YAML

Red Hat | 8,602,581 installs | ★★★½☆ (54) | Free

YAML Language Support by Red Hat, with built-in Kubernetes syntax support

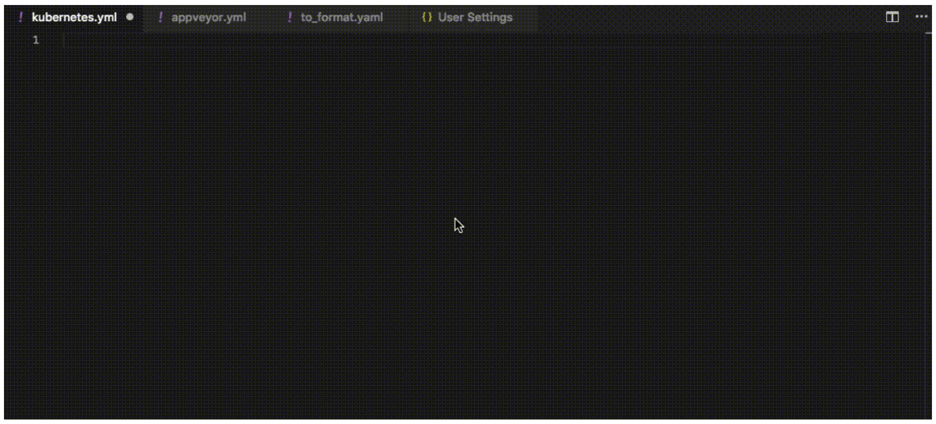
Install Trouble Installing?

CI passing Visual Studio Marketplace v1.8.0

YAML Language Support by Red Hat

Provides comprehensive YAML Language support to Visual Studio Code, via the yaml-language-server, with built-in Kubernetes syntax support.

Features



1. YAML validation:
   - Detects whether the entire file is valid yaml
   - Detects errors such as:
     - Node is not found
     - Node has an invalid key node type
     - Node has an invalid type
     - Node is not a valid child node
2. Document Outlining (Ctrl + Shift + O):
   - Provides the document outlining of all completed nodes in the file
3. Auto completion (Ctrl + Space):
   - Auto completes on all commands
   - Scalar nodes autocomplete to schema's defaults if they exist
4. Hover support:
   - Hovering over a node shows description *if provided by schema*
5. Formatter:
   - Allows for formatting the current file
   - On type formatting auto indent for array items

Auto completion and hover support are provided by the schema. Please refer to Language Server Settings to setup a schema

YAML version support

Starting from `1.0.0` the extension uses eemeli/yaml as the new YAML parser, which strictly enforces the specified YAML spec version. Default YAML spec version is `1.2`, it can be changed with `yaml.yamlVersion` setting.

Categories

Programming Languages Snippets Linters Formatters

Tags

autocompletion dockercompose kubernetes validation yaml

Works with

Universal, Web

Resources

Issues
Repository
Homepage
License
Changelog
Download Extension

Project Details

redhat-developer/vscode-yaml
Last Commit: 2 weeks ago
6 Pull Requests
131 Open Issues

More Info

Version	1.8.0
Released on	10/4/2017, 3:41:49 PM
Last updated	6/9/2022, 7:02:05 AM
Publisher	Red Hat
Unique Identifier	redhat.vscode-yaml
Report	Report Abuse

Extension Settings

The following settings are supported:

- `yaml.yamlVersion`: Set default YAML spec version (`1.2` or `1.1`)
- `yaml.format.enable`: Enable/disable default YAML formatter (requires restart)
- `yaml.format.singleQuote`: Use single quotes instead of double quotes
- `yaml.format.bracketSpacing`: Print spaces between brackets in objects
- `yaml.format.proseWrap`: Always: wrap prose if it exceeds the print width, Never: never wrap the prose, Preserve: wrap prose as-is
- `yaml.format.printWidth`: Specify the line length that the printer will wrap on
- `yaml.validate`: Enable/disable validation feature
- `yaml.hover`: Enable/disable hover
- `yaml.completion`: Enable/disable autocompletion
- `yaml.schemas`: Helps you associate schemas with files in a glob pattern
- `yaml.schemaStore.enable`: When set to true, the YAML language server will pull in all available schemas from JSON Schema Store
- `yaml.schemaStore.url`: URL of a schema store catalog to use when downloading schemas.
- `yaml.customTags`: Array of custom tags that the parser will validate against. It has two ways to be used. Either an item in the array is a custom tag such as "!Ref" and it will automatically map !Ref to a scalar, or you can specify the type of the object !Ref should be, e.g. "!Ref sequence". The type of object can be either scalar (for strings and booleans), sequence (for arrays), mapping (for objects).
- `yaml.maxComputedItems`: The maximum number of outline symbols and folding regions computed (limited for performance reasons).
- `yaml.disableDefaultProperties`: Disable adding not required properties with default values into completion text (default is false).
- `yaml.suggest.parentSkeletonSelectedFirst`: If true, the user must select some parent skeleton first before autocompletion starts to suggest the rest of the properties. When the YAML object is not empty, autocompletion ignores this setting and returns all properties and skeletons.

- `[yaml]`: VSCode-YAML adds default configuration for all YAML files. More specifically, it converts tabs to spaces to ensure valid YAML, sets the tab size, allows live typing autocompletion and formatting, and also allows code lens. These settings can be modified via the corresponding settings inside the `[yaml]` section in the settings:
  - `editor.tabSize`
  - `editor.formatOnType`
  - `editor.codeLens`

- `http.proxy`: The URL of the proxy server that will be used when attempting to download a schema. If it is not set or it is undefined no proxy server will be used.

