

THE WAY TO



0.0.0.0/ME



CÎRLIG ŞTEFAN
Go developer @ Ellation



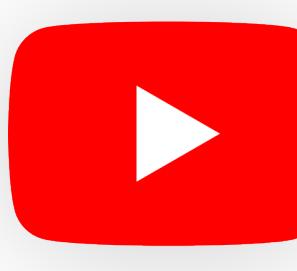
@steevehook



steevehookmd



@steevehook

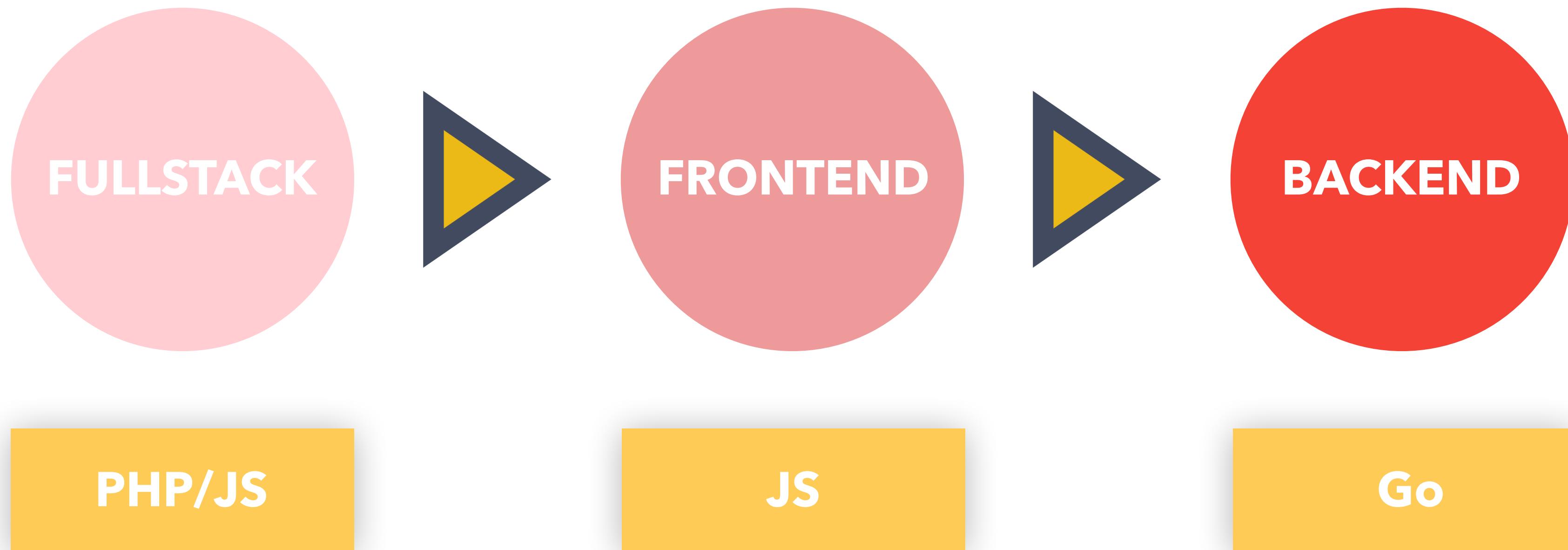


GopherTuts

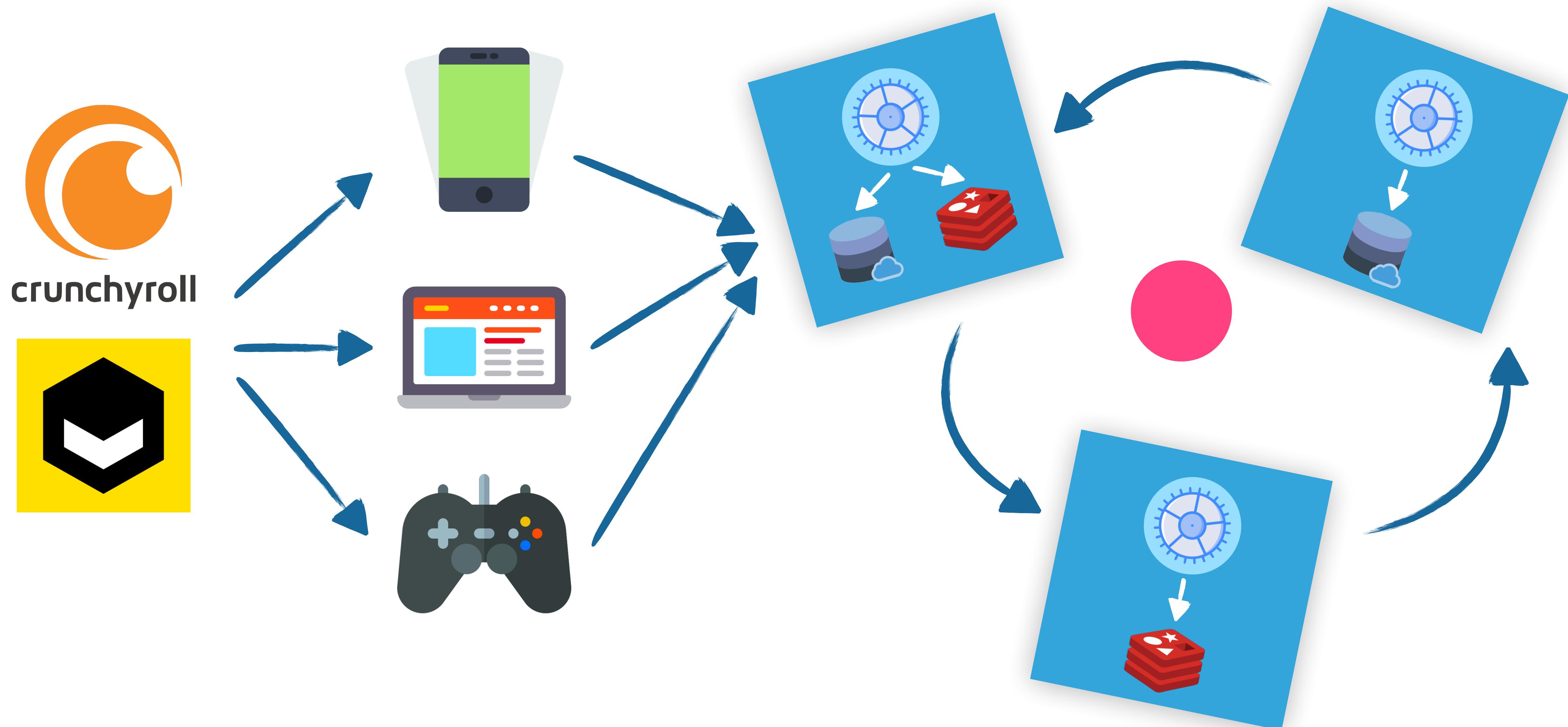


steevehook/mdc19

MY JOURNEY



WHAT DO I DO @ ELLATION



MY EXPERIENCE SO FAR



9 months

Perfect System?



**“Software engineering is what
happens to programming when you
add time and other programmers”**

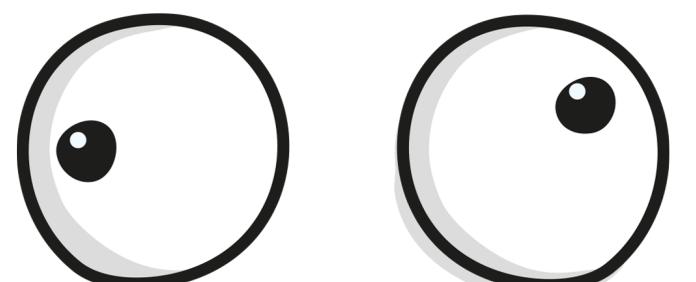
Russ Cox

WILL YOUR COMPANY INVEST IN GO?



