java
- name: Loans
  technology: Ruby
```

YAML

YAML Aint Markup Language

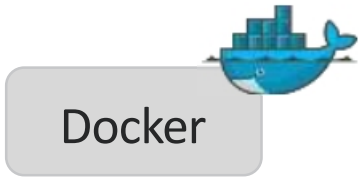
XML vs JSON vs YAML

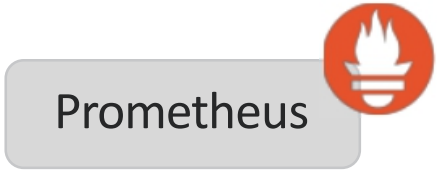
	XML	JSON	YAML
HUMAN-READABLE	Harder to read	Harder to read	Easier to read and understand
SYNTAX	More verbose	Explicit, strict syntax requirements	Minimalist syntax
COMMENTS	Allows comments	Comments are not allowed	Allows comments
HIERARCHY	Hierarchy is denoted by using open & close tags.	Hierarchy is denoted by using braces and brackets.	Hierarchy is denoted by using double space characters.
STORAGE	Will take lot of storage and network bandwidth.	Lighter compared to XML	Lighter compared to XML & JSON
USECASES	Best for complex projects that require fine control over schema	Favored in web development & transmitting data over HTTP	Best for configuration files, along with JSON features.

YAML USECASES

- Since YAML is a human friendly readable format, it is widely used in writing configurations files in different DevOps tools, cloud platforms and applications.

FEW OF THE MOST FAMOUS TOOLS/APPLICATIONS THAT FOLLOWS YAML HEAVILY





THUMB RULES IN WRITING YAML

sample.yaml

```
applications:
- name: Accounts
  technology: Python
- name: Cards
  technology: Java
- name: Loans
  technology: Ruby
```

8

Indentation of whitespace is used to denote structure. This is very similar to Python uses indentation to highlight the blocks of code.

The basic structure of a YAML file is a map. You might call this a dictionary, hash or object, depending on your favorite programming language.

Tabs are not included as indentation for YAML files. So please be careful with space and tab inside YAML files.

YAML is case sensitive in nature. [name: accounts != Name: Accounts]

SCALARS IN YAML

KEY-VALUE PAIRS

- Most things in a YAML file are a form of key-value pair. Key-value pairs are the basis for all other YAML constructions.

SCALARS

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

SCALARS IN YAML

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

```yaml
name: eazybytes
languagge: "Java"
area: 'Online'
major-version: 2
minor-version: 0.5
hex: 0x12d6 #evaluates to 4822
octal: 024432 #evaluates to 10522
goodDay: true
badday: false
niceweekend: Yes
boringweekend: No
positive-infinity: .inf # evaluates to infinity
negative-infinity: -.Inf #evaluates to negative infinity
invalidNumber: .NAN #Not a Number
no-value: null
```

10

YAML TO JSON

```json
{
  "name": "eazybytes",
  "languagge": "Java",
  "area": "Online",
  "major-version": 2,
  "minor-version": 0.5,
  "hex": 4822,
  "octal": 10522,
  "goodDay": true,
  "badday": false,
  "niceweekend": true,
  "boringweekend": false,
  "positive-infinity": Infinity,
  "negative-infinity": -Infinity,
  "invalidNumber": NaN,
  "no-value": null
}
```

YAML

JSON

References: https://yaml - online -parser.appspot.com/ , https://www.json2yaml.com/

STRINGS IN YAML

STRINGS

- Strings in YAML doesn’t need explicit double or single quotes.
- Use single or double quotes in YAML if your value includes special characters. For example, these special characters may require quotes: {, }, [,], ,, &, :, *, #, ?, |. -, <. >, =, !, %, @, \.
- "Yes" and "No" should be enclosed in quotes (single or double) or else they will be interpreted as True and False boolean values.

STRINGS IN YAML

STRINGS

- We can use > (Folded style) to removes newlines within the string
- We can use | (Literal style) to turn every newline within the string into a literal newline.
- You can control the handling of the final new line in the string, and any trailing blank lines (\n\n) by adding a block chomping indicator character:
    - >, |: "clip": keep the line feed, remove the trailing blank lines.
    - >-, |-: "strip": remove the line feed, remove the trailing blank lines.
    - >+, |+: "keep": keep the line feed, keep trailing blank lines.

STRINGS IN YAML

- Sample YAML file with String related examples and it's corresponding JSON file,

