

# The "Rabbit Hole" of Dependencies

in Golang



Andrii, Principal Software Engineer at ***Delivery Hero***



<https://de.linkedin.com/in/andrii-raikov>

<https://x.com/andriiraikov>

<https://andrii-raikov.medium.com>

<https://sessionize.com/andrii-raikov>



Can you?

**Can you clearly name couple of dependencies in your Golang project and why do you need them?**



Can you?

**Can you clearly name couple of indirect dependencies in your Golang project and why do you need them?**



# Simplest Go program

```
package main
```

```
func main() {  
    println("Hello world")  
}
```



# Simplest Go program

Kind: Unix Executable File

Size: 1.553.168 bytes (1,6 MB on disk)



# One small common dependency

```
package main
```

```
import "fmt"
```

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



# One small common dependency

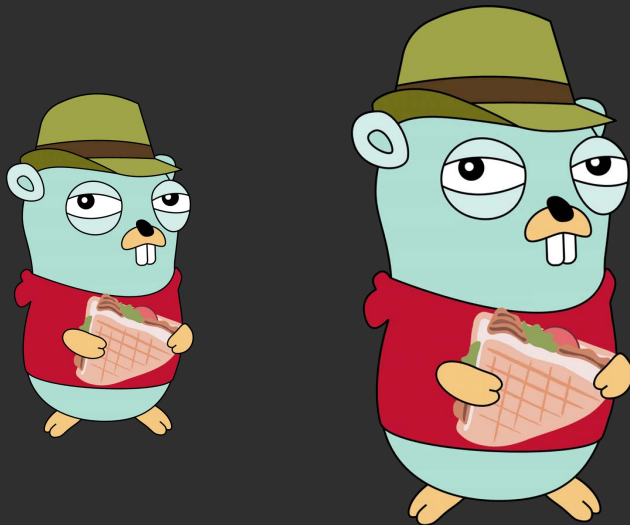
Kind: Unix Executable File

Size: 2.353.904 bytes (2,4 MB on disk)





+51% to the previous result



# Add more dependencies

```
package main
```

```
import (  
    "fmt"  
    "os"  
)
```

```
func main() {  
    fmt.Printf("Hello %s\n", os.Args[1])  
}
```



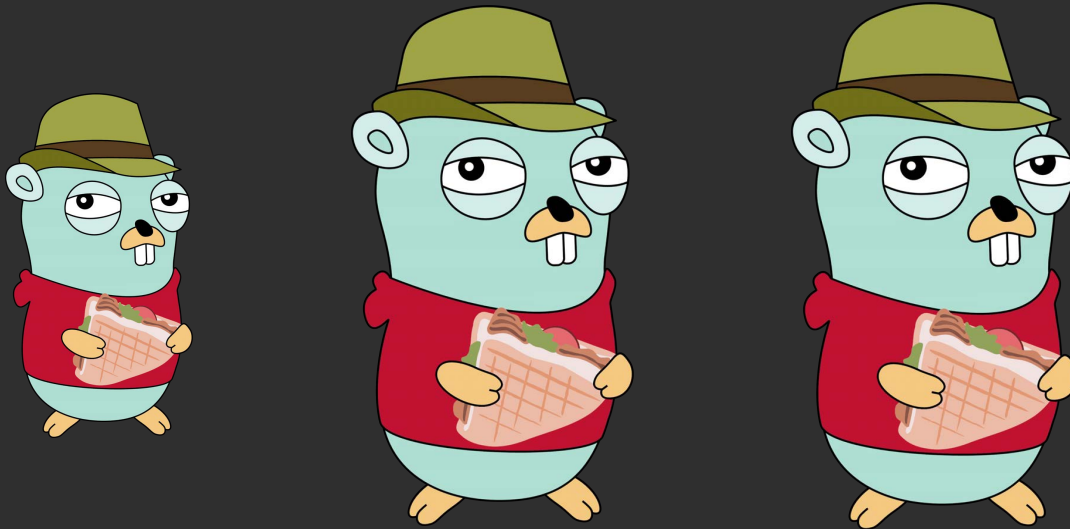
# Add more dependencies

Kind: Unix Executable File

Size: 2.370.464 bytes (2,4 MB on disk)




+53% to the original result




# fmt already includes os


> flag

✓ fmt

 doc.go


 errors.go


 errors\_test.go

 example\_test.go


 export\_test.go

 fmt\_test.go

 format.go


 gostringer\_example\_test.go

 print.go

 scan.go

 scan\_test.go

 state\_test.go

 stringer\_example\_test.go

 stringer\_test.go

> go

> hash

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

```
package fmt
```

```
import (
```

```
    "internal/fmtsort"
```

```
    "io"
```

```
    "os"
```

```
    "reflect"
```

```
    "strconv"
```

```
    "sync"
```

```
    "unicode/utf8"
```

```
)
```

```
// Strings for use with buffer.WriteString.
```

```
// This is less overhead than using buffer.Write with byte arrays.
```

```
const (
```

```
    commaSpaceString = ", "
```

```
    nilAngleString   = "<nil>"
```

```
    nilParenString   = "(nil)"
```

```
    nilString        = "nil"
```

# Hidden dependencies

By adding just “fmt” we already bring a lot of hidden dependencies and these bring even more and more...



