aws re/start

Introduction to JSON and YAML

This lesson introduces you to JavaScript Object Notation (JSON) and YAML Ain't Markup Language (YAML).

JSON and YAML are data formats that are commonly used in automated and repeatable deployments.

What you will learn

At the core of the lesson

You will learn how to:

- Describe the basic syntax of JavaScript Object Notation (JSON) and YAML Ain't Markup Language (YAML)
- Explain the advantages and disadvantages of JSON and YAML data formats

Goals:

- Be able to identify JSON & YAML formats
- Understand the basic syntax of both formats

YAML & JSON: Syntaxes for defining cloud infrastructure

The language of infrastructure

JSON & YAML: Declarative languages for creating cloud infrastructures

- **Infrastructure as code** (IaC) is a method to declare the resources that are needed in the cloud in using text files
- **JavaScript Object Notation** (JSON) and **YAML Ain't Markup** (YAML) syntaxes are used by AWS to declare resources in the cloud
- Enables the definition of simple to complex infrastructures in text
- Understanding the syntaxes of JSON & YAML are required to build infrastructure as code

Before studying how to define infrastructure in code it is necessary to study the syntax of the two formats used by AWS CloudFormation.

Introduction to JSON

What is JSON?

JavaScript Object Notation (JSON)

- Syntax for storing and transporting data that contains:
  - Key-value pairs
    - » `{Key1: "Value1", Key2: "Value2"}`
  - Arrays (lists) and other objects
    - » `{Array: [1, 2, 3]}`
- Enables the exchange of complex data structures in a single document

aws re/start

Some of the AWS services covered in this module—such as AWS CloudFormation—support JavaScript Object Notation (JSON) as a data format. AWS CloudFormation is a service for writing or changing templates that create and delete related AWS resources together as a unit (stack). To provision and configure your stack resources, you must understand AWS CloudFormation templates, which are formatted text files in JSON or YAML. Therefore, in this section, you will learn the essentials of JSON.

JSON is a syntax for storing and transporting data. It is a text-based format, so it is human-readable. JSON documents are easily written. They store key-value pairs and arrays of data. A key-value pair is a set of two linked data items:

- **Key** – Unique identifier for an item of data.
- **Value** – Data that is identified or a pointer to the location of that data.

The data can be structured so that you can store complex data structures in a single JSON document. The key use case for JSON is that it supports the simple exchange of complex data between computer systems.

Representation of data

StudentName	Location	FavoriteSport
María	USA	Tennis
John	UK	Cricket
Diego	USA	Basketball
Kwesi	India	Hockey

Data in rows and columns

```
{
  "sport_choices":[
    {
      "StudentName": "María",
      "Location": "USA",
      "FavoriteSport": "Tennis"
    },
    {
      "StudentName": "John",
      "Location": "UK",
      "FavoriteSport": "Cricket"
    },
    {
      "StudentName": "Diego",
      "Location": "USA",
      "FavoriteSport": "Basketball"
    },
    {
      "StudentName": "Kwesi",
      "Location": "India",
      "FavoriteSport": "Hockey"
    }
  ]
}
```

Data in JSON

You commonly see data in a columnar format, such as in a table. However, to store table data in a way that is easy for machines to ingest, parse, and distribute, it must be stored in a non-graphical way.

JSON provides a way to store data objects in a human-readable text document. As illustrated in the example, columnar data from a table—which consists of three columns and five rows—can be stored in JSON format.

Advantages and disadvantages of JSON

Advantages:

- Lightweight (minimal syntax and mockup)—good for application programming interfaces (APIs).
- Easy for humans to read and write.
- Easy for machines to parse and generate.

Disadvantages:

- No native support for binary data (such as image files).

Like any standard, JSON has both advantages and disadvantages.

One key advantage of JSON is that it is *lightweight*. JSON has only a few syntax rules that you must follow so that the JSON document is valid. It is also a good choice for use with APIs. In fact, JSON is a common data exchange format for modern RESTful services.

JSON is also easy for humans to read, and it is easy for machines to parse and generate.

However, JSON also has some disadvantages. It is more verbose. As a result, you might have long files, where many lines in the file consist of only an opening brace ({) or closing brace (}). Another disadvantage of JSON is that it does not have built-in support for storing multimedia formats (such as images) in JSON.