```yaml
organization: eazybank
about: |
  EazyBank
  is an
  Bank based
  application.
history: >
  EazyBank
  established
  in 1989.
products: "[Accounts] & {Cards} - <Loans>"
hasMobileApp: "Yes"
```

13

YAML

YAML TO JSON

```json
{
  "organization": "eazybank",
  "about": "EazyBank\nis an\nBank based\napplication.\n",
  "history": "EazyBank established in 1989.\n",
  "products": "[Accounts] & {Cards} - <Loans>",
  "hasMobileApp": "Yes"
}
```

JSON

References: https://yaml - online -parser.appspot.com/ , https://www.json2yaml.com/

COMMENTS SUPPORT IN YAML

ADDING COMMENTS INSIDE YAML FILES

- YAML allows you to add comments to files using the hash symbol (#) similar to Python comments.

- YAML supports only single line comments. Its structure is explained below with the help of an example:

```
#14this is the best YAML course for you
```

- YAML does not support multi line comments. If we need to provide comments for multiple lines, we can do so by writing multiple single line comments as shown below,

```
# this
# is the best YAML
# course for you
```

COMMENTS SUPPORT IN YAML

- A commented block is skipped during execution and helps to add description for specified code block.
- We can add comments with in YAML code as shown below. This will help others to understand your YAML configurations quickly,

```yaml
# this is the best YAML course for you
isSingleLine : true
# this
# is the best YAML
# course for you
isMultipleLine : true
supportingComments : This has more comments # This property has more comments
```

15

YAML TO JSON

```json
{
  "supportingComments": "This has more comments",
  "isMultipleLine": true,
  "isSingleLine": true
}
```

- One of the key difference and advantage with YAML compared to JSON is it will allow comments inside documents. In JSON, comments are not allowed.

IMPLICIT AND EXPLICIT TYPING IN YAML

- YAML offers versatility in typing by auto-detecting data types while also supporting explicit typing options.
- To tag data as a certain type, simply include !![typeName] before the value.

```yaml
---
# Boolean explicit declaration
YAML is a superset of JSON: !!bool true
Pluto is a planet: !!bool false
# Integer explicit declaration
negative: !!int -16
zero: !!int 0
positive: !!int 30
# Float explicit declaration
negative-flot: !!float -3.14
zero-flot: !!float 0
positive-float: !!float 16.
# String explicit declaration
sampleNumString: !!str 8.14
sampleString: !!str EazyBytes
sampleDate: !!str 1989-06-16
# Timestamp explicit declaration
randomdate: !!timestamp 2001-12-15 2:59:43.10
# Null explicit declaration
key with null value: !!null null
...
```

YAML TO JSON

```json
{
  "YAML is a superset of JSON": true,
  "zero-flot": 0.0,
  "sampleDate": "1989-06-16",
  "positive": 30,
  "randomdate": "2001-12-15 02:59:43",
  "negative": -16,
  "zero": 0,
  "Pluto is a planet": false,
  "positive-float": 16.0,
  "negative-flot": -3.14,
  "key with null value": null,
  "sampleNumString": "8.14",
  "sampleString": "EazyBytes"
}
```

TIMESTAMP IN YAML

!!timestamp

- A timestamp value represents a single point in time. It is a helpful data type that lets you store times as a unit rather than as a collection of different numbers.

- Using !!timestamp data tag we can hold various levels of date formats as per our requirements. Below are the few examples,

  - ✓ canonical: 2001-12-15T02:59:43.1Z
  - ✓ iso8601: 2001-12-14t21:59:43.10-05:00
  - ✓ space separated: 2001-12-14 21:59:43.10 -5
  - ✓ no time zone (Z): 2001-12-15 2:59:43.10
  - ✓ date (00:00:00Z): 2002-12-14

TIMESTAMP IN YAML

- If the time zone is omitted, the timestamp is assumed to be specified in UTC. The time part may be omitted altogether, resulting in a date format. In such a case, the time part is assumed to be 00:00:00Z (start of day, UTC).

- We can define the timezone by including how many hours it is ahead or behind UTC. For example, we can set a timestamp as EST with a -5 at the end.

