Go über den Wolken

CLOUD NATIVE LANDSCAPE

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

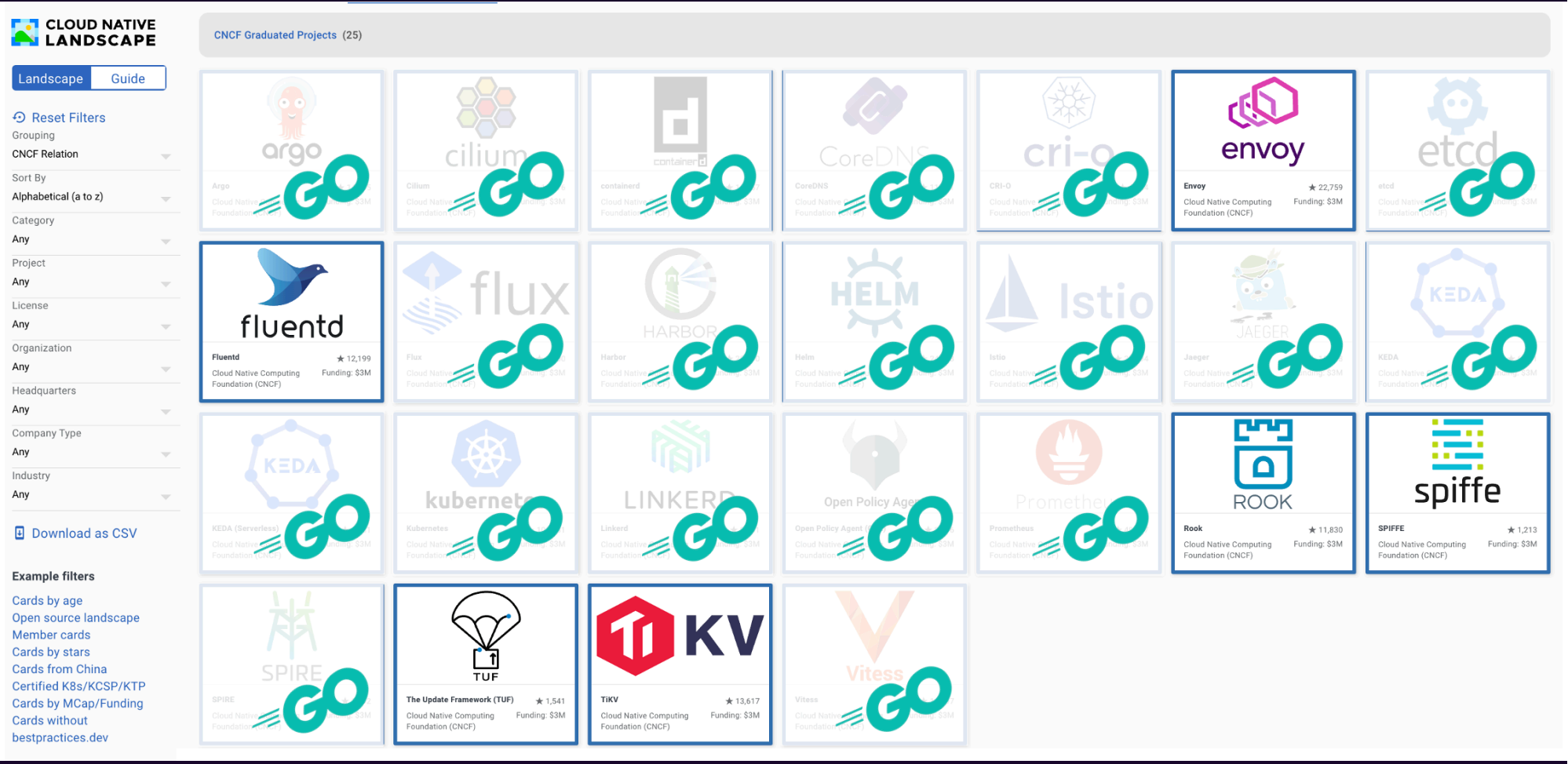
Download as CSV

Example filters

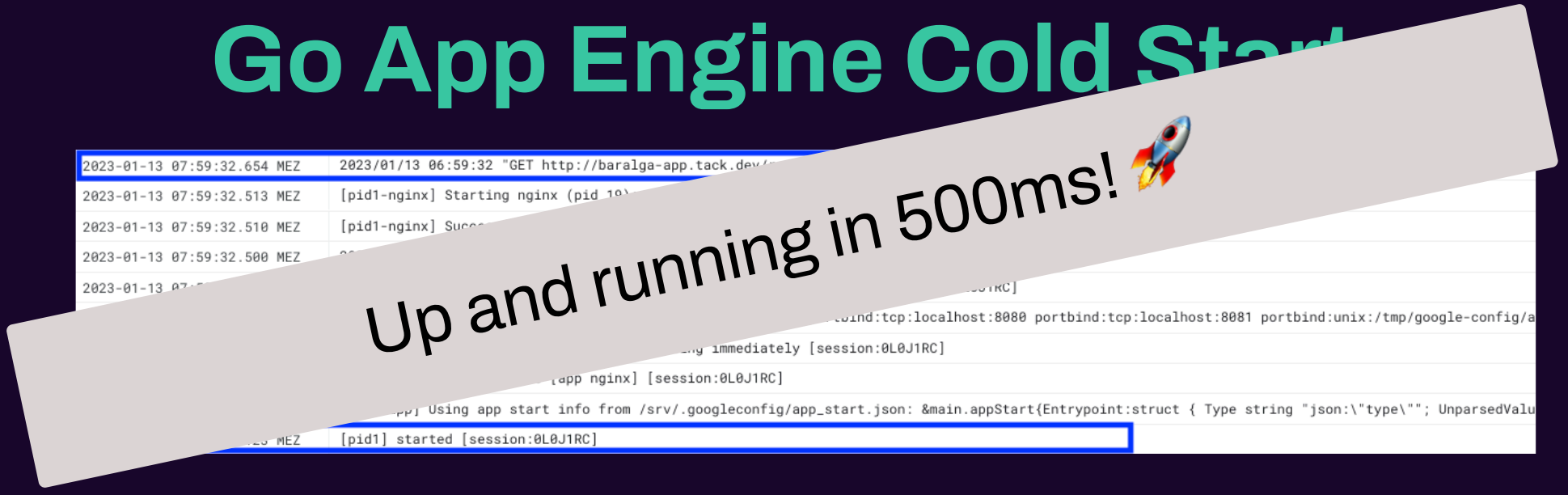
Cards by age
Open source landscape
Member cards
Cards by stars
Cards from China
Certified K8s/KCSP/KTP
Cards by MCap/Funding
Cards without bestpractices.dev

CNCF Graduated Projects (25)

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


CLOUD NATIVE LANDSCAPE
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
CNCF Graduated Projects (25)
envoy
Envoy
★ 22,759
Cloud Native Computing Foundation (CNCF)
Funding: $3M
fluentd
Fluentd
★ 12,199
Cloud Native Computing Foundation (CNCF)
Funding: $3M
ROOK
Rook
★ 11,830
Cloud Native Computing Foundation (CNCF)
Funding: $3M
spiffe
SPIFFE
★ 1,213
Cloud Native Computing Foundation (CNCF)
Funding: $3M
TUF
The Update Framework (TUF)
★ 1,541
Cloud Native Computing Foundation (CNCF)
Funding: $3M
TiKV
★ 13,617
Cloud Native Computing Foundation (CNCF)
Funding: $3M

