



# Go über den Wolken



CNCF Graduated Projects (25)

Landscape Guide

Reset Filters

Grouping  
CNCF Relation

Sort By  
Alphabetical (a to z)

Category  
Any

Project  
Any

License  
Any

Organization  
Any

Headquarters  
Any


























Company Type  
Any

Industry  
Any

Download as CSV

Example filters

Cards by age  
Open source landscape  
Member cards  
Cards by stars  
Cards from China  
Certified K8s/KCSP/KTP  
Cards by MCap/Funding  
Cards without  
bestpractices.dev

 <b>Argo</b> ★ 14,145 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Cilium</b> ★ 16,516 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>containerd</b> ★ 14,857 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>CoreDNS</b> ★ 11,071 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>cri-o</b> ★ 4,754 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Envoy</b> ★ 22,759 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>etcd</b> ★ 44,537 Cloud Native Computing Foundation (CNCF) Funding: \$3M
 <b>Fluentd</b> ★ 12,199 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Flux</b> ★ 5,300 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Harbor</b> ★ 20,890 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Helm</b> ★ 24,953 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Istio</b> ★ 33,694 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Jaeger</b> ★ 18,318 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>KEDA</b> ★ 6,831 Cloud Native Computing Foundation (CNCF) Funding: \$3M
 <b>KEDA (Serverless)</b> ★ 6,831 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Kubernetes</b> ★ 101,881 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Linkerd</b> ★ 9,881 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Open Policy Agent</b> ★ 8,496 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Prometheus</b> ★ 49,987 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Rook</b> ★ 11,830 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>SPIFFE</b> ★ 1,213 Cloud Native Computing Foundation (CNCF) Funding: \$3M
 <b>SPIRE</b> ★ 1,392 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>The Update Framework (TUF)</b> ★ 1,541 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>TiKV</b> ★ 13,617 Cloud Native Computing Foundation (CNCF) Funding: \$3M	 <b>Vitess</b> ★ 16,897 Cloud Native Computing Foundation (CNCF) Funding: \$3M			



CNCF Graduated Projects (25)

Landscape Guide


Reset Filters

- Grouping
- CNCF Relation
- Sort By
- Alphabetical (a to z)
- Category
- Any
- Project
- Any
- License
- Any
- Organization
- Any
- Headquarters
- Any
- Company Type
- Any
- Industry
- Any

Download as CSV


Example filters

- Cards by age
- Open source landscape
- Member cards
- Cards by stars
- Cards from China
- Certified K8s/KCSP/KTP
- Cards by MCap/Funding
- Cards without bestpractices.dev



Argo

Cloud Native Computing Foundation (CNCF)




Cilium

Cloud Native Computing Foundation (CNCF)




containerd

Cloud Native Computing Foundation (CNCF)




CoreDNS

Cloud Native Computing Foundation (CNCF)



cri-o

Cloud Native Computing Foundation (CNCF)



envoy

Cloud Native Computing Foundation (CNCF)


★ 22,759

Funding: \$3M



etcd

Cloud Native Computing Foundation (CNCF)




fluentd

Cloud Native Computing Foundation (CNCF)


★ 12,199

Funding: \$3M




flux

Cloud Native Computing Foundation (CNCF)




Harbor

Cloud Native Computing Foundation (CNCF)




Helm

Cloud Native Computing Foundation (CNCF)




Istio

Cloud Native Computing Foundation (CNCF)




Jaeger

Cloud Native Computing Foundation (CNCF)




KEDA

Cloud Native Computing Foundation (CNCF)




KEDA (Serverless)

Cloud Native Computing Foundation (CNCF)




Kubernetes

Cloud Native Computing Foundation (CNCF)




Linkerd

Cloud Native Computing Foundation (CNCF)



Open Policy Agent

Cloud Native Computing Foundation (CNCF)



Prometheus

Cloud Native Computing Foundation (CNCF)




ROOK

Cloud Native Computing Foundation (CNCF)

★ 11,830

Funding: \$3M




SPIFFE

Cloud Native Computing Foundation (CNCF)

★ 1,213

Funding: \$3M



SPIRE

Cloud Native Computing Foundation (CNCF)




The Update Framework (TUF)

Cloud Native Computing Foundation (CNCF)

★ 1,541

Funding: \$3M




TiKV

Cloud Native Computing Foundation (CNCF)