# Let's make our dependency external

```
package message
```

```
import (  
    "fmt"  
    "os"  
)
```

```
func Print() {  
    fmt.Printf("Hello %s\n", os.Args[1])  
}
```



# go.mod

```
module github.com/raykov/dependencies_004
```

```
go 1.24.0
```





# program

```
package main
```

```
import (  
    message "github.com/raykov/dependencies_004"  
)
```

```
func main() {  
    message.Print()  
}
```



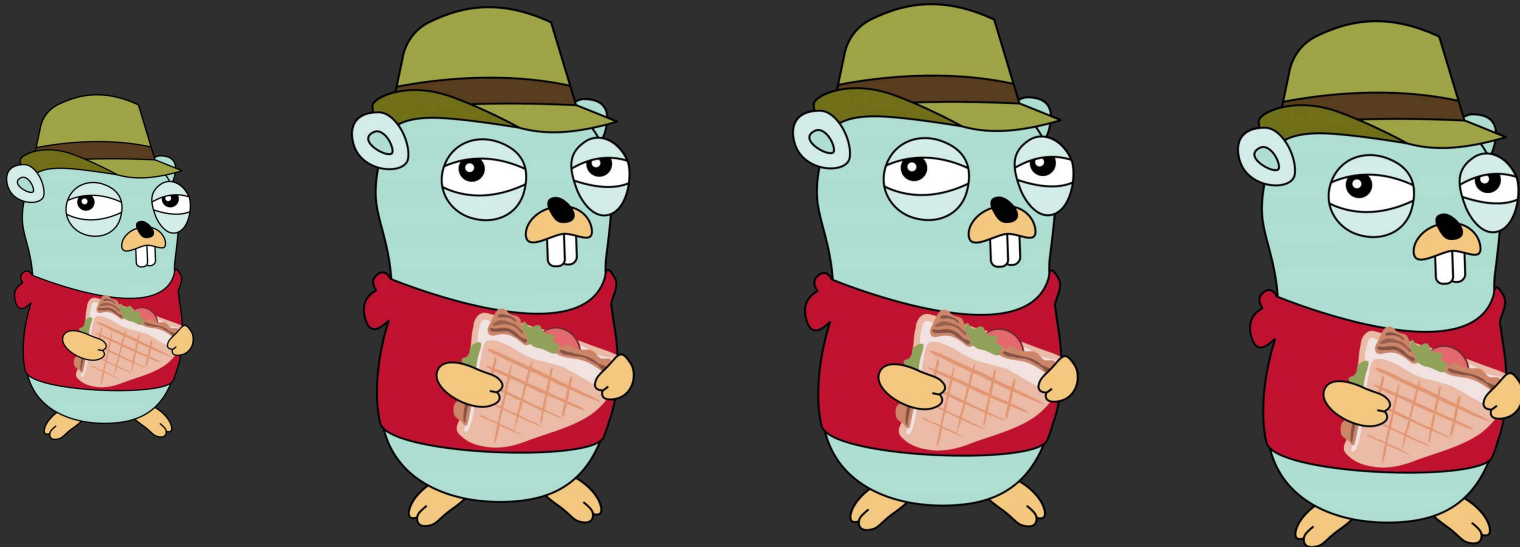
# result

Kind: Unix Executable File

Size: 2.370.512 bytes (2,4 MB on disk)



Didn't change much



# Let's try something more interesting

Let's list Buckets from AWS account using publicly available packages.



# List of buckets from S3

```
package main

import (
    "fmt"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
)

func main() {
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String("us-west-2")},
    )

    svc := s3.New(sess)

    result, _ := svc.ListBuckets(nil)

    for _, b := range result.Buckets {
        fmt.Printf("* %s\n",
            aws.StringValue(b.Name))
    }
}
```



# List of buckets from S3

Kind: Unix Executable File  
Size: 14.682.880 bytes (14,7 MB  
on disk)



+845% to the original size



???

Ok, these packages might be having other dependencies and leads to inflated build. AWS, at the end, stands for Web Services, so there is an API we can call. Let's try to implement ListObjects without using external dependencies...





# (c) Chat GPT

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# ChatGPT.py
# Author: ChatGPT
# Description: A simple chatbot using the ChatGPT API.

import sys
import os
import json
import time
import random
import string
import hashlib
import hmac
import io/ioutil
import net/http
import os
import time
```



```
import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/hex"
    "fmt"
    "io/ioutil"
    "net/http"
    "os"
    "time"
)
```



# Own ListBucket implementation



Kind: Unix Executable File  
Size: 8.275.904 bytes (8,3 MB on disk)



-413% to the package



# Well, we decreased size

But, we implemented **only one** small API call... So, most probably, to implement all of them will take take exactly this difference between 8 and 14 MB.

So, then, what takes other 6 MB of your program?



# What takes other 6 MB of your program?

```
import (  
    "crypto/hmac"  
    "crypto/sha256"  
    "encoding/hex"  
    "fmt"  
    "io/ioutil"  
    "net/http"  
    "os"  
    "time"  
)
```



# Commented crypto related code

```
func sha256Hex(data []byte) string {  
    //hash := sha256.Sum256(data)  
    //return hex.EncodeToString(hash[:])  
  
    return ""  
}
```

```
func hmacSHA256(key []byte, data string) []byte {  
    //h := hmac.New(sha256.New, key)  
    //h.Write([]byte(data))  
    //return h.Sum(nil)  
  
    return []byte{}  
}
```



# Commented crypto related code

```
import (  
    "encoding/hex"  
    "fmt"  
    "io/ioutil"  
    "net/http"  
    "os"  
    "time"  
)
```



# Only 4 KB were removed

Kind: Unix Executable File

Size: 8.275.904 bytes (8,3 MB on disk)

Kind: Unix Executable File

Size: 8.271.568 bytes (8,3 MB on disk)





# Comment encoding/hex

```
func hmacSHA256Hex(key []byte, data string) string {  
    //return hex.EncodeToString(hmacSHA256(key, data))  
  
    return ""  
}
```



# Comment encoding/hex

```
import (  
    "fmt"  
    "io/ioutil"  
    "net/http"  
    "os"  
    "time"  
)
```



# Nothing has changed!

Kind: Unix Executable File

Size: 8.275.904 bytes (8,3 MB on disk)

Kind: Unix Executable File

Size: 8.271.568 bytes (8,3 MB on disk)

Kind: Unix Executable File

Size: 8.271.568 bytes (8,3 MB on disk)



# Comment time

```
import (  
    "fmt"  
    "io/ioutil"  
    "net/http"  
    "os"  
)
```



# Comment io

```
import (  
    "fmt"  
    "net/http"  
    "os"  
)
```



# So, from where these 6 MB come?

