52Kubernetes 系列（四十九）使用 YQ 解析 Kubernetes 清单文件

Kubernetes 系列（四十九）使用 YQ 解析 Kubernetes 清单文件

yq 是一个主要关注 YAML 数据处理和高亮显示的工具，使我们能够更有效地查询和操作YAML。

在处理Kubernetes资源方面，“yq”是仅次于kubectl的最强大的工具，特别是在操作YAML文件方面，这是无可替代的。

为什么我会有这样的信念?从下面的演示和查询、更新和删除' yq '处理的最佳实践中，您将得到答案。

基本信息

本文中提到的“yq”工具是 mikefarah/yq ，而不是 kislyuk/yq 。 mikefarah/yq 具有强大的操作符并提供更广泛的支持，而 kislyuk/yq 只能支持YAML读取。

```
yq eval [expression] [yaml_file1]… [flags]
```

查询、更新、删除等操作在表达式和可使用的操作符上有所不同。由于eval是默认命令， `yq e '.metadata.name' a.yaml` 和 `yq '.metadata.name' a.yaml` 效果是一样的。

`yq` 以类似于树解析的方式解析YAML文件，将YAML文件划分为不同的节点类型，这些节点类型彼此嵌套。

总共三种节点类型：

- **Scalar** , the common node, such as `.metadata.name` , whose value can be `string` , `int` , `boolean` , etc.
- **标量** ，基础常用节点，如 `.metadata.name` ，其值可以是 `string` , `int` , `boolean` 等。
- **数组** ，例如 `.spec.containers` 和 `.spec.volumes` 都是数组。前面有 `-` 的每个项的值通常是一个kv结构的YAML，或者一个单独的字符串，例如容器中的 ENV 设置。
- **Map** , `.metadata.labels` 以及 `.metadata.annotations` 都是 Map 类型

查询

查询主要是遍历 YAML，或者换句话说，用 `.(dot)` 操作符搜索YAML文件中的键和值。

字段筛选

使用 `.key1.key2.xxx` 的表达式，我们可以得到YAML支持的所有数据格式，如通用字符串、数组、map、对象等。下面列出了一些合理的yq命令。

```
$ yq e '.metadata.name' pod.yaml
test-pod
$ yq e '.metadata.labels' pod.yaml# return labels map
l1: v1
app.name: test-pod
$ yq e '.metadata pod.yaml# return whole metadata field
name: test-pod
labels:
l1: v1
app.name: test-pod
annotations:
com.team.owner: abc
```

通常，我们用双引号标记键，以忽略由特殊字符如 `app.name` 引起的问题。

Array/map 扩展查询

与只有一个值的字符串不同，数组在YAML中可以有多个表达式。在YAML中，只需一个简单的键就可以获得完全相同的返回值，并且通过一些特殊处理可以获得更高精度的更多结果。

• 在键后面加上 `[]` ，返回的map或-可以被删除。

```
$ yq e '.metadata.labels[]' pod.yaml
v1
test-pod
$ yq e '.spec.containers[]' pod.yaml
name: c1
image: nginx
volumeMounts:
  - name: html
mountPath: /usr/share/nginx/html
name: c2
image: debian
```

• 使用collect ([key/index])，可以从Map或数组获得一个或多个键或索引值，如果索引或键不存在，则返回null。

```
$ yq e '.metadata.labels["l1"]' pod.yaml
v1
# Get multiple keys
$ yq e '.metadata.labels["l1", "app", "app.name"]' pod.yaml
v1
null
test-pod

$ yq e '.spec.containers[1]' pod.yaml
name: c2
image: debian
volumeMounts:
- name: html
mountPath: /html
command: ["/bin/sh", "-c"]
args:
- while true; do date >> /html/index.html; sleep 1; done
```

当需要搜索多个键或索引时，`pick` 操作符用于获取映射/数组中键和索引的对应值。

```
$ yq e '.metadata.labels | pick(["com.team.owner", "l1", "app.name"])' pod.yaml
l1: v1
app.name: test-pod
```

`with_entries` 操作符可以处理 Map，允许我们获取Map的键/值，然后执行后续处理。当与 `select` 操作符协作时，它可以实现非常强大的过滤查询。

条件查询

`yq` 支持两个常用的查询操作符，`select` 和 `contains`，它们通常与 `.key` 结合使用：

- `has(key)`，如果键存在返回true。
- `contains(obj)`，返回true/false。' obj '可以是一个简单的字符串(查询任何匹配的键和值)，KV组合，数组，对象等。
- `select(.=="some pattern here")`。`.` 表示当前字段，如果存在，则返回否则返回'null"。`select` 可以与比较操作符(' > ', ' < ', ' == '等)或布尔操作符(and, or, not等)一起使用，它还支持通配符(' *')。

```
$ yq '.metadata.labels | has("l1")' pod.yaml
true
$ yq '.metadata.labels | contains({"l1":"v1"})' pod.yaml
true
# combine select and has
$ yq '.metadata.labels | select(. | has("l1")) ' pod.yaml
l1: v1
app.name: test-pod
$ yq '.metadata.name | select(. == "test")' pod.yaml# no match
# wildcard match
yq '.spec.containers[] | select(.name == "*1" )' pod.yaml
name: c1
image: nginx
volumeMounts:
- name: html
mountPath: /usr/share/nginx/html
# select, with_entries and not, search for labels don't match
yq '.metadata.labels | with_entries(select(.key == "l1" |not))' pod.yaml
app.name: test-pod
```

对于更复杂的场景，如匹配正则表达式，`select` 总是与字符串操作符组合，如 `test`，`match`，`capture` 等。

```
# test regular
yq '.spec.containers[] | select(.image | test("nginx|debian"))' pod.yaml # return all containers# capture
yq '.spec.containers[] | select(.command | .[] | capture("/bin/sh"))' pod.yaml
```

