

Viper и Cleanenv для конфигурирования структур vs Велосипед

АНОСОВ КОСТЯ



Viper и Cleanenv для конфигурирования структур vs Велосипед

АНОСОВ КОСТЯ





Предистория

История началась до Контура

Предистория

История началась до Контура

Проект - 36 микросервисов и сервисов

Предистория

История началась до Контура

Проект - 36 микросервисов и сервисов

Стек: Python, Golang, Kotlin

Глоссарий

Точечная конфигурация - конфигурация каждого отдельного поля

Глоссарий

Точечная конфигурация - конфигурация каждого отдельного поля

Источник конфигурации - ресурс

Какие бывают источники

1. json, toml, yaml, ini, json

Какие бывают источники

1. json, toml, yaml, ini, json
2. дефолтные значения, переменные окружения

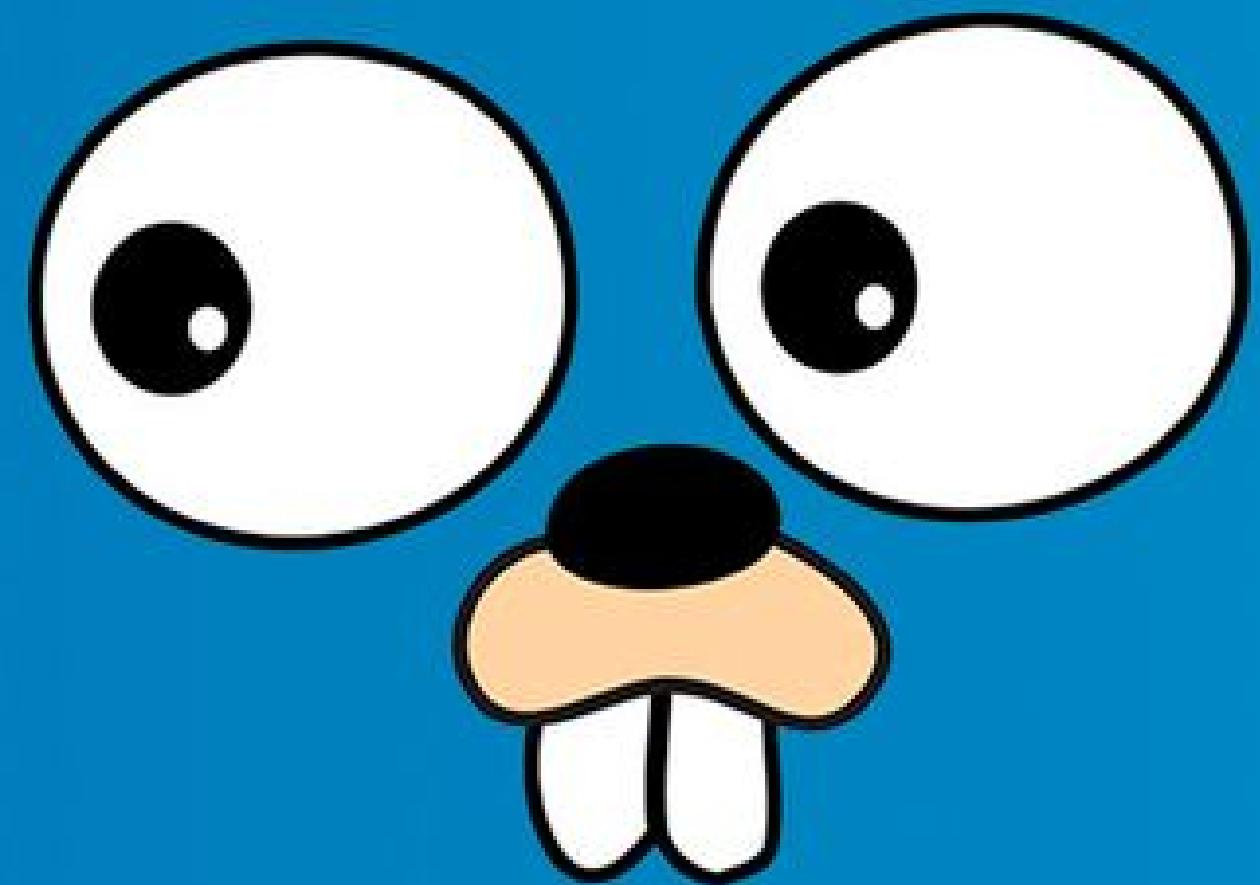
Какие бывают источники

1. json, toml, yaml, ini, json
2. дефолтные значения, переменные окружения
3. vault

Какие бывают источники

1. hocon, toml, yaml, ini, json
2. дефолтные значения, переменные окружения
3. vault
4. config file server(spring cloud config server), key\value store(consul, etcd, firestore)

