



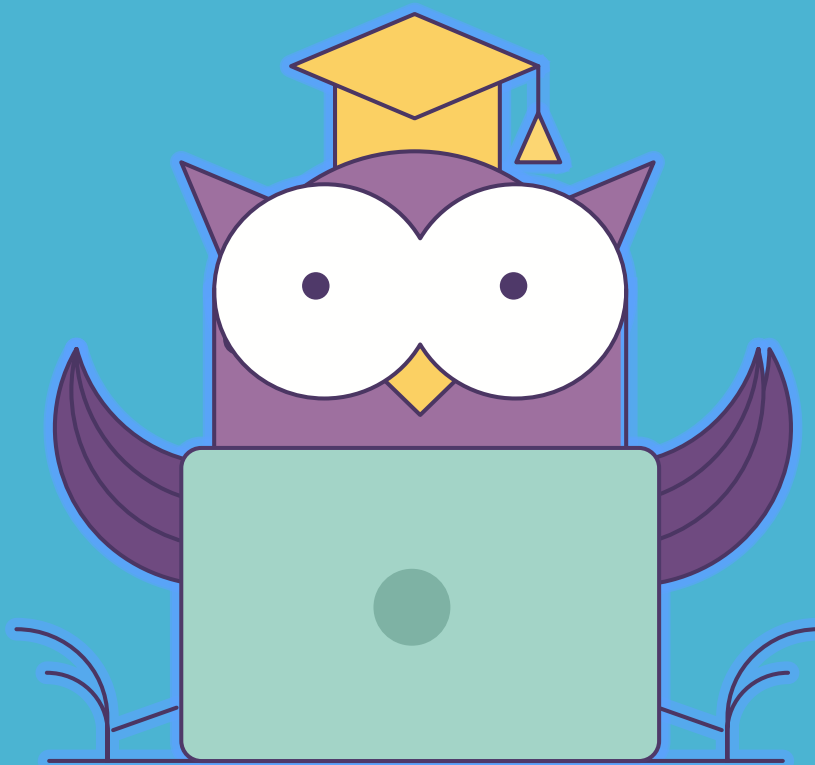
ОНЛАЙН-ОБРАЗОВАНИЕ

Конфигурация и логирование

Артем Желтак



Как меня слышно и видно?



> Напишите в чат

+ если все хорошо

- если есть проблемы со звуком или с видео

!проверить запись!

- Различные варианты конфигурации
- Использование простых форматов: ini, yaml и т.п.
- Библиотеки для работы с конфигурацией: viper и confita
- Стандартная библиотека для логирования
- Расширенное логирование с помощью Zap

Конфигурация приложения – это всё, что может меняться между развёртываниями

- Идентификаторы подключения к ресурсам / сторонним службам
- Регистрационные данные для подключения к внешним сервисам, (Amazon S3 / Twitter)
- Значения зависящие от среды развёртывания (каноническое имя хоста)

- файлы

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
```

- переменные окружения

```
env

TRMPLN_IDENTITY_HANDLE=http://127.0.0.1:8080/matches
VISITOR_TTL=60
WORKERS=10
```

```
domain: abs.com
blacklist:
  - evil.com
  - bad.com
```

```
type Config struct {
  Domain string `yaml:"domain"`
  Blacklist []string `yaml:"blacklist"`
}
```

```
import (  
    "gopkg.in/yaml.v2"  
    "io/ioutil"  
    "log"  
)
```

```
func main() {  
    var c Config  
    yamlFile, err := ioutil.ReadFile("conf.yaml")  
    if err != nil {  
        log.Fatal(err)  
    }  
    err = yaml.Unmarshal(yamlFile, &c)  
    if err != nil {  
        log.Fatalf("Unmarshal: %v", err)  
    }  
}
```

- легко менять между деплоями
- сложно случайно закоммитить в репозиторий
- независимы от языка

