
Go in der Praxis

Take GitHub to the command line

GitHub CLI brings GitHub to your terminal. Free and open source.

```
brew install gh
```

or **Download for Mac**

View installation instructions →

$ gh pr checkout

Check out pull requests locally.

```
$ gh pr checkout 12
remote: Enumerating objects: 66, done.
remote: Counting objects: 100% (66/66), done.
```

gh Github CLI

149.298 Zeilen Code

490 Contributors

1.  CLI Tools
2.  Web App
3. λ Lambda Function

5 Fakten zu Go

1. statisches Typsystem
2. Garbage Collection
3. keine Vererbung
4. Concurrency eingebaut
5. native Ausführung
   Linux, Win, z/OS, 386, amd64, ARM, wasm, ...

CLI Tool
Go Proverbs

Go Proverbs

18 Prinzipien zur Go Entwicklung
von Rob Pike 2015 vorgestellt
"Clear is better than clever."

CLI Proverbs ohne Argumente

```
> pro
```

```
Clear is better than clever.
```

CLI Projekt aufsetzen

```
// 1. Go Modul erstellen (erzeugt go.mod)
> go mod init crossnative.com/pro
```

```
// 2. Datei main.go erstellen
package main

import "fmt"

func main() {
    fmt.Println("Hello Go Pro!")
}
```

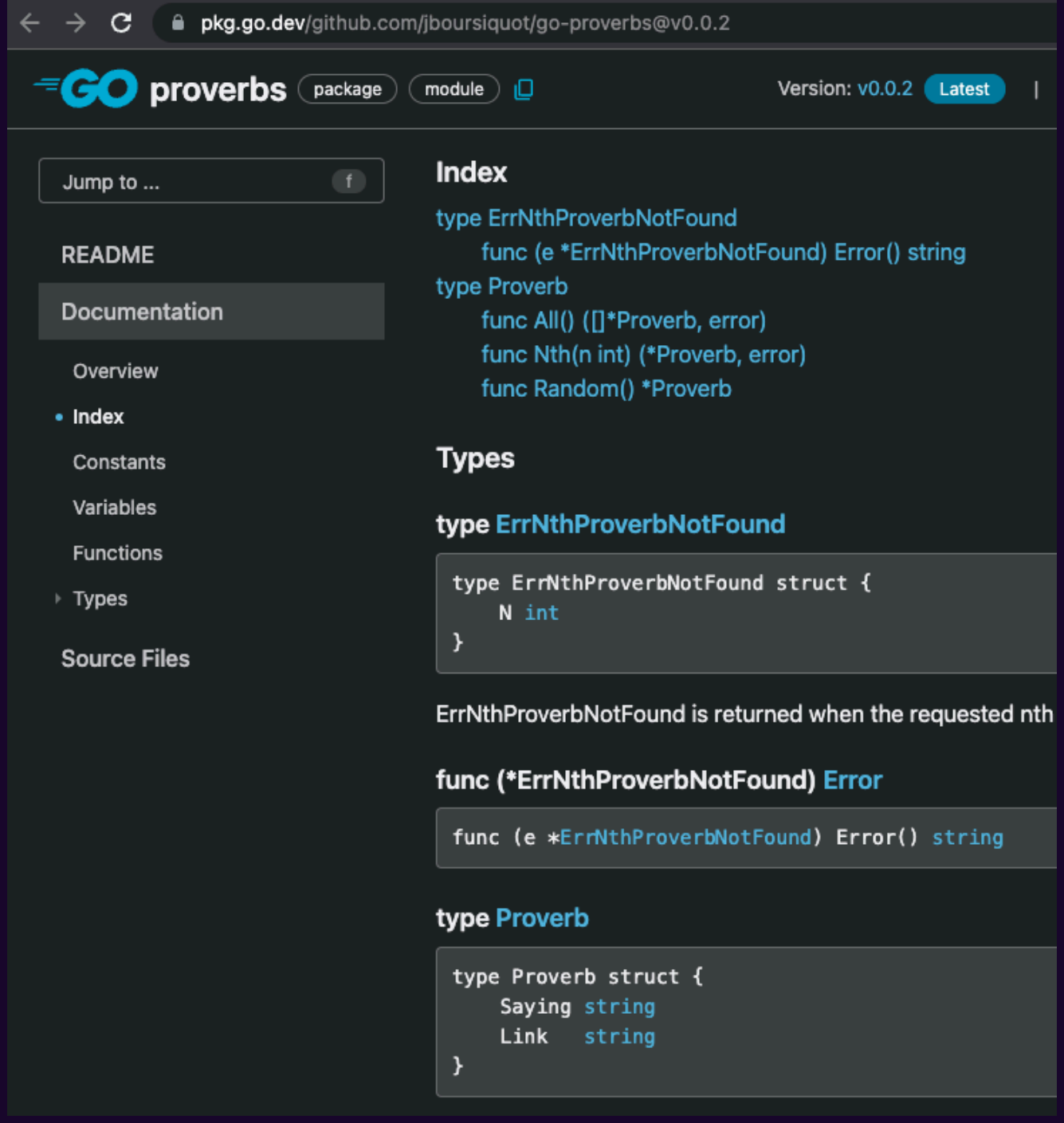
Ausführen

```
go build . // 1. Code kompilieren
./pro      // 2. Binary ausführen
```

```
go run .   // Code kompilieren und ausführen
```