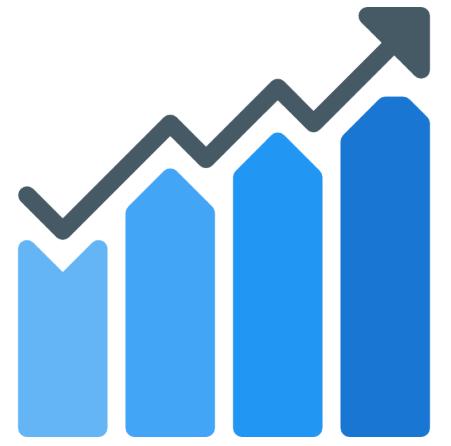
WHY THE F GO?

simplicity



readability

productivity



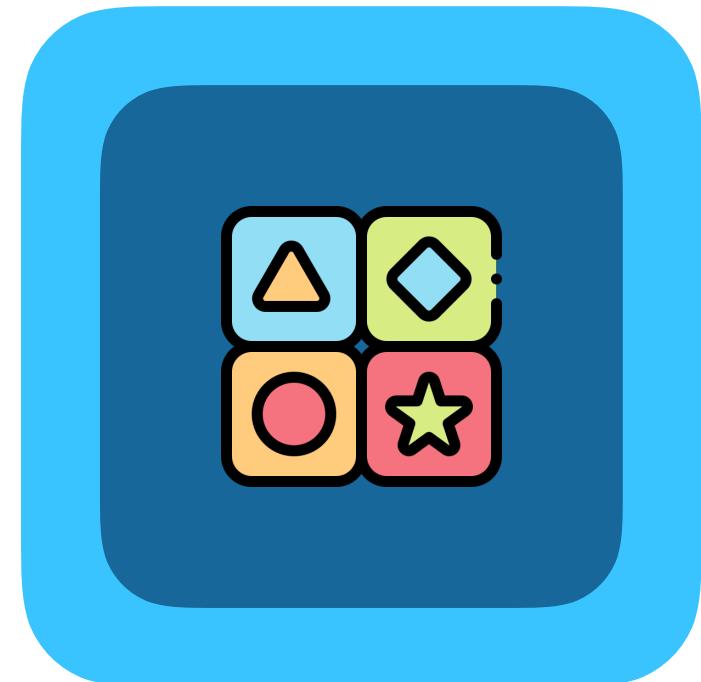
LET'S GO

HELLO WORLD

hello-world.go

```
1  package main
2
3  import "fmt"
4
5  const (
6      hello      = "你好"
7      exclamation = "!"
8 )
9
10 var world string
11
12 func init() {
13     world = "世界"
14 }
15
16 ► func main() {
17     fmt.Println(hello + world + exclamation)
18 }
```

BASIC CONCEPTS



Types



Functions



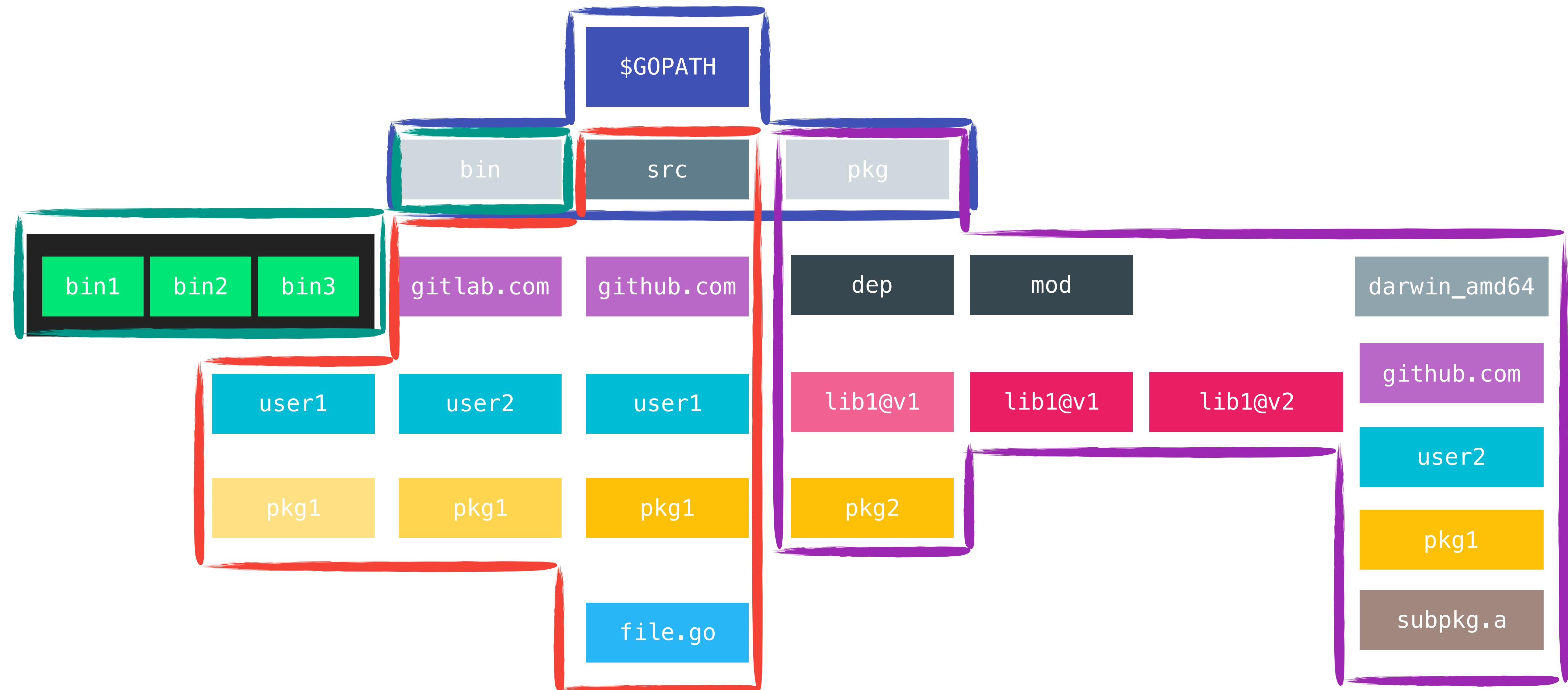
Pointers



Interfaces **Concurrency**



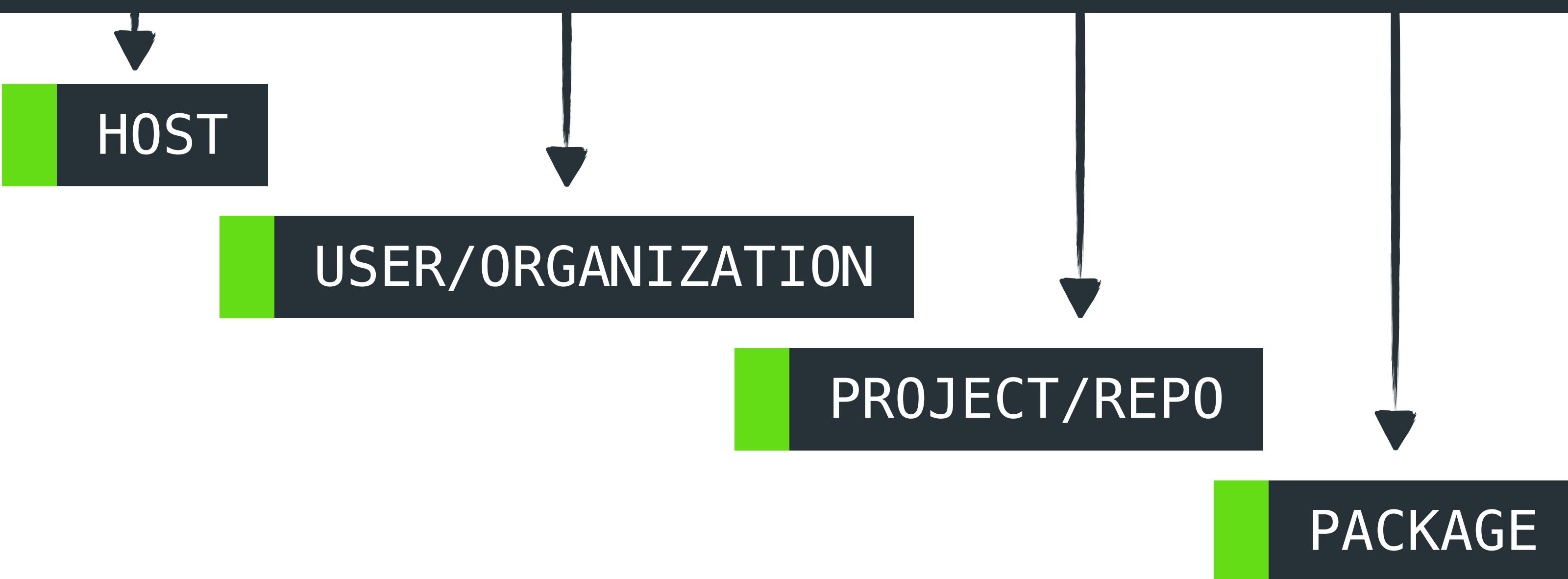
WHAT IS \$GOPATH



PACKAGE IMPORTS



```
package pkg  
import "github.com/steevehook/mdc19/structs"
```



PACKAGES

main.go

```
package main

import (
    "github.com/steevehook/mdc19/examples/packages/pkg1"
    // bad practice, avoid naming the package differently than the dir
    pkg_another "github.com/steevehook/mdc19/examples/packages/pkg2"
)

func main() {
    pkg1.F1()
    pkg_another.F2()
}
```

pkg1/pkg1.go

```
package pkg1
```

```
import "fmt"
```

```
func F1() {
    fmt.Println(a...: "f1")
}
```

pkg1/pkg_another.go

```
package pkg_another
```

```
import "fmt"
```

```
func F2() {
    fmt.Println(a...: "f2")
}
```



PACKAGES BEST PRACTICES



Name them **concise** and **meaningful**

Avoid naming like **utils** and **helpers**

Package name = **directory** name

Forget about **naming collision**

No **cross reference** allowed

Create a **file** exactly as **package name** for **common** things

Create a **dedicated package** for **test utils**

PACKAGE MANAGEMENT



Go mod

Dep

Glide

\$GOPATH

GO MOD

```
go mod help
```

```
go mod init github.com/steevehook/dummy
```

```
go get -u package@v1.0.0
```

```
go get ./...
```

```
go mod tidy
```

```
go mod vendor
```

