

YAML with Kubernetes

YAML, which stands for Yet Another Markup Language, or YAML Ain’t Markup Language is a human-readable text-based format for specifying configuration-type information.

it is a human friendly data serialization standard for all programming languages.

**YAML** (YAML Ain't Markup Language) is a human-readable data serialization language. It is commonly used for configuration files, but could be used in many applications where data is being stored (e.g. debugging output) or transmitted (e.g. document headers). YAML targets many of the same communications applications as XML  . It uses both Python-style indentation to indicate nesting, and a more compact format that uses [] for lists and {} for maps making YAML 1.2 a superset of JSON.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | YAML | | | [**Filename extension**](https://en.wikipedia.org/wiki/Filename_extension) | .yaml, .yml | | [**Internet media type**](https://en.wikipedia.org/wiki/Media_type) | Not registered | | **Initial release** | 11 May 2001; 17 years ago | | [**Latest release**](https://en.wikipedia.org/wiki/Software_release_life_cycle) | 1.2 (Third Edition) (1 October 2009; 8 years ago) | | **Type of format** | Data interchange | | [**Open format**](https://en.wikipedia.org/wiki/Open_format)**?** | Yes | | **Website** | [yaml.org](http://yaml.org) | |

1. YAML is a human-readable text-based format that let’s you easily specify configuration-type information by using a combination of maps of name-value pairs and lists of items (and nested versions of each).
2. YAML is the most convenient way to work with Kubernetes objects like pods,rc,services.

Using YAML for K8s definitions gives you a number of advantages, including:

* **Convenience:** You’ll no longer have to add all of your parameters to the command line
* **Maintenance:** YAML files can be added to source control, so you can track changes
* **Flexibility:** You’ll be able to create much more complex structures using YAML than you can on the command line

YAML is a superset of JSON, which means that any valid JSON file is also a valid YAML file.

There are only two types of structures in YAML:

* Seperator (optional)
* Lists
* Maps

**YAML Maps**

Maps let you associate name-value pairs, which of course is convenient when you’re trying to set up configuration information.  For example, you might have a config file that starts like this:

---

apiVersion: v1

kind: Pod

The first line is a separator, and is optional unless you’re trying to define multiple structures in a single file. From there, as you can see, we have two values, v1 and Pod, mapped to two keys, apiVersion and kind.

specify more complicated structures by creating a key that maps to another map, rather than a string, as in:

---

apiVersion: v1

kind: Pod

**metadata:**

**name: oracle**

**labels:**

**app: database**

In this case, we have a key, metadata, that has as its value a map with 2 more keys, name and labels. The labels key itself has a map as its value. we can nest these as far as you want to.

## YAML lists

YAML lists are literally a sequence of objects.  For example:

args

 - india

 - "200"

 - welcome

 - "firefox browser!"

we can have virtually any number of items in a list, which is defined as items that start with a dash (-) indented from the parent

---

apiVersion: v1

kind: Pod

metadata:

 name: oracle

 labels:

   app: database

spec:

**containers:**

**- name: back-end**

**image: oracle**

**ports:**

**- containerPort: 1521**

**- name: dbservice**

**image:** oracle:v1

**ports:**

**- containerPort: 8080**

So as we can see here, we have a list of containers “objects”, each of which consists of a name, an image, and a list of ports.  Each list item under ports is itself a map that lists the containerPort and its value.

 We have:

* maps, which are groups of name-value pairs
* lists, which are individual items
* maps of maps
* maps of lists
* lists of lists
* lists of maps

Basically, whatever structure you want to put together, you can do it with those two structures.

Properties you can set for a Container:

* name
* image
* command
* args
* workingDir
* ports
* env
* resources
* volumeMounts
* livenessProbe
* readinessProbe
* livecycle
* terminationMessagePath
* imagePullPolicy
* securityContext
* stdin
* stdinOnce
* tty

**JavaScript Object Notation** or **JSON** is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other [serializable](https://en.wikipedia.org/wiki/Serialization) value). It is a very common data format used for asynchronous browser–server communication, including as a replacement for XML in some AJAX-style systems.

JSON is a language-independent data format. It was derived from JavaScript, but as of 2017 many programming languages include code to generate and parse JSON-format data. The official Internet media type for JSON is application/json. JSON filenames use the extension .json.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | JSON | | |  | | | [**Filename extension**](https://en.wikipedia.org/wiki/Filename_extension) | .json | | [**Internet media type**](https://en.wikipedia.org/wiki/Media_type) | application/json | | [**Type code**](https://en.wikipedia.org/wiki/Type_code) | TEXT | | [**Uniform Type Identifier (UTI)**](https://en.wikipedia.org/wiki/Uniform_Type_Identifier) | public.json | | **Type of format** | Data interchange | | **Extended from** | [JavaScript](https://en.wikipedia.org/wiki/JavaScript) | | [**Standard**](https://en.wikipedia.org/wiki/International_standard) | [RFC](https://en.wikipedia.org/wiki/Request_for_Comments_(identifier)) [8259](https://tools.ietf.org/html/rfc8259), [ECMA-404](http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf) | | **Website** | [json.org](http://json.org/) | |

{

"apiVersion": "v1",

"kind": "Pod"

}

{

"apiVersion": "v1",

"kind": "Pod",

**"metadata": {**

**"name": "oracle",**

**"labels": {**

**"app": "database"**

**}**

**}**

}

### YAML vs JSON vs XML

#### JSON

a subset of the JavaScript object notation syntax

* data stored in name/value pairs
* records separated by commas
* field names & strings are wrapped by double quotes

#### YAML

stands for YAML ain't markup language and is a superset of JSON

* .yml files begin with '---', marking the start of the document
* key value pairs are separated by colon
* lists begin with a hyphen
* YAML uses indentation to define the structured data. So the each block in the YAML is differentiated by the number of white spaces.

XML:

* XML is “*eXtensible Markup Language”* whereas YML is not a markup language.
* XML uses a tag to define the structure just like HTML.

**YAML vs JSON vs XML:**

* All three mentioned serialization language has same extension as their name. (.yaml for YAML, .json for JSON, .xml for XML). So it is easy to remember.
* In fact, file extensions are arbitrary for all the three data serialization standard. It is useful for the application and users know what files format, type of the content and their data structure.

|  |  |  |
| --- | --- | --- |
| Yaml (.yml or .yaml) | JSON (.json) | XML (.xml) |
| # EmpRecord  - Employee one:  id: emp01  name: Ramesh  job: DBA  skills:  - oracle  - rac  - asm | {  "EmpRecord": {  "Employee": [  {  "-id": "emp01",  "name": "Ramesh",  "job": "DBA",  "skills": " oracle,rac,asm"  },  {  "-id": "emp02",  "name": "Naveen",  "job": "Tester",  "skills": "QTP, selenium, Jmeter"  }  ]  }  } | <?xml version="1.0"?>  <EmpRecord>  <Employee id="emp01">  <name>Ramesh</name>  <job>DBA</job>  <skills>oracle, rac, asm</skills>  </Employee>    <Employee id="emp02">  <name>Naveen</name>  <job>Tester</job>  <skills>qtp, selenium, jmeter</skills>  </Employee>    </EmpRecord> |