Go App Engine Cold Start



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

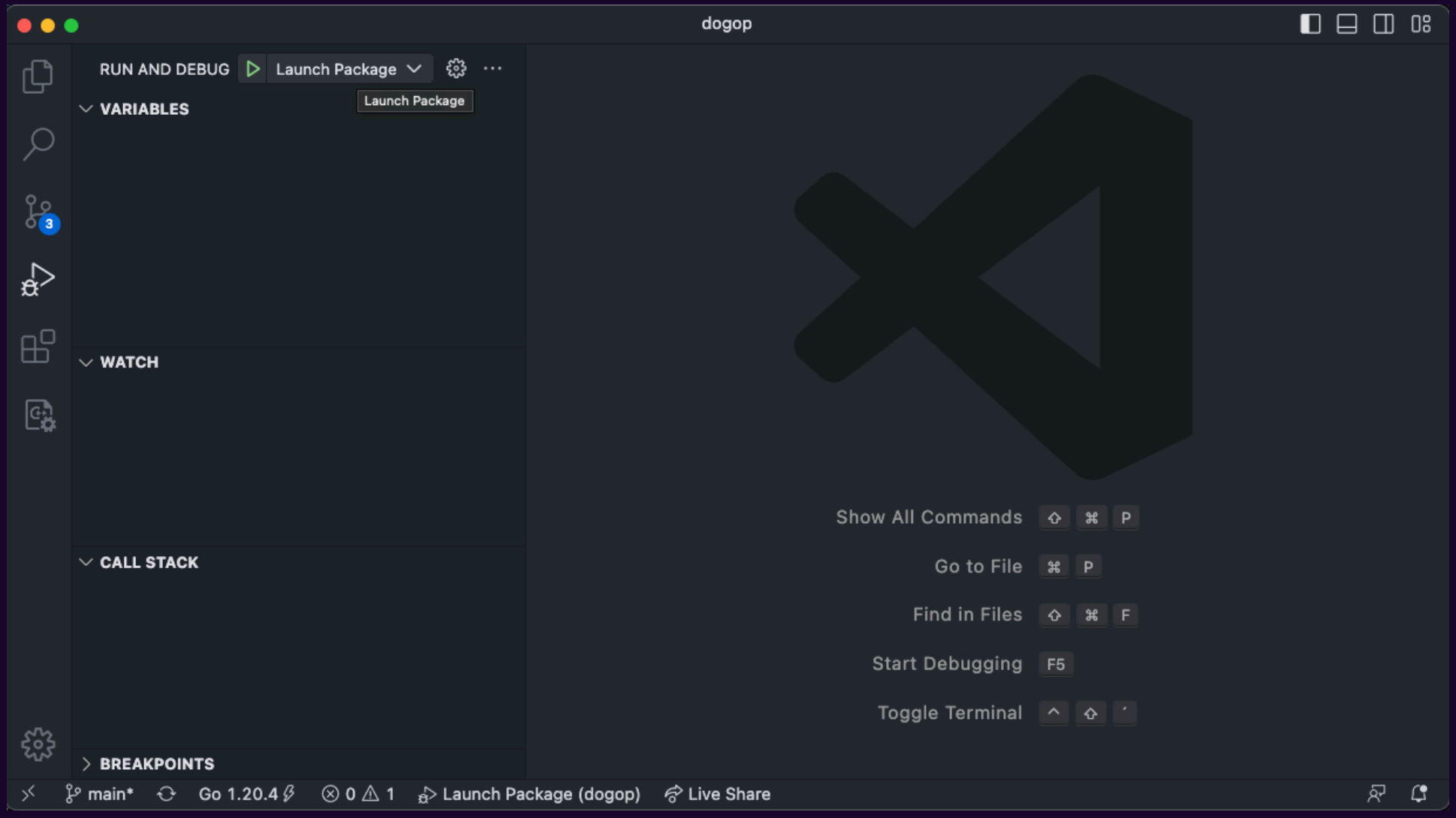
```go
package main

import (
  "net/http"
  "github.com/go-chi/chi/v5"
)

func main() {
  r := chi.NewRouter()
  r.HandleFunc("GET /", func(w http.ResponseWriter, req *h
    w.Write([]byte("Hello DogOp!"))
  })
  http.ListenAndServe(":8080", r)
}
```

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

```go
func HandleQuote(w http.ResponseWriter, r *http.Request) {
  // 1. JSON Request lesen
  var q Quote
  json.NewDecoder(r.Body).Decode(&q)

  // 2. Tarif berechnen
  tariff := Tariff{Name: "Dog OP _ Basic", Rate: 12.4}
  quote.Tariffs = []Tariff{tariff}

  // 3. JSON Response schreiben
  json.NewEncoder(w).Encode(quote)
}

func m

                    /api/quote", HandleQuote)

    r.HandleFunc("GET /", func(w http.ResponseWriter, r *http.Request) {
      w.Write([]byte("Hello DogOp!"))
    })

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

Achtung, noch ohne Fehlerhandling! 💣

Struct statt Klasse

```go
type Tariff struct {
  Name string  `json:"name"`
  Rate float64 `json:"rate"`
}

type Quote struct {
  Age     int       `json:"age"`
  Breed   string    `json:"breed"`
  Tariffs []Tariff `json:"tariffs"`
}
```

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

Fehler 💣

```go
func HandleQuote(w http.ResponseWriter, r *http.Request) {
  // 1. JSON Request lesen
  var q Quote
  json.NewDecoder(r.Body).Decode(&q) // 💣 Fehler möglich!