Does **not** have to be inside **\$GOPATH**

Migrates well from **dep** or **glide**



GO MOD EXAMPLE



main.go

```
package main

import "github.com/sirupsen/logrus"

func main() {
    logger := logrus.WithField(key: "some_key", value: "some_value")
    logger.Println(args...: "printing some args")
}
```

go.mod

```
module github.com/steevehook/mdc19
```

```
go 1.12
```

```
require github.com/sirupsen/logrus v1.4.2
```

go.sum

```
github.com/davecgh/go-spew v1.1.1/go.mod h1:J7Y8YcW2NihsgmVo/mv3lAwl/skON4iLHjSsI+c5H38=
github.com/konsorten/go-windows-terminal-sequences v1.0.1 h1:mweAR1A6xJ3oS2pRaGiHgQ4008tzTaLawm8vnODuwDk=
github.com/konsorten/go-windows-terminal-sequences v1.0.1/go.mod h1:T0+1ngSBFLxvqU3pZ+m/2kptfBszLMUKC4ZK/EgS/cQ=
github.com/pmezard/go-difflib v1.0.0/go.mod h1:iKH77koFhYxTK1pcRnkKkqfTogsbg7gZNVY4sRDYZ/4=
github.com/sirupsen/logrus v1.4.2 h1:SPIRibHv4MatM3XXN02BJeFLZwZ2LvZgfQ5+UNI2im4=
github.com/sirupsen/logrus v1.4.2/go.mod h1:tLMulIdttU9McNUsp0xgXVQah82FyeX6MwdIuYE2rE=
github.com/stretchr/objx v0.1.1/go.mod h1:HFkY916IF+rwdDfMAkV70twuqBVzrE8GR6GFx+wExME=
github.com/stretchr/testify v1.2.2/go.mod h1:a80nRcib4nhh0OaRAV+Yts87kKdq0PP7pXfy6kDkUVs=
golang.org/x/sys v0.0.0-20190422165155-953cdadca894 h1:Cz4ceDQGXuKRnVBDTs23GTn/pU5OE2C0WrNTOYK1Uuc=
golang.org/x/sys v0.0.0-20190422165155-953cdadca894/go.mod h1:h1NjWce9XRLGQEsw7wpKNCjG9DtNlclVuFLEZdDNbEs=
```

BASIC TYPES



bool

uint

float32

int

uint8

float64

int8

uint16

complex64

int16

uint32

complex128

int32/rune

uint64

string

int64

uintptr

CONST/VAR DECLARATIONS



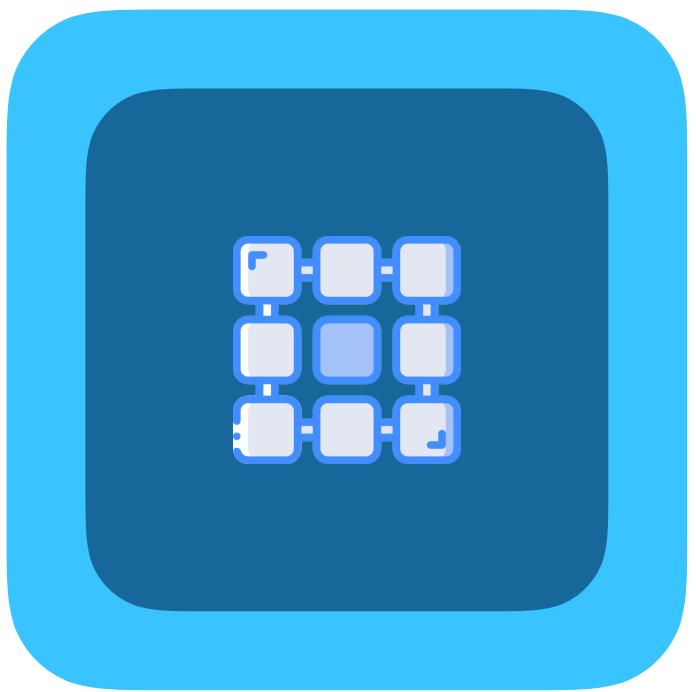
constants.go

```
1 package main
2
3 import "fmt"
4
5 const c1 = 10
6 const s string = "Hello"
7
8 const (
9     g1      = 10
10    g2 int8 = 32
11 )
12
13 func main() {
14     fmt.Println(c1, s, g1, g2)
15 }
```

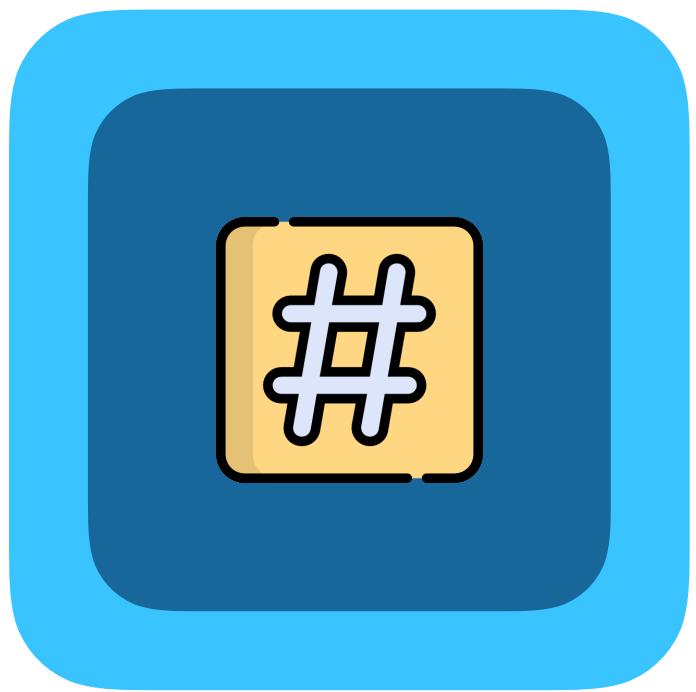
variables.go

```
1 package main
2
3 import "fmt"
4
5 var s1 string = "s1"
6 var s2 = "s2"
7 var s3 string
8
9 // group
10 var (
11     g1 = 1
12     g2 = 2
13 )
14
15 func main() {
16     s3 = "s3"
17     s4 := "s4"
18     fmt.Sprintf(format: "%s\n%s\n%s\n%s\n%d\n%d", s1, s2, s3, s4, g1, g2)
19 }
```

AGGREGATE TYPES



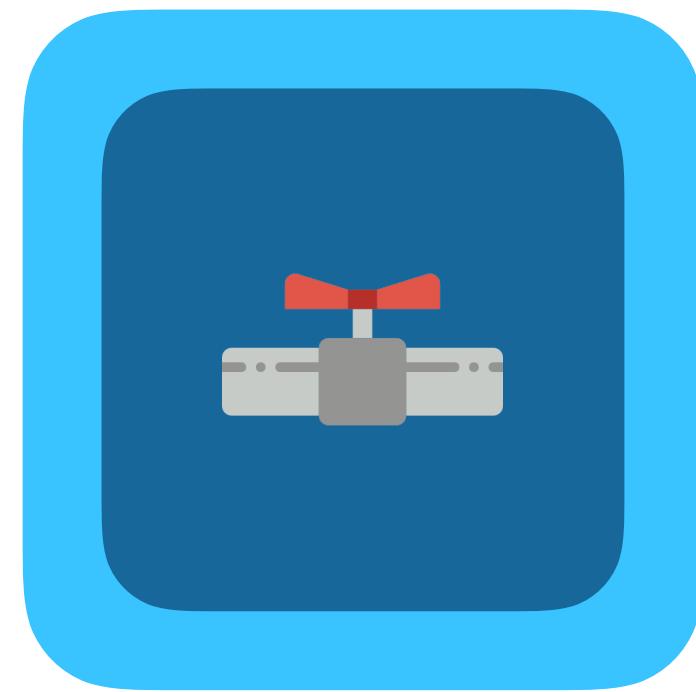
Arrays/Slices



Maps

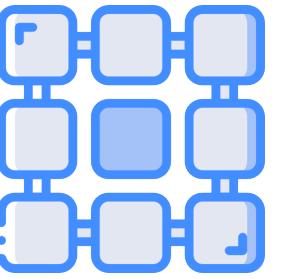


Structs



Channels

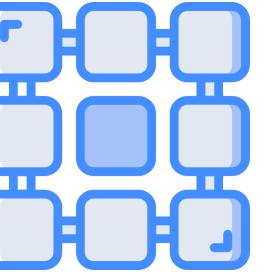
ARRAYS



arrays.go

```
1 package main
2
3 import "fmt"
4
5 ► func main() {
6     arrInt, arrString := [3]int{1, 2, 3}, [2]string{"Hello", "World"}
7     //arrIntErr := [2]int{10, 20, 30} // index out of bounds: 2
8     arrVar := [...]string{
9         "Some", "very", "long", "string", "array", "Heck", "knows", "the", "end", "of", "it",
10    }
11    arrVarCap := cap(arrVar)
12    fmt.Printf( format: "%d\n%s\n%s\n", arrInt, arrString, arrVar)
13    fmt.Printf( format: "The type of arrVar: %T", arrVar) // [11]string
14    fmt.Printf( format: "The capacity of arrVar: %d", arrVarCap)
15 }
```