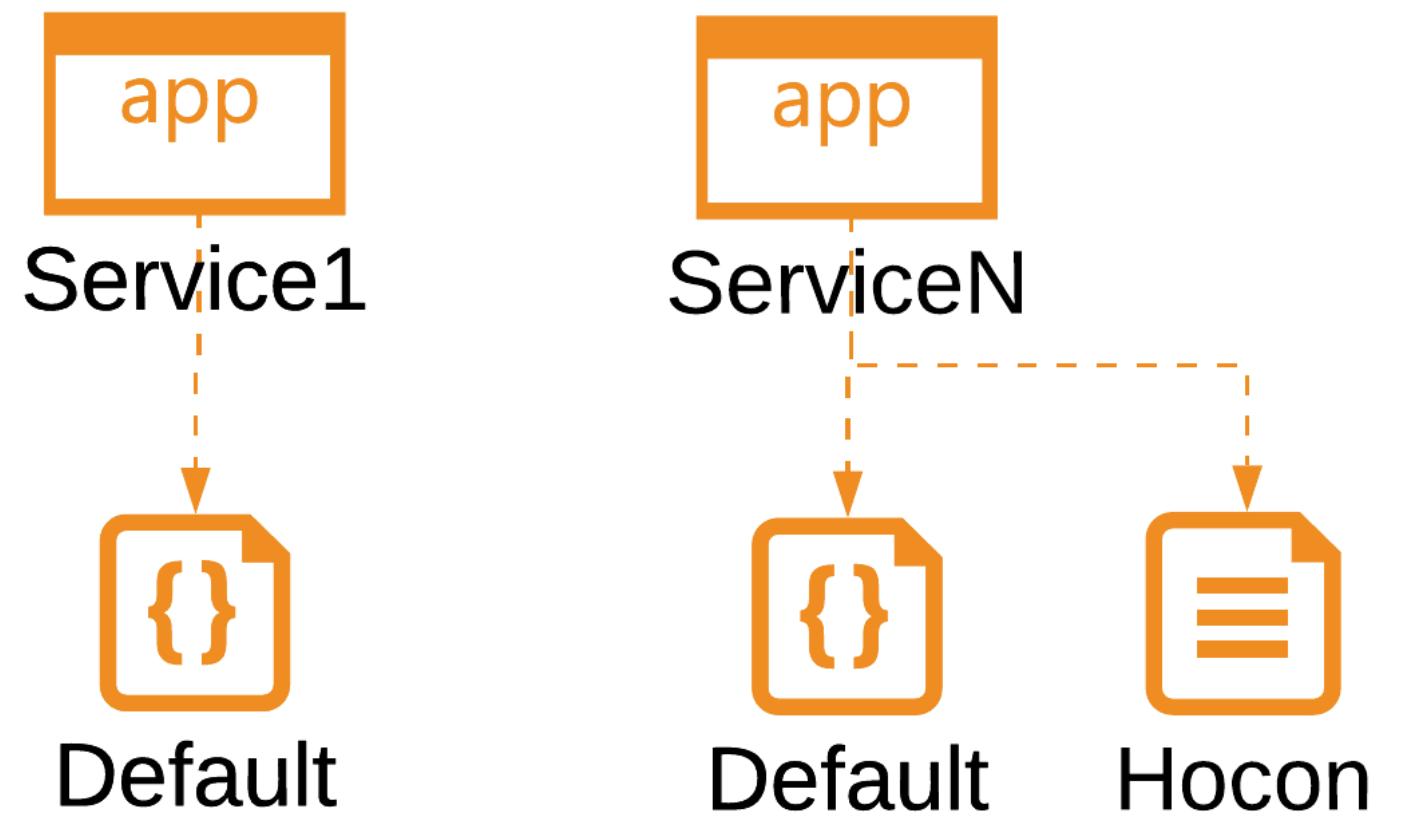
Так а причём здесь го?



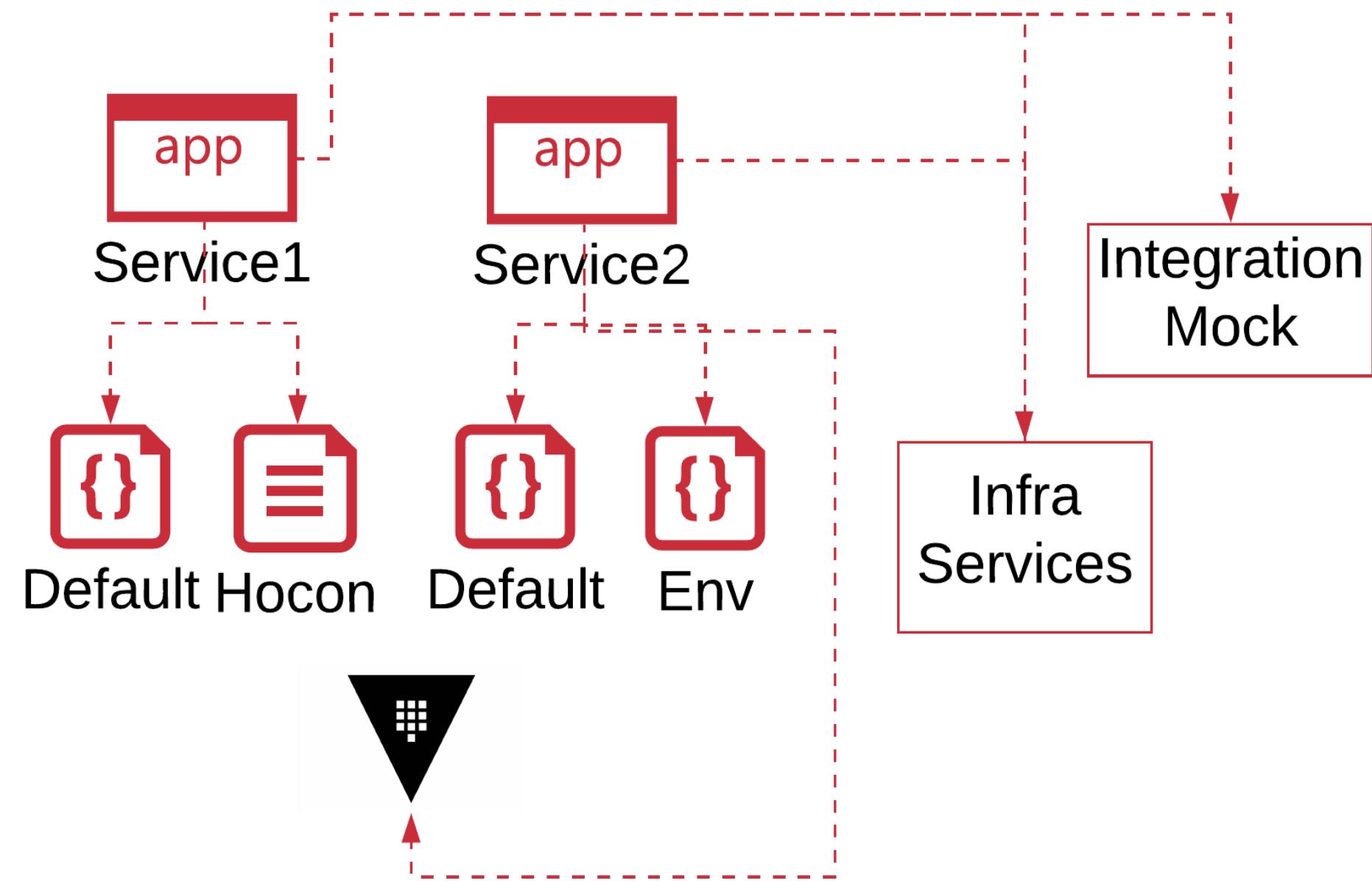
Главная боль - Стейджинги



Dev



Test



Prod

Критерии отбора решения

1. Есть ли точечная конфигурация?