★ 13,617

Funding: \$3M



Vitess

Cloud Native Computing Foundation (CNCF)



Up and running in 500ms! 🚀

Up and running in 500ms! 🚀



## 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, WebAssembly, ...





# Was ist cloud-native?



Cloud Native Technologien befähigen  
**skalierbare Anwendungen**  
zu entwickeln und betreiben,  
**in modernen, dynamischen Umgebungen.**

Cloud Native Computing Foundation



# Cloud Native Booster



Skalierbar



Zuverlässig



Einfach





# Tarifrechner DogOP



# Tarifrechner DogOp v0.1

- # API die Hunde OP Versicherung berechnet
- # Bereitstellung als Container
- # Konfiguration über Umgebungsvariablen



# Tarifrechner DogOp v0.2

- # CRUD Operationen in REST API

- # Angebote in Postgres speichern

## Fallschirm und Rettungsgurt

- # Health Check

- # Problem Details

- # Middleware



# **Tarifrechner DogOp v0.1**



# Projekt DogOP aufsetzen

```
go mod init crossnative/dogop // Go Modul initialisieren
```

```
go get github.com/go-chi/chi/v5 // Chi Dependency einbinden
```

## Projektstruktur

```
go.mod // Modul Deskriptor mit Dependencies  
go.sum // Checksummen der Dependencies
```



# Projekt DogOP aufsetzen

```
1 package main
2
3 import (
4     "net/http"
5     "github.com/go-chi/chi/v5"
6 )
7
8 func main() {
9     r := chi.NewRouter()
10    r.HandleFunc("GET /", func(w http.ResponseWriter, req *http.Request) {
11        w.Write([]byte("Hello DogOp!"))
12    })
13    http.ListenAndServe(":8080", r)
14 }
```



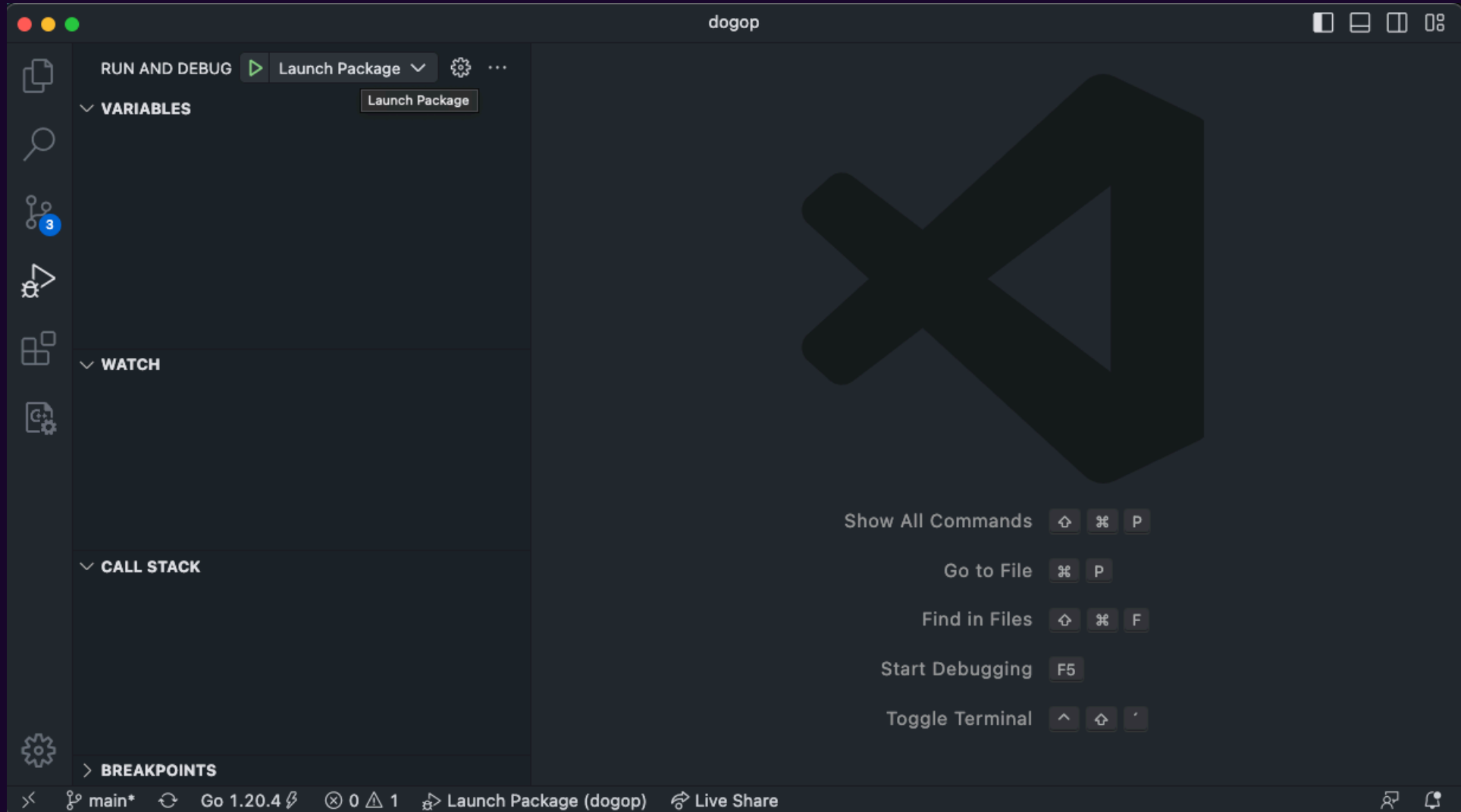
# Bauen und Ausführen

```
// 1. Kompilieren in Binary  
go build -o build/dogop .
```

```
// 2. Binary ausführen  
./build/dogop
```