SLICES



slices.go

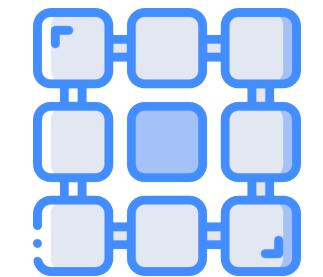
```
package main

import "fmt"

var sComplex []complex128

func main() {
    sInt, sString := []int{1,2,3}, []string{"Hello", "World"}
    s1, s2, s3 := make([]int, 100), make([]int, 0, 100), make([]int, 10, 100)
    sComplex = append(sComplex, elems...: 2 + 3i, 4+2i, 5, 6)
    newIntSlice := append([]int{10, 20}, sInt...)
    fmt.Printf(format: "%d\n%s\n%v\n%d\n", sInt, sString, sComplex, newIntSlice)
    fmt.Printf(format: "s1|len:%d<=>cap:%d\ns2|len:%d<=>cap:%d\ns3|len:%d<=>cap:%d\n",
        len(s1), cap(s1), len(s2), cap(s2), len(s3), cap(s3),
    )
}
```

LEN/CAP/APPEND/MAKE



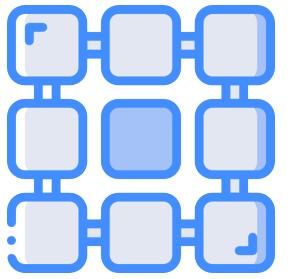
`len(array/slice)`

`cap(array/slice)`

`append(slice , elem , elem , ...)`

`make(slice/map/chan , [len , cap])`

ARRAY/Slice ADDRESSING



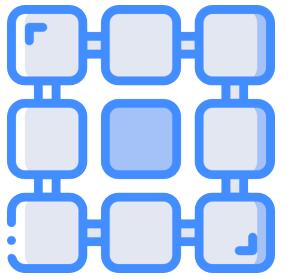
arrays-slices-addressing.go

```
package main

import "fmt"

func main() {
    s := make([]int, 10, 12)
    fmt.Println(a...:"s:", s) // [0 0 0 0 0 0 0 0 0 0]
    s[0], s[5] = 10, 15
    fmt.Printf(format: "s[0]:%d\ns[5]:%d\ns[9]:%d\n", s[0], s[5], s[9])
    fmt.Println(a...:"cap(s):", cap(s)) // 12
    s = append(s, elems...:1, 2, 3)
    fmt.Println(a...:"cap(s):", cap(s)) // 24
    //fmt.Println(s[11]) // index out of range
}
```

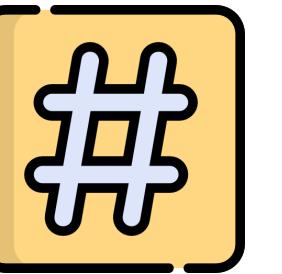
ARRAY/SLICE LOOPING



array-slice-looping.go

```
1  package main
2
3  import "fmt"
4
5  ► func main() {
6      names := []string{"John", "Mike", "Jane"}
7      for i := 0; i < len(names); i++ {
8          fmt.Println(names[i])
9      }
10     for index, name := range names {
11         fmt.Println(index, name)
12     }
13     for _, name := range names {
14         fmt.Println(name)
15     }
16 }
```

MAPS



maps.go

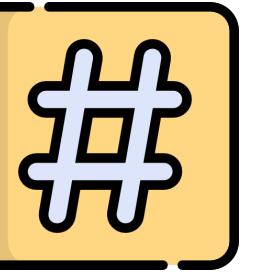
```
package main

import "fmt"

var dictionary map[string]string // nil map (either init it later or avoid this)

func main() {
    //dictionary["some_key"] = "some_value" // panic: assignment to entry in nil map
    wiktictionary := make(map[string]string)
    wiktictionary["some_key"] = "some_value"
    grades := map[string]int{
        "Math":    10,
        "English": 10,
    }
    fmt.Printf( format: "dictionary: %+v\nwiktictionary: %+v\ngrades: %+v\n", dictionary, wiktictionary, grades)
}
```

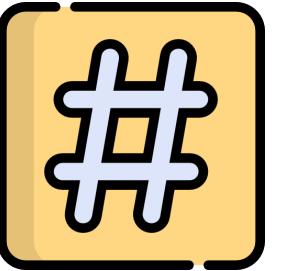
MAPS LOOPING



maps-looping.go

```
1 package main
2
3 import "fmt"
4
5 ► func main() {
6     grades := map[string]int{
7         "Math": 8,
8         "English": 10,
9         "Physics": 9,
10    }
11    // the order may always be different (because it's hashmap)
12    for subject, grade := range grades {
13        fmt.Printf(format: "%s ==> %d\n", subject, grade)
14    }
15 }
```

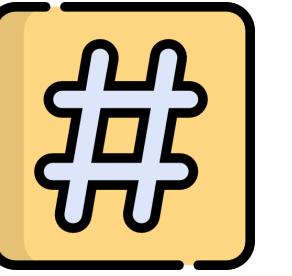
MAPS COMMA, OK IDIOM



maps-comma-ok.go

```
1 package main
2
3 import "fmt"
4
5 ► ⌄ func main() {
6     dictionary := map[string]string{"some_key": "some_value"}
7     val, ok := dictionary["another_key"]
8     fmt.Printf(format: "value: %v <=> ok: %v\n", val, ok)
9     val, ok = dictionary["some_key"]
10    fmt.Printf(format: "value: %v <=> ok: %v\n", val, ok)
11 }
```

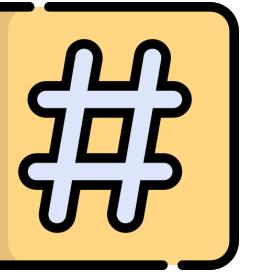
NIL MAP



nil-map.go

```
1 package main
2
3 import "fmt"
4
5 var robot map[string]func() // nil map
6
7 func init() {
8     robot = map[string]func{}{
9         "start": func() {
10             fmt.Println("robot started")
11         },
12     }
13 }
14
15 ► func main() {
16     robot["start"]()
17     // panic: runtime error: invalid memory address or nil pointer dereference
18     [robot["stop"]()]
19 }
```

MAPS DELETE



maps-delete.go

```
1 package main
2
3 import "fmt"
4
5 ► func main() {
6     dictionary := map[string]string{
7         "human": "being that has 1 life",
8         "programmer": "being that has no life",
9     }
10    fmt.Printf( format: "%+v\n", dictionary)
11    delete(dictionary, key: "programmer")
12    delete(dictionary, key: "non-existent") // no error
13    fmt.Printf( format: "%+v\n", dictionary)
14 }
```

ZERO VALUE



bool

false

pointer

nil

int/uint

0

error

nil

float

0

map

nil

complex

0 + 0i

chan

nil

string

""

func

nil

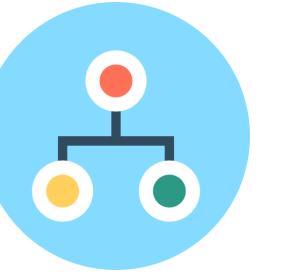
interface{}

nil

struct

empty struct with zero value fields

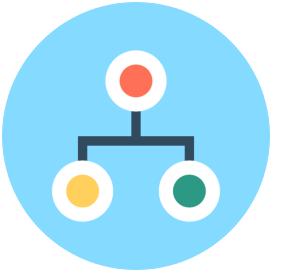
STRUCTS



structs.go

```
1 package main
2
3 import "fmt"
4
5 type person struct {
6     Name string
7     Age  uint
8 }
9
10 func main() {
11     people := []struct{
12         name string
13         age  uint
14     } {
15         {name: "John", age: 21},
16         {name: "Jane", age: 19}, // don't use this
17     }
18     fmt.Printf( format: "%+v\n", person{Name: "Steve", Age: 24})
19     fmt.Printf( format: "%+v", people)
20 }
```

STRUCTS - EXPORTED/UNEXPORTED FIELDS



person/person.go

```
package person

// Person represents the person type
type Person struct {
    name string
    age  uint
    Hobby string
}
func NewPerson(name string, age uint, hobby string) Person {
    return Person{name: name, age: age, Hobby: hobby}
}
func (p Person) Name() string {
    return p.name
}
func (p Person) Age() uint {
    return p.age
}
```

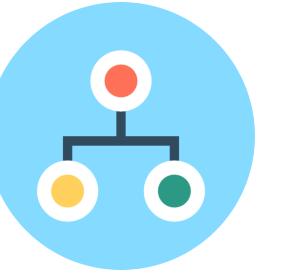
main.go

```
package main

import (
    "fmt"
    "github.com/steevehook/mdc19/examples/struct-exported-unexported/person"
)

func main() {
    p1 := struct {
        name string
        age  uint
    }{name: "Steve", age: 24}
    p2 := person.NewPerson( name: "John", age: 19, hobby: "developer")
    p3 := person.Person{Hobby: "human"}
    fmt.Printf( format: "p1:\nname: %s\nage: %d\n\n", p1.name, p1.age)
    fmt.Printf( format: "p2:\nname: %s\nage: %d\nHobby: %s\n\n", p2.Name(), p2.Age(), p2.Hobby)
    fmt.Printf( format: "p3:\nname: %s\nage: %d\nHobby: %s\n", p3.Name(), p3.Age(), p3.Hobby)
}
```

