Workshop *Go for Microservices*

Rainer Stropek (software architects)

31. August 2020

Inhaltsverzeichnis

Einleitung

Workshop *Go for Microservices*

In diesem Dokument wird der Go-Workshop beschrieben, der am 31. August 2020 im Rahmen der Magdeburger Developer Days[1] stattfindet.

Hier die Beschreibung des Workshops wie im Programm der Konferenz angekündigt:

> Die Programmiersprache Go hat in den letzten Jahren viel Aufsehen erregt. Im Microservices-Umfeld wurde sie besonders bekannt, weil Docker mit Go geschrieben ist. Go zeichnet sich durch einfache und klare Syntax, hohe Laufzeiteffizienz, gute Unterstützung von nebenläufiger Programmierung und viele weitere Dinge aus, die die Sprache besonders geeignet für Web APIs machen.

> Dieser Workshop ist gedacht für Entwicklerinnen und Entwickler, die in Go einsteigen möchten. Vorkenntnisse in anderen objektorientierten Sprachen wie Java, C# oder TypeScript sind notwendig. Darauf aufbauend lernen die Teilnehmerinnen und Teilnehmer die Tools von Go und die Syntax der Sprache kennen. Natürlich wird auch das Verpacken von Go-Programmen in Docker Containern ein Thema sein. Nach dem Behandeln der Grundlagen sprechen wir über die Umsetzung von Microservices mit Go. Gestartet wird, indem wir ohne besonderem Framework eine einfache, HTTP-basierende Web API erstellen und in Docker betreiben. Anschließend diskutieren wir, welche Aspekte für eine vollwertige Microservice-Infrastruktur fehlen. Rainer Stropek wird einige populäre Microservice-Frameworks für Go gegenüberstellen und anschließend im Besonderen auf *Go kit* genauer eingehen. Ziel ist, ein einfaches Microservice mit Go kit zu erstellen und es in der Cloud zu hosten. Im Workshop steht praktisches Programmieren statt Slides im Vordergrund. Wir werden Go anhand vieler live ausprobierter Beispiele kennenlernen.

Trainer

Ihr Trainer für den Go-Workshop ist Rainer Stropek[2]. Seine Aufgabe ist es, Ihnen das notwendige Basiswissen zu vermitteln und die praktischen Übungen, die im Rahmen des Workshops durchgeführt werden, vorzubereiten.

Hier eine kurze Vorstellung von Rainer Stropek:

[1] https://md-devdays.de/
[2] https://rainerstropek.me

> Rainer Stropek ist seit über 25 Jahren als Unternehmer in der IT-Industrie tätig. Er gründete und führte in dieser Zeit mehrere IT-Dienstleistungsunternehmen und entwickelt im Augenblick in seiner Firma *software architects* mit seinem Team die preisgekrönte Software *time cockpit*.

> Rainer hat Abschlüsse an der Höheren Technischen Schule für MIS, Leonding (AT) sowie der University of Derby (UK). Er ist Autor mehrerer Fachbücher und Artikel zu Themen im Bereich Softwareentwicklung und Softwarearchitektur. Seine technischen Schwerpunkte sind Cloud Computing, Software-as-a-Service und damit verbundene Technologien. Rainer tritt regelmäßig als Speaker und Trainer auf namhaften Konferenzen in Europa und den USA auf. 2010 wurde Rainer von Microsoft zu einem der ersten MVPs für die Windows Azure Plattform ernannt. Seit 2015 ist Rainer Microsoft Regional Director. 2016 hat Rainer zusätzlich den MVP Award für Visual Studio und Developer Technologies erhalten.

Bei Fragen erreichen Sie Rainer Stropek per Email unter rainer@software-architects.at[3] oder per Twitter unter *@rstropek*.

[3] mailto:rainer@software-architects.at

Organisatorisches

Web Meeting

Der Workshop findet über Webmeeting statt. Als Webmeeting-Plattform wird *Microsoft Teams* verwendet. Hier der Teilnahmelink:

> http://bit.ly/go-md-devdays

Teilnehmerinnen und Teilnehmer können über den Teams Web Client oder die Teams Clientsoftware teilnehmen.

Zeitplan

Der Workshop findet in vier Blöcken zu je ca. 1,5 Stunden statt:

Zeit (MESZ)	Inhalt
09:00 - 10:30	Workshop Teil 1
10:30 - 11:00	Pause
11:00 - 12:30