# Bauen und Ausführen







# REST API für Rechner

## Request

```
POST /api/quote  
content-type: application/json
```

```
{  
  "age": 8,  
  "breed": "chow"  
}
```

## Response

```
HTTP/1.1 200 OK  
Content-Type: application/json
```

```
{  
  "age": 8,  
  "breed": "chow",  
  "tariffs": [  
    {  
      "name": "Dog OP _ Basic",  
      "rate": 12.4  
    }  
  ]  
}
```

# REST API für Rechner



```
1 func HandleQuote(w http.ResponseWriter, r *http.Request) {
2     // 1. JSON Request lesen
3     var q Quote
4     json.NewDecoder(r.Body).Decode(&q)
5
6     // 2. Tarif berechnen
7     tariff := Tariff{Name: "Dog OP _ Basic", Rate: 12.4}
8     quote.Tariffs = []Tariff{tariff}
9
10    // 3. JSON Response schreiben
11    json.NewEncoder(w).Encode(quote)
12 }
13
14 func main() {
```

Achtung, noch ohne Fehlerhandling! 💣

```
15     http.HandleFunc("/api/quote", HandleQuote)
16
17     http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
18         w.Write([]byte("Hello DogOp!"))
19     })
20
21     http.ListenAndServe(":8080", r)
22 }
23
24 }
```







# Struct statt Klasse

```
1 type Tariff struct {
2     Name string `json:"name"`
3     Rate float64 `json:"rate"`
4 }
5
6 type Quote struct {
7     Age      int      `json:"age"`
8     Breed    string    `json:"breed"`
9     Tariffs []Tariff `json:"tariffs"`
10 }
```

```
// Struct erzeugen
tariff := Tariff{Name: "Dog OP _ Basic", Rate: 12.4}
```



# Fehler

```
1 func HandleQuote(w http.ResponseWriter, r *http.Request) {  
2     // 1. JSON Request lesen  
3     var q Quote  
4     json.NewDecoder(r.Body).Decode(&q) //  Fehler möglich!  
5  
6     // 2. Tarif berechnen  
7     tariff := Tariff{Name: "Dog OP _ Basic", Rate: 12.4}  
8     quote.Tariffs = []Tariff{tariff}  
9  
10    // 3. JSON Response schreiben  
11    json.NewEncoder(w).Encode(quote) //  Fehler möglich!  
12 }
```



Jump to ...



## Documentation

Overview

▸ Index

Constants

Variables

▸ Functions

▾ Types

▾ type Decoder

NewDecoder(r)

(dec) Buffered()

(dec) Decode(v)

(dec) DisallowUnknownFiel...

(dec) InputOffset()

(dec) More()

(dec) Token()

(dec) UseNumber()

▸ type Delim

▸ type Encoder

▸ type InvalidUTF8Error

▸ type InvalidUnmarshalError

Source Files

**func (\*Decoder) Decode****func** (dec \*Decoder) Decode(v any) error

Decode reads the next JSON-encoded value from its input and stores it in the value pointed to by v.