JSON building blocks: Objects

How would you describe this dessert?

Cake

Chocolate

Feeds 8 people

$20

Consider how you would describe an object, such as the dessert shown here. What characteristics does it have? Four characteristics that describe this dessert are highlighted.

JSON building blocks: Objects

You can structure that information into a **JSON object**.

Key Value

```
{
  "type": "cake",
  "flavor": "chocolate",
  "price": "20",
  "feeds": "8"
}
```


Here, you see that the cake characteristics that were identified are now captured in a JSON object.

JSON building blocks: Objects

- **Braces { }** act as a container.
- JSON uses universal data structures:
  - An object is an unordered set of key-value pairs.
  - An object begins and ends with braces ({ }).
  - Each key is followed by a colon (:).
  - Each key-value pair is separated by a comma (,).

```
{
  "type": "cake",
  "flavor": "chocolate",
  "price": "20",
  "feeds": "8"
}
```


JSON objects are enclosed in braces. JSON data structures must follow specific rules.

First, a JSON object consists of an unordered set of key-value pairs. A colon is used to separate the keys from the values, and all keys and values are enclosed in quotation marks. Key-value pairs in an object are separated by commas.

The last key-value pair does *not* have a comma after it.

In the example, suppose that the *flavor* key-value pair came before the *type* key-value pair. A program or person that receives this JSON object would still consider the object valid.

If you had another dessert and wanted to describe that dessert by using the same keys, it might look something like this example:

```
{
  "type": "pie",
  "flavor": "apple",
  "price": "16",
  "feeds": "6"
}
```

Notice that the same keys are used, but the values are different.

JSON building blocks: Arrays (1 of 3)

What additional attributes could you use to describe this dessert?

Mint

Blueberries

If you wanted to be more specific about this dessert, you could document additional characteristics. For example, it has both blueberries and mint as additional ingredients.

JSON building blocks: Arrays (2 of 3)

You can put the additional ingredients information in an **array**.

```
{
  "type": "cake",
  "flavor": "chocolate",
  "price": "20",
  "feeds": "8",
  "additional_ingredients": [
      "blueberries",
      "mint"
  ]
}
```


Here, an `additional_ingredients` array was added to the JSON object. Arrays are formatted differently than key-value pairs.

Arrays are used when a key can have multiple values. From the example, “additional_ingredients” has the values “blueberries” and “mint”. Some other examples of useful arrays could a list of names in a group, chemicals in formula, models available from a car manufacturer, and so on.

JSON building blocks: Arrays (3 of 3)

- **Brackets ([ ])** hold arrays.
- An array is an ordered list of values.
- An array begins with a left bracket ([) and ends with a right bracket (]).
- Values are separated by a comma (,).
- Values can consist of multiple types.
  - String, number, object, array, or Boolean

```
{
  "type": "cake",
  "flavor": "chocolate",
  "price": "20",
  "feeds": "8",
  "additional_ingredients": [
      "blueberries",
      "mint"
  ]
}
```


To define an array in JSON, enclose it in brackets.

An array in JSON is an ordered collection of values. Unlike the key-value pairs at the root of a JSON object, JSON arrays are ordered, so that you can parse them by index value. For example, if you want to pull mint from the example object, you would refer to `objectName.additional_ingredients[1]`.

Array values can be one of these types: string, number, object, array, or Boolean. The example array has two values, both of which are strings (`blueberries` and `mint`). However, in longer and more complex JSON documents, you will often see arrays that contain other JSON objects or other JSON arrays.