```
package main
```

```
import "net/http"
```

```
func main() {  
    _, _ = http.NewRequest("GET", "/", nil)  
}
```



net/http

Kind: Unix Executable File

Size: 4.915.776 bytes (4,9 MB on disk)



# Why am I bringing this up?

I'm not encourage you remove all dependencies from your codebase

or remove / re-implement net/http, but I bet you were never bother yourself about are coincidences of adding one or another dependency to your code.

Before today most of you, didn't know how much net/http increases your application size. What about indirect dependencies it brings?





# Dependency management culture

Your team should have strong culture about dependency management.

Otherwise you risk to have very poor code quality and will stuck with technical debt forever!

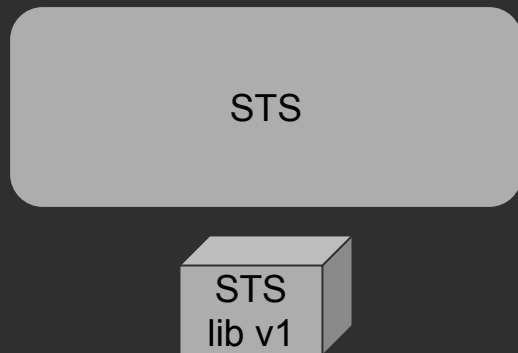


# Why?

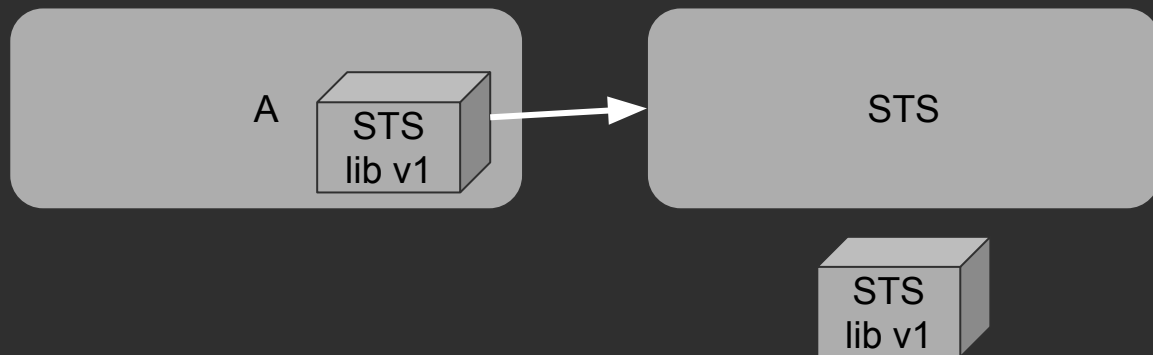
STS



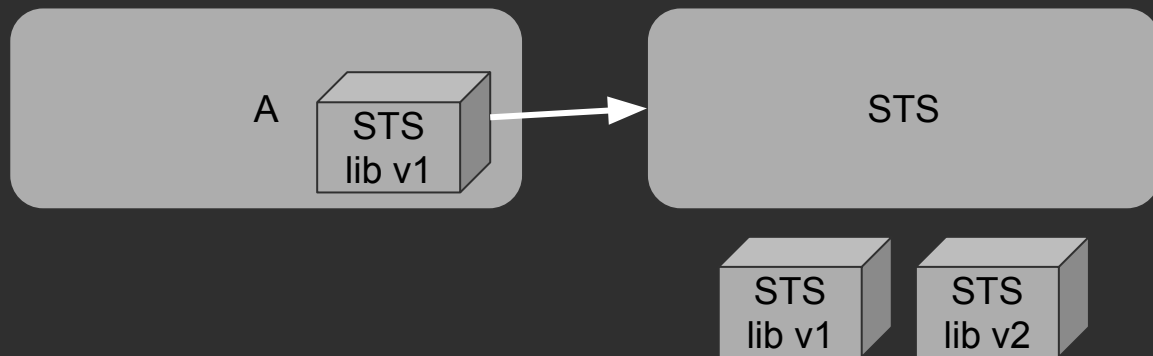
# Why?



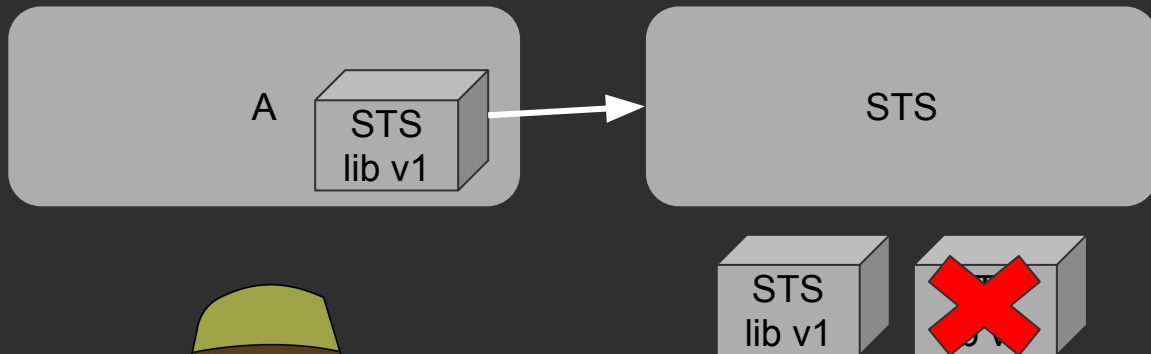
# Why?



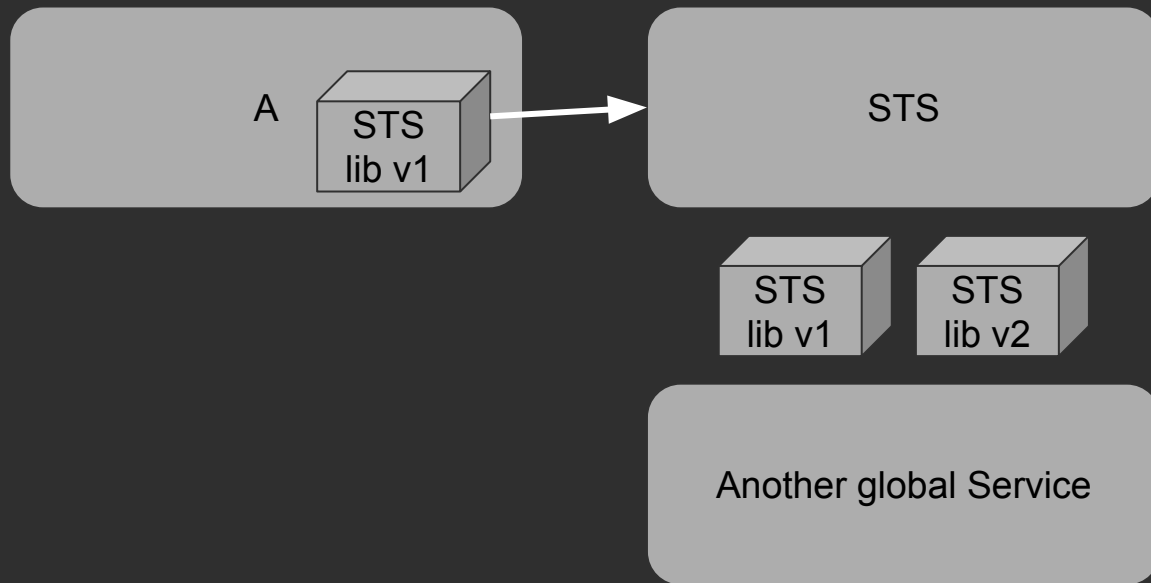
# Why?



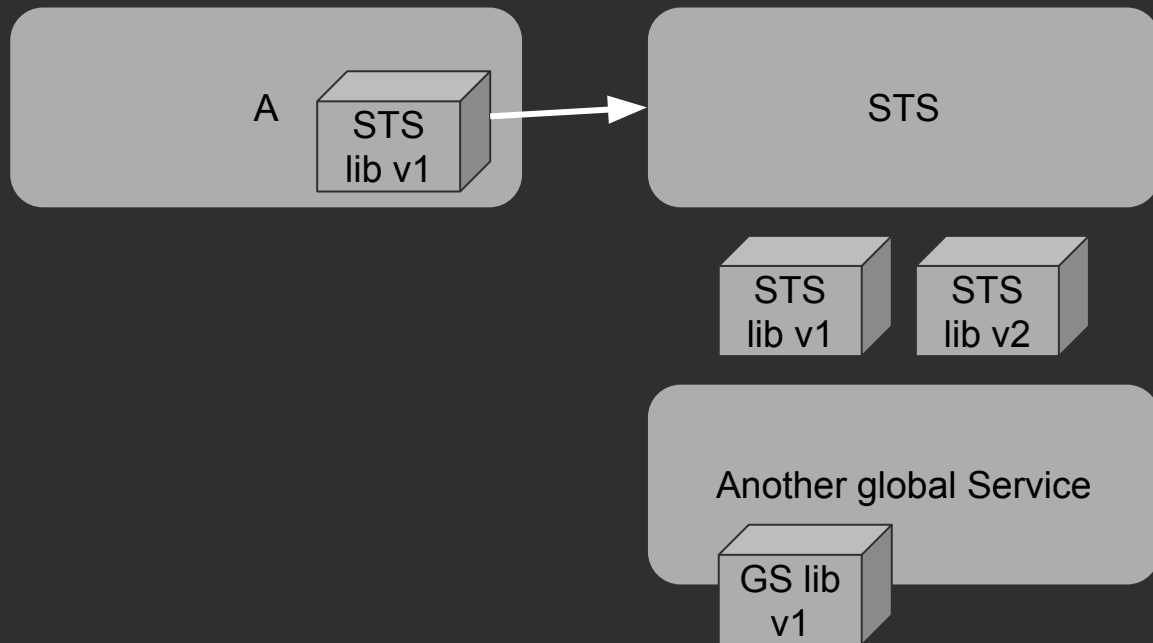
# Why?



# Why?

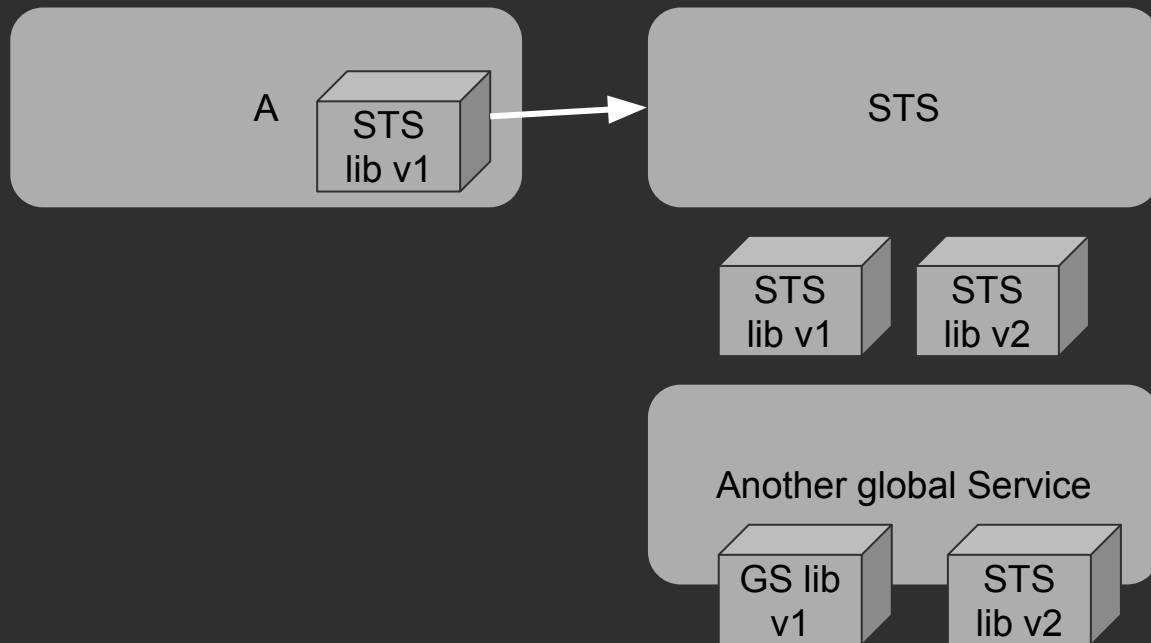


# Why?

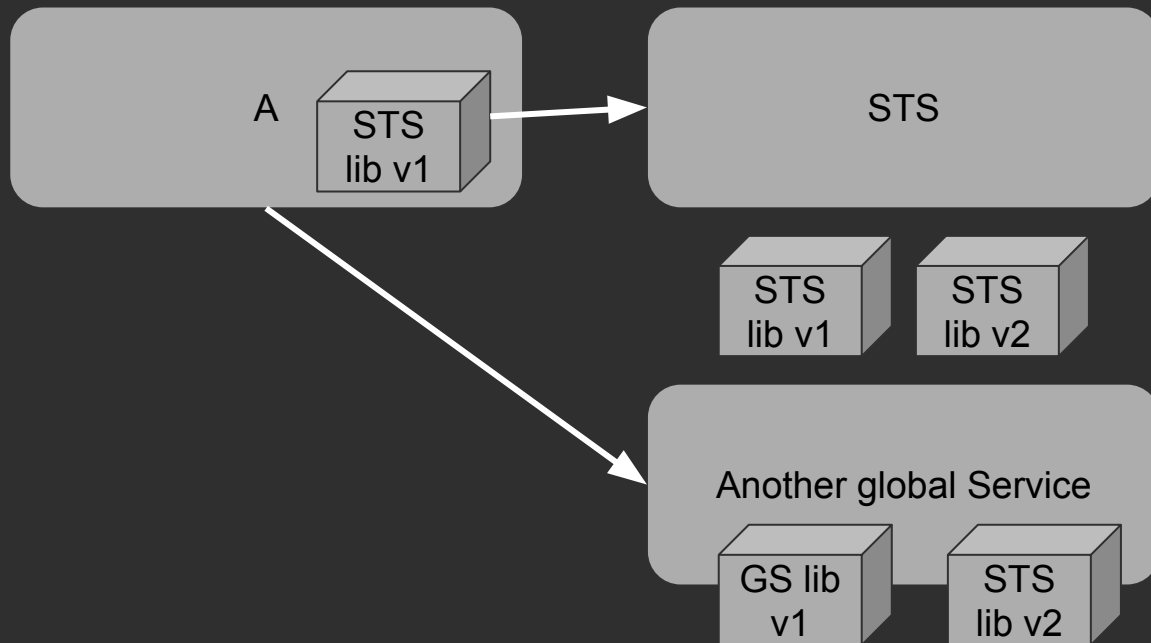




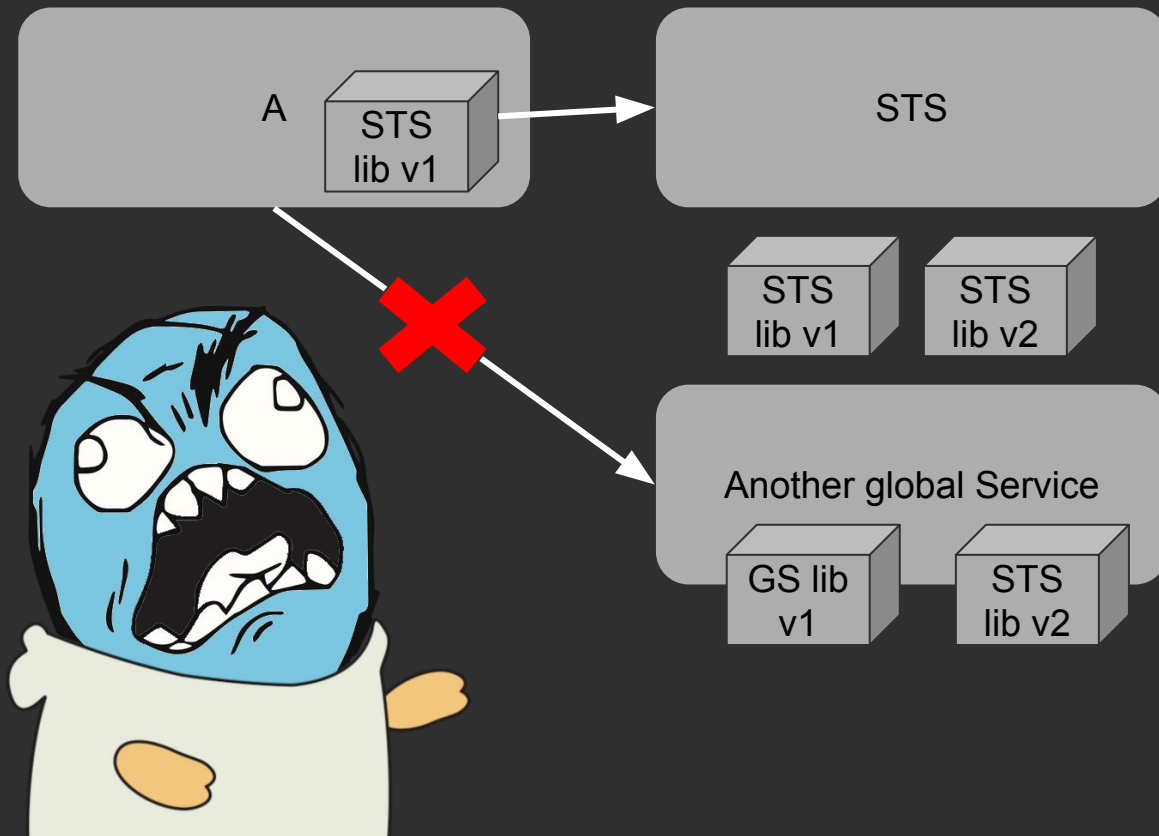
# Why?



# Why?



# Why?



# Problems

- Hard to maintain multiple version. Especially if they have breaking changes.
- Hard to persuade teams to migrate to the new version.
- At some point the loop will be closed and you endup with some services are able to use you libs.
- It actually takes a lot of time to endup in such situation and amount of development around this is so huge, so it is very hard to escape this tech debt.
- Huge investment required to fix the problem.



# What is the biggest problem?

- They didn't release a new version of the package, they just increased version
- 1.1.1 -> 1.2.1
- But didn't change "my-project/lib" -> "my-project/lib/v2" with separate go.mod



# go.sum of the real project