遍历及变量

- 如何读取上一层的信息?
- 如何比较不同层之间的内容?例如，验证 `volumeMounts` 是否与 `spec.volumes` 一致

`yq` 提供了答案:

- `parent`，返回上一层的节点内容。
- `variable`，用于定义在后续管道中使用的变量。

```
yq '.spec.volumes[] | .name |  parent' pod.yaml
name: html
emptyDir: {}

# find all volumeMounts match the volumes
yq '.spec | .volumes[].name as $volumeName | .containers[] | select(.volumeMounts[] | .name == $volumeName
```

复杂流运算符

`yq` 表达式还支持一组用于管道内处理的操作符(`map` , `flattern` , `groupby` , `union` , `reduce` 等)，类似于函数式编程。' map '、' flattern '和' groupby '都是用于数组节点处理的，其中' flattern '和' groupby '很少使用，很难想象它们会在kubernetes相关的YAML中使用。因此，我建议理解它们所扮演的角色，只在需要时检查它们的文档。

而map是你应该掌握的操作符。它结合' string '操作符或' math '操作符来处理数组中的项，或添加组合字符串。例如，将容器中的名称修改为大写。

```
yq e '.spec.containers | map(.name | upcase) ' pod.yaml
```

更新

查询、搜索和过滤现有文件可以满足我们的大部分需求。但是，如果我们需要更新或创建YAML呢? `yq` 已经准备好了，别着急。

首先，让我们看看一组相关的标志。

- `i/-inplace` 。默认情况下，结果被发送到标准输出，对当前文件的修改可以通过' i '完成。虽然我们需要文件重定向操作符(' > ')来输出到一个新文件。
- `o/-output-format` 。该输出格式默认为YAML，但也支持' json/j '、' xml/x '和' props/p '。
- `I/-indent` ， YAML的缩进。默认值是2，这也是YAML规范的标准。
- `-from-file` ，从文件中读取表达式。

更新/新增 Key

`yq` 支持YAML更新，通常模式为 `yq e -i '.key1 | .key2 … = new_value' file` 如果要更新的字段不存在，' update '操作将变为' add '。除了最常见的' = '运算符外，' yq '还支持' |= '等其他运算符。

它们在大多数情况下具有相同的效果，除了' |= '可以基于旧值更新。

```
# add suffix to pod’s name using |= operator
yq '.metadata.name |= . + "abc" | .metadata.name' pod.yaml
test-podabc
```

默认情况下，我们在末尾添加新字段。但是在index:0处添加它们也是可行的，例如将nodeName字段添加到.spec中。

```
yq  '.spec = {"nodeName":"master"} + .spec ' pod.yaml
```

至于向数组和Map添加新内容，我们可以使用+或+=操作符，例如在. matdata .labels的起始位置添加一个新标签。

```
yq '.metadata.labels |= {"pos-1":"val-1"} + . ' pod.yaml
pos-1: val-1
l1: v1
app.name: test-pod
```

同时，可以使用一组字符串操作符(sub/replace、split、join、大写、小写等)进行更新。

从ENV读取变量更新

这是一种非常常见的操作，因为我们使用各种bash/kubectl命令来获取用于更新yq中的字段的有用信息。

Comment 更新

Comment 操作符允许您以以下3种方式更新注释。

- `line_comment` 更新某些行中的注释
- `head_comment` 在YAML文件的开头添加一个注释，后面跟着一个空行。
- `foot_comment` 在YAML文件的末尾添加一个注释，上面有一个空行。

```
yq '.metadata.annotations["com.team.owner"] line_comment="owner squad"' pod.yaml
yq '. head_comment="Pod example start"'  pod.yaml
yq '. foot_comment="Pod example end"' pod.yaml
```

删除

虽然删除可以单独使用，但一般与查询和条件查询结合使用。支持删除的操作符包括' subtract '和' del '。

- `del` 是最常用的' delete '操作符，用于删除匹配的键。
- `subtract` 是一个数学减法运算符，但也可以用于从数组中删除项。

删除一个字段

如果知道字段路径，就可以直接删除它，比如删除.spec中的nodeName。

```
yq -i 'del(.spec.nodeName)' pod.yaml
```

当然，我们可以在select/match/has之后执行delete。

```
# only deletes the labels when select matches
yq 'del(select(.metadata.name == "test-pod") | .metadata.labels)' pod.yaml
```

删除数组元素

删除 pod 中的 nginx 容器

```
yq '.spec.containers - [{"image":"nginx", "name":"c1", "volumeMounts":[{"name":"html","mountPath":"/usr/sh
```

在这里，需要声明整个数组项的内容。太复杂了，对吧?select + del可以让它更简单。

```
yq 'del(.spec.containers[] | select(.image == "nginx"))' pod.yaml
```

输出

有一些标记可用于增强输出。

- `C` 强制输出颜色。
- `in` 设置输出缩进为n个空格。
- `P` 代表漂亮的打印。
- `v` 支持详细输出，并可用于调试。

它默认输出到标准输出并以漂亮的格式打印。但是样式操作符允许将打印样式更改为其他5种格式:双格式、单格式、标记格式、文字格式和折叠格式。下面的示例输出一个json样式的字符串。

总结

`yq` 命令作为集群治理的有效工具，可以解放我们的双手，将我们从枯燥的复制和粘贴中拯救出来。它是用 Go 写的，这提高了我对它实现的兴趣。之后，我也会尝试添加一些功能来满足自己的需求。

感谢阅读!

欢迎关注我的公众号“ **云原生拓展** ”，原创技术文章第一时间推送。