STRUCTS - FIELD PROMOTION



`struct-field-promotion.go`

```
package main

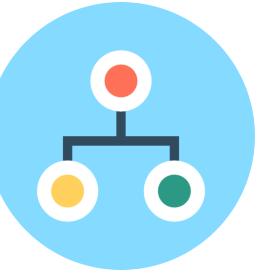
import "fmt"

type person struct {
    name string
    age  uint
}

type developer struct {
    person
    //age uint // this will override the previous field
    language string
}

func main() {
    d := developer{
        language: "Go",
        person:   person{name: "Steve", age: 24},
    }
    fmt.Printf( format: "developer info\n---\nname: %s\nage: %d\nlanguage: %s", [d.name, d.person.age], d.language)
}
```

STRUCTS - JSON



structs-json.go

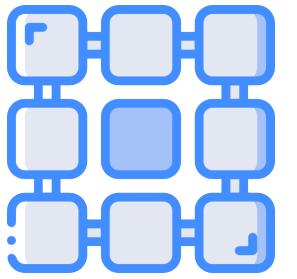
```
package main

import (
    "encoding/json"
    "fmt"
)

type person struct {
    Name string `json:"name"`
    Age  uint   `json:"age,omitempty"`
    //sex  bool   `json:"sex"` // not gonna work
    sex  bool   `json:"sex"`
    Hobby string `json:"-"`
}

func main() {
    // always handle errors :D
    bs1, _ := json.Marshal(person{Name: "Steve", Age: 24, sex: true, Hobby: "developer"})
    bs2, _ := json.Marshal(person{Name: "Jane", sex: false, Hobby: "human"})
    fmt.Println(string(bs1)) // {"name": "Steve", "age": 24}
    fmt.Println(string(bs2)) // {"name": "Jane"}
}
```

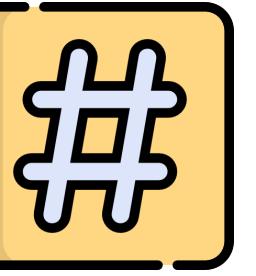
ARRAYS/SLICES - JSON



arrays-slices-json.go

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6 )
7
8 ► func main() {
9     grades := []int{10, 20, 30, 40}
10    people := []struct {
11        Name string `json:"name"`
12    }{{Name: "Steve"}, {Name: "John"}, {Name: "Jane"}, {Name: "Mary"}}
13    // handle errors you bustard
14    gradesBS, _ := json.Marshal(grades)
15    peopleBS, _ := json.Marshal(people)
16    // [10, 20, 30, 40]
17    // [{"name": "Steve"}, {"name": "John"}, {"name": "Jane"}, {"name": "Mary"}]
18    fmt.Printf(format: "grades:\n%s\n\npeople:\n%s", string(gradesBS), string(peopleBS))
19 }
```

MAPS - JSON



maps-json.go

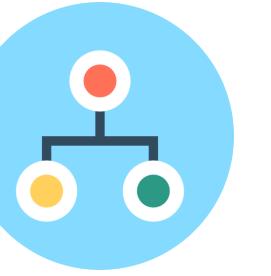
```
package main

import (
    "encoding/json"
    "fmt"
)

type person struct {
    Name string `json:"name"`
    Age  uint   `json:"age,omitempty"`
}

func main() {
    people := map[string]person{
        "fa1b4f6f-beab-5e55-acdf-6da3a03aabec": {
            Name: "Steve", Age: 24,
        },
        "fa1b4f6f-beab-5e55-acdf-6da3a03aabed": {
            Name: "John",
        },
    }
    // handle errors, PLEASE :D
    bs, _ := json.Marshal(people)
    // {
    //     "fa1b4f6f-beab-5e55-acdf-6da3a03aabec": {"name": "Steve", "age": 24},
    //     "fa1b4f6f-beab-5e55-acdf-6da3a03aabed": {"name": "John"}
    // }
    fmt.Println(string(bs))
}
```

JSON UNMARSHALLING



json-unmarshal.go

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6 )
7
8 type person struct {
9     Name string `json:"name"`
10    Age uint `json:"age,omitempty"`
11    //sex bool `json:"sex"` // not gonna work
12    sex bool
13    Hobby string `json:"-"`
14 }
15
16 func main() {
17     j1 := `{"name": "Mary", "age": 21, "sex": true, "hobby": "human"}`
18     j2 := `{"name": "Steve", "age": 24, "anything_else": "not_gonna_unmarshal"}`
19     var p1 person
20     _ = json.Unmarshal([]byte(j1), &p1)
21     var p2 person
22     _ = json.Unmarshal([]byte(j2), &p2)
23     fmt.Printf( format: "%+v\n\n%+v", p1, p2)
24 }
```

FUNCTIONS



main.go

```
package main

import (
    "fmt"
    "log"
)

func main() {
    nums := []int{10, 15, -2, 4}
    for _, n := range nums {
        i, err := identity(n)
        if err != nil {
            log.Fatal(err)
        }
        fmt.Println(i)
    }
}
```

identity.go

```
package main

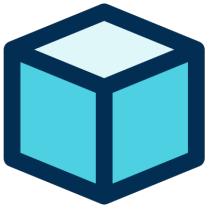
import "errors"

func identity(num int) (int, error) {
    if num < 0 {
        return 0, errors.New(text: "number is lower than 0")
    }
    return num, nil
}
```

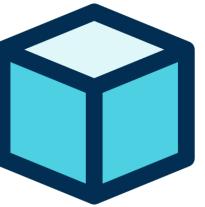
CUSTOM TYPES

custom-types.go

```
1 package main
2
3 import "fmt"
4
5 type age int
6 type person struct {
7     name string
8     age  uint
9 }
10
11 ► func main() {
12     var p person
13     p1 := struct {
14         name string
15         age  uint
16     }{name: "Jane", age: 19}
17     p = p1
18     var a1 int
19     var a2 age
20     a1 = int(a2)
21     //a1 = a2 // won't work
22     fmt.Printf( format: "person: %+v\nage: %d\n\n", person{name: "Steve", age: 24}, age(1))
23     fmt.Printf( format: "p: %+v\na1:%d", p, a1)
24 }
```



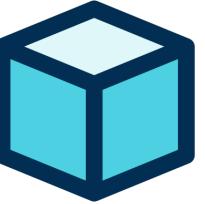
METHODS / FUNCTION RECEIVERS



type-methods.go

```
1 package main
2
3 import "fmt"
4
5 type age uint
6
7 func (a *age) BirthDay() *age {
8     *a += 1
9     return a
10 }
11 func (a age) IsDead() bool {
12     return a > 150
13 }
14
15 func main() {
16     a := age(148)
17     a.BirthDay().BirthDay()
18     fmt.Printf(format: "age: %d\nis dead: %t\n\n", a, a.IsDead())
19     a.BirthDay()
20     fmt.Printf(format: "age: %d\nis dead: %t", a, a.IsDead())
21 }
```

POINTERS



main.go

```
package main

import "fmt"

func main() {
    var a, b *int
    iPtr := intPtr(i: 10)
    a = iPtr
    b = iPtr
    *a = 10
    fmt.Printf(format: "a: %d\nb: %d\niPtr: %d", *a, *b, *iPtr)
}
```

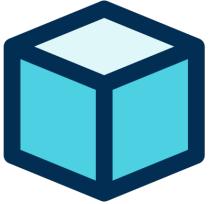
int-pointer.go

```
package main

func intPtr(i int) *int {
    var n int
    n = i
    return &n
}
```

POINTER VS VALUE

pointer-vs-value.go



```
package main

import "fmt"

type person struct {
    name string
}

func (p person) changeNameValue(name string) {
    p.name = name
}

func (p *person) changeNamePointer(name string) {
    p.name = name
}

func main() {
    p1, p2 := person{name: "Steve"}, &person{name: "George"}
    p1.changeNameValue( name: "John")
    fmt.Println(p1.name)
    p1.changeNamePointer( name: "John")
    fmt.Println(p1.name)
    p2.changeNameValue( name: "Unknown")
    fmt.Println(p2.name)
}
```

INTERFACES



types.go

```
package main

import "fmt"

type person struct {
    name string
}

func (p person) Run() {
    fmt.Println(p.name, "is running")
}

type runner interface {
    Run()
}
```

main.go

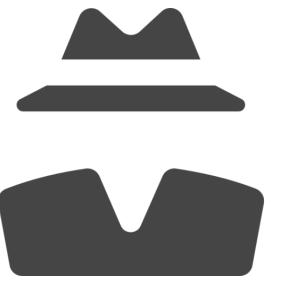
```
package main

import "fmt"

func marathon(runners ...runner) {
    for _, r := range runners {
        r.Run()
    }
}

func main() {
    steve := person{name: "Steve"}
    john := struct {
        name string
    }{
        name: "John",
    }
    marathon(steve)
    //marathon(steve, john) // incompatible types
    fmt.Println(john.name, "is running")
}
```