Introduction to YAML

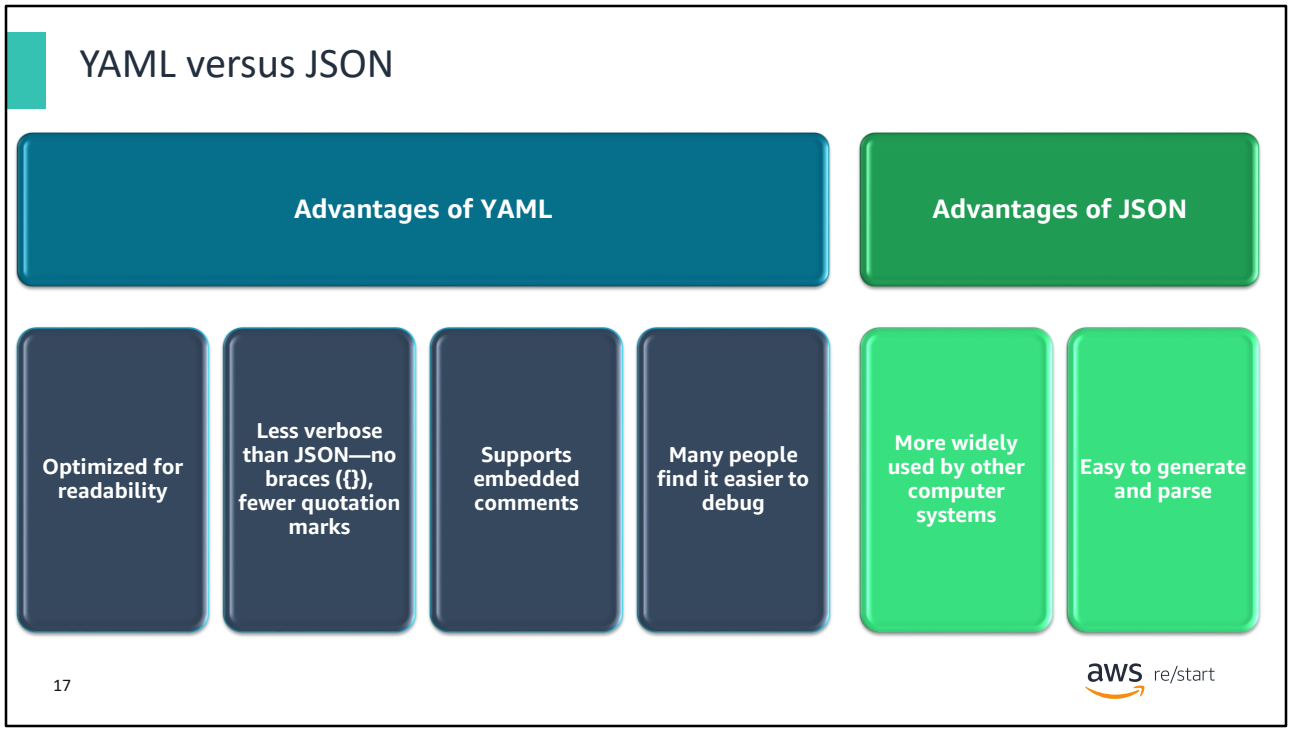
What is YAML?

YAML Ain't Markup Language (YAML)

- Is a human-readable data serialization language
  - Often used for *configuration files*
  - Used to describe data
- Is designed to be portable
  - Works well with C, Java, Perl, Python, Ruby, and other languages
- Provides similar capabilities to JSON
  - Complex data structures can be exchanged in a single document.

Some of the AWS services that are discussed later in this lesson—such as AWS CloudFormation—also support YAML as a data format. To use AWS CloudFormation, is it important to understand the YAML format. Similar to JSON, you should be able to understand the structure of a YAML document to author some AWS CloudFormation template sections in YAML. Therefore, in this section, you will be introduced to the essentials of YAML.

The creators of YAML wanted to distinguish YAML from languages—such as Markdown and XML—that are often used to mark up text. Like JSON, YAML is a syntax for storing data. It is a text-based format, so it is human-readable. YAML documents are easily written. They store key-value pairs, lists, and associative arrays of data. You can store complex data structures in a single YAML document.



Compared to JSON, YAML is optimized for readability. It is less verbose than JSON. The same data that is stored in JSON will take fewer lines to store in YAML. The reason is that YAML does not use braces, and it uses fewer quotation marks. Another advantage of YAML is that it natively supports embedded comments. Finally, many find it easier to debug YAML documents compared with JSON. With JSON, it can be difficult to find missing or misplaced commas or braces. YAML does not have this issue.

Despite the many advantages of YAML, JSON offers some unique advantages. First, it is widely used by computer systems. Its ubiquity is an advantage because data that is stored in JSON can reliably be used with many systems without needing transformation. Also, it is usually easier to generate and parse JSON than it is to generate and parse YAML.