```yaml
# Date & Time representation using String
canonical_string: 2001-12-15T02:59:43.1Z
iso8601_string: 2001-12-14t21:59:43.10-05:00
space separated_string: 2001-12-14 21:59:43.10 -5
no time zone (Z)_string: 2001-12-15 2:59:43.10
date (00:00:00Z)_string: 2002-12-14
# Date & Time representation using timestamp
canonical: !!timestamp 2001-12-15T02:59:43.1Z
iso8601: !!timestamp 2001-12-14t21:59:43.10-05:00
space separated: !!timestamp 2001-12-14 21:59:43.10 -5
no time zone (Z): !!timestamp 2001-12-15 2:59:43.10
date (00:00:00Z): !!timestamp 2002-12-14
```

YAML TO JSON

```json
{
  "canonical_string": "2001-12-15 02:59:43 UTC",
  "iso8601_string": "2001-12-14 21:59:43 -0500",
  "space separated_string": "2001-12-14 21:59:43 -0500",
  "no time zone (Z)_string": "2001-12-15 02:59:43 +0000",
  "date (00:00:00Z)_string": "2002-12-14",
  "canonical": "2001-12-15 02:59:43 UTC",
  "iso8601": "2001-12-14 21:59:43 -0500",
  "space separated": "2001-12-14 21:59:43 -0500",
  "no time zone (Z)": "2001-12-15 02:59:43 +0000",
  "date (00:00:00Z)": "2002-12-14"
}
```

SEQUENCES/COLLECTIONS IN YAML

SEQUENCES/COLLECTIONS

- Sequences are 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. They can be defined with either block style or inline flow style.
- Block style uses spaces to structure the list or array. It's easier to read but is less compact compared to flow style.
- In Flow styles we can write sequences inline using square brackets, similar to an array declaration in a programming language like Python, Java or JavaScript. Flow style is more compact but harder to read at a glance.
- We can always, embed a sequence into another sequence.

SEQUENCES/COLLECTIONS IN YAML

- Sample YAML file with Sequence/list related examples and it’s corresponding JSON file,

```yaml
# List Sequence in Block Style
departments:
  - Marketing
  - Insurance
  - Security
# List Sequence in flow Sty
locations: [Hyderabad, NewYork, Berlin, "Paris"]
# Embeded/Nested Sequences
products:
  - Accounts:
      - Savings Account
      - Current Account
  - Loans:
      - Home Loan
      - Car Loan
      - Personal Loan
  - Cards:
      - Credit Card
      - Debit Card
```


YAML TO JSON