INTERFACE{} TYPE



Represents **any type** in Go

Any type **implements** it

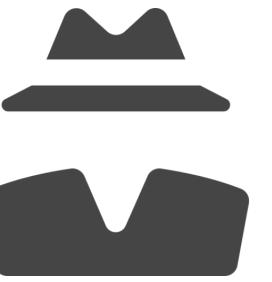
Does not equal to **nil**

Supports **type assertion**

Has **type** and **value** fields

The **value** has to be a **concrete type**

INTERFACE{} EXAMPLE



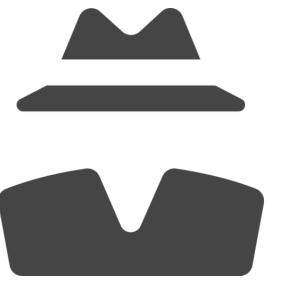
interface{}.go

```
package main

import "fmt"

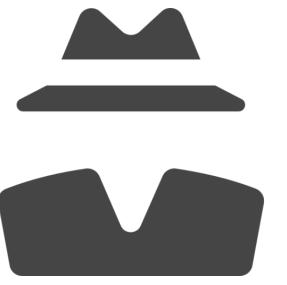
func main() {
    anything := map[string]interface{}{
        "animals": []string{"lion", "bear"},
        "grades": []int{10, 8, 9},
        "people": []struct {
            name string
        }{
            {name: "Steve"},
            {name: "John"},
        },
    }
    fmt.Printf("Anything: %+v", anything)
}
```

THE POWER OF INTERFACES



```
type Reader {  
    Read(p []byte) (n int, err error)  
}  
  
type Writer {  
    Write(p []byte) (n int, err error)  
}
```

IMPLEMENT ERROR INTERFACE



main.go

```
package main

import "log"

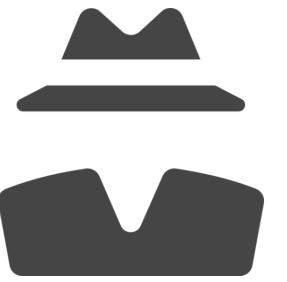
func handle(err error) {
    log.Fatal(err)
}

func main() {
    handle(CustomErr{Message: "some error"})
}
```

custom-error.go

```
1 package main
2
3 fx type CustomErr struct {
4     Message string
5 }
6
7 fx func (e CustomErr) Error() string {
8     return e.Message
9 }
```

TYPE ASSERTION



main.go

```
package main

import (
    "errors"
    "log"
    "net/http"
)

func main() {
    http.HandleFunc("/http", func(w http.ResponseWriter, req *http.Request) {
        SendError(w, http.StatusInternalServerError, HTTPError{Message: "something bad happen"})
    })

    http.HandleFunc("/data", func(w http.ResponseWriter, req *http.Request) {
        SendError(w, http.StatusBadRequest, DataValidationError{
            Message: "something bad happen",
            Constraints: map[string]interface{}{
                "field1": "some error on field1",
            },
        })
    })

    http.HandleFunc("/error", func(w http.ResponseWriter, req *http.Request) {
        SendError(w, http.StatusInternalServerError, errors.New("test"))
    })
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

encoder.go

```
package main

import (
    "encoding/json"
    "net/http"
)

func toJSON(w http.ResponseWriter, code int, data interface{}) {
    w.WriteHeader(code)
    // handle errors, bustard
    _ = json.NewEncoder(w).Encode(data)
}

func SendError(w http.ResponseWriter, code int, e error) {
    switch err := e.(type) {
    case DataValidationError:
        err.Type = DataValidationType
        toJSON(w, code, err)
    case HTTPError:
        err.Type = HTTPErrorType
        toJSON(w, code, err)
    default:
        toJSON(w, code, HTTPError{
            Type:    HTTPErrorType,
            Message: "some http error",
        })
    }
}
```

errors.go

```
package main

const (
    HTTPErrorType      = "http_error"
    DataValidationType = "data_validation"
)

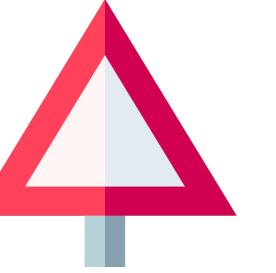
type DataValidationError struct {
    Type      string `json:"type"`
    Message   string `json:"message"`
    Constraints map[string]interface{} `json:"constraints"`
}

func (e DataValidationError) Error() string {
    return e.Message
}

type HTTPError struct {
    Type      string `json:"type"`
    Message   string `json:"message"`
}

func (e HTTPError) Error() string {
    return e.Message
}
```

ERRORS



Errors are just **values**

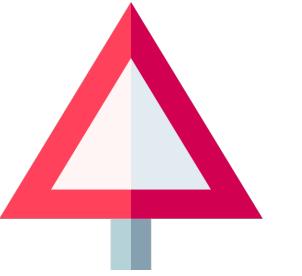
There are **no exceptions** and no catching

The **error** usually comes as the **2nd return**

error is just another **interface** (`Error()` string)

Error messages don't start with **Uppercase** or have **weird spacing**

ERRORS EXAMPLE



main.go

```
package main

import (
    "fmt"
    "log"
)

func main() {
    nums := []int{10, -15}
    for _, n := range nums {
        val, err := identity(n)
        if err != nil {
            log.Fatal(err)
            return
        }
        fmt.Println("identity:", val)
    }
}
```

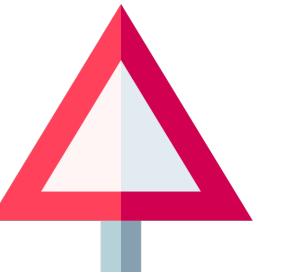
identity.go

```
package main

import "errors"

func identity(n int) (int, error) {
    if n < 0 {
        return 0, errors.New("the number is less than zero")
    }
    return n, nil
}
```

DEFER



defer.go

```
package main

import "fmt"

type operation struct{}

func (op operation) SideEffects() {
    fmt.Println("doing some side effects")
}

func (op operation) CleanUp() {
    fmt.Println("cleaning up after the side effects")
}

func main() {
    o := operation{}
    o.SideEffects()
    defer o.CleanUp()
    fmt.Println("other operations")
    panic(v: "some serious error")
}
```

defer-loop.go

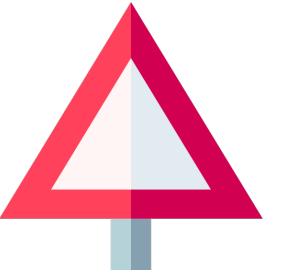
```
package main

import "fmt"

func clean(i int) {
    fmt.Printf("cleanup for operation: %d\n", i)
}

func main() {
    for i := 0; i < 10; i++ {
        defer clean(i)
    }
}
```

PANIC & RECOVER



panic-recover.go

```
package main

import "fmt"

func yikes() {
    panic("can't do anything")
}

func main() {
    defer func() {
        err := recover()
        fmt.Println(err)
    }()
    yikes()
}
```

Don't **panic**! Avoid panicking

You can **recover** from any **panic**

A **panic** shuts down the entire app with an **exit code**

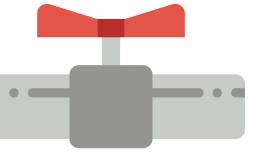
Have a **panic recovery middleware**, for graceful handling

You can't **recover** on the same **function level**

panic can contain **any value**

Can't **recover** without **defer**

CHANNELS



main.go

```
package main

import "time"

type message struct {
    id   string
    text string
}

func main() {
    messages := make(chan message)
    go listen(messages)
    messages <- message{id: "1", text: "first message"}
    messages <- message{id: "2", text: "second message"}
    time.Sleep(1 * time.Second)
}
```

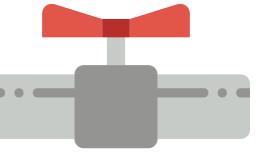
listen.go

```
package main

import "fmt"

func listen(messages chan message) {
    for {
        select {
        case msg := <-messages:
            fmt.Printf(format: "%+v", msg)
        default:
            return
        }
    }
}
```

CHANNELS FACTS



They **block** when the **buffer** is **full**

Can cause **deadlocks** if no routine reads it

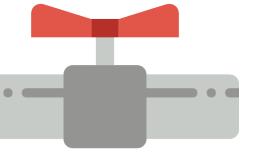
Should be **closed** at the **end**

Can't be **written to** if **closed**

If **main go routine** does not **block** it **exits**

Ranging over **unclosed chan** causes **deadlock**

BUFFERED CHANNELS



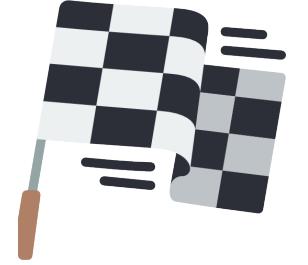
buffered-channels.go

```
package main

import (
    "fmt"
    "time"
)

func main() {
    buffered := make(chan int, 2)
    buffered <- 1 // will not block - buffer = 1
    buffered <- 2 // will not block - buffer = 2
    //buffered <- 3 // WILL BLOCK - buffer > 2
    go func() {
        for i := range buffered {
            fmt.Println(i)
        }
    }()
    buffered <- 3
    time.Sleep(1 * time.Second)
}
```

DEADLOCK



deadlock.go