  // 2. Tarif berechnen
  tariff := Tariff{Name: "Dog OP _ Basic", Rate: 12.4}
  quote.Tariffs = []Tariff{tariff}

  // 3. JSON Response schreiben
  json.NewEncoder(w).Encode(quote) // 💣 Fehler möglich!
}
```

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 💣

```go
func HandleQuote(w http.ResponseWriter, r *http.Request) {
  // 1. JSON Request lesen
  var q Quote

  // Potentieller Fehler
  err := json.NewDecoder(r.Body).Decode(&q)

  // Auf Fehler prüfen
  if err != nil {
    // Fehler behandeln
    http.Error(w, "Could not decode quote.😔", http.StatusBadRequest)
    return
  }

  // ...
}
```

HTTP Handler testen

Unit Tests in Standardlib enthalten
Tests in Datei `main_test.go`

HTTP Handler testen

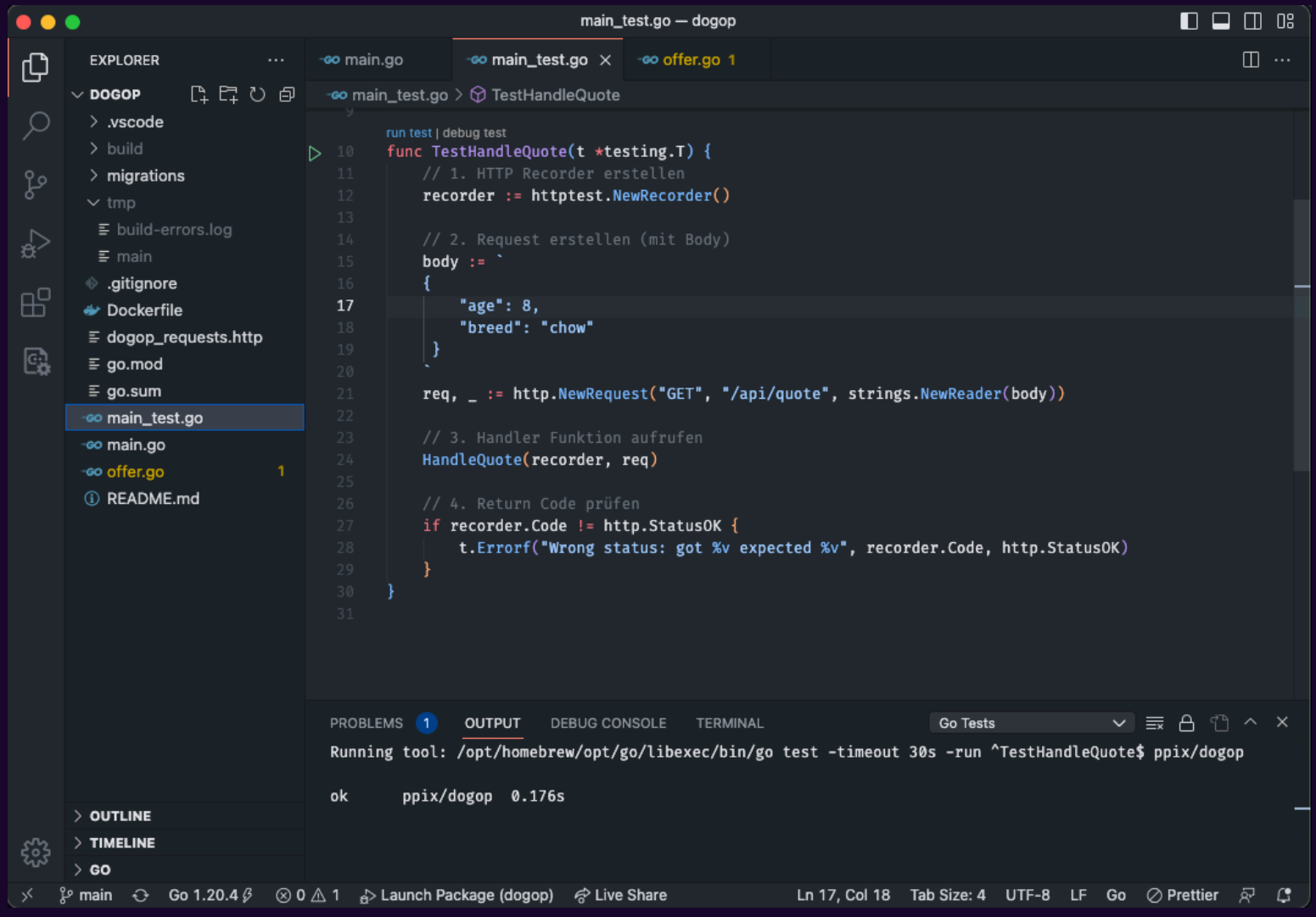
```go
func TestHandleQuote(t *testing.T) {
  // 1. HTTP Recorder erstellen
  recorder := httptest.NewRecorder()

