DWARVES FOUNDATION

Advanced Topics

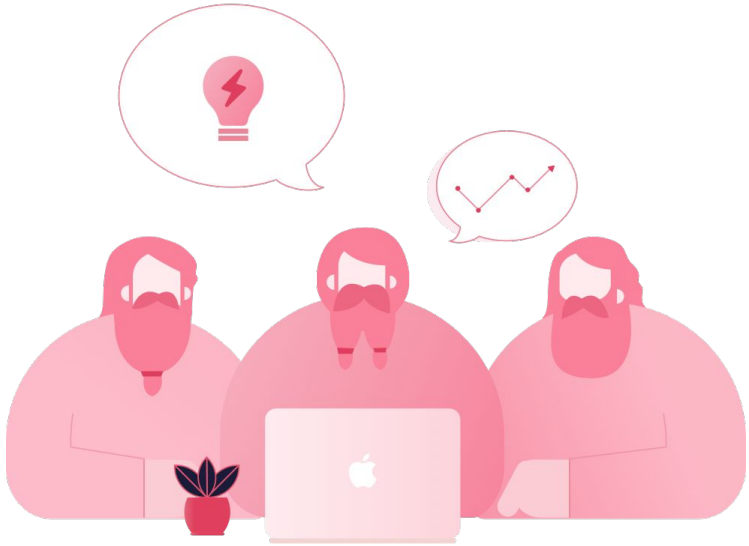
Go Reflection. Simple deployment

Hieu Phan

Engineer Lead

@hieuphq

andy@d.foundation



DWARVES FOUNDATION

Agenda

Topic summary

1. Go Reflection and Interface System
2. Concurrency Pattern
3. Performance Optimization Technique
4. Demo

Go Reflection

What is Go Reflection?

- Allowing a program to manipulate objects with arbitrary types.
- Ability of a program to introspect and analyze its structure during run-time

Why ?

- Interface{} say nothings since it has no method.
- The typical use is to take a value with static type interface{} and extract its dynamic type information by calling TypeOf, which returns a Type.

Reflect Package

The foundation of Go reflection is based around these term Values, Types and Kinds. Also defined as the type reflect.Value, reflect.Kind, reflect.Type.

- reflect.ValueOf(x interface{})
- reflect.TypeOf(x interface{})
- Type.Kind()

DWARVES FOUNDATION

```go
type Student struct {
    ID      int
    Name    string
    DoB     string
    Address string
}

func main() {
    s := Student{
        ID:      1,
        Name:    "John Doe",
        DoB:     "2006-01-02",
        Address: "HCMC",
    }
    v := reflect.ValueOf(s)
    t := reflect.TypeOf(s)
    fmt.Println(v)
    fmt.Println(t)
    fmt.Println(t.Kind())
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  zsh

→ reflect go run main.go
{1 John Doe 2006-01-02 HCMC}
→ reflect go run main.go
{1 John Doe 2006-01-02 HCMC}
main.Student
struct
→ reflect go run main.go
{1 John Doe 2006-01-02 HCMC}
main.Student
struct
→ reflect
```

Reflect Package

These are some common method provide by the package

- NumField()
- Field()

```go
func main() {
    s := Student{
        ID:      9948,
        Name:    "John Doe",
        DoB:     "2006-01-02",
        Address: "HCMC",
    }

    v := reflect.ValueOf(s)
    for i := 0; i < v.NumField(); i++ {
        fmt.Print(v.Type().Field(i).Name + ": ")
        fmt.Println(v.Field(i))
    }
}
```

```
PROBLEMS  OUTPUT  TERMINAL  ···  zsh - refle

→  reflect go run main.go
ID: 9948
Name: John Doe
DoB: 2006-01-02
Address: HCMC
→  reflect
```