```json
{
  "products": [
    {
      "Accounts": [
        "Savings Account",
        "Current Account"
      ]
    },
    {
      "Loans": [
        "Home Loan",
        "Car Loan",
        "Personal Loan"
      ]
    },
    {
      "Cards": [
        "Credit Card",
        "Debit Card"
      ]
    }
  ],
  "locations": [
    "Hyderabad",
    "NewYork",
    "Berlin",
    "Paris"
  ],
  "departments": [
    "Marketing",
    "Insurance",
    "Security"
  ]
}
```

DICTIONARIES/MAPPINGS IN YAML

DICTIONARIES/MAPPINGS

- Dictionaries are collections of key-value pairs all nested under the same subgroup. They’re helpful to divide data into logical categories for later use.

- Dictionaries are defined with a name, a colon, and a space followed by 1 or more indented key-value pairs.

- Using Dictionaries, we can build information related objects like Person, Application, Vehicle etc.

- The key-value is YAML's basic building block. Every item in a YAML document is a member of at least one dictionary. The key is always a string. The value is a scalar so that it can be any datatype. So, as we've already seen, the value can be a string, a number, or another dictionary.

DICTIONARIES/MAPPINGS IN YAML

- Sample YAML file with dictionaries related examples and it’s corresponding JSON file,

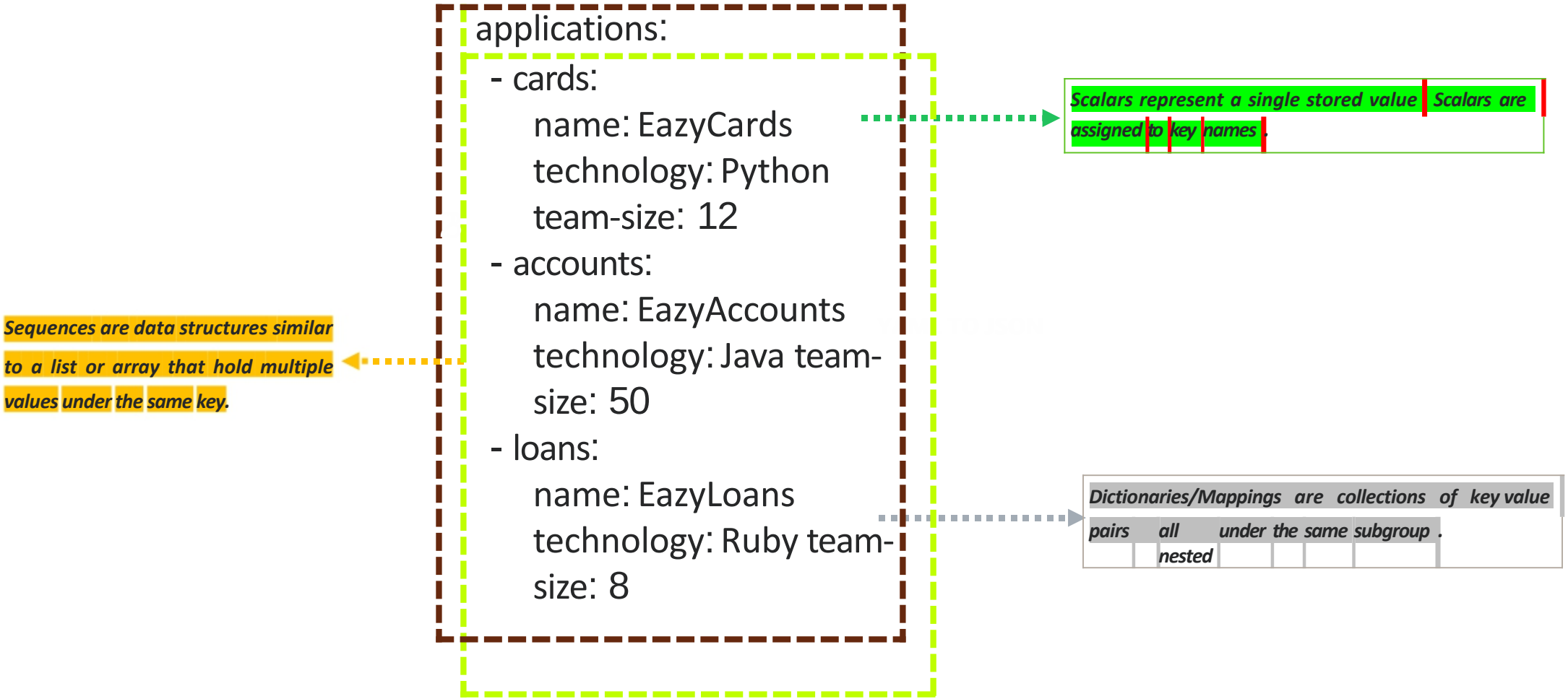
```yaml
# An Application record
applications:
  - cards:
      name: EazyCards
      technology: Python
      team-size: 12
  - accounts:
      name: EazyAccounts
      technology: Java
      team-size: 50
  - loans:
      name: EazyLoans
      technology: Ruby
      team-size: 8
```


YAML TO JSON

```json
{
  "applications": [
    {
      "cards": {
        "technology": "Python",
        "name": "EazyCards",
        "team-size": 12
      }
    },
    {
      "accounts": {
        "technology": "Java",
        "name": "EazyAccounts",
        "team-size": 50
      }
    },
    {
      "loans": {
        "technology": "Ruby",
        "name": "EazyLoans",
        "team-size": 8
      }
    }
  ]
}
```

COMMON STRUCTURE OF YAML

- Below is the most common structure of YAML file,



COMPLEX KEYS IN YAML

- Keys can also be complex inside YAML, like multi-line string. We use ? followed by a space to indicate the start of a complex key.

- Sample YAML file with complex key related examples and it's corresponding JSON file,