  // 2. Request erstellen (mit Body)
  body := `
  {
    "age": 8,
    "breed": "chow"
  }
  `
  req, _ := http.NewRequest("GET", "/api/quote", strings.NewReader(body))

  // 3. Handler Funktion aufrufen
  HandleQuote(recorder, req)

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

HTTP Handler testen

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

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

HTTP Handler testen



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

```go
func HandleCreateOffer(offerRepository *OfferRepository) http.HandlerFunc {
  return func(w http.ResponseWriter, req *http.Request) {
    // 1. JSON Request lesen
    var offer Offer
    json.NewDecoder(req.Body).Decode(&offer)

    // 2. Offer speichern
    createdOffer, _ := offerRepository.Insert(req.Context(), &offer)

    // 3. JSON Response schreiben
    json.NewEncoder(w).Encode(createdOffer)
  }
}
```

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

```go
func (r *OfferRepository) Insert(ctx context.Context, offer *Offer) (*Offer, error) {
  // 1. ID generieren
  offer.ID = uuid.New().String()

  // 2. Transaktion beginnen
  tx, _ := r.connPool.Begin(ctx)
  defer tx.Rollback(ctx)

  // 3. Offer per Insert speichern
  _, err := tx.Exec(
  ctx,
  `INSERT INTO offers
  (id, customer, age, breed, name)
  VALUES
  ($1, $2, $3, $4, $5)`,
  offer.ID, offer.Customer, offer.Age, offer.Breed, offer.Name,
  )
  if err != nil {
    return nil, err
  }

  // 4. Transaktion commiten
  tx.Commit(ctx)

  // 5. Gespeicherte Offer zurückgeben
  return offer, nil
}
```

Aus dem Context gerissen

```
return func(w http.ResponseWriter, req *http.Request) {
  // ...

  // 2. Offer speichern
  createdOffer, _ := offerRepository.Insert(req.Context(), &offer)

  // ...
}

func (r *OfferRepository) Insert(ctx context.Context, offer *Offer) (*Offer, error) {
  // ..

  // 2. Transaktion beginnen
  tx, _ := r.connPool.Begin(ctx)
  defer tx.Rollback(ctx)

  // 3. Offer per Insert speichern
  _, err := tx.Exec(
  ctx,
  `INSERT INTO offers
  (id, customer, age, breed, name)
  VALUES
  ($1, $2, $3, $4, $5)`,
  offer.ID, offer.Customer, offer.Age, offer.Breed, offer.Name,
  )
  // ...

  // 4. Transaktion commiten
  tx.Commit(ctx)

  // ...
}
```

Fallschirm und Rettungsgurt

Health Check

Nutzung von `health-go` (hellofresh)
Integration des Checks für `pgx`

Health Check

```
// Register Health Check
h, _ := health.New(health.WithChecks(
    health.Config{
        Name:      "db",
        Timeout:   time.Second * 2,
        SkipOnErr: false,
        Check: healthPgx.New(healthPgx.Config{
            DSN: config.Db,
        }),
    },
))

 // Register Handler Function
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

Internet Engineering Task Force (IETF)
Request for Comments: 7807
Category: Standards Track
ISSN: 2070-1721

M. Nottingham
Akamai
E. Wilde

March 2016

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.

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

```go
func loggingMiddleware(next http.Handler) http.Handler {
  return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    log.Printf("%v requested URL %v", r.Host, r.URL)
    next.ServeHTTP(w, r)
  })
}

func main() {
  r := chi.NewRouter()

  // Nutze Logging Middleware
  r.Use(loggingMiddleware)

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

Middlewares nutzen

```go
func main() {
  r := chi.NewRouter()

  // Basis Middleware
  r.Use(middleware.RequestID)
  r.Use(middleware.RealIP)
  r.Use(middleware.Logger)
  r.Use(middleware.Recoverer)

  // Timeout über Request Context setzen
  r.Use(middleware.Timeout(60 * time.Second))

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

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

Go über den Wolken