DWARVES FOUNDATION

Use cases

Data Validation and Parsing

Custom Serialization

Dependency Injection

Database ORM

API Documentation

Code Generation

main.go

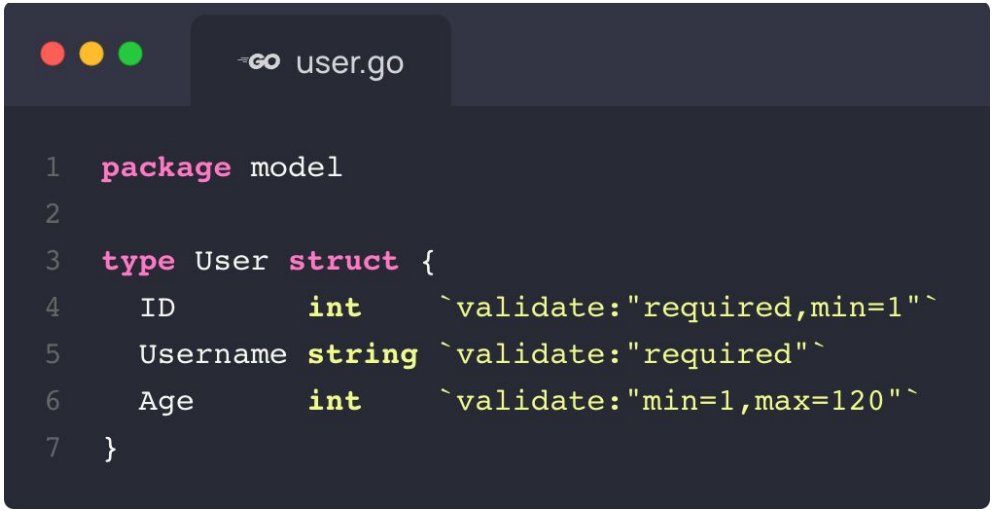
```go
package main

import (
  "fmt"

  "github.com/dwarvesf/go23/ex8/model"
  "github.com/dwarvesf/go23/ex8/validation"
)

func main() {
  u := model.User{
    ID:       0,
    Username: "dwarvesf",
    Age:      0,
  }

  rs := validation.ValidateStruct(u)

  fmt.Println(rs)
  // [ID is required ID should be at least 1 Age should be at least 1]
}
```

Data Validation

Define struct with validation tags

Create validation function

Implement validation logic



Data Validation

Define struct with validation tags

Create validation function

Implement validation logic

validator.go

```go
func ValidateStruct(data interface{}) []string {
    var validationErrors []string

    value := reflect.ValueOf(data)
    valueType := value.Type()

    for i := 0; i < value.NumField(); i++ {
        field := value.Field(i)
        fieldName := valueType.Field(i).Name
        fieldTag := valueType.Field(i).Tag.Get("validate")

        if fieldTag != "" {
            tags := strings.Split(fieldTag, ",")
            for _, tag := range tags {
                validationError := validateTag(tag, fieldName, field)
                if validationError != "" {
                    validationErrors = append(validationErrors, validationError)
                }
            }
        }
    }

    return validationErrors
}
```

Data Validation

Define struct with validation tags

Create validation function

Implement validation logic

validator.go

```go
func validateTag(tag string, fieldName string, field reflect.Value) string {
    parts := strings.Split(tag, "=")
    switch parts[0] {
    case "required":
        if field.Interface() == reflect.Zero(field.Type()).Interface() {
            return fieldName + " is required"
        }
    case "min":
        minValue, _ := strconv.Atoi(parts[1])
        if field.Int() < int64(minValue) {
            return fieldName + " should be at least " + parts[1]
        }
    case "max":
        maxValue, _ := strconv.Atoi(parts[1])
        if field.Int() > int64(maxValue) {
            return fieldName + " should be at most " + parts[1]
        }
    }
    return ""
}
```

Deep Equality

Unexported Fields: DeepEqual won't be able to access unexported fields

Slice and Map Order: DeepEqual compares slices and maps by their content, but not necessarily by order. If order matters, consider comparing them manually.

Interface Values: The DeepEqual function doesn't compare two interface values that hold different concrete types, even if they have the same underlying values.

Nil Comparisons: DeepEqual returns true when comparing a nil value against an uninitialized struct.