```
cloud.google.com/go v0.26.0/go.mod h1:aQUYkXzVsufM+DwF1aE+0xfcU+56JwCaLick0C1mMTw=
cloud.google.com/go v0.34.0/go.mod h1:aQUYkXzVsufM+DwF1aE+0xfcU+56JwCaLick0C1mMTw=
cloud.google.com/go v0.38.0/go.mod h1:990N+gfupTy94rShfmMCWGDn0LpTmnzTp2qbd1dvSRU=
cloud.google.com/go v0.44.1/go.mod h1:iSa0KzasP4Uvy3f1mN/7PiObzGgflwredwwASm/v6AU=
cloud.google.com/go v0.44.2/go.mod h1:60680Gw3Yr4ikxnPRS/oxxkBccT6SAlyMk63TGekxKY=
cloud.google.com/go v0.45.1/go.mod h1:RpBamKRgapWJb87xiFSdk4g1CME7QZg3uwTez+TSTjc=
cloud.google.com/go v0.46.3/go.mod h1:a6bKKbmY7er1mI7TEI41sAks/mkhTSZK8w33B4RAg0=
cloud.google.com/go v0.50.0/go.mod h1:r9sluTvynVuxRIOHXQEhMFFfphuXHOMZMycpNR5e6To=
cloud.google.com/go v0.52.0/go.mod h1:pXajvrRH/6o3+F9jDHZWQ5PbGhn+o8w9qiu/CffaVdO4=
cloud.google.com/go v0.53.0/go.mod h1:fp/UouUEsRkN6ryDKNW/Upv/JBKnv6WDthjR6+vze6M=
cloud.google.com/go v0.54.0/go.mod h1:1rq2OEKv3YMf6n/9ZvGWI3GWw0VoqH/1x2nd8Is/bPc=
cloud.google.com/go v0.56.0/go.mod h1:jr7tqZxxKOVYizybht9+26Z/gUq7tiRzu+ACVAMbKVk=
cloud.google.com/go v0.57.0/go.mod h1:oXiQ6Rzq3RAkkY7N6t3TcE6jE+CIBBbA36lwQ1JyzZs=
cloud.google.com/go v0.62.0/go.mod h1:jmCYTdRCQuc1PHI1J/maLIInMho30T/Y0M4hTdTShOYc=
cloud.google.com/go v0.65.0/go.mod h1:O5N8zS7uWY9vkA9vayVHs65eM1ubvY4h553ofrNHObY=
cloud.google.com/go v0.110.0 h1:Zc8gqp3+a9/Eyph2KDmcGaPtBKRioqq4YT1L4NMD0Ys=
cloud.google.com/go/bigquery v1.0.1/go.mod h1:i/xBL2U1R5RvWAURpBYZTtm/cXjCha91lfbpx4poX+o=
cloud.google.com/go/bigquery v1.3.0/go.mod h1:PjpwJns1EMmckchkHffq+HTD2DmtT67aNFKH1/VBDHE=
cloud.google.com/go/bigquery v1.4.0/go.mod h1:S8dzgnTigyfTmLBfrtrhyYhwRxG72rYxvftPBK2Dvzc=
cloud.google.com/go/bigquery v1.5.0/go.mod h1:snEHRnqQbz117VIFhE8bmtwIDY80NLUZUMb4Nv6dBig=
cloud.google.com/go/bigquery v1.7.0/go.mod h1://okPTzCYNXSlb24MZs83e2Do+h+VXtc4gLoIoXIAPc=
cloud.google.com/go/bigquery v1.8.0/go.mod h1:J5hqkt3O0uAFnINi6JXValWIb1v0goeZM77hZzJN/fQ=
```



# This is very common example

I have seen similar situation 4 times in DH with different services but result was always similar - unmaintainable dependencies and complete team ignorance of the problem due to other priorities.



# Another example: “helper” packages