See the documentation for Unmarshal for details about the conversion of JSON into a Go value.

## ▼ Example (Stream)

This example uses a Decoder to decode a streaming array of JSON objects.

```
package main

import (
    "encoding/json"
    "fmt"
    "log"
    "strings"
)

func main() {
    const jsonStream = `
    [
        {"Name": "Ed", "Text": "Knock knock."},
        {"Name": "Sam", "Text": "Who's there?"},
        {"Name": "Ed", "Text": "Go fmt."},
        {"Name": "Sam", "Text": "Go fmt who?"},
        {"Name": "Ed", "Text": "Go fmt yourself!"}
    ]
`

    type Message struct {
        Name, Text string
    }
}
```



# Fehler

- # Go kennt kein spezielles Konstrukt zur Fehlerbehandlung
- # Fehler sind normale Rückgabewerte



# Fehler

```
1 func HandleQuote(w http.ResponseWriter, r *http.Request) {
2     // 1. JSON Request lesen
3     var q Quote
4
5     // Potentieller Fehler
6     err := json.NewDecoder(r.Body).Decode(&q)
7
8     // Auf Fehler prüfen
9     if err != nil {
10         // Fehler behandeln
11         http.Error(w, "Could not decode quote.😞", http.StatusBadRequest)
12         return
13     }
14
15     // ...
16 }
```





# HTTP Handler testen

# Unit Tests in Standardlib enthalten

# Tests in Datei `main_test.go`



# HTTP Handler testen

```
1 func TestHandleQuote(t *testing.T) {
2     // 1. HTTP Recorder erstellen
3     recorder := httptest.NewRecorder()
4
5     // 2. Request erstellen (mit Body)
6     body := `
7     {
8         "age": 8,
9         "breed": "chow"
10    }
11    `
12    req, _ := http.NewRequest("GET", "/api/quote", strings.NewReader(body))
13
14    // 3. Handler Funktion aufrufen
15    HandleQuote(recorder, req)
16
17    // 4. Return Code prüfen
18    if recorder.Code != http.StatusOK {
19        t.Errorf("Wrong status: got %v expected %v", recorder.Code, http.StatusOK)
20    }
21 }
```



# HTTP Handler testen

```
go test -v ./...
```

```
=== RUN TestHandleQuote  
--- PASS: TestHandleQuote (0.00s)  
PASS  
ok crossnative/dogop 0.228s
```



# HTTP Handler testen

The screenshot shows a VS Code editor window with the title 'main\_test.go — dogop'. The Explorer sidebar on the left shows a project named 'DOGOP' with files like '.vscode', 'build', 'migrations', 'tmp', 'build-errors.log', 'main', '.gitignore', 'Dockerfile', 'dogop\_requests.http', 'go.mod', 'go.sum', 'main\_test.go', 'main.go', 'offer.go', and 'README.md'. The 'main\_test.go' file is selected and open in the editor. The code in the editor is as follows:

```
run test | debug test
10 func TestHandleQuote(t *testing.T) {
11     // 1. HTTP Recorder erstellen
12     recorder := httptest.NewRecorder()
13
14     // 2. Request erstellen (mit Body)
15     body := `
16     {
17         "age": 8,
18         "breed": "chow"
19     }
20
21     req, _ := http.NewRequest("GET", "/api/quote", strings.NewReader(body))
22
23     // 3. Handler Funktion aufrufen
24     HandleQuote(recorder, req)
25
26     // 4. Return Code prüfen
27     if recorder.Code != http.StatusOK {
28         t.Errorf("Wrong status: got %v expected %v", recorder.Code, http.StatusOK)
29     }
30 }
31
```

Below the editor, the 'OUTPUT' panel shows the test results:

```
Running tool: /opt/homebrew/opt/go/libexec/bin/go test -timeout 30s -run ^TestHandleQuote$ ppix/dogop

ok      ppix/dogop  0.176s
```

The status bar at the bottom indicates the current file is 'main', the Go version is 'Go 1.20.4', and the package is 'Launch Package (dogop)'. The cursor is at 'Ln 17, Col 18'.