Критерии отбора решения

1. Есть ли точечная конфигурация?
2. vault, hocon, default, env, spring config server

Критерии отбора решения

1. Есть ли точечная конфигурация?
2. vault, hocon, default, env, spring config server
3. Понятная структура ошибок

Критерии отбора решения

1. Есть ли точечная конфигурация?
2. vault, hocon, default, env, spring config server
3. Понятная структура ошибок
4. Небольшой объём написания инфраструктурного кода

Конфигурация окружения

Мапим значения из источников

Что будем конфигурировать?

```
type ServiceConfiguration struct {  
    Main          MainConfig  
    Db            DatabaseConfig  
    ServiceA     IntegrationConf  
    BusinessSecret BusinessConf  
}
```

Что будем конфигурировать?

```
type MainConfig struct {  
    IsKubernetes      bool  
    ServiceName       string  
    StrategyRequest   string  
}
```

Что будем конфигурировать?

```
type ServiceConfiguration struct {  
    Main          MainConfig  
    Db            DatabaseConfig  
    ServiceA     IntegrationConf  
    BusinessSecret BusinessConf  
}
```

Что будем конфигурировать?

```
type DatabaseConfig struct {  
    Replicas []string  
    Username string  
    Password string  
}
```

Что будем конфигурировать?

```
type ServiceConfiguration struct {  
    Main          MainConfig  
    Db            DatabaseConfig  
    ServiceA     IntegrationConf  
    BusinessSecret BusinessConf  
}
```

Что будем конфигурировать?

```
type IntegrationConf struct {  
    ApiKey           string  
    StrategyRequest string  
    RateLimit       int16  
}
```

Что будем конфигурировать?

```
type ServiceConfiguration struct {  
    Main          MainConfig  
    Db            DatabaseConfig  
    ServiceA     IntegrationConf  
    BusinessSecret BusinessConf  
}
```

Что будем конфигурировать?

```
type BusinessConf struct {  
    PrivateKeyPath string  
    PrivateCertPath string  
}
```

Как будет выглядеть конфигурация окружения?

```
sources:  
  vault:  
    Address: $VAULT_ADDRESS  
    Token: $VAULT_TOKEN  
    Path: $VAULT_PATH  
  
  json:  
    pathFile: $PATH_TO_CONFIG  
  hocon:  
    pathFile: $PATH_TO_CONFIG  
  env:  
  
  default:
```

А теперь посмотрим на код, который придётся написать руками

```
func ReadConfig(configSources string) (
    ServiceConfiguration,
    error) {
    configSource, err := readSourceConfig(configSources)
    if err != nil {
        return ServiceConfiguration{}, err
    }
    // на следующем слайде
}
```

Читаем настройки окружения

```
func readSourceConfig(file string) (
    map[string]interface{}, error) {
    readBytes, errReading := ioutil.ReadFile(file)
    if errReading != nil {
        return nil, errReading
    }
    var result map[string]interface{}
    if err := yaml.Unmarshal(readBytes, &result); err != nil {
        return nil, err
    }
    return result, nil
}
```

Продолжим описывать ReadConfig

```
func ReadConfig(configSources string) (
    ServiceConfiguration,
    error) {
    // чтение конфигурации окружения
    var configuration ServiceConfiguration
    sources := configSource["sources"]
        .(map[interface{}][]interface{})
    // продолжение на следующем слайде
}
```

Разбираем источники

```
for key, source := range sources {
    switch key {
    case "json":
        if err := configuration.addFile(source.(map[string]interface{})["pathToFile"].(string)); err != nil {
            return ServiceConfiguration{}, err
        }
    break
    ...
}
```

Метод для чтения и мержа с получателем конфигурации

```
func (config *ServiceConfiguration) addFile(file string)
error {
    configurationByFile, err := readFile(file)
    if err != nil {
        return err
    }
    return mergo.MergeWithOverwrite(config, &configurationByF:
}
```

Как будем читать конфигурацию из файлика

```
func readFile(file string) (ServiceConfiguration, error) {
    readBytes, errReading := ioutil.ReadFile(file)
    if errReading != nil {
        return ServiceConfiguration{}, errReading
    }
    var data ServiceConfiguration
    errUnmarshaling := json.Unmarshal(readBytes, &data)
    if errUnmarshaling != nil {
        return ServiceConfiguration{}, errUnmarshaling
    }
    return data, nil
```

Мержим с нашей конфигурацией

```
func (config *ServiceConfiguration) addFile(file string)
error {
    configurationByFile, err := readFile(file)
    if err != nil {
        return err
    }
    return mergo.MergeWithOverwrite(config,
        &configurationByFile)
}
```

Идентично для оставшихся источников

```
case "vault":  
    if err := configuration.addVault(source  
        .(map[interface{}]interface{})); err != nil {  
        return ServiceConfiguration{}, err  
    }  
    break
```

Конфигурируем из волта

```
func (config *ServiceConfiguration) addVault(  
    dataFromEnv map[interface{}][]interface{}) error {  
    configurationVault := ConfigVault{  
        VaultAddress: dataFromEnv["Address"].(string),  
        VaultPath:    dataFromEnv["Path"].(string),  
        VaultToken:   dataFromEnv["Token"].(string),  
    }  
    ...  
}
```

Конфигурируем из волта

```
...
if err := configurationVault.connectToVault(); err != nil {
    return err
}

configFromVaultDatabase, err := vaultConfig.getPath(configurationVault.Vau
if err != nil {
    return err
}

config.BusinessSecret.PrivateCertPath = configFromVaultDatabase["certPath"].
config.BusinessSecret.PrivateKeyPath = configFromVaultDatabase["keyPath"].(s
config.ServiceA.ApiKey = configFromVaultDatabase["ApiKey"].(string)
return nil
}
```

Подключение к волту

```
func (cfg *ConfigVault) connectToVault() error {
    conn, errConnection := vault.NewClient(cfg.VaultAddress,
        vault.WithCaPath(""),
        vault.WithAuthToken(cfg.VaultToken))
    if errConnection != nil {
        return errConnection
    }
    conn.SetToken(cfg.VaultToken)
    cfg.Connection = conn
    log.Println("Connection are initialized")
    return nil
}
```

Конфигурируем из волта

```
...
if err := configurationVault.connectToVault(); err != nil {
    return err
}

configFromVaultDatabase, err := vaultConfig.getPath(configurationVault.Vau
if err != nil {
    return err
}

config.BusinessSecret.PrivateCertPath = configFromVaultDatabase["certPath"].
config.BusinessSecret.PrivateKeyPath = configFromVaultDatabase["keyPath"].(s
config.ServiceA.ApiKey = configFromVaultDatabase["ApiKey"].(string)
return nil
}
```

Получение данных из волта

```
func (cfg ConfigVault) getPath(path string) (
    map[string]interface{}, error) {
    pathSecrets, err := cfg.Connection.Logical().Read(path)
    if err != nil {
        return nil, err
    }
    return pathSecrets.Data, nil
}
```