```yaml
? This is a key
  that has multiple lines
: and this is its value
? - Development
  - UAT
  - PROD
: - http://dev.eazybank.com
  - http://uat.eazybank.com
  - http://prod.eazybank.com
```


YAML TO JSON

```json
{
  "This is a key that has multiple lines": "and this is its value",
  "[\"Development\", \"UAT\", \"PROD\"]": [
    "http://dev.eazybank.com",
    "http://uat.eazybank.com",
    "http://prod.eazybank.com"
  ]
}
```

ANCHORS & ALIAS IN YAML

- Anchors and aliases let you identify an item with an anchor in a YAML document, and then refer to that item with an alias later in the same document. Anchors are identified by an & character, and aliases by an * character.

  - ✓ The anchor '&' which defines a chunk of values/configuration
  - ✓ The alias '*' used to refer to that chunk elsewhere

- Anchors & Aliases can be considered If we have repeated sections inside our yaml files. They can reduce effort and make updating in bulk, easier.

- The Alias essentially acts as a "see above'' command, which makes the program pause standard traversal, return to the anchor point, then resume standard traversal after the Anchored portion is finished.

- YAML anchors and aliases cannot contain the ' [', '] ', ' { ', ' } ', and ' , ' characters.

ANCHORS & ALIAS IN YAML

- Sample YAML file with anchors/alias related examples and it’s corresponding JSON file,

```yaml
definitions:
  days:
    - weekday: &working
        wakeup: 6:00 AM
        activites:
          - workout
          - meetings
          - work
          - sleep
        sleeptime: 10:00 PM

schedules:
  days:
    weekdays:
      - monday: *working
```


YAML TO JSON

OVERRIDE ANCHORS VALUES IN YAML

- After defining anchor values inside your YAML file, what if you want essentially the same block of code with one small change?
- You can use overrides with the characters '<<:' before the Alias to add more values, or override existing ones.
- When using overrides mappings are overridden if the new mapping has the same name or is added afterward if different.
- Overriding is also called as merge.

OVERRIDE ANCHORS VALUES IN YAML

- Sample YAML file with anchors/alias/override related example and it’s corresponding JSON file,

```yaml
definitions:
  days:
    - weekday: &working
        wakeup: 6:00 AM
        activites:
          - workout
          - meetings
          - work
          - sleep
        sleeptime: 10:00 PM

schedules:
  days:
    weekdays:
      - friday:
          <<: *working
          sleeptime: 12:00 AM
```


YAML TO JSON

```json
{
  "definitions": {
    "days": [
      {
        "weekday": {
          "wakeup": "6:00 AM",
          "activites": [
            "workout",
            "meetings",
            "work",
            "sleep"
          ],
          "sleeptime": "10:00 PM"
        }
      }
    ]
  },
  "schedules": {
    "days": {
      "weekdays": [
        {
          "friday": {
            "wakeup": "6:00 AM",
            "activites": [
              "workout",
              "meetings",
              "work",
              "sleep"
            ],
            "sleeptime": "12:00 AM"
          }
        }
      ]
    }
  }
}
```

MULTI-DOCUMENT SUPPORT IN YAML

- A document starts with three dashes (---) and ends with three periods (...). Some YAML processors require the document start operator. The end operator is usually optional. For example, Java's Jackson will not process a YAML document without the start, but Python's PyYAML will.

- We 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 (---).

```yaml
# Development Applications Details
---
applications:
  - cards:
      name: EazyCards
      technology: Python
      team-size: 12
  - accounts:
      name: EazyAccounts
      technology: Java
      team-size: 50
...
```

YAML TO JSON

```json
{
  "applications": [
    {
      "cards": {
        "technology": "Python",
        "name": "EazyCards",
        "team-size": 12
      }
    },
    {
      "accounts": {
        "technology": "Java",
        "name": "EazyAccounts",
        "team-size": 50
      }
    }
  ]
}
```

THANK YOU & CONGRATULATIONS

YO U ARE N O W A M A S T E R OF YAML