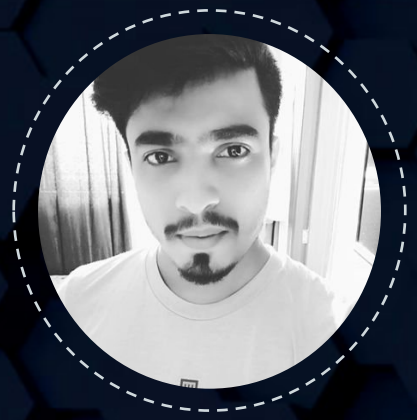
An Ultimate GopherLabs Hands-on Labs



Who Am I?

Sangam Biradar

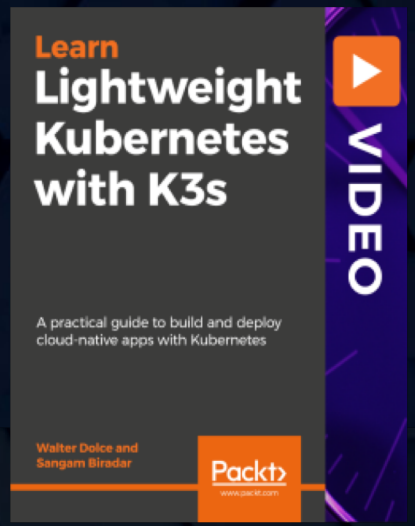
Sangam Biradar

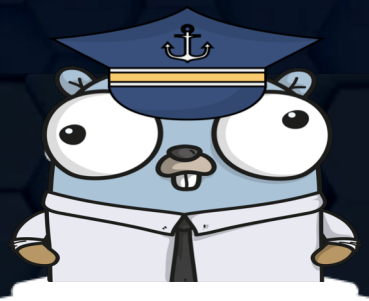
@BiradarSangam

EngineITops.com

- Docker Community Leader , Bangalore
- Author :- lightweight Kubernetes with k3s with packt Publication
- Gopherlabs – 500+ tutorials
- Okteto – Kubernetes For Developer , Bangalore

Meetup Organizer





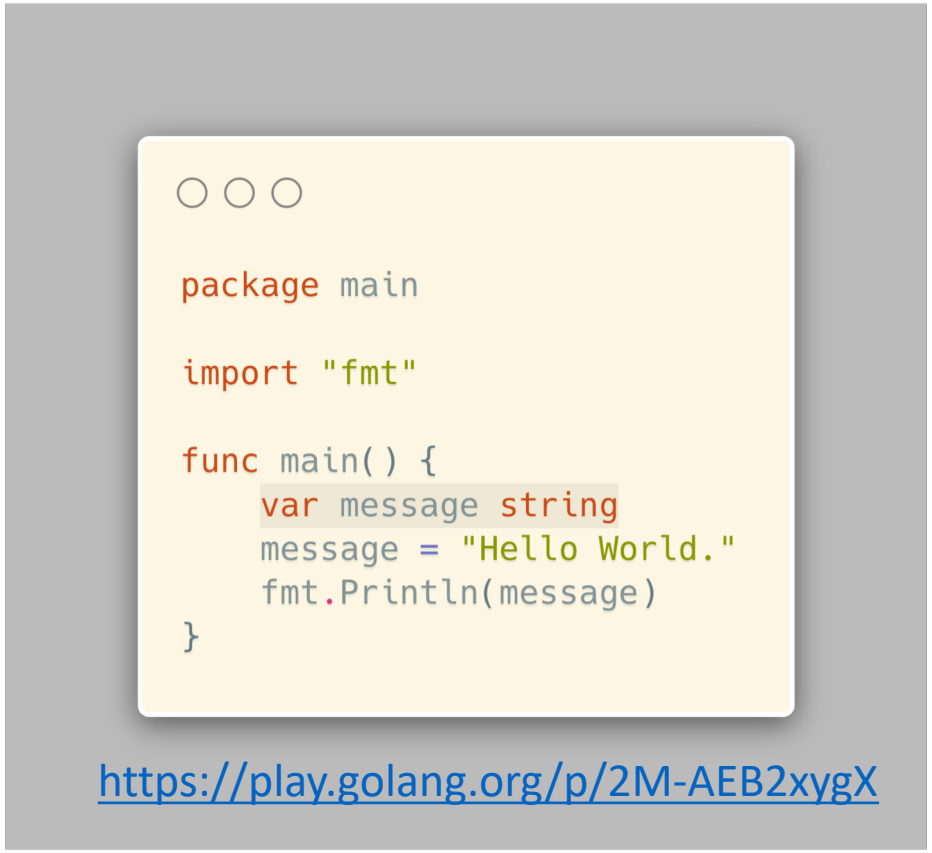
• Variables

“There are only two hard things in Computer Science:
cache invalidation and naming things.”