Конфигурируем из волта

```
...
if err := configurationVault.connectToVault(); err != nil {
    return err
}

configFromVaultDatabase, err := vaultConfig.getPath(configurationVault.Vau
if err != nil {
    return err
}

config.BusinessSecret.PrivateCertPath = configFromVaultDatabase["certPath"].
config.BusinessSecret.PrivateKeyPath = configFromVaultDatabase["keyPath"].(s
config.ServiceA.ApiKey = configFromVaultDatabase["ApiKey"].(string)
return nil
}
```


Для переменных окружения

```
case "env":  
    if err := configuration.addEnv(); err != nil {  
        return ServiceConfiguration{}, err  
    }
```

Опишем чтение из переменных окружения

```
func (config *ServiceConfiguration) addEnv() error {
    configurationByEnv := ServiceConfiguration{
        Main: MainConfig{
            ServiceName:     os.Getenv("SERVICE_NAME"),
            StrategyRequest: os.Getenv("STRATEGY_REQUEST"),
        },
        Db: DatabaseConfig{
            Replicas: strings.Split(os.Getenv("MONGO_REPLICAS"), ","),
        },
        ServiceA: IntegrationConf{
            StrategyRequest: os.Getenv("GOOGLE_AI_STRATEGY_REQUEST"),
        },
    }
    return mergo.MergeWithOverwrite(config, configurationByEnv)
}
```

Для значений по умолчанию

```
case "default":  
    configuration.addDefault()  
}
```

Дефолтные значения конфигурации

```
func (config *ServiceConfiguration) addDefault() {  
    config.Main.StrategyRequest = "Sequential"  
    config.ServiceA.StrategyRequest = "Sequential"  
}
```

Удовлетворило ли такое решение?

1. 167 строк кода

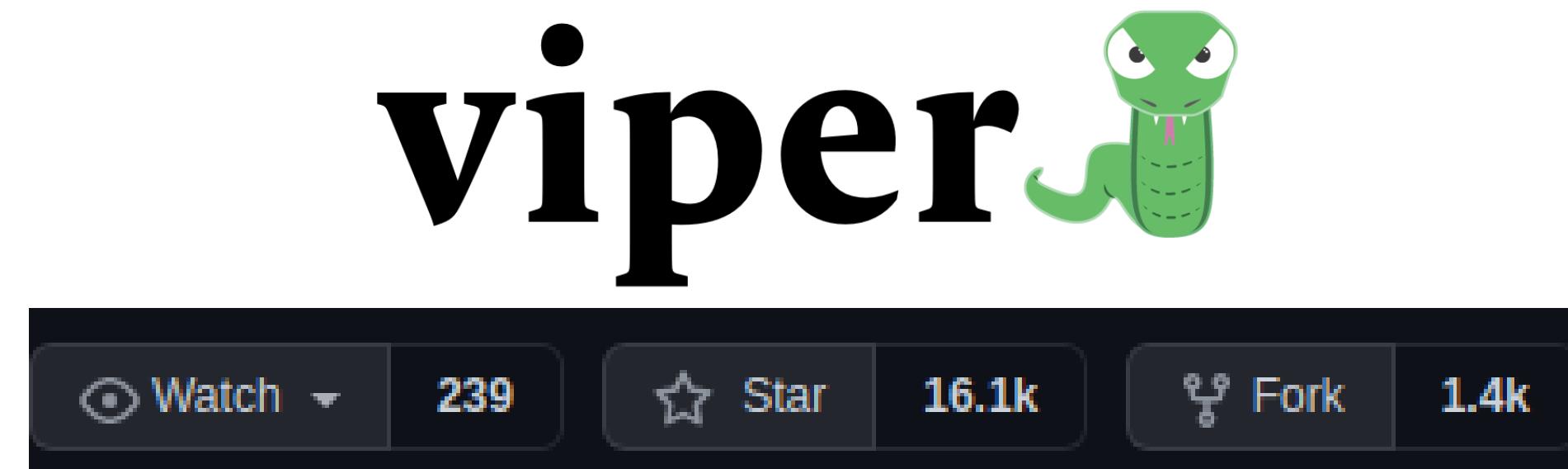
Удовлетворило ли такое решение?

1. 167 строк кода
2. Точечной конфигурации нет

Удовлетворило ли такое решение?

1. 167 строк кода
2. Точечной конфигурации нет
3. Дополнительный код

Viper



Viper



1 Шаг настройка

2 Шаг получение значений

Сконфигурируем viper

```
func settingViper() {  
    viper.SetConfigName("ServiceConfiguration")  
    viper.SetConfigType("json")  
    viper.AddConfigPath(".")  
    viper.BindEnv("service_name", "Main.ServiceName")  
    viper.BindEnv("strategy_request", "Main.StrategyRequest")  
    viper.SetDefault("Main.StrategyRequest", "Sequential")  
    viper.BindEnv("google_ai_strategy_request", "ServiceA.StrategyReques  
    viper.SetDefault("ServiceA.StrategyRequest", "Sequential")  
    viper.BindEnv("mongo_replicas", "Db.Replicas")  
}
```

Опишем функционал по чтению конфигурации

```
func ReadConfig(path string) (ServiceConfiguration, error) {
    sourceConfig := readSourceConfig(path)
    var configuration ServiceConfiguration
    if err := viper.Unmarshal(&configuration); err != nil {
        return ServiceConfiguration{}, err
    }
    for key, _ := range data {
        switch key {
        case "vault":
            if err := configuration.addVault(configurationSource.GetStringMap(
                return ServiceConfiguration{}, err
            )
        }
    }
}
```

Прочтём конфигурацию окружения

```
func readSourceConfig(fileName string)(map[string]interface{}) {  
    configurationSource := viper.New()  
    configurationSource.AddConfigPath(".")  
    configurationSource.SetConfigFile(fileName)  
    data := configurationSource.GetStringMap("sources")  
}
```

Описываем функциона по чтению конфигурации

```
func ReadConfig(path string) (ServiceConfiguration, error) {
    sourceConfig := readSourceConfig(path)
    var configuration ServiceConfiguration
    if err := viper.Unmarshal(&configuration); err != nil {
        return ServiceConfiguration{}, err
    }
    for key, _ := range data {
        switch key {
        case "vault":
            if err := configuration.addVault(configurationSource.G...
```