Entwicklung
GO
JetBrains
GoLand
Visual
Studio Code
Vim
Vim Go

Proverbs als Go Modul


pkg.go.dev/github.com/jboursiquot/go-proverbs@v0.0.2
proverbs
package
module
Version: v0.0.2
Latest
Jump to ...
f
README
Documentation
Overview
Index
Constants
Variables
Functions
Types
Source Files
Index
type ErrNthProverbNotFound
func (e *ErrNthProverbNotFound) Error() string
type Proverb
func All() ([]*Proverb, error)
func Nth(n int) (*Proverb, error)
func Random() *Proverb
Types
type ErrNthProverbNotFound
type ErrNthProverbNotFound struct {
N int
}
ErrNthProverbNotFound is returned when the requested nth
func (*ErrNthProverbNotFound) Error
func (e *ErrNthProverbNotFound) Error() string
type Proverb
type Proverb struct {
Saying string
Link string
}

Proverbs Modul nutzen

```
// 1. Proverbs Modul Dependency
> go get github.com/jboursiquot/go-proverbs
```

```
// 2. Go Modul Descriptor go.mod
module crossnative.com/pro

go 1.22

require github.com/jboursiquot/go-proverbs v0.0.2
```

Zufälliges Proverb ausgeben

```go
package main

import (
    "fmt"
    "github.com/jboursiquot/go-proverbs"
)

func main() {
    fmt.Println(proverbs.Random())
}
```

```
// Ausgabe `go run .`
&{Make the zero value useful. http://youtube.com/322}
```

💡 Structs und Pointer

```go
package main

import (
    "fmt"
    "github.com/jboursiquot/go-proverbs"
)

func main() {
    // Pointer Variable auf Proverb Struct
    var p *proverbs.Proverb = proverbs.Random()

    fmt.Println(p)
}
```

```go
// Struct statt Klasse
type Proverb struct {
    Saying string
    Link   string
}
```

Zufälliges Proverb ausgeben

```go
package main

import (
    "fmt"
    "github.com/jboursiquot/go-proverbs"
)

func main() {
    var p *proverbs.Proverb = proverbs.Random()

    // Zugriff auf Property des Structs
    fmt.Println(p.Saying)
}
```

```
// Ausgabe `go run .`
Make the zero value useful.
```

CLI Proverbs mit Count

```
> pro -count=3
```

```
Clear is better than clever.
Documentation is for users.
Don't panic.
```

Flag für Count

```go
import (
    "flag"
    "fmt"
    "github.com/jboursiquot/go-proverbs"
)

func main() {
    // 1. Flags definieren
    var count *int = flag.Int("count", 1, "proverb count")

    // 2. Flags parsen
    flag.Parse()

    // 3. Ausgabe in Schleife
    for i := 0; i < *count; i++ {
        var p *proverbs.Proverb = proverbs.Random()
        fmt.Println(p.Saying)
    }
}
```

```
// Compile mit `go build .`, Ausgabe mit `pro -count=2`
Make the zero value useful.
Reflection is never clear.
```