~ Phil Karlton

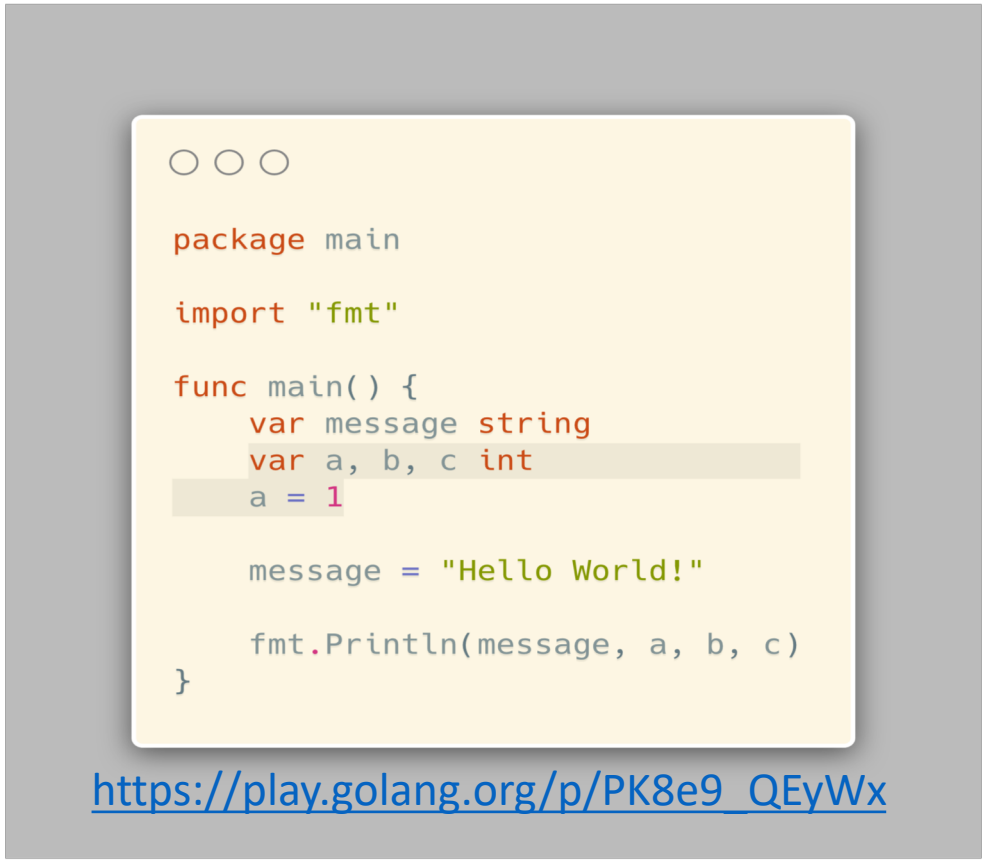
• Variables

Naming a variable properly is an important part of software development. Names must start with a letter and may contain letters, numbers, or the underscore symbol (_). The Go compiler doesn't care what you name a variable, but you should choose names that clearly describe the variable's purpose



```go
package main

import "fmt"

func main() {
    var message string
    message = "Hello World."
    fmt.Println(message)
}
```

https://play.golang.org/p/2M-AEB2xygX



```go
package main

import "fmt"

func main() {
    var message string
    var a, b, c int
    a = 1

    message = "Hello World!"

    fmt.Println(message, a, b, c)
}
```

https://play.golang.org/p/PK8e9_QEyWx

```go
package main

import "fmt"

func main() {

    var message = "Hello World!"
    var a, b, c int = 1, 2, 3

    fmt.Println(message, a, b, c)
}
```

https://play.golang.org/p/JxOe5GG1aP7

```go
package main

import "fmt"

func main() {

    var message = "Hello World!"
    var a, b, c = 1, false, 3

    fmt.Println(message, a, b, c)
}
```

https://play.golang.org/p/duf-rUXwl9f

• variable with zero value

```go
package main

import "fmt"

func main() {

    var a int
    var b string
    var c float64
    var d bool

    fmt.Printf("%v \n", a)
    fmt.Printf("%v \n", b)
    fmt.Printf("%v \n", c)
    fmt.Printf("%v \n", d)

    fmt.Println()
}
```

https://play.golang.org/p/eGBvFSlbu_b

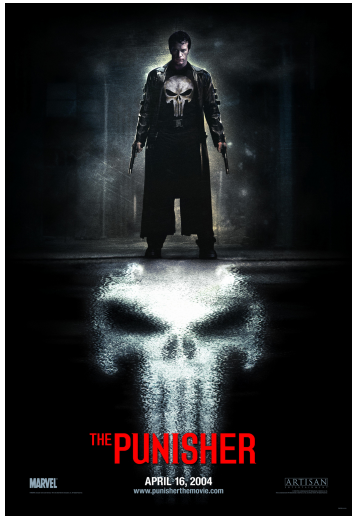
• Short variable declarations

```go
package main

import "fmt"

func main() {
    a := 10
    b := "golang"
    c := 4.17
    d := true
    e := "Hello"
    f := `Do you like my hat?`
    g := 'M'
    fmt.Printf("%v \n", a)
    fmt.Printf("%v \n", b)
    fmt.Printf("%v \n", c)
    fmt.Printf("%v \n", d)
    fmt.Printf("%v \n", e)
    fmt.Printf("%v \n", f)
    fmt.Printf("%v \n", g)
}
```