Чтение из Vault

```
func (config *ServiceConfiguration) addVault(dataFromEnv  
map[interface{}]{}) error {  
    configurationVault := ConfigVault{  
        VaultAddress: dataFromEnv["Address"].(string),  
        VaultPath:    dataFromEnv["Path"].(string),  
        VaultToken:   dataFromEnv["Token"].(string),  
    }  
  
    if err := configurationVault.connectToVault(); err != nil {  
        return err  
    }  
  
    configFromVaultDatabase, err := vaultConfig  
        .getPath(configurationVault.VaultPath)  
    if err != nil {  
        return err  
    }  
    config.BusinessSecret.PrivateCertPath = configFromVaultDatabase["certPath"].(string)  
    config.BusinessSecret.PrivateKeyPath = configFromVaultDatabase["keyPath"].(string)  
    config.ServiceA.ApiKey = configFromVaultDatabase["ApiKey"].(string)  
    return nil
```

Теперь к метрикам

1. 113 строчек кода

Теперь к метрикам

1. 113 строчек кода
2. Точечной конфигурации нет

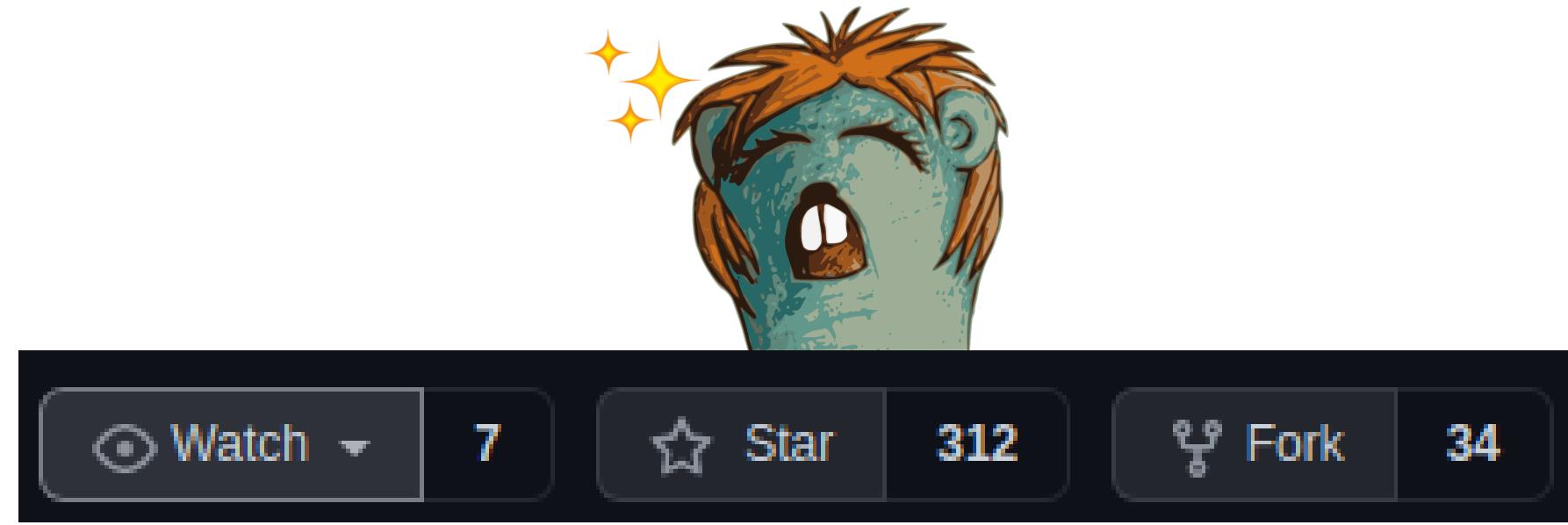
Теперь к метрикам

1. 113 строчек кода
2. Точечной конфигурации нет
3. Дополнительный инфровый код

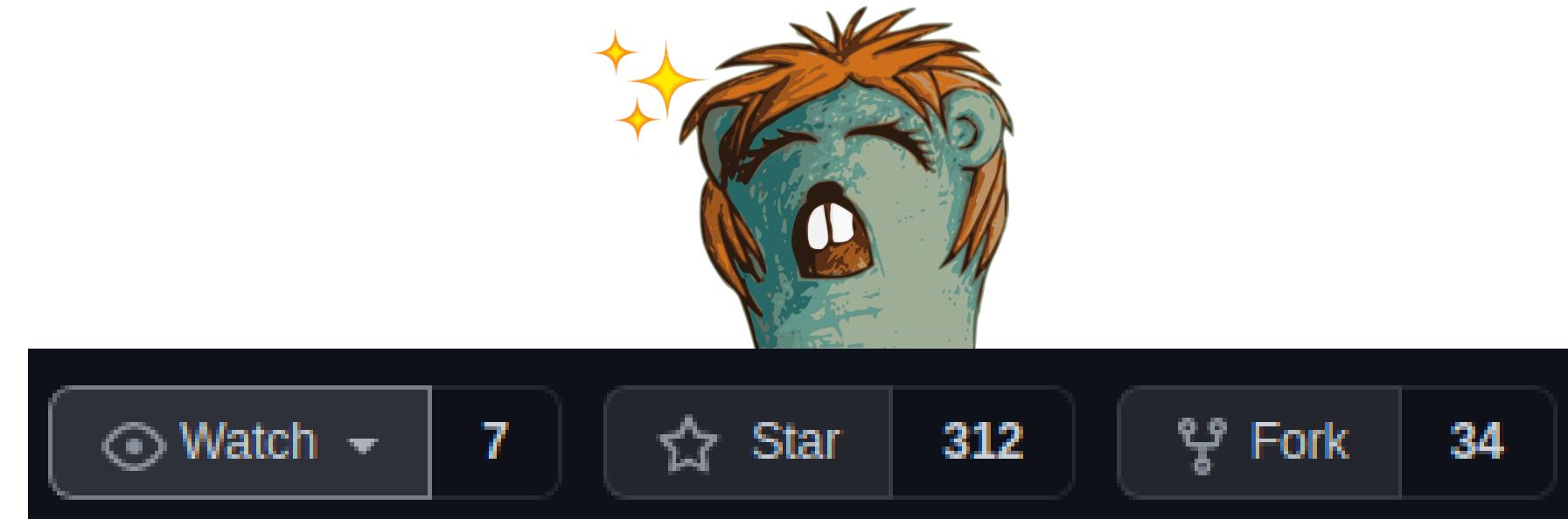
Теперь к метрикам

1. 113 строчек кода
2. Точечной конфигурации нет
3. Дополнительный инфровый код
4. Зафиксированный порядок источников