```
package main

func main() {
    ch := make(chan int)
    ch <- 10
}
```

deadlock-fix.go

```
package main

import "fmt"

func main() {
    ch := make(chan int)
    go func() {
        fmt.Println(<-ch)
    }()
    ch <- 10
}
```

“Concurrency is not parallelism”

CONCURRENCY (GO ROUTINES)



concurrency.go

```
package main

import (
    "fmt"
    "time"
)

func main() {
    go func() {
        fmt.Println("routine 1")
    }()
    go func() {
        fmt.Println("routine 2")
    }()
    go func() {
        fmt.Println("routine 3")
    }()
    time.Sleep(1 * time.Second)
}
```

Start a go routing using **go** in front of **func call**

They are **not threads**

They **don't start** in the **order** they **appear**

They can cause **data races**

Scheduler creates **1 thread** per **CPU core**

RACE CONDITIONS



race-conditions.go

```
package main

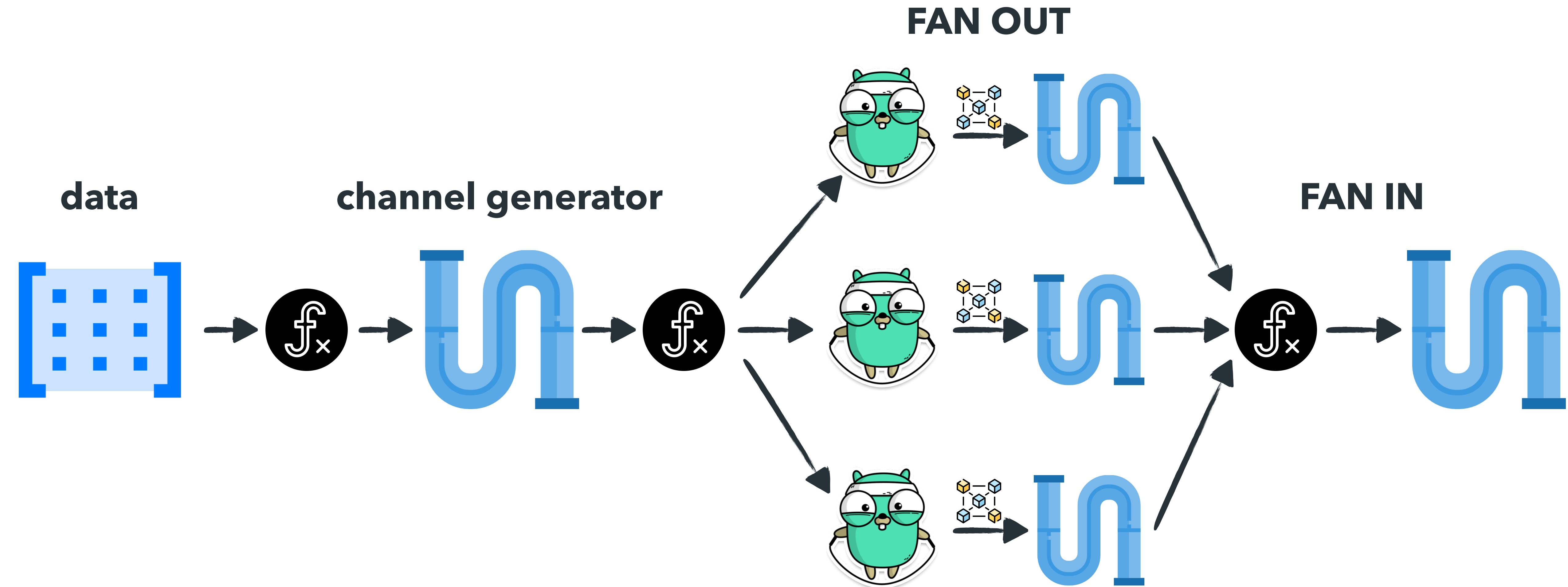
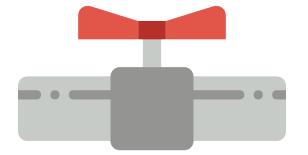
import (
    "fmt"
    "time"
)

func main() {
    n := 10
    go func() {
        n = 15
    }()
    go func() {
        n = 12
    }()
    go func() {
        fmt.Println(n)
    }()
    time.Sleep(1 * time.Second)
}
```

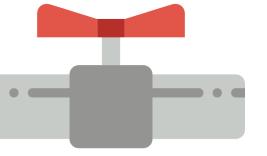
go run -race *.go

go test -race ./...

CHANNELS FAN OUT / FAN IN PATTERN



CHANNELS FAN IN/FAN OUT



main.go

```
package main

import (
    "fmt"
    "time"
)

func generate(numbers ...int) chan int {
    out := make(chan int)
    go func() {
        for _, num := range numbers {
            out <- num
        }
        close(out)
    }()
    return out
}

func main() {
    numbers := generate(numbers...: 3, 6, 9)
    c1 := fanOut(numbers)
    c2 := fanOut(numbers)
    result := fanIn(c1, c2)
    for i := range result {
        fmt.Println(i)
    }
    time.Sleep(1 * time.Second)
}
```

fan-out.go

```
package main

func fanOut(in chan int) chan int {
    out := make(chan int)
    go func() {
        for i := range in {
            out <- square(i)
        }
        close(out)
    }()
    return out
}

func square(num int) int {
    return num * num
}
```

fan-in.go

```
package main

func fanIn(channels ...chan int) chan int {
    out := make(chan int)
    done := make(chan struct{})
    for _, c := range channels {
        go func(in chan int) {
            for num := range in {
                out <- num
            }
            done <- struct{}{}
        }(c)
    }
    go func() {
        for i := 0; i < len(channels); i++ {
            <-done
        }
    }()
    close(out)
    return out
}
```

HTTP



main.go

```
package main

import (
    "log"
    "net/http"
)

func main() {
    mux := http.NewServeMux()
    mux.HandleFunc(pattern: "/", home())
    mux.HandleFunc(pattern: "/about", about())
    log.Fatal(http.ListenAndServe(addr: ":8080", mux))
}
```

about.go

```
func about() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        // handle errors, son of the bitch
        _, _ = fmt.Fprint(w, a...: "About")
    }
}
```

home.go

```
func home() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        // handle errors, son of the bitch
        _, _ = fmt.Fprint(w, a...: "Welcome Home!")
    }
}
```

TCP (MEM CACHE)

main.go

```
package main

import (
    "fmt"
    "log"
    "net"
)

func main() {
    cache := cache{}

    listener, err := net.Listen(
        network: "tcp",
        address: ":8080",
    )

    defer listener.Close()

    if err != nil {
        log.Fatal(err)
    }

    for {
        // handle errors, you bustard
        conn, _ := listener.Accept()
        fmt.Fprintf(
            conn,
            format: "Welcome to CACHE 1.0\n-> ",
        )
        go listen(conn, cache)
    }
}
```

listen.go

```
package main

import (
    "bufio"
    "fmt"
    "net"
    "strings"
)

func listen(c net.Conn, mem cache) {
    scanner := bufio.NewScanner(c)
    for scanner.Scan() {
        l := strings.ToLower(strings.TrimSpace(scanner.Text()))
        values := strings.Split(l, sep: " ")
        switch {
        case len(values) == 3 && values[0] == "set":
            mem.Set(values[1], values[2])
            fmt.Fprintf(c, format: "OK\n-> ")
        case len(values) == 2 && values[0] == "get":
            fmt.Fprintf(c, format: "%s\n-> "
                mem.Get(values[1]))
        case len(values) == 1 && values[0] == "exit":
            c.Close()
        default:
            fmt.Fprintf(c, format: "UNKNOWN: %s\n-> ", l)
        }
    }
}
```

cache.go

```
package main

import "sync"

var mu sync.RWMutex

type cache map[string]string

func (c cache) Set(key, value string) {
    mu.Lock()
    defer mu.Unlock()
    c[key] = value
}

func (c cache) Get(key string) string {
    mu.RLock()
    defer mu.RUnlock()
    return c[key]
}
```



USEFUL GO COMMANDS



```
go run *.go
```

```
go build -o exec
```

```
go install
```

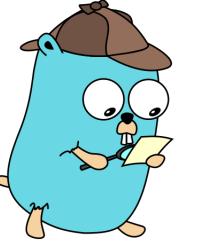
```
go get ./...
```

```
go test ./...
```

```
go fmt ./...
```

```
go vet ./...
```

HOW WE DEVELOP?



We use a **Makefile**

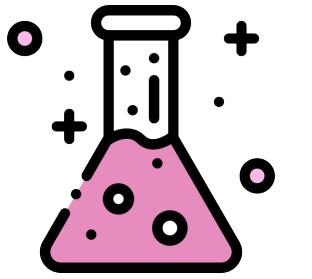
Write the **functionality**

Unit test each component

Integration test the whole service

Write **swagger** docs

HOW WE TEST?