AWS CloudFormation, which will be discussed in detail later in this module, supports templates that are written in JSON or YAML.

YAML syntax basics

A few YAML syntax basics:

- Use *white-space indentation* to denote and create data structures.
  - Do not use tabs.
  - Two spaces is the standard indentation.
- Add a *comment* by using a number sign (#) at the beginning of the line.
- ***Lists***:
  - Create a list by starting the line with a hyphen (one list item per line).
  - Alternatively, define the list on a single line, using brackets ([]) and commas (,) between items.
- Create *associative arrays* by using a colon followed by a space.
  - *<key>: <value>* (one item per line)
- ***Strings*** are normally not in quotes.
  - However, they might be enclosed in double or single quotes.

aws re/start

Denoting data structures
YAML syntax uses white-space indentation to denote and create data structures. It is important to know that YAML interprets tabs as white space. You must be careful to avoid using tab spaces when you create or edit YAML documents. Inserting two white spaces is the standard indentation. However, as long as your indentation approach is consistent, the document will be valid.

Adding comments
To add a comment, use a number sign (#) at the beginning of the line. Keep a blank line both preceding and following any commented line.

Creating lists
To create a list, start the line with a hyphen, and write only one list item per line. Alternatively, you can define an entire list by putting multiple line items on a single line. However, to use this list structure, you must surround the list in brackets ([]) and use commas between the list items.

Creating arrays
To create an associative array, use a colon followed by a space. You will see an example of this syntax next.

Defining key-value pairs

Just like JSON, YAML documents store key-value pairs. To define a single key-value pair, use a colon between the key and the value (use one item per line).

Creating strings

Finally, unlike JSON, strings in YAML are normally *not* enclosed in quotation marks. However, you might sometimes see them enclosed in double or single quotation marks. Usually, quotation marks are used to escape characters that appear in the string, and that would otherwise confuse a system reading a YAML document.

Examples: JSON and YAML

JSON*

```
"WebServer": {
    "Type": "AWS::EC2::Instance",
    "Properties": {
        "ImageId": "ami-09ead922c1dad67e4",
        "InstanceType": "t2.micro",
        "KeyName": "myKey",
        "SecurityGroupIds": [
            {
                "Ref": "SecurityGroupIDs"
            }
        ],
        "SubnetId": "subnet-42b01138"
    }
}
```

*Not actual code

YAML*

```
WebServer:
  Type: AWS::EC2::Instance
  Properties:
    ImageId: ami-09ead922c1dad67e4
    InstanceType: t2.micro
    KeyName: myKey
    SecurityGroupIds:  !Ref SecurityGroupID
    SubnetId: subnet-42b01138
```

*Not actual code

The examples show the same configuration information documented in both JSON and YAML formats.

The JSON version has more characters and includes many quotation marks, braces, and commas. The YAML version has fewer characters and uses indentation with spaces (not tabs) to create parent-child relationships between pieces of data.

- The JSON file uses braces to indicate that `Type` and `Properties` are children of the `WebServer` parent.

- The YAML file uses white-space indentation to indicate `Type` and `Properties` are children of the `WebServer` parent.

Notice the difference in how the key-value pairs are documented. In YAML, the key and the value are *not* in quotation marks. The ends of the lines do not have commas, and the keys and values do not need to be enclosed in quotation marks.

You now have a general understanding of the syntax of both JSON and YAML. This understanding should help you while you start exploring the features of AWS CloudFormation in this module.

Key takeaways

- JSON
  - Syntax for storing and transporting data.
  - Text-based format, so it is human-readable.
  - Documents are easily written.
  - Stores key-value pairs and arrays of data.
- YAML
  - Syntax for storing data.
  - Text-based format, so it is human-readable.
  - Documents are easily written.
  - Store key-value pairs, lists, and associative arrays of data.
  - Store complex data structures in a single YAML document.

aws re/start

- JSON
  - Syntax for storing and transporting data.
  - Text-based format, so it is human-readable.
  - Documents are easily written.
  - Stores key-value pairs and arrays of data.
- YAML
  - Syntax for storing data.
  - Text-based format, so it is human-readable.
  - Documents are easily written.
  - Store key-value pairs, lists, and associative arrays of data.
  - Store complex data structures in a single YAML document.