maps, slices package

Go 1.21.0 Provide compare functions

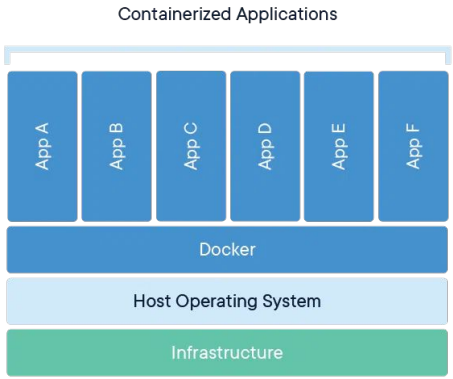
validator.go

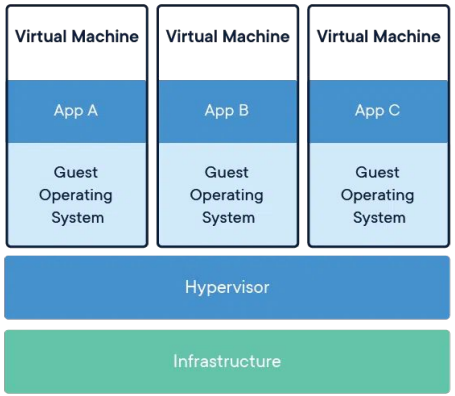
```go
func validateTag(tag string, fieldName string, field reflect.Value) string {
    parts := strings.Split(tag, "=")
    switch parts[0] {
    case "required":
        if field.Interface() == reflect.Zero(field.Type()).Interface() {
            return fieldName + " is required"
        }
    case "min":
        minValue, _ := strconv.Atoi(parts[1])
        if field.Int() < int64(minValue) {
            return fieldName + " should be at least " + parts[1]
        }
    case "max":
        maxValue, _ := strconv.Atoi(parts[1])
        if field.Int() > int64(maxValue) {
            return fieldName + " should be at most " + parts[1]
        }
    }
    return ""
}
```

Docker

Docker

Docker is a platform that enables developers to automate the deployment of applications within lightweight, portable containers.



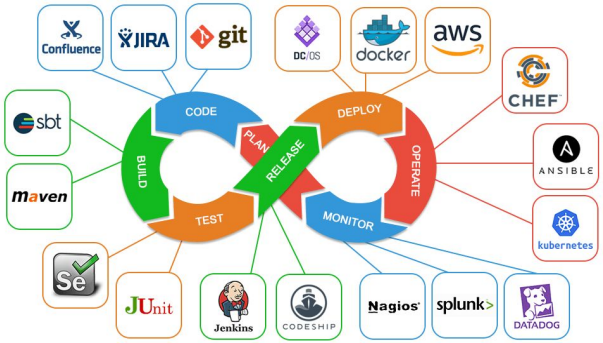


Use Cases for Docker

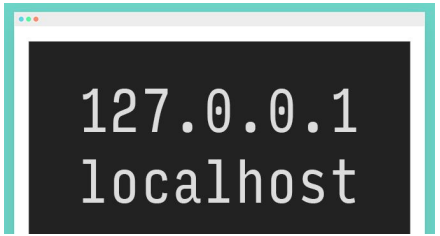
Microservices

Microservices: Docker's lightweight nature is well-suited for microservices architecture.

DevOps Automation: Docker simplifies application deployment, scaling, and management.



Testing and CI/CD: Consistent environments make testing and continuous integration easier.



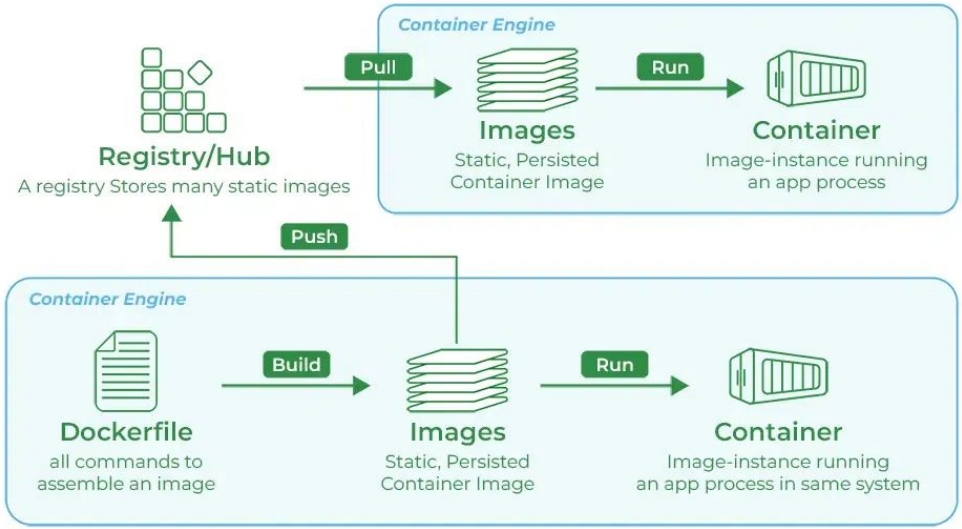
Local Development: Developers can replicate production environments locally using containers.