CLI Proverbs mit ungültigem Count

```
> pro -count=
```

```
flag needs an argument: -count
Usage of ./pro:
  -count int
        proverb count (default 1)
```

Flag für Count Refactored

```go
import (
    "flag"
    "fmt"
    "github.com/jboursiquot/go-proverbs"
)

func main() {
    // 1. Flags definieren
    count := flag.Int("count", 1, "proverb count")

    // 2. Flags parsen
    flag.Parse()

    // 3. Ausgabe in Schleife
    for range *count {
        p := proverbs.Random()
        fmt.Println(p.Saying)
    }
}
```

```
// Compile mit `go build .`, Ausgabe mit `pro -count=2`
Make the zero value useful.
Reflection is never clear.
```

CLI Tool
File Downloader

File Downloader

Paralleler File Download
Nutzt Goroutinen und Channels
Context is King

File Downloader 1/3

```go
func main() {
  urls := []string{
    "https://go.dev/dl/go1.22.2.src.tar.gz",
    "https://go.dev/dl/go1.22.2.darwin-amd64.tar.gz",
  }

  // URL Download
  ctx := context.Background()
  downloadURLs(ctx, urls)
}

func downloadURLs(ctx context.Context, urls []string) {
  // ...
}
```

File Downloader 2/3

```go
func downloadURLs(ctx context.Context, urls []string) {
  // 1. Channel für Download Jobs
  jobs := make(chan int, len(urls))

  // 2. Go Routinen für Downloads starten
  for _, url := range urls {
    go func() {
      downloadURL(ctx, url)
      jobs <- 1 // Send an Channel (blockiert)
    }()
  }

  // 3. Warten auf Ende aller Go Routinen
  for range len(urls) {
    <-jobs // Receive von Channel (blockiert)
  }
}
```

File Downloader 3/3

```go
func downloadURL(ctx context.Context, url string) {
  // 1. HTTP Request mit Context erstellen
  req, _ := http.NewRequestWithContext(ctx, http.MethodGet, url, nil)

  // 2. HTTP Aufruf durchführen
  httpClient := http.Client{}
  resp, _ := httpClient.Do(req)
  defer resp.Body.Close()

  // 3. Ausgabe in Datei
  file, _ := os.Create(filepath.Base(url))
  defer file.Close()

  // 4. HTTP Body in Datei streamen
  io.Copy(file, resp.Body)
}
```

File Downloader 3/3 mit Fehlern

```go
func downloadURL(ctx context.Context, url string) error {
  // 1. HTTP Request mit Context erstellen
  req, err := http.NewRequestWithContext(ctx, http.MethodGet, url, nil)
  if err != nil {
    return err
  }

  // 2. HTTP Aufruf durchführen
  // ...
}
```

File Downloader mit Timeout

```go
func main() {
  urls := []string{
    "https://go.dev/dl/go1.22.2.src.tar.gz",
    "https://go.dev/dl/go1.22.2.darwin-amd64.tar.gz",
  }

  // 1. Context mit Cancel Func
  ctx, cancel := context.WithCancel(context.Background())

  // 2. Timer Channel nutzen
  timer := time.After(3 * time.Second)
  go func() {
    <-timer // Receive timer Channel
    log.Println("Canceled due to timeout")
    cancel()
  }()

  // 3. URL Download
  downloadURLs(ctx, urls)
}
```

File Downloader mit Interrupt

```go
func main() {
  urls := []string{
    "https://go.dev/dl/go1.22.2.src.tar.gz",
    "https://go.dev/dl/go1.22.2.darwin-amd64.tar.gz",
  }

  // 1. Context mit Cancel Func
  ctx, cancel := context.WithCancel(context.Background())

  // 2. Interrupt Channel nutzen
  interrupt := make(chan os.Signal, 1)
  signal.Notify(interrupt, os.Interrupt, syscall.SIGTERM)
  go func() {
    <-interrupt // Receive interrupt Channel
    log.Println("Canceled due to interrupt")
    cancel()
  }()

  // 3. URL Download
  downloadURLs(ctx, urls)
}
```