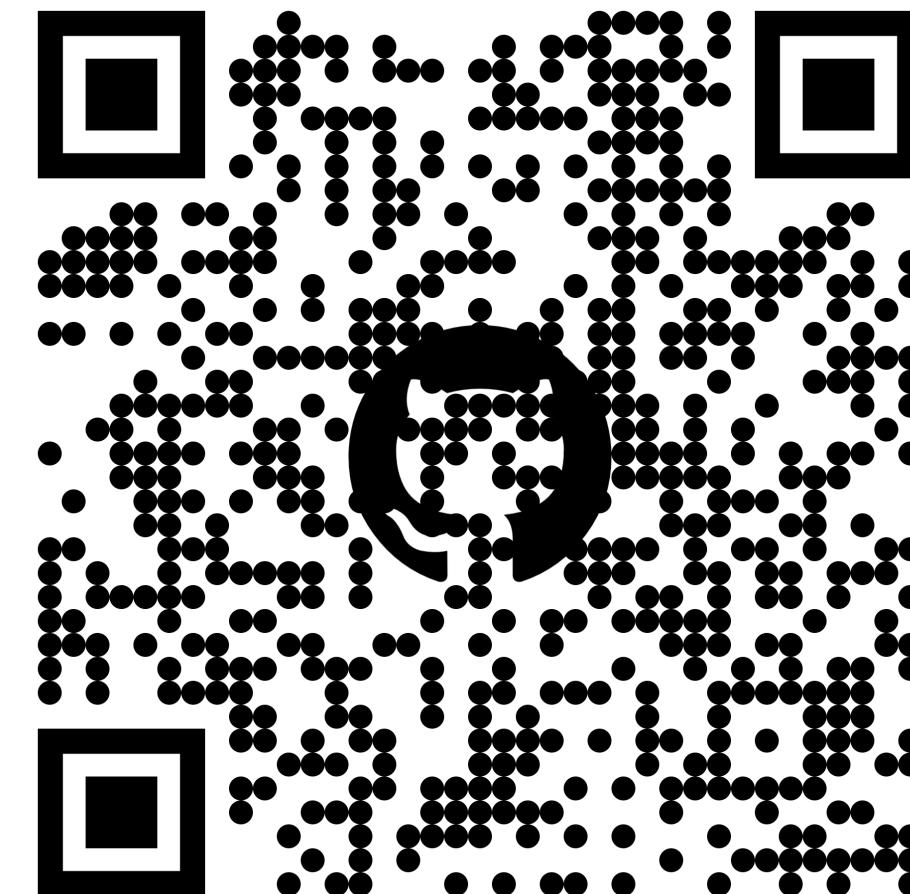
Cleanenv



Cleanenv



Ссылка на гитхаб: <https://github.com/ilyakaznacheev/cleanenv>



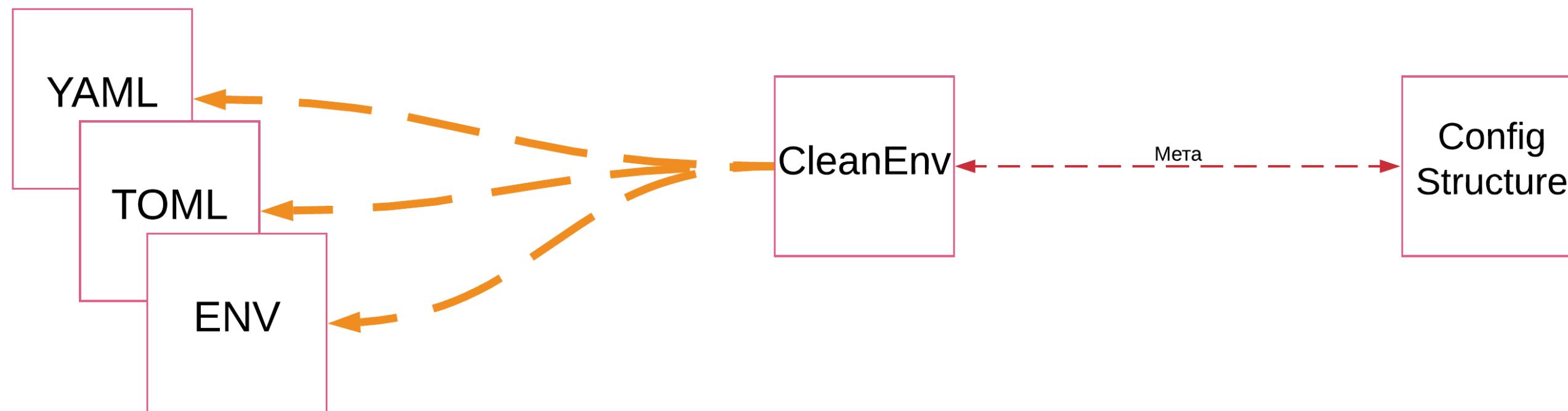
Теги в Go

field1 Type `tagKey:"tagValue"

Получение мета информации из тегов



Получение значений в порядке указания тегов



Используем следующие теги из этой библиотеки

1. env

Используем следующие теги из этой библиотеки

1. env
2. env-default

Модифицируем наши конфиг-структуры

```
type MainConfig struct {  
    IsKubernetes     bool   `env-default:"true"  
    ServiceName      string `env:"SERVICE_NAME"  
    StrategyRequest  string `env:"MAIN_STRATEGY_REQUEST"  
                           env-default:"sequential"  
}
```

Модифицируем конфиг-структуры

```
type DatabaseConfig struct {  
    Replicass []string `env:"MONGO_REPLICAS" `  
    Username  string  `json:"username" `  
    Password  string  `json:"password" `  
}
```

Модифицируем конфиг-структуры

```
type IntegrationConf struct {
    ApiKey           string
    StrategyRequest string `env:"GOOGLE_AI_STRATEGY_REQUEST" env-default:"sequential"`
    RateLimit        int16
}
```

Как будем читать конфигурацию

```
func ReadConfig(configSources string) (ServiceConfiguration, error) {
    configSource, err := readSourceConfig(configSources)
    if err != nil {
        return ServiceConfiguration{}, err
    }
    var configuration ServiceConfiguration
    sources := configSource["sources"].(map[interface{}]interface{})
    for key, source := range sources {
        switch key {
        case "vault":
            if err := configuration.addVault(source.(map[interface{}]interface{}))
                return ServiceConfiguration{}, err
        }
    }
}
return configuration, nil
```