Docker components

Docker Image: A snapshot of a filesystem.

Docker Container: An instance of a Docker image that runs as a process on the host machine.

Docker Registry: A repository for storing and sharing Docker images



Deployment

Fly.IO

- The fastest way to deployment new thir to the internet.
- Not required to much the deployment knowledge.
- Suitable to hobby projects.

Fly.IO

> Install flyctl on Mac OS

$ brew install flyctl

> Sign Up

$ fly auth signup

> Sign In

$ fly auth login

> Launch. a config file fly.toml will be created

$ fly launch

> If have existing fly.toml file

$ fly deploy

```toml
app = "demo"
primary_region = "sin"

[build]
  builder = "paketobuildpacks/builder:base"
  buildpacks = ["gcr.io/paketo-buildpacks/go"]

[env]
  DEBUG = "true"
  ENV = "prod"
  PORT = "8080"
  SERVICE_NAME = "demo"

[http_service]
  internal_port = 8080
  force_https = true
  auto_stop_machines = true
  auto_start_machines = true
  min_machines_running = 0
  processes = ["app"]
```

Fly.IO - Database

Fly Postgres is a Fly app with flyctl using bootstrap and manage a database cluster.

Feature: replication, failover, metrics, monitoring and daily snapshots.

Fly.IO - ENV and SECRET

Local Environment Variables: using .env file

Environment Variables on fly.io:

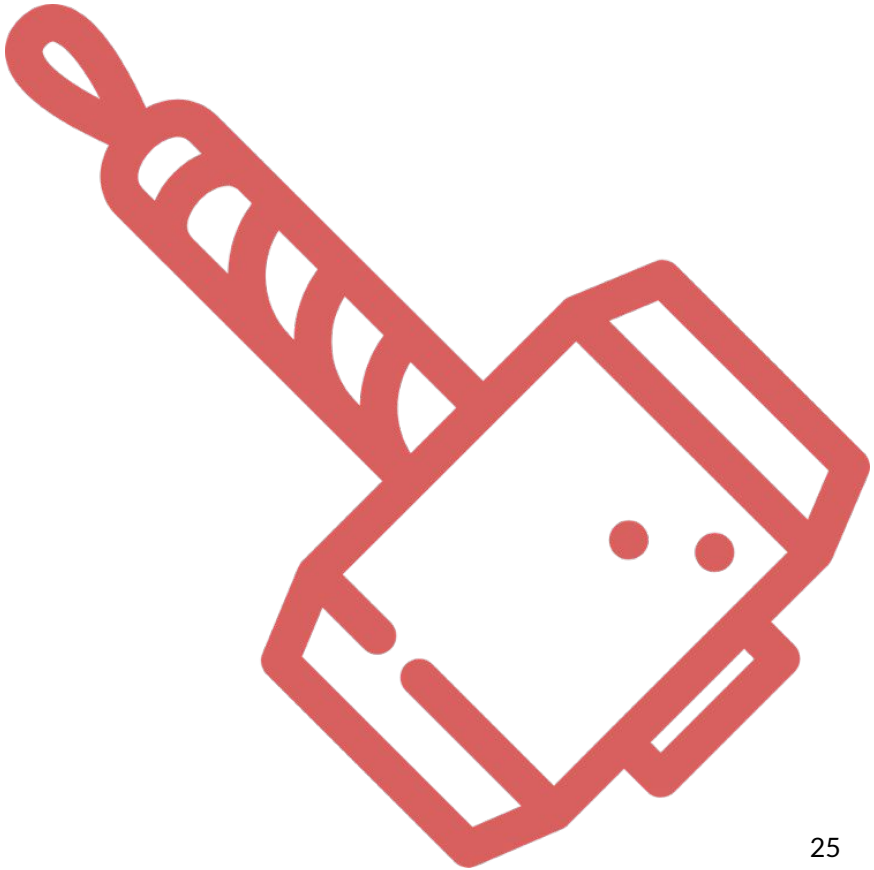
- ENV: input to fly.toml
- Secret
  - using feature on web
  - `fly secrets set DB_PASSWORD=topsecret`

Demo

Demo - Zer0 to Hero

Run app locally

Deploy app to fly.io



Reference

Resources & Reference links

- https://go.dev/blog/laws-of-reflection
- https://fly.io/docs/reference/configuration/

DWARVES FOUNDATION

Thank You

DWARVES FOUNDATION

Q&A