File Downloader mit Select

```go
func main() {
  // 1. Context mit Cancel Func
  //    + Timeout Channel
  //    + Interrupt Channel

  go func() {
    select {
      // 2a. Receive timer Channel
      case <-timer:
        log.Println("Canceled due to timeout")
        cancel()
      // 2b. Receive interrupt Channel
      case <-interrupt:
        log.Println("Canceled due to interrupt")
        cancel()
      }
  }()

  // 3. URL Download
  downloadURLs(ctx, urls)
}
```

CLI Tools

Standardlib Flags
einfache Releases mit Goreleaser
Cobra CLI Lib wenn's komplexer wird
Github, Kubernetes sind Go CLIs

Web App Cats

Set up Cats App

```
# 1. Verzeichnis erstellen
mkdir cats
cd cats
```

```
# 2. Go Modul aufsetzen
go mod init crossnative.com/cats
```

```
# 3. Go Datei erstellen
touch main.go
```

Cat API simple

```go
func catAPIHandler(w http.ResponseWriter, r *http.Request) {
  fmt.Fprintf(w, "Meow!")
  w.WriteHeader(http.StatusOK)
}

func main() {
  http.HandleFunc("GET /api/cats", catAPIHandler)
  http.ListenAndServe(":8080", nil)
}
```

Cat API mit JSON

```go
// 1. Struct mit JSON Tag definieren
type Cat struct {
  Name string `json:"name"`
}

func catAPIHandler(w http.ResponseWriter, r *http.Request) {
  // 2. Slice erstellen
  cats := make([]Cat, 1)

  // 3. Struct erstellen und einfügen
  cats[0] = Cat{Name: "Ginger"}

  // 4. JSON rendern
  json.NewEncoder(w).Encode(cats)
}

func main() {
  http.HandleFunc("GET /api/cats", catAPIHandler)
  http.ListenAndServe(":8080", nil)
}
```

Cat API mit JSON

```
# Query cat API
curl -s http://localhost:8080/api/cats | jq
[
  {
    "name": "Ginger"
  }
]
```

Test Cat API `cat_api_test.go`

```go
func TestCatAPIHandler(t *testing.T) {
  // 1. Test Request erstellen
  req, _ := http.NewRequest("GET", "/api/cats", nil)

  // 2. HTTP Recorder erstellen (ist http.ResponseWriter)
  rec := httptest.NewRecorder()

  // 3. Handler aufrufen
  catAPIHandler(rec, req)

  // 4. Response prüfen
  if rec.Code != http.StatusOK {
    t.Errorf("got status %v expected %v", rec.Code, http.StatusOK)
  }
}
```

Test ausführen

```
1 go test -v ./...
2 === RUN   TestCatAPIHandler
3 --- PASS: TestCatAPIHandler (0.00s)
4 PASS
5 coverage: 50.0% of statements
6 ok      crossnative.com/cats      0.127s  coverage: 50.0% of statements
```

Cats App
Web

Cats Web App

```go
func indexHandler(w http.ResponseWriter, r *http.Request) {
  tpl := template.Must(template.ParseFiles("index.html"))
  tpl.Execute(w, nil)
}

func main() {
  // 1. Mulitplexer erzeugen
  mux := http.NewServeMux()

  // 2. Handler registrieren
  mux.HandleFunc("/", indexHandler)

  // 3. Server mit Mulitplexer starten
  http.ListenAndServe(":8080", mux)
}
```

File Server aus Binary

```go
//go:embed assets
var assets embed.FS

func main() {
  mux := http.NewServeMux()
  mux.HandleFunc("/", indexHandler)

  // Serve Files
  mux.Handle("/assets/", http.FileServer(http.FS(assets)))

  http.ListenAndServe(":8080", mux)
}
```

Index Handler Template mit Daten

```go
type Cat struct {
  Name string
}

func indexHandler(w ResponseWriter, r *Request) {
  // 1. Slice erstellen
  cat := make([]Cat, 1)

  // 2. Struct erstellen und einfügen
  cat[0] = Cat{Name: "Ginger"}

  // 3. Template rendern
  tpl := template.Must(template.ParseFiles("index.html"))
  tpl.Execute(w, cat)
}
```

```html
<body>
  <h1>Cats App</h1>
  {{ range . }}
    <h2>{{ .Name }}</h2>
  {{ end }}
</body>
```

The Cat API

Cats as a service.

cause everyday is a Caturday.

