A simple way to migrate configuration files
or rather, non-bloat configuration tool
10 March 2016

Luka Napotnik
Platform Developer, Visionect d.o.o.
Challenges

- non-incremental upgrades: from version 1.x to 4.x
- structural changes: block changes
Idea

- using reflection
- recursive walk
- building a migration path
Should We Migrate?

- Retrieve the current document version

```go
var objmap map[string]*json.RawMessage
json.Unmarshal(config, &objmap); err != nil {
    v := "" // Current document version
    json.Unmarshal(*objmap["ConfigVersion"], &v)
```
The Interface

type Configuration interface {
    Version() string
    Migrations() []string
    Migrate(from Configuration) error
    Depends() string
}

- every possible configuration

Configs = make(map[string]Configuration)
We Need Rules

- A function that accepts a value that is being migrated and returns an error if the migration couldn't be done

```go
type MigrationRule func(value reflect.Value) error
```

- rules are provided by target

```go
rules map[string]MigrationRule
```

- what we now have
Traversing And Rules

- run handler if we hit a rule
- otherwise, copy values

```go
func traverse(src, dest) {
    switch src.Kind() {
    case reflect.Ptr:
        // dereference pointers
        fallthrough
    case reflect.Struct
        if rule, ok := rules[srcName]; ok { // apply rule to whole struct
            err := rule(src)
        } else { // traverse elements
            traverse(...
        }
    default:
        // find value in target
        value := findField(srcName, srcType, dest)
        value.Set(src)
    }
}
Traversing And Rules - a problem

- values can have same name and same type

```go
type CfgV1 struct {
    Clients []struct {
        Host string
    }
    Host string
}
type CfgV2 struct {
    Host string
}
```

- shows up as a problem in previous attempts to traverse
- keep a map of already migrated values (map[parent]values)
Build Migration Path

- We want a sequence of migration we need to perform to get from version A to C

```go
sequence := []Configuration{}

- Go from target downwards to the minimum dependency that is required to migrate

```for depends := Configs[target.Deps()]; depends != nil; {
    if len(depends.Deps()) > 0 {
        sequence = append([]Configuration{depends}, sequence...)
    }
    target = depends
    depends = Configs[target.Deps()]
}
Migrate!

- migrate from A to C

```go
oldConfig := source // current configuration
for i, _ = range sequence {
    // Incremental upgrade
    if err = sequence[i].Migrate(oldConfig); err != nil {
        return err
    }
    oldConfig = sequence[i]
}
```

- the sequence is A->B, B->C
Thanks!

- SEARCHING FOR A JOB?
  
  www.visionect.com/about#careers (https://www.visionect.com/about#careers)
Thank you

Luka Napotnik
Platform Developer, Visionect d.o.o.
luka.napotnik@visionect.com
http://www.visionect.com
@napsy