```
SHORTENER_HOST=localhost SHORTENER_PORT=7777 go run main.go
```

Храним секреты в переменных окружения:

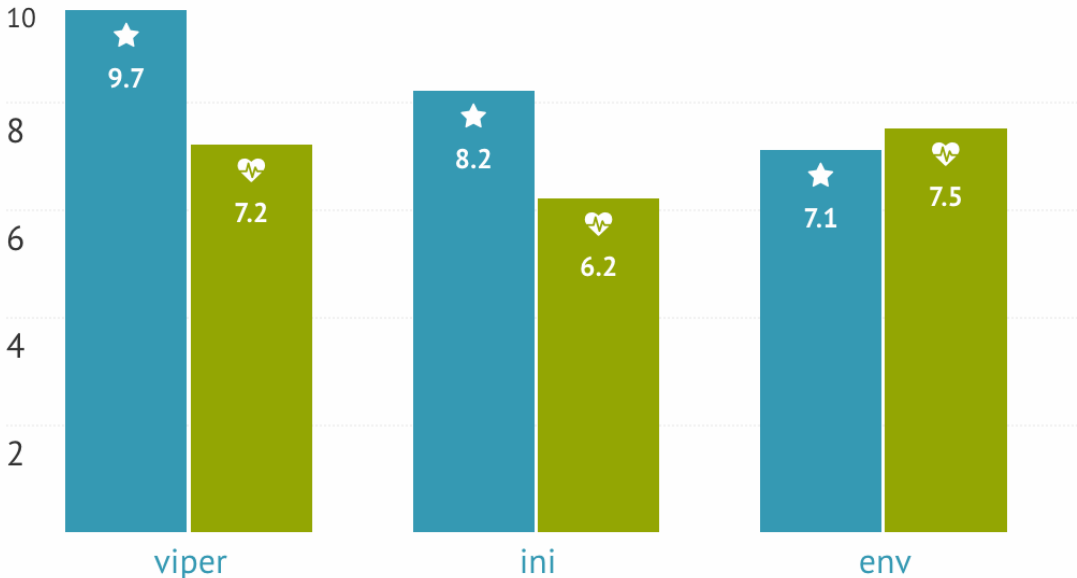
```
heroku run env

ALGOLIA_APPLICATION_ID=PYML6VQ4
GOOGLE_MAPS_KEY=AIzaSyAmfj18ZSEAV_mJkc2_eCkkksf5o
CLOUDINARY_API_SECRET=kS7s0RMfPmhM7zyIvS4
ALGOLIA_INDEX_PREFIX=dev
CLOUDINARY_API_KEY=6443243932
REDISCLLOUD_URL=redis://rediscloud:EjfdZBtSdHpIU6zyXzTMdM@redis-1242.c6.eu-west-1-1.ec2.cloud.redislabs.
ALGOLIA_API_KEY=a0348f6d80345caed21509be2247
STREAM_API_SECRET=ewu3g7mepbttb39x2d2tb5v88jbfudennunda8hp77xruy73phbkdnngwt
```

<https://12factor.net/config>

```
type Config struct {  
    Port int  
    Host string  
}
```

```
httpPort, err := strconv.Atoi(os.Getenv("SHORTENER_PORT"))  
if err != nil {  
    panic(fmt.Sprintf("SHORTENER_PORT not defined"))  
}  
  
shortenerHost := os.Getenv("SHORTENER_HOST")  
if shortenerHost == "" {  
    panic(fmt.Sprintf("SHORTENER_HOST not defined"))  
}  
  
config := Config{httpPort, shortenerHost}
```



<https://go.libhunt.com/categories/463-configuration> <https://github.com/avelino/awesome-go#configuration>

Библиотеки для чтения переменных окружения в структуры:

- envcfg - Un-marshaling environment variables to Go structs.
- envconf - Configuration from environment.
- envconfig - Read your configuration from environment variables.
- envh - Helpers to manage environment variables.
- godotenv - Go port of Ruby's dotenv library (Loads environment variables from .env).

<https://github.com/avelino/awesome-go#configuration>

```
type Config struct {  
    Port int    `env:"SHORTENER_PORT,required"`  
    Host string `env:"SHORTENER_HOST,required"`  
}
```