Читаем настройки окружения

```
func readSourceConfig(file string) (
    map[string]interface{}, error) {
    readBytes, errReading := ioutil.ReadFile(file)
    if errReading != nil {
        return nil, errReading
    }
    var result map[string]interface{}
    if err := yaml.Unmarshal(readBytes, &result); err != nil {
        return nil, err
    }
    return result, nil
}
```

Инфра для волта та же

```
func (config *ServiceConfiguration) addVault(dataFromEnv map[interface{}]interface{}) error {
    configurationVault := ConfigVault{
        VaultAddress: dataFromEnv["Address"].(string),
        VaultPath:    dataFromEnv["Path"].(string),
        VaultToken:   dataFromEnv["Token"].(string),
    }

    if err := configurationVault.connectToVault(); err != nil {
        return err
    }

    configFromVaultDatabase, err := vaultConfig.getPath(configurationVault.VaultPath)
    if err != nil {
        return err
    }
    config.BusinessSecret.PrivateCertPath = configFromVaultDatabase["certPath"].(string)
    config.BusinessSecret.PrivateKeyPath = configFromVaultDatabase["keyPath"].(string)
    config.ServiceA.ApiKey = configFromVaultDatabase["ApiKey"].(string)
    return nil
}
```

Метрики

1. 94 строчки кода

Метрики

1. 94 строчки кода
2. Точечная конфигурация есть

Метрики

1. 94 строчки кода
2. Точечная конфигурация есть
3. Мало источников

Метрики

1. 94 строчки кода
2. Точечная конфигурация есть
3. Мало источников
4. Порядок конфигурации жёстко зафиксирован