https://play.golang.org/p/PJtgbew0DhC

```go
package main

import "fmt"

func main() {

    a := 10
    b := "golang"
    c := 4.17
    d := true
    e := "Hello"
    f := `Do you like my hat?`
    g := 'M'

    fmt.Printf("%T \n", a)
    fmt.Printf("%T \n", b)
    fmt.Printf("%T \n", c)
    fmt.Printf("%T \n", d)
    fmt.Printf("%T \n", e)
    fmt.Printf("%T \n", f)
    fmt.Printf("%T \n", g)
}
```

https://play.golang.org/p/y5G1JaTxlKh



🤣🤣🤣🤣

• Scope

universe → package → file → function → curly braces

• Universe, capitalization (Public, Private)

```
package main

import (
    "fmt"

    "github.com/collabnix/gopherlabs/Beginners/workshop/scope/01_package-scope/02_visibility/vis"
)

func main() {
    fmt.Println(vis.MyName)
    vis.PrintVar()
}
```

https://play.golang.org/p/yUpMBsPUvWW

• How it works .....

```
sangam:02_visibility sangam$ tree
.
├── main
│   └── main.go
└── vis
    ├── name.go
    └── printer.go

// github.com/collabnix/gopherlabs/Beginners/workshop/scope/01_package-scope/02_visibility/vis
```

```
package vis

// MyName is exported because it starts with a capital letter
var MyName = "Sangam Biradar"
var yourName = "Future Rock Star Programmer"
```

```
package vis

import "fmt"

// PrintVar is exported because it starts with a capital letter
func PrintVar() {
    fmt.Println(MyName)
    fmt.Println(yourName)
}
```

• Package

```go
package main

import "fmt"

var x = 42

func main() {
    fmt.Println(x)
    foo()
}

func foo() {
    fmt.Println(x)
}
```

https://play.golang.org/p/3-AGJ6Ldspg

• _ blank identifier

- for a variable you don't want to use

variables.go

```
1  package main
2
3  import "fmt"
4
5  var a string = "this is stored in the variable a"
6  var b, c string = "stored in b", "stored in c"
7
8  func main() {
9
10     fmt.Println("a - ", a)
11     fmt.Println("b - ", b)
12     fmt.Println("c is not being used - ") // invalid code
13 }
```

variables.go

```
1  package main
2
3  import "fmt"
4
5  var a string = "this is stored in the variable a"
6  var b, _ string = "stored in b", "stored then thrown away"
7
8  func main() {
9
10     fmt.Println("a - ", a)
11     fmt.Println("b - ", b)
12     fmt.Println("c is not being used -  and this is no problem")
13 }
```

• Scope

scope.go

```
1  package main
2
3  import "fmt"
4
5  var x string = "Boss Hogg"
6
7  func oneFunc() {
8      fmt.Println(x)
9  }
10
11 func twoFunc() {
12     fmt.Println(x)
13 }
14
15 func main() {
16     oneFunc()
17     twoFunc()
18 }
```

package level scope

scope.go

```
1  package main
2
3  import "fmt"
4
5  func oneFunc() {
6      var x string = "Boss Hogg"
7      fmt.Println(x)
8  }
9
10 func twoFunc() {
11     fmt.Println(x)
12 }
13
14 func main() {
15     oneFunc()
16     twoFunc()
17 }
18
```

scope.go

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      x := 42
7      fmt.Println(x)
8  }
```

func level scope

invalid

scope of variable x is within the function in which it is declared

• curly-braces level scope

scope.go

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      fmt.Println("Hello")
7      {
8          x := 42
9          fmt.Println(x)
10     }
11 }
```

curly-braces level scope

scope.go

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      fmt.Println("Hello")
7      {
8          x := 42
9          fmt.Println(x)
10     }
11         fmt.Println(x)
12 }
```

invalid

scope of variable x is within the curly-braces in which it is declared

• Constants

```
package main

import (
    "fmt"
)

const

func main() {
    fmt.Println("Hello, playground")
}
```

Declare Constant by const keyword ……. as similar to import

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

```go
package main

import (
    "fmt"
)

const a = 42
const b = 42.78
const c = "James Bond"

func main() {
    fmt.Println(a)
    fmt.Println(b)
    fmt.Println(c)
    fmt.Printf("%T\n", a)
    fmt.Printf("%T\n", b)
    fmt.Printf("%T\n", c)
}
```

https://play.golang.org/p/5ShMrj1Eg2e

Declare Constant by const keyword ……. as similar to import

1st its will the variable and then its print the type of variable

References

- https://gopherlabs.collabnix.com
- https://godoc.org/
- https://golang.org/doc/

Thanks!

Any questions?

 Sangam Biradar

 @BiradarSangam

 @sangambiradar