PI all about cat.

- Images. Breeds. Facts.

ET YOUR API KEY READ OUR GUIDES


```
const headers = new Headers({
  "Content-Type": "application/json",
```

Index Handler mit Cats API

Query Cats API

```
GET https://api.thecatapi.com/v1/breeds?limit=5
```

```
[
   {
      "id": "abys",
      "name": "Abyssinian",
      "image": {
        "url": "https://cdn2.thecatapi.com/0XYvRd7oD.jpg"
      }
   },
   {
      "id": "aege",
      "name": "Aegean",
      "image": {
        "url": "https://cdn2.thecatapi.com/ozEvzdVM-.jpg"
      }
   },
   ...
]
```

Map JSON auf Struct

```
type Cat struct {
  ID    string `json:"id"`
  Name  string `json:"name"`
  Image struct {
    URL string `json:"url"`
  } `json:"image"`
}
```

Index Handler mit Cats API

```go
func indexHandler(w http.ResponseWriter, r *http.Request) {
  // 1. Cat API Aufruf (Fehler ignorieren)
  resp, _ := http.Get("https://api.thecatapi.com/v1/breeds?limit=5")

  // 2. Slice für Cat Structs
  cat := make([]Cat, 5)

  // 3. Response Body parsen
  defer resp.Body.Close()
  body, _ := ioutil.ReadAll(resp.Body)
  json.Unmarshal(body, &cat)

  // 4. Template rendern
  tpl.Execute(w, cat)
}
```

Index Handler mit Fehlerhandling 💣

```go
func indexHandler(w http.ResponseWriter, r *http.Request) {
  // 1. Cat API Aufruf
  resp, err := http.Get("https://api.thecatapi.com/v1/breeds?limit=5")
  //    Fehler behandeln
  if err != nil {
    http.Error(w, "Cats API error", http.StatusInternalServerError)
    return
  }

  // 2. Slice für Cat Structs
  cat := make([]Cat, 5)

  // 3. Response Body parsen
  defer resp.Body.Close()
  body, _ := ioutil.ReadAll(resp.Body)
  err := json.Unmarshal(body, &cat)
  // TODO: Fehler behandeln

  // 4. Template rendern
  err := tpl.Execute(w, cat)
  // TODO: Fehler behandeln
}
```

