YAML marshaling and unmarshaling support for Go

build passing

kubernetes-sigs/yaml is a permanent fork of ghodss/yaml.

Introduction

A wrapper around go-yaml designed to enable a better way of handling YAML when marshaling to and from structs.

In short, this library first converts YAML to JSON using go-yaml and then uses `json.Marshal` and `json.Unmarshal` to convert to or from the struct. This means that it effectively reuses the JSON struct tags as well as the custom JSON methods `MarshalJSON` and `UnmarshalJSON` unlike go-yaml. For a detailed overview of the rationale behind this method, see this blog post.

Compatibility

This package uses go-yaml and therefore supports everything go-yaml supports.

Caveats

Caveat #1: When using `yaml.Marshal` and `yaml.Unmarshal`, binary data should NOT be preceded with the `!!binary` YAML tag. If you do, go-yaml will convert the binary data from base64 to native binary data, which is not compatible with JSON. You can still use binary in your YAML files though - just store them without the `!!binary` tag and decode the base64 in your code (e.g. in the custom JSON methods `MarshalJSON` and `UnmarshalJSON`). This also has the benefit that your YAML and your JSON binary data will be decoded exactly the same way. As an example:

```
BAD:
    exampleKey: !!binary gIGC

GOOD:
    exampleKey: gIGC
... and decode the base64 data in your code.
```

Caveat #2: When using `YAMLToJSON` directly, maps with keys that are maps will result in an error since this is not supported by JSON. This error will occur in `Unmarshal` as well since you can't unmarshal map keys anyways since struct fields can't be keys.

Installation and usage

To install, run:

```
$ go get sigs.k8s.io/yaml
```

And import using:

```
import "sigs.k8s.io/yaml"
```

Usage is very similar to the JSON library:

```go
package main

import (
    "fmt"

    "sigs.k8s.io/yaml"
)

type Person struct {
    Name string `json:"name"` // Affects YAML field names too.
    Age  int    `json:"age"`
}

func main() {
    // Marshal a Person struct to YAML.
    p := Person{"John", 30}
    y, err := yaml.Marshal(p)
    if err != nil {
        fmt.Printf("err: %v\n", err)
        return
    }
    fmt.Println(string(y))
    /* Output:
    age: 30
    name: John
    */

    // Unmarshal the YAML back into a Person struct.
    var p2 Person
    err = yaml.Unmarshal(y, &p2)
    if err != nil {
        fmt.Printf("err: %v\n", err)
        return
    }
    fmt.Println(p2)
    /* Output:
    {John 30}
    */
}
```

`yaml.YAMLToJSON` and `yaml.JSONToYAML` methods are also available:

```go
package main

import (
    "fmt"

    "sigs.k8s.io/yaml"
)
```

```go
func main() {
    j := []byte(`{"name": "John", "age": 30}`)
    y, err := yaml.JSONToYAML(j)
    if err != nil {
        fmt.Printf("err: %v\n", err)
        return
    }
    fmt.Println(string(y))
    /* Output:
    name: John
    age: 30
    */
    j2, err := yaml.YAMLToJSON(y)
    if err != nil {
        fmt.Printf("err: %v\n", err)
        return
    }
    fmt.Println(string(j2))
    /* Output:
    {"age":30,"name":"John"}
    */
}
```