# Port per Umgebungsvariable konfigurieren

```
1 // Einfach per Standardlib
2
3 function main() {
4     port := os.Getenv("DOGOP_PORT")
5     http.ListenAndServe(fmt.Sprintf(":%v", config.Port), r)
6 }
```



# Port per Umgebungsvariable konfigurieren

```
1 // Oder mit envconfig von Kelsey Hightower
2
3 type Config struct {
4     Port string `default:"8080"`
5 }
6
7 function main() {
8     var config Config
9     err := envconfig.Process("dogop", &config)
10    if err != nil {
11        log.Fatal(err.Error())
12    }
13
14    http.ListenAndServe(fmt.Sprintf(":%v", config.Port), r)
15 }
```



# Dockerfile für DogOP

```
1 # 1. DogOp Builder
2 FROM golang as builder
3 WORKDIR /app
4 ADD . /app
5 RUN CGO_ENABLED=0 go build -ldflags="-w -s" -o build/dogop .
6
7 # 2. DogOp Container
8 FROM alpine
9 COPY --from=builder /app/build/dogop /usr/bin/
10 EXPOSE 8080
11 ENTRYPOINT ["/usr/bin/dogop"]
```



# Docker DogOP mit



Buildpacks.io

```
// 1. Docker Image bauen  
>_ pack build dogop-cnb  
    --buildpack paketo-buildpacks/go  
    --builder paketobuildpacks/builder-jammy-base
```

```
// 2. Docker Image ausühren  
>_ docker run dogop-cnb
```





# Tarifrechner DogOp v0.2

# CRUD API für Angebote



## Create Request

```
POST /api/offer
content-type: application/json

{
  "name": "Rollo",
  "age": 8,
  "breed": "chow",
  "customer": "jan"
}
```

## Create Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "427ed4de",
  "customer": "jan",
  "age": 8,
  "breed": "chow",
  "name": "Rollo"
}
```

## Read Request

```
GET /api/offer/427ed4de
```

## Read Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "427ed4de",
  "customer": "jan",
  "age": 8,
  "breed": "chow",
  "name": "Rollo"
}
```



# CRUD API für Angebote

```
r.HandleFunc("POST /api/offer", HandleCreateOffer(offerRepository))  
r.HandleFunc("GET /api/offer/{ID}", HandleReadOffer(offerRepository))
```



# Create Request mit Handler Function

```
1 func HandleCreateOffer(offerRepository *OfferRepository) http.HandlerFunc {
2     return func(w http.ResponseWriter, req *http.Request) {
3         // 1. JSON Request lesen
4         var offer Offer
5         json.NewDecoder(req.Body).Decode(&offer)
6
7         // 2. Offer speichern
8         createdOffer, _ := offerRepository.Insert(req.Context(), &offer)
9
10        // 3. JSON Response schreiben
11        json.NewEncoder(w).Encode(createdOffer)
12    }
13 }
```



# Angebote speichern in Postgres

# pgx als Postgres Driver und Toolkit

# Interface database/sql in Go Standardlib



# Angebote speichern in Postgres

```
1 type OfferRepository struct {  
2     connPool *pgxpool.Pool  
3 }  
4  
5 func (r *OfferRepository) Insert(ctx context.Context, offer *Offer)  
6     // ...  
7 }
```



# Angebote speichern in Postgres

```
1 func (r *OfferRepository) Insert(ctx context.Context, offer *Offer) (*Offer, error) {
2     // 1. ID generieren
3     offer.ID = uuid.New().String()
4
5     // 2. Transaktion beginnen
6     tx, _ := r.connPool.Begin(ctx)
7     defer tx.Rollback(ctx)
8
9     // 3. Offer per Insert speichern
10    _, err := tx.Exec(
11        ctx,
12        `INSERT INTO offers
13        (id, customer, age, breed, name)
14        VALUES
15        ($1, $2, $3, $4, $5)`,
16        offer.ID, offer.Customer, offer.Age, offer.Breed, offer.Name,
17    )
18    if err != nil {
19        return nil, err
20    }
21
22    // 4. Transaktion commiten
23    tx.Commit(ctx)
24
25    // 5. Gespeicherte Offer zurückgeben
26    return offer, nil
27 }
```