Web App

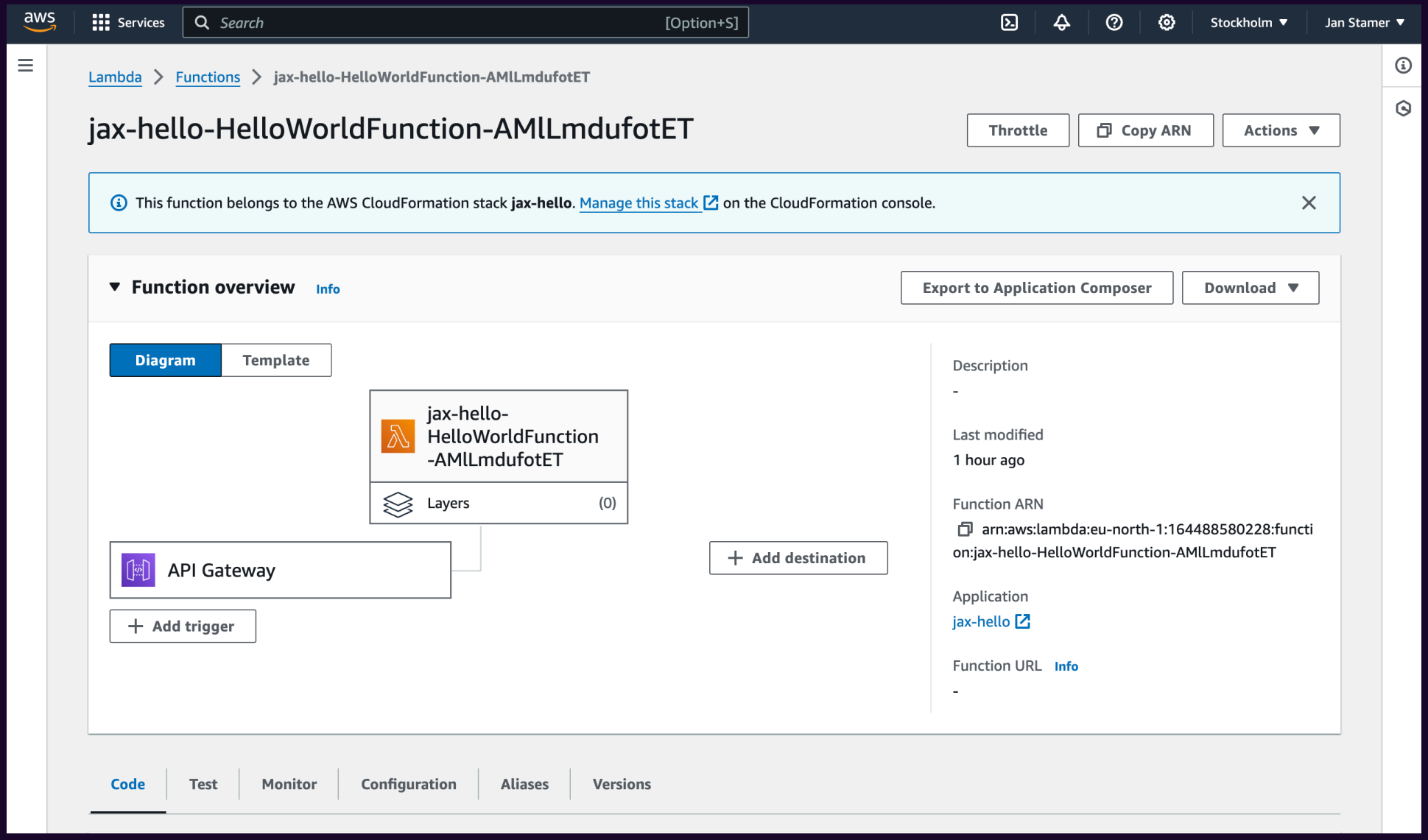
Standardlib JSON und Router
HTTP Tests
Querschnittsfeatures durch Middleware

λ Lambda Function

λ Lambda Function

Go ideal für Serverless Functions
schneller Start, geringer Speicherbedarf
unterstützt von AWS, Azure, Google