- `http.proxyStrictSSL`: If true the proxy server certificate should be verified against the list of supplied CAs. Default is false.

In to your settings.

Adding custom tags

To use the custom tags in your YAML file, you need to first specify the custom tags in the setting of your code editor. For example, you can have the following custom tags:

```
"yaml.customTags": [
    "!Scalar-example scalar",
    "!Seq-example sequence",
    "!Mapping-example mapping"
]
```

The !Scalar-example would map to a scalar custom tag, the !Seq-example would map to a sequence custom tag, the !Mapping-example would map to a mapping custom tag.

You can then use the newly defined custom tags inside the YAML file:

```
some_key: !Scalar-example some_value
some_sequence: !Seq-example
  - some_seq_key_1: some_seq_value_1
  - some_seq_key_2: some_seq_value_2
some_mapping: !Mapping-example
  some_mapping_key_1: some_mapping_value_1
  some_mapping_key_2: some_mapping_value_2
```

Associating schemas

YAML Language support uses JSON Schemas to understand the shape of a YAML file, including its value sets, defaults and descriptions. The schema support is shipped with JSON Schema Draft 7.

We support schemas provided through JSON Schema Store. However, schemas can also be defined in a workspace.

The association of a YAML file to a schema can be done either in the YAML file itself using a modeline or in the User or Workspace settings under the property `yaml.schemas`.

Associating a schema in the YAML file

It is possible to specify a yaml schema using a modeline. Schema url can be a relative path. If a relative path is specified, it is calculated from yaml file path, not from workspace root path

```
# yaml-language-server: $schema=<urlToTheSchema>
```

Associating a schema to a glob pattern via yaml.schemas:

`yaml.schemas` applies a schema to a file. In other words, the schema (placed on the left) is applied to the glob pattern on the right. Your schema can be local or online. Your schema must be a relative path and not an absolute path. The entrance point for `yaml.schemas` is a location in user and workspace settings

When associating a schema it should follow the format below

```
"yaml.schemas": {
    "url": "globPattern",
    "Kubernetes": "globPattern"
}
```

e.g.

```
yaml.schemas: {
    "https://json.schemastore.org/composer": "/*"
}
```

e.g.

```
yaml.schemas: {
    "kubernetes": "/myYamlFile.yaml"
}
```

e.g.

```
yaml.schemas: {
    "https://json.schemastore.org/composer": "/*",
    "kubernetes": "/myYamlFile.yaml"
}
```

On Windows with full path:

```
yaml.schemas: {
    "C:\\Users\\user\\Documents\\custom_schema.json": "someFilePattern.yaml",
    "file:///C:/Users/user/Documents/custom_schema.json": "someFilePattern.yaml",
}
```

On Mac/Linux with full path:

```
yaml.schemas: {
    "/home/user/custom_schema.json": "someFilePattern.yaml",
}
```

Since `0.11.0` YAML Schemas can be used for validation:

```
 "/home/user/custom_schema.yaml": "someFilePattern.yaml"
```

A schema can be associated with multiple globs using a json array, e.g.

```
yaml.schemas: {
    "kubernetes": ["filePattern1.yaml", "filePattern2.yaml"]
}
```

e.g.

```
"yaml.schemas": {
    "http://json.schemastore.org/composer": ["/*"],
    "file:///home/johnd/some-schema.json": ["some.yaml"],
    "../relative/path/schema.json": ["/config*.yaml"],
    "/Users/johnd/some-schema.json": ["some.yaml"],
}
```

e.g.

```
"yaml.schemas": {
    "kubernetes": ["/myYamlFile.yaml"]
}
```

e.g.

```
"yaml.schemas": {
    "http://json.schemastore.org/composer": ["/*"],
    "kubernetes": ["/myYamlFile.yaml"]
}
```

Multi root schema association:

You can also use relative paths when working with multi-root workspaces.

Suppose you have a multi-root workspace that is laid out like:

```
My_first_project:
   test.yaml
   my_schema.json
My_second_project:
   test2.yaml
   my_schema2.json
```

You must then associate schemas relative to the root of the multi root workspace project.

```
yaml.schemas: {
    "My_first_project/my_schema.json": "test.yaml",
    "My_second_project/my_schema2.json": "test2.yaml"
}
```

`yaml.schemas` allows you to specify JSON schemas that you want to validate against the YAML you write. *Kubernetes* is a reserved keyword field. It does not require a URL, as the language server will provide that. You need the keyword `kubernetes` and a glob pattern.

Mapping a schema in an extension

- Supports `yamlValidation` point, which allows you to contribute a schema for a specific type of YAML file (Similar to jsonValidation) e.g.

```
{
  "contributes": {
    "yamlValidation": [
      {
        "fileMatch": "yourfile.yml",
        "url": "./schema.json"
      }
    ]
  }
}
```

Data and Telemetry

The `vscode-yaml` extension collects anonymous usage data and sends it to Red Hat servers to help improve our products and services. Read our privacy statement to learn more. This extension respects the `redhat.telemetry.enabled` setting, which you can learn more about at https://github.com/redhat-developer/vscode-redhat-telemetry#how-to-disable-telemetry-reporting

How to contribute