# Aus dem Context gerissen



```
1 return func(w http.ResponseWriter, req *http.Request) {
2     // ...
3
4     // 2. Offer speichern
5     createdOffer, _ := offerRepository.Insert(req.Context(), &offer)
6
7     // ...
8 }
9
10 func (r *OfferRepository) Insert(ctx context.Context, offer *Offer) (*Offer, error) {
11     // ..
12
13     // 2. Transaktion beginnen
14     tx, _ := r.connPool.Begin(ctx)
15     defer tx.Rollback(ctx)
16
17     // 3. Offer per Insert speichern
18     _, err := tx.Exec(
19         ctx,
20         `INSERT INTO offers
21         (id, customer, age, breed, name)
22         VALUES
23         ($1, $2, $3, $4, $5)`,
24         offer.ID, offer.Customer, offer.Age, offer.Breed, offer.Name,
25     )
26     // ...
27
28     // 4. Transaktion commiten
29     tx.Commit(ctx)
30
31     // ...
32 }
```





# Fallschirm und Rettungsgurt



# Health Check

# Nutzung von health-go (hellofresh)

# Integration des Checks für pgx



# Health Check

```
1 // Register Health Check
2 h, _ := health.New(health.WithChecks(
3     health.Config{
4         Name: "db",
5         Timeout: time.Second * 2,
6         SkipOnError: false,
7         Check: healthPgx.New(healthPgx.Config{
8             DSN: config.Db,
9         }),
10    },
11 ))
12
13 // Register Handler Function
14 r.HandleFunc("GET /health", h.HandlerFunc)
```



# Problem Details

## Ungültiger Request

```
GET /api/offer/-invalid-
```

## Response mit Problem Details

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json

{
  "reason": "invalid UUID format",
  "status": 400,
  "title": "invalid request"
}
```

# Problem Details



## Problem Details for HTTP APIs

### Abstract

This document defines a "problem detail" as a way to carry machine-readable details of errors in a HTTP response to avoid the need to define new error response formats for HTTP APIs.

### Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in [Section 2 of RFC 5741](#).

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <http://www.rfc-editor.org/info/rfc7807>.

### Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents



# Fehler behandeln

## Mit Problem Details

```
if err != nil {  
    problem.New(  
        problem.Title("invalid request"),  
        problem.Wrap(err),  
        problem.Status(http.StatusBadRequest),  
    ).WriteTo(responseWriter)  
    return  
}
```

## Mit Standardbibliothek

```
if err != nil {  
    http.Error(responseWriter, "invalid request", http.StatusBadRequest)  
    return  
}
```



# Middleware

# cross-cutting Features für alle Requests  
(z.B. Logging, Authentifizierung)

# erzeugen eine Kette von Handlern

router => middleware handler => application handler

# implementieren das Interface `http.Handler`





# Middleware loggt Requests

```
1 func loggingMiddleware(next http.Handler) http.Handler {
2     return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
3         log.Printf("%v requested URL %v", r.Host, r.URL)
4         next.ServeHTTP(w, r)
5     })
6 }
7
8 func main() {
9     r := chi.NewRouter()
10
11     // Nutze Logging Middleware
12     r.Use(loggingMiddleware)
13
14     http.ListenAndServe(":8080", r)
15 }
```



# Middlewares nutzen

```
1 func main() {  
2     r := chi.NewRouter()  
3  
4     // Basis Middleware  
5     r.Use(middleware.RequestID)  
6     r.Use(middleware.RealIP)  
7     r.Use(middleware.Logger)  
8     r.Use(middleware.Recoverer)  
9  
10    // Timeout über Request Context setzen  
11    r.Use(middleware.Timeout(60 * time.Second))  
12  
13    http.ListenAndServe(":8080", r)  
14 }
```



# Zusammenfassung DogOP

## v0.1

1. Go Projekt aufsetzen
2. einfache Rest API Handler
3. Konfiguration per Umgebungsvariable
4. Structs
5. Fehler behandeln
6. Unit Tests
7. Docker Container bauen

## v0.2

1. CRUD API für Angebote
2. Angebote in Postgres speichern
3. Context
4. Health Check
5. Problem Details
6. Middleware



# Cloud Native Booster



Skalierbar



Zuverlässig



Einfach



# 3 Gründe für Go

1. Einfach
2. Mächtig
3. Langweilig



# Jan Stamer

[jan.stamer@crossnative.com](mailto:jan.stamer@crossnative.com)

in





# Go über den Wolken