λ Lambda Function



λ Lambda Function

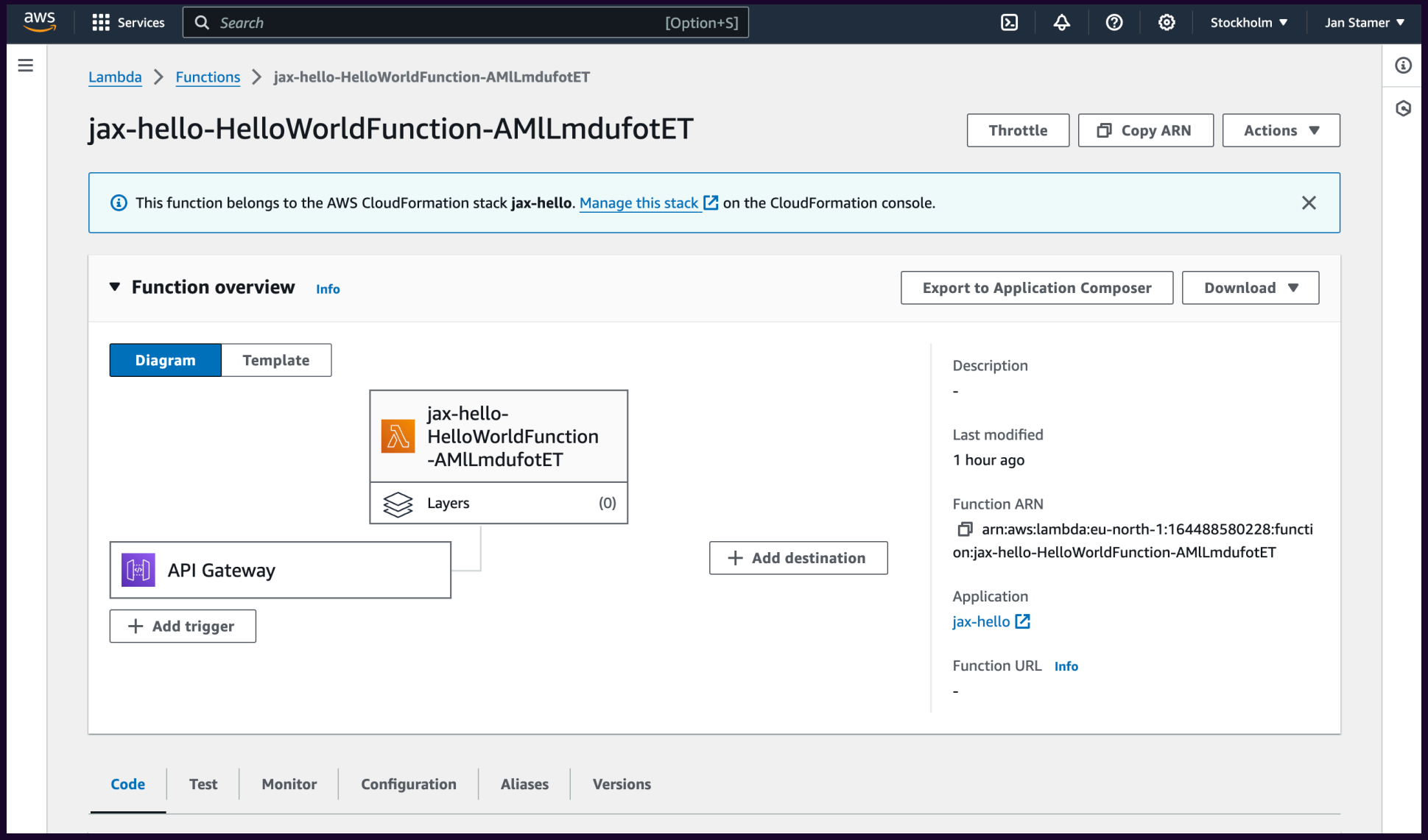
```
// 1. Lambda Go Projekt aufsetzen
> sam init

// 2. Lambda bauen
> sam build

// 3. Lambda ausführen
> sam local invoke -e events/event.json
Invoking bootstrap (provided.al2023)
Local image is up-to-date
Using local image: public.ecr.aws/lambda/provided:al2023-rapid-x86_64.

Mounting .aws-sam/build/HelloWorldFunction, inside runtime container
REPORT RequestId: 37876f36  Init Duration: 0.52 ms  Duration: 80.04 ms
Billed Duration: 81 ms  Memory Size: 128 MB   Max Memory Used: 128 MB
{"statusCode": 200, "body": "Hello JAX!"}
```

λ Lambda Function



λ Lambda Function

```
1 # Query AWS Lambda
2 curl -s https://0up9.eu-north-1.amazonaws.com/Prod/hello
3 Hello JAX!
```

Footprint

Package Size 5,3 MB
Billed Duration 70 ms
Max Memory Used 18 MB
Init Duration 63.66 ms

λ Lambda Function

```go
import (
  "fmt"
  "github.com/aws/aws-lambda-go/events"
  "github.com/aws/aws-lambda-go/lambda"
)

func handler(req events.APIGatewayProxyRequest) (events.APIGatewayProxyResponse, error)
  return events.APIGatewayProxyResponse{
    Body:       "Hello JAX!",
    StatusCode: 200,
  }, nil
}

func main() {
  lambda.Start(handler)
}
```

λ Lambda Function Projekt

```
|-events
   | event.json // Beispiel Request
|-hello-world   // Go App
   | go.mod
   | go.sum
   | main.go
   | main_test.go
| Makefile       // Build File
| samconfig.toml // SAM Config
| template.yaml  // SAM Config
| README.md
```

CLI Tools

Projekt Setup
Variablen, Schleifen
Goroutinen, Channels
Context
Structs

Web App

Web Server
Slice
Unit Test
JSON
Fehlerhandling

λ Lambda Func

AWS Setup
Serverless Experience

3 Gründe für Go

1. Einfach
2. Mächtig
3. Langweilig

Go liebt

Microservices

Serverless Functions

Kommandozeilen-Tools

DevOps und Cloud


GO!
GO!

Jan Stamer

jan.stamer@crossnative.com

Go in der Praxis