```
go test ./...
```

Use **testify** for test **suites, assertions & mocks**

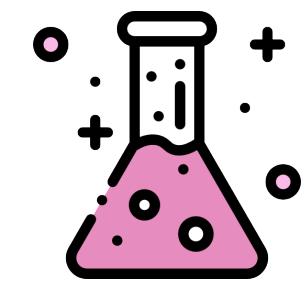
controllers/comments.go

```
type commentsService interface {
    GetComments() ([]string, error)
}

func GetComments(service commentsService) http.HandlerFunc {
    return func(w http.ResponseWriter, req *http.Request) {
        switch req.Method {
        case http.MethodGet:
            comments, err := service.GetComments()
            if err != nil {
                w.WriteHeader(http.StatusServiceUnavailable)
                return
            }
            err = json.NewEncoder(w).Encode(comments)
            if err != nil {
                w.WriteHeader(http.StatusBadRequest)
            }
        default:
            w.WriteHeader(http.StatusMethodNotAllowed)
        }
    }
}
```

main.go

```
func main() {
    commentsService := services.NewComments()
    http.HandleFunc(
        pattern: "/comments",
        controllers.GetComments(commentsService),
    )
    log.Fatal(http.ListenAndServe(
        addr: ":8080",
        handler: nil,
    ))
}
```



services/comments.go

```
type CommentsService struct{}

func NewComments() CommentsService {
    return CommentsService{}
}

func (c CommentsService) GetComments() ([]string, error) {
    return []string{
        "comment1",
        "comment2",
    }, nil
}
```

services/comments_test.go

```
type commentsSuite struct {
    suite.Suite
    service CommentsService
}

func (s *commentsSuite) SetupSuite() {
    s.service = NewComments()
}

func (s *commentsSuite) Test_GetComments_Success() {
    expected := []string{
        "comment1",
        "comment2",
    }

    comments, err := s.service.GetComments()

    s.Require().NoError(err)
    s.Equal(expected, comments)
}

func TestComments(t *testing.T) {
    suite.Run(t, new(commentsSuite))
}
```

controllers/comments_test.go

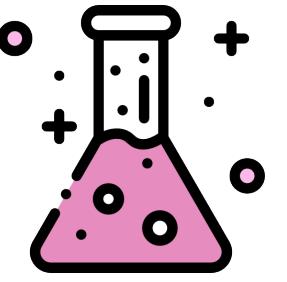
```
type commentsSuite struct {
    suite.Suite
    service *commentsServiceMock
}

func (s *commentsSuite) SetupSuite() {
    s.service = new(commentsServiceMock)
}

func (s *commentsSuite) TearDownSuite() {
    s.service.AssertExpectations(s.T())
}

func (s *commentsSuite) Test_GetComments_Success() {
    body := `["c1", "c2"]`
    req := httptest.NewRequest(
        http.MethodGet,
        "/comments",
        body,
    )
    w := httptest.NewRecorder()
    handler := http.HandlerFunc(GetComments(s.service))
    s.service.
        On(methodName: "GetComments").
        Return([]string{"c1", "c2"}, nil).Once()
    handler.ServeHTTP(w, req)
    s.Equal(expected: 200, w.Code)
    s.JSONEq(body, w.Body.String())
}

func TestComments(t *testing.T) {
    suite.Run(t, new(commentsSuite))
}
```

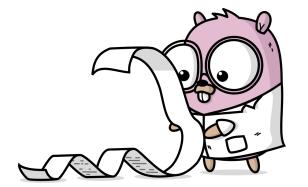


controllers/mocks_test.go

```
type commentsServiceMock struct {
    mock.Mock
}

func (m *commentsServiceMock) GetComments(
) ([]string, error) {
    args := m.Called()
    return args.Get(index: 0).([]string),
    args.Error(index: 1)
}
```

HOW WE DOCUMENT?

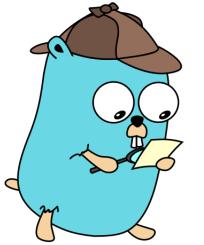


We use **// comments**

We document any **exported symbols**, except implementations

We optionally document **un exported**(private) symbols

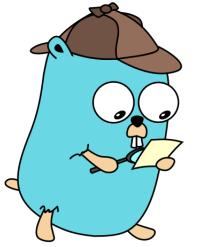
WHAT CODE STYLE WE USE?



```
go fmt ./...
```

```
go vet ./...
```

HOW WE DEPLOY?

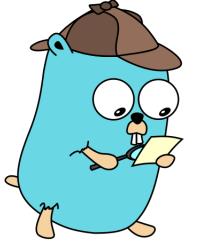


Compile the code & generate the **binary**

Create a **daemon** for the binary

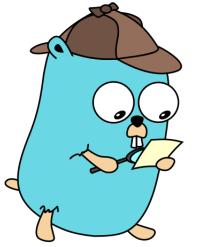
Proxy it using i. e. **Nginx**

WHICH ORM WE USE?



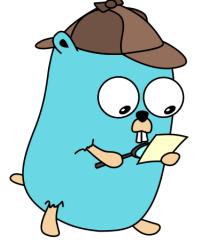
We Don't

WHAT FRAMEWORKS WE USE?



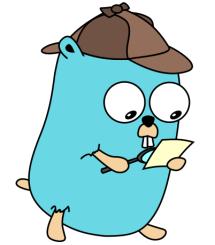
We Don't

LIBRARIES?



Yes, PLEASE!

HOW WE HANDLE ERRORS?



```
if err != nil {  
    // do something  
}
```

APPS WRITTEN IN GO



Core OS

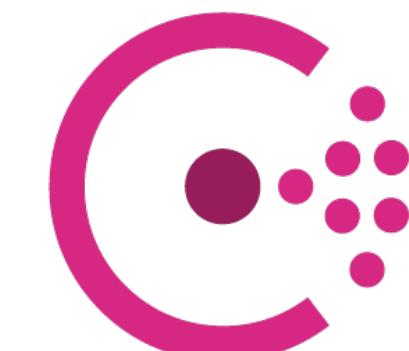


Cockroach LABS

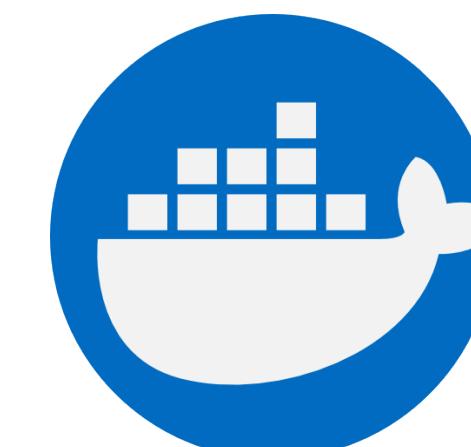
etcd



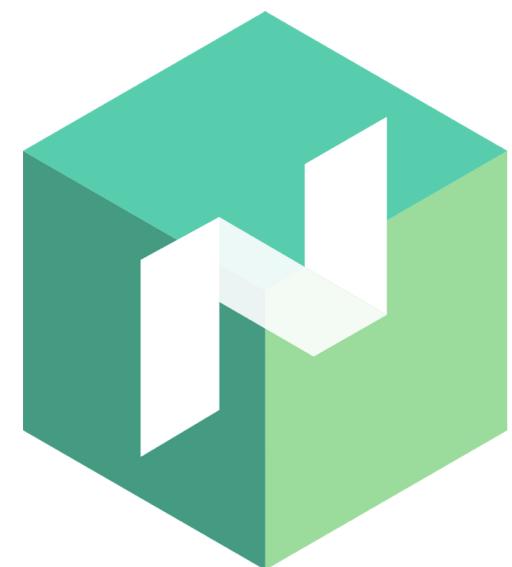
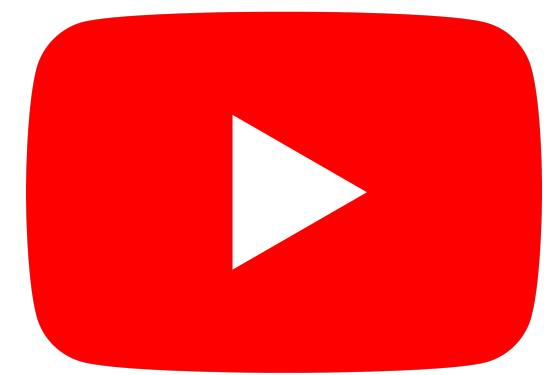
DEIS



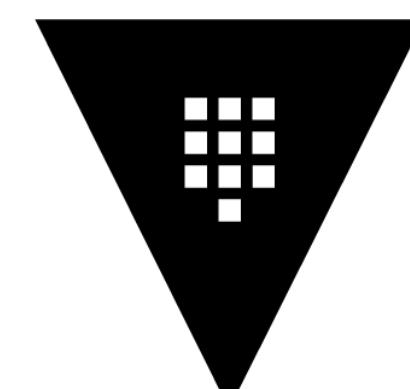
HashiCorp
Consul



HashiCorp
Packer



circleci



HashiCorp
Vault



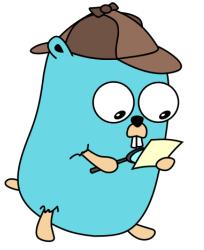
Terraform



influxdata



THINGS I WISH GO HAD



Generics

Better error handling

Ternary operator

Filter,Map,Reduce

Struct spread

Go in gaming

Go in UI

Go in AI

RESOURCES



A tour of Go



GopherTuts



steevehook
/mdc19

“We don’t stop playing because we get old, We get old because we stop playing”

George Bernard Shaw

DEMO