```
package gopher
```

```
type Gopher struct {  
    Name      string  
    Age       int  
    Position  string  
}
```





# helpers

```
package helpers
```

```
import (  
    "time"  
)
```

```
func FormatTime(t time.Time) string {  
    return t.Format(time.RFC3339)  
}
```



# Extend some more helpers

```
import (  
    "deps/gopher"  
    "fmt"  
    "time"  
)  
  
func Info(g gopher.Gopher) string {  
    return fmt.Sprintf("Gopher: %s %d %s\n", g.Name, g.Age,  
g.Position)  
}  
  
func FormatTime(t time.Time) string {  
    return t.Format(time.RFC3339)  
}
```

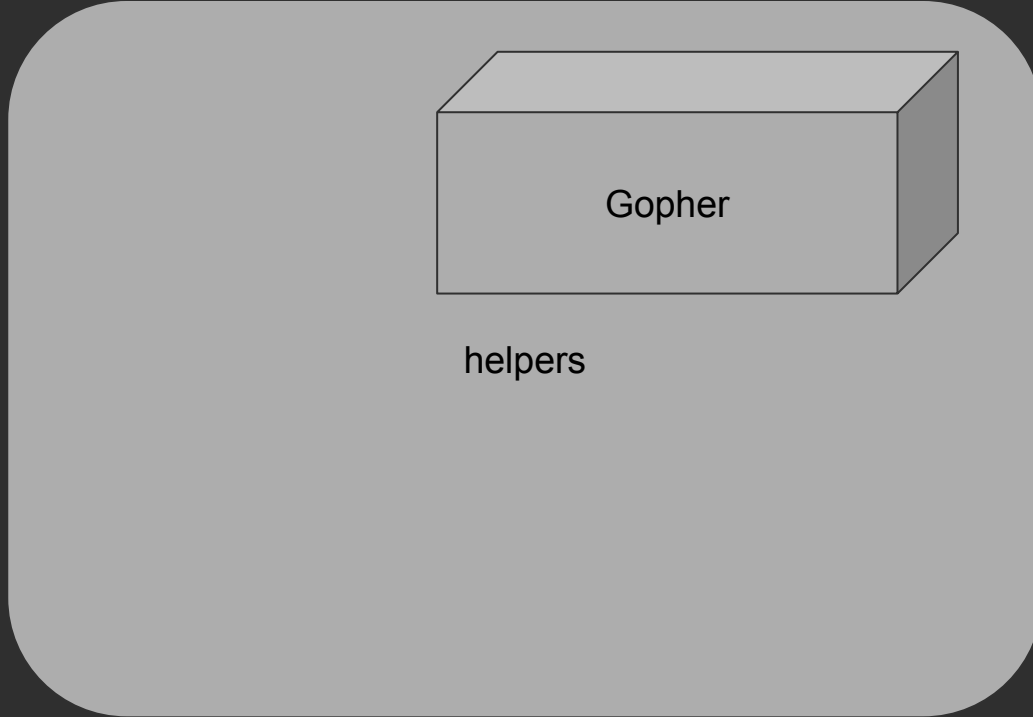


# The usage

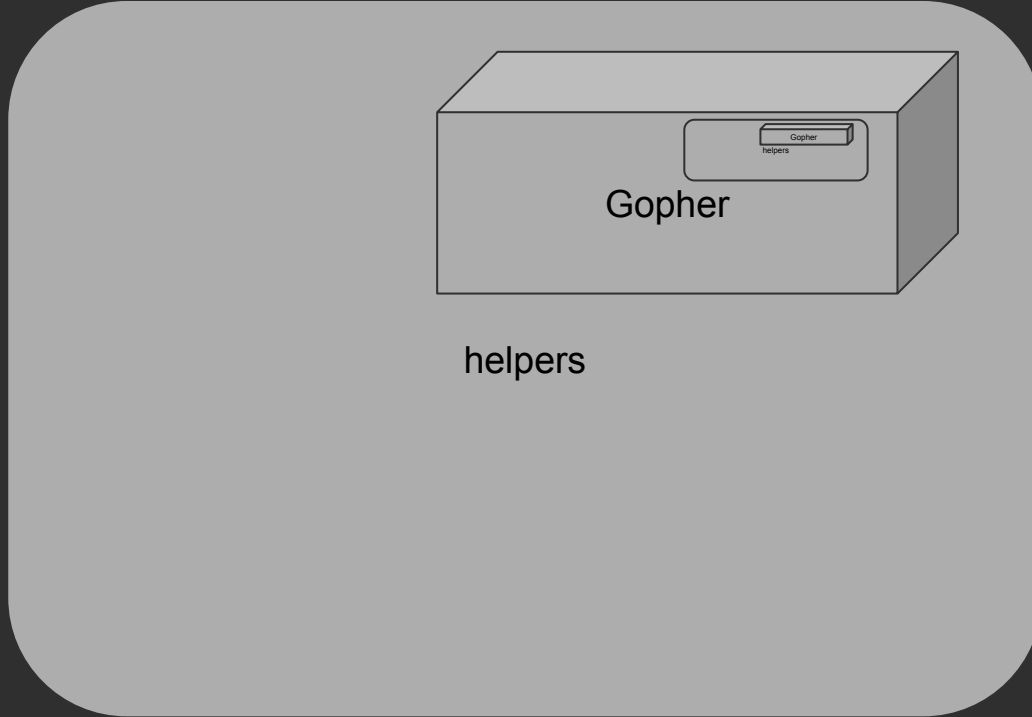
```
import (  
    "deps/gopher"  
    "deps/helpers"  
    "fmt"  
)  
  
func main() {  
    g := gopher.Gopher{  
        Name:      "Gopher",  
        Age:        42,  
        Position:   "Engineer",  
    }  
  
    fmt.Println(helpers.Info(g))  
}
```



# What is wrong in this example?



# What is wrong in this example?



# Our helper has dependency on the gopher package

And if we use it somewhere, it automatically adds dependency to “gopher” pkg to any other package using helpers.

At some point (soon or later) you will face circular dependency error, because of this uncontrolled usage.



# How to solve this particular example?

```
package gopher

import (
    "fmt"
)

type Gopher struct {
    Name      string
    Age       int
    Position  string
}

func (g Gopher) Info() string {
    return fmt.Sprintf("Gopher: %s %d %s\n", g.Name, g.Age, g.Position)
}
```



# Solution

```
func main() {  
    g := gopher.Gopher{  
        Name:      "Gopher",  
        Age:       42,  
        Position:  "Engineer",  
    }  
  