```
cfg := config.Config{}  
if err := env.Parse(&cfg); err != nil {  
    log.Fatalf("%+v\n", err)  
}
```

```
type config struct {  
    Home          string          `env:"HOME"``  
    Port          int             `env:"PORT" envDefault:"3000"``  
    IsProduction bool           `env:"PRODUCTION"``  
    Hosts         []string       `env:"HOSTS" envSeparator:":"``  
    Duration      time.Duration  `env:"DURATION"``  
    TempFolder    string         `env:"TEMP_FOLDER" envDefault:"${HOME}/tmp" envExpand:"true"``  
}
```

```
go get -u github.com/heetch/confita
```

- поддерживает примитивы Go
- поддерживает несколько "бэкэндов"

```
loader := confita.NewLoader(  
    env.NewBackend(),  
    file.NewBackend("/path/to/config.json"),  
    file.NewBackend("/path/to/config.yaml"),  
    flags.NewBackend(),  
    etcd.NewBackend(etcdClientv3),  
    consul.NewBackend(consulClient),  
    vault.NewBackend(vaultClient),  
)
```

```
go get -u github.com/spf13/viper
```

- setting defaults
- reading from JSON, TOML, YAML, HCL, envfile and Java properties config files
- live watching and re-reading of config files (optional)
- reading from environment variables
- reading from remote config systems (etcd or Consul), and watching changes
- reading from command line flags
- reading from buffer
- setting explicit values

- explicit call to Set
- flag
- env
- config
- key/value store
- [default](#)

```
viper.Set("port", 8080)
./myprogram --port 8080
PORT=8080 ./myprogram
conf.yaml: port: 8080
etcd/consul
viper.SetDefault("port", 777)
```

```
viper.SetConfigName("config") // name of config file (without extension)
viper.AddConfigPath("/etc/appname/") // path to look for the config file in
viper.AddConfigPath("$HOME/.appname") // call multiple times to add many search paths
viper.ReadInConfig() // Find and read the config file
```

```
viper.BindEnv("port") // bind to ENV "PORT"
viper.BindEnv("name", USERNAME) // bind to ENV "USERNAME"

os.Setenv("PORT", "13") // typically done outside of the app
os.Setenv("USERNAME", "spf13") // typically done outside of the app

port := viper.GetInt("port") // 13
name := viper.GetString("name") // "spf13"
```

```
viper.SetEnvPrefix("foo") // Becomes "FOO_"  
os.Setenv("FOO_PORT", "1313") // typically done outside of the app  
viperAutomaticEnv()  
port := viper.GetInt("port") // 1313
```

```
viper.GetString("logfile") // case insensitive Setting & Getting
if viper.GetBool("verbose") {
    fmt.Println("verbose enabled")
}

if viper.IsSet("foo") {
    i := viper.GetInt("foo")
}
```

```
pflag.Int("flagname", 1234, "help message for flagname")  
  
pflag.Parse()  
viper.BindPFlags(pflag.CommandLine)  
  
i := viper.GetInt("flagname") // retrieve values from viper instead of pflag
```

```
type config struct {  
    Port int  
    Name string  
}  
  
var C config  
  
err := Marshal(&C)  
if err != nil {  
    t.Fatalf("unable to decode into struct, %v", err)  
}
```

```
viper.WatchConfig()  
viper.OnConfigChange(func(e fsnotify.Event) {  
    fmt.Println("Config file changed:", e.Name)  
})
```

```
viper.SetConfigType("yaml") // or viper.SetConfigType("YAML")

// any approach to require this configuration into your program.
var yamlExample = []byte(`
Hacker: true
name: steve
hobbies:
- skateboarding
- snowboarding
- go
clothing:
  jacket: leather
  trousers: denim
age: 35
eyes : brown
beard: true
`)

viper.ReadConfig(bytes.NewBuffer(yamlExample))

viper.Get("name") // this would be "steve"
```

```
viper.WriteConfig() // writes current config to predefined path set by 'viper.AddConfigPath()' and 'viper.ConfigFile()'
viper.SafeWriteConfig()
viper.WriteConfigAs("/path/to/my/.config")
viper.SafeWriteConfigAs("/path/to/my/.config") // will error since it has already been written
viper.SafeWriteConfigAs("/path/to/my/.other_config")
```

Журнал - это поток агрегированных, упорядоченных по времени событий, собранных из потоков вывода всех запущенных процессов и вспомогательных сервисов. Журнал в своём сыром виде обычно представлен текстовым форматом с одним событием на строчку (хотя трассировки исключений могут занимать несколько строк). Журнал не имеет фиксированного начала и конца, поток сообщений непрерывен, пока работает приложение.

```
import (  
    "log"  
)  
  
func init(){  
    log.SetPrefix("LOG: ")  
    log.SetFlags(log.Ldate | log.Lmicroseconds | log.Llongfile)  
    log.Println("init started")  
}  
func main() {  
    // Println writes to the standard logger.  
    log.Println("main started")  
  
    // Fatalln is Println() followed by a call to os.Exit(1)  
    log.Fatalln("fatal message")  
  
    // Panicln is Println() followed by a call to panic()  
    log.Panicln("panic message")  
}
```

```
package main

import (
    "log"
    "os"
)

func main() {
    f, err := os.OpenFile("info.log", os.O_RDWR | os.O_CREATE | os.O_APPEND, 0666)
    if err != nil {
        log.Fatalf("error opening file: %v", err)
    }
    defer f.Close()

    log.SetOutput(f)
    log.Println("This is a test log entry")
}
```

<https://github.com/uber-go/zap>

- есть и printf, и structured logging
- очень быстрый

Package	Time	Time % to zap	Objects Allocated
⚡ zap	118 ns/op	+0%	0 allocs/op
⚡ zap (sugared)	191 ns/op	+62%	2 allocs/op
zerolog	93 ns/op	-21%	0 allocs/op
go-kit	280 ns/op	+137%	11 allocs/op
standard library	499 ns/op	+323%	2 allocs/op
apex/log	1990 ns/op	+1586%	10 allocs/op
logrus	3129 ns/op	+2552%	24 allocs/op
log15	3887 ns/op	+3194%	23 allocs/op

```
logger := zap.NewExample() // для тестов, пишет в stdout
```

```
logger, err := zap.NewProduction()
if err != nil {
    log.Fatalf("can't initialize zap logger: %v", err)
}
defer logger.Sync()
```

- Example and Production используют JSON Encoder, Development - Console
- The `logger.DPanic()` паникует в Development, и не паникует - в Example / Production

Development:

- Печатает stack trace для Warn и выше.
- Всегда печатает файл/номер строки
- Печатает уровень сообщения в верхнем регистре
- Печатает время в формате ISO8601 с миллисекундами

Production:

- не логирует на уровне debug
- Добавляет stack trace в качестве поля на уровне Error/DPanic levels
- Добавляет caller в качестве поля
- Печатает timestamp в формате epoch
- Печатает уровень сообщения в нижнем регистре

```
logger.Info("This is an INFO message with fields",
            zap.String("region", "us-west"),
            zap.Int("id", 2))
```

```
slogger := logger.Sugar()
slogger.Info("Info() uses sprint")
slogger.Infof("Infof() uses %s", "sprintf")
slogger.Infow("Infow() allows tags", "name", "Legolas", "type", 1)
```

Можем переключиться обратно в любой момент:

```
logger := zap.NewExample()  
defer logger.Sync()  
sugar := logger.Sugar()  
plain := sugar.Desugar()
```

Обязательные поля: encoder, level, sink.

```
logger, _ = zap.Config{
    Encoding:      "json",
    Level:         zap.NewAtomicLevelAt(zapcore.DebugLevel),
    OutputPaths:  []string{"stdout"},
    EncoderConfig: zapcore.EncoderConfig{
        MessageKey: "message", // <--
    },
}.Build()

logger.Info("This is an INFO message with fields", zap.String("region", "us-west"), zap.Int("id", 2))
```

- `logger.AddCaller()` - добавить аннотацию
- `logger.AddStacktrace()` - стектрейс для сообщений выше определенного уровня
- `logger.Fields()` - добавить поля (Creating loggers this way, and not specifying additional fields during the actual log call can make your logging faster with less memory allocations.)
- `logger.WrapCore()` - замена ядра логера:

```
logger.WithOptions(  
    zap.WrapCore(  
        func(zapcore.Core) zapcore.Core {  
            return zapcore.NewCore(zapcore.NewConsoleEncoder(cfg.EncoderConfig), zapcore.AddSync(os.Std  
        })).Info("This is an INFO message")  
    })
```

```
func SyslogTimeEncoder(t time.Time, enc zapcore.PrimitiveArrayEncoder) {
    enc.AppendString(t.Format("Jan  2 15:04:05"))
}

cfg.EncoderConfig.EncodeTime = SyslogTimeEncoder

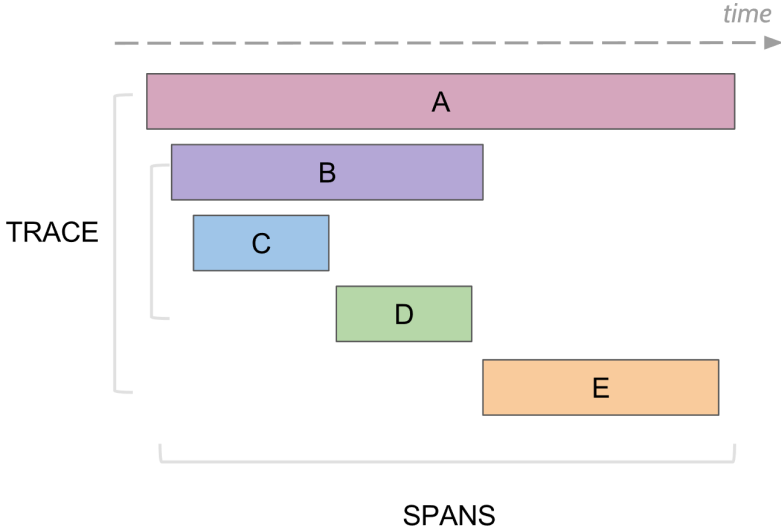
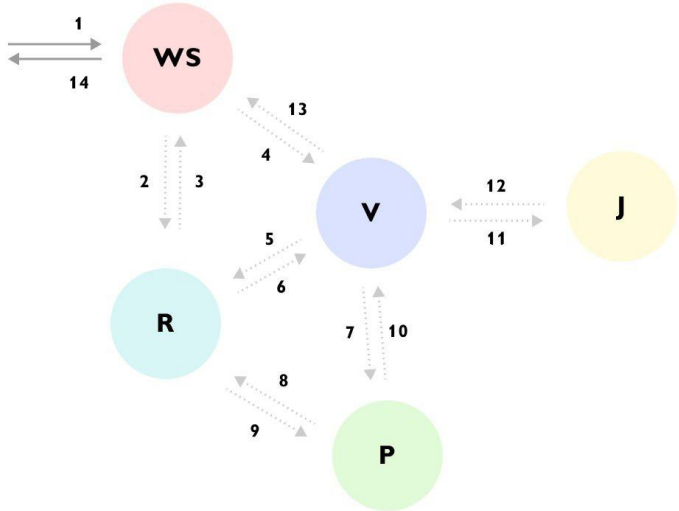
logger, _ = cfg.Build()
logger.Info("This should have a syslog style timestamp")
```

```
func CustomLevelEncoder(level zapcore.Level, enc zapcore.PrimitiveArrayEncoder) {
    enc.AppendString("[ " + level.CapitalString() + " ]")
}

cfg.EncoderConfig.EncodeLevel = CustomLevelEncoder

logger, _ = cfg.Build()
logger.Info("This should have a bracketed level name")
```

Opentracing/jaeger. Логирование меж микросервисов



Заполните пожалуйста опрос

<https://otus.ru/polls/4903/>



Спасибо за внимание!