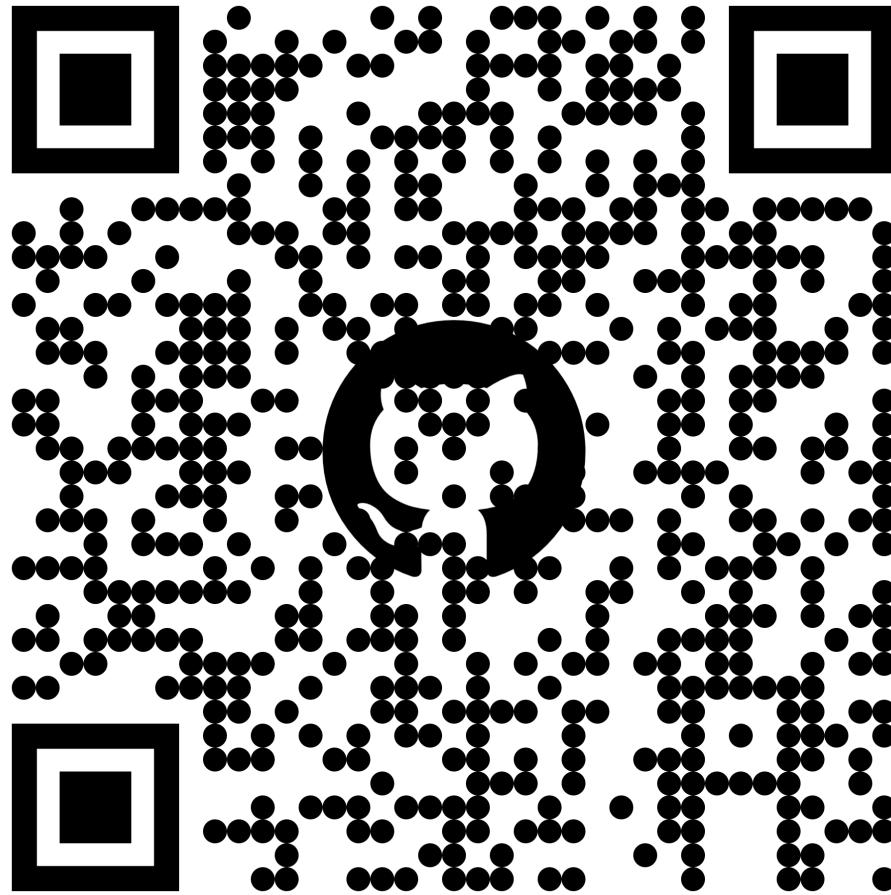
Библиотека Gostructor



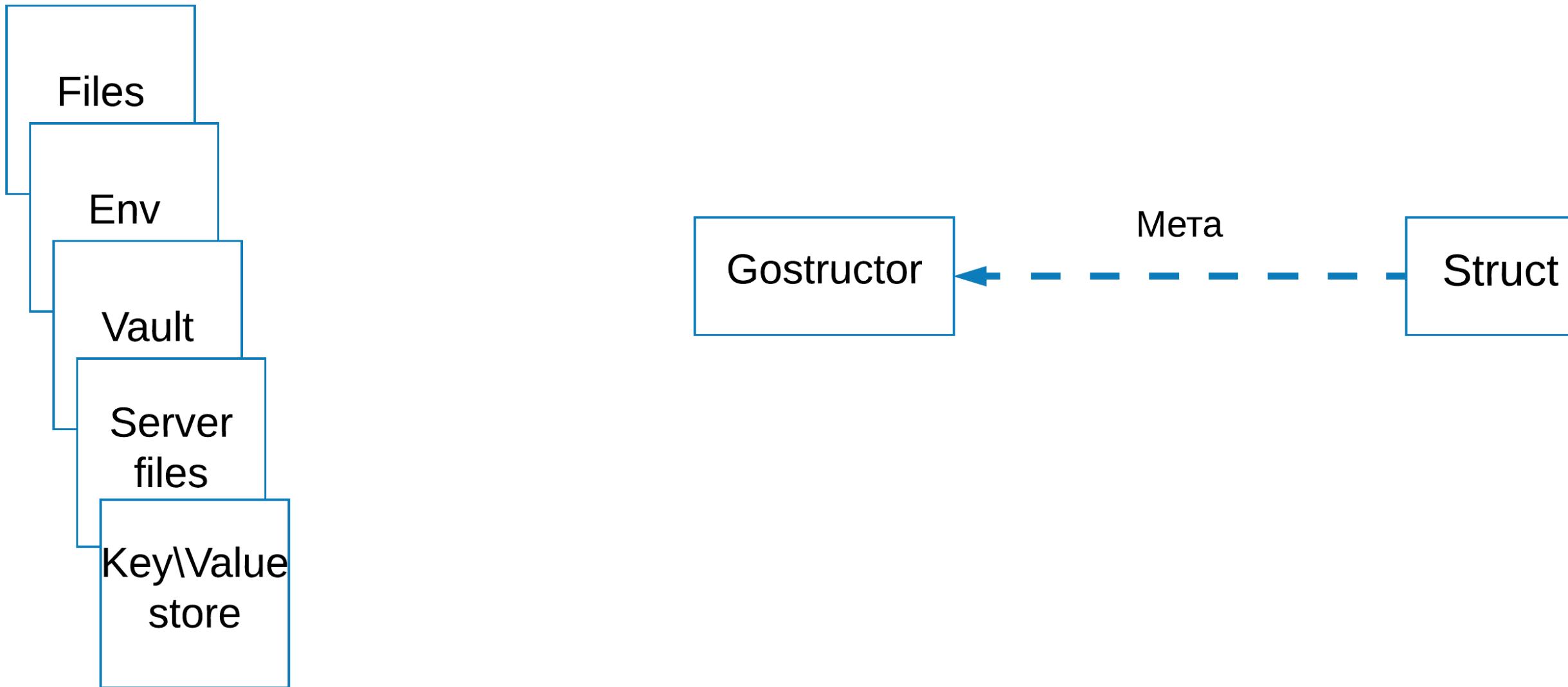
Библиотека GoStructor



Ссылка на гитхаб: <https://github.com/goreflect/gostructor>



Получаем мета информацию из тегов



Получаем значения

Как выглядит мета информация в тегах

Приоритизация источников

Кастомизация приоритетов в зависимости от окружения

Модифицируем наши структуры

```
type MainConfig struct {
    IsKubernetes     bool      `cf_env:"KUBERNETES"
                                cf_default:"false"`
    ServiceName       string   `cf_json:"name"
                                cf_env:"SERVICE_NAME" cf_default:"test"`
    StrategyRequest  string   `cf_default:"sequential"`
}
```

Модифицируем наши структуры

```
type DatabaseConfig struct {
    Replicas []string `cf_env:"MONGO_REPLICAS"
                  cf_default:"localhost:27017"`
    Username string   `cf_vault:"infra/mongo#username"`
    Password string   `cf_vault:"infra/mongo#password"`
}
```

Модифицируем наши структуры

```
type IntegrationConf struct {
    APIKey           string `cf_vault:"integration/serviceA#api-`  
                      cf_env:"SERVICE_A_KEY" cf_default:"testApi"`
    StrategyRequest string `cf_default:"sequential"`
    RateLimit       int16  `cf_default:"100"`
}
```

Кастомизируем последовательность источников

```
type IntegrationConf struct {
    APIKey string `... cf_priority:"SEQUENCE_1:cf_env,
                    cf_default;SEQUENCE_2:cf_vault"`
    ...
}
```

Модифицируем наши структуры

```
type BusinessConf struct {  
    PrivateKeyPath string `cf_vault:"services/  
        businessService1#keyPath" cf_default:"/var/test.pem"  
    PrivateCertPath string `cf_vault:"services/  
        businessService1#certPath" cf_default:"/var/test.crt"  
}
```

Кастомизируем последовательность источников

```
type BusinessConf struct {
    PrivateKeyPath string `cf_priority:"SEQUENCE_1:cf_default;
                           SEQUENCE_2:cf_vault"`
    PrivateCertPath string `cf_priority:"SEQUENCE_1:cf_default;
                           SEQUENCE_2:cf_vault"`
}
```

Как будем прокидывать настройку окружения?

Добавим в переменные окружения PRIORITY

Читаем конфигурацию

```
func ReadConfig() (ServiceConfiguration, error) {  
    var configuration ServiceConfiguration  
    _, err := gostructor.ConfigureSmart(&configuration)  
    if err != nil {  
        return ServiceConfiguration{}, nil  
    }  
    return configuration, nil  
}
```

Плюсы gostructor

1. Автоматизация вне зависимости от окружения

Плюсы gostructor

1. Автоматизация вне зависимости от окружения
2. Точечная конфигурация

Плюсы gostructor

1. Автоматизация вне зависимости от окружения
2. Точечная конфигурация
3. Множество источников

Плюсы gostructor

1. Автоматизация вне зависимости от окружения
2. Точечная конфигурация
3. Множество источников
4. Понятная система ошибок

Минусы

1. Дополнительная зависимость + транзитивные зависимости в проект

Минусы

1. Дополнительная зависимость + транзитивные зависимости в проект
2. Нет горячих настроек

Минусы

1. Дополнительная зависимость + транзитивные зависимости в проект
2. Нет горячих настроек
3. Не поддержан Spring Cloud Config Server, key\value

Минусы

1. Дополнительная зависимость + транзитивные зависимости в проект
2. Нет горячих настроек
3. Не поддержан Spring Cloud Config Server, key\value
4. Небольшая популярность

В каких проектах я использовал gostructor

1. Skillmap

В каких проектах я использовал gostructor

1. Skillmap
2. AuthEvent

В каких проектах я использовал gostructor

1. Skillmap
2. AuthEvent
3. ЕССМ компании Eltex

Какие планы на библиотеку

1. Spring cloud config server

Какие планы на библиотеку

1. Spring cloud config server
2. Key\Value (Consul & etcd)

Какие планы на библиотеку

1. Spring cloud config server
2. Key\Value (Consul & etcd)
3. Упрощение синтаксиса в тегах

Какие планы на библиотеку

1. Spring cloud config server
2. Key\Value (Consul & etcd)
3. Упрощение синтаксиса в тегах
4. Краткая документашка по полям

Какие планы на библиотеку

1. Spring cloud config server
2. Key\Value (Consul & etcd)
3. Упрощение синтаксиса в тегах
4. Краткая документашка по полям
5. Отслеживание измнений в файлах

Какие планы на библиотеку

1. Spring cloud config server
2. Key\Value (Consul & etcd)
3. Упрощение синтаксиса в тегах
4. Краткая документашка по полям
5. Отслеживание измнений в файлах

...

Кому может подойти

1. Больше 1 стейджинга

Кому может подойти

1. Больше 1 стейджинга
2. Больше 1 источника конфигурации

Кому может подойти

1. Больше 1 стейджинга
2. Больше 1 источника конфигурации
3. Кто не хочет писать лишний код

Кому может подойти

1. Больше 1 стейджинга
2. Больше 1 источника конфигурации
3. Кто не хочет писать лишний код
4. У кого конфигурация - строгий контракт

**Лучший код с точки
зрения качества**



**ЭТО ТОТ КОД, КОТОРОГО
НЕТ**