    fmt.Println(g.Info())  
}
```





# This one I also see a lot in our codebase

This one is supper common problem I see in the codebase



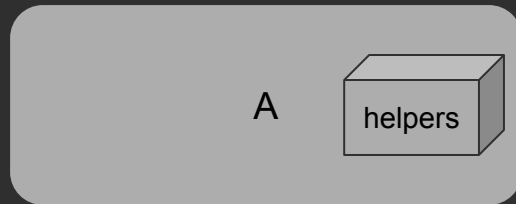
# Another common example related to the helpers



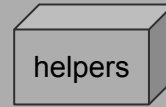
A



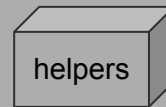
# Creating a helper package



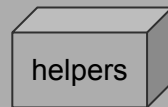
A



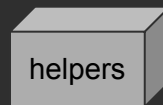
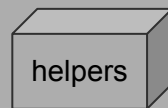
B

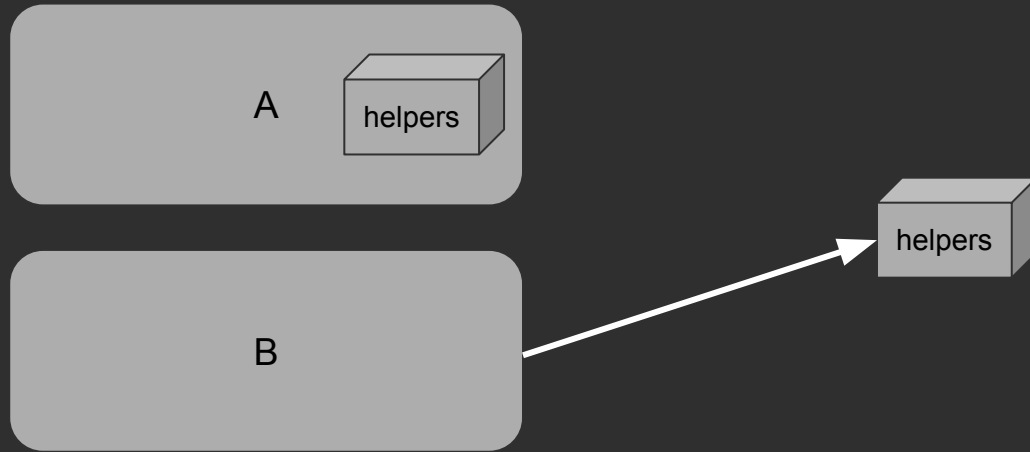


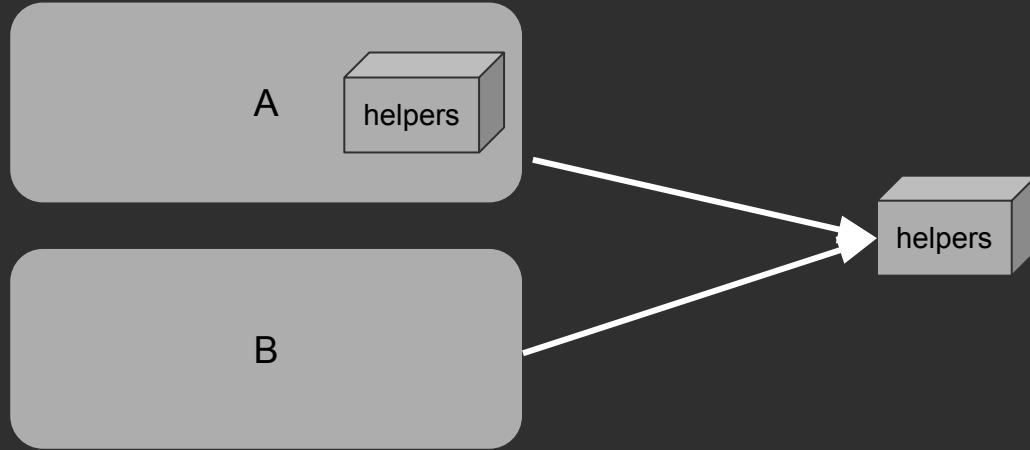
A



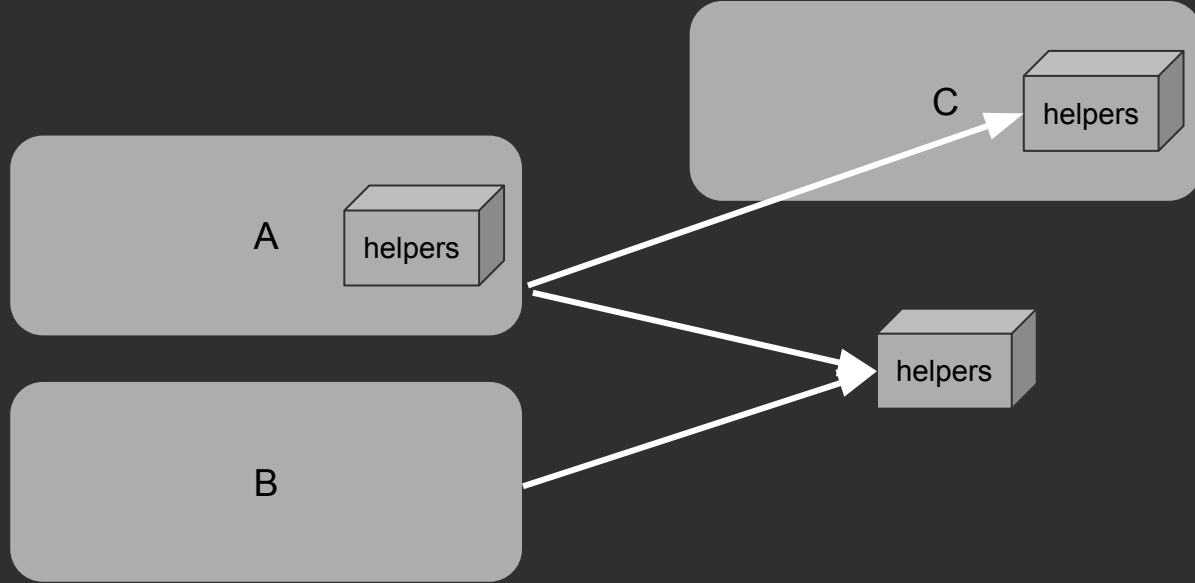
B







# Real case











# Conclusion

Your team requires strong dependency management culture!

- Document Dependency Policies
- Using guidance minimize and defined Dependencies
- Do regular Dependency Audits:
  - Update versions
  - Detect and fix Vulnerabilities
- Follow best practices
- Don't skip "import" section during PR review



We are hiring: <https://careers.deliveryhero.com>



***Delivery Hero***



# Questions?

Ask me

<https://de.linkedin.com/in/andrii-raikov>

<https://x.com/andriiraikov>

<https://andrii-raikov.medium.com>

<https://sessionize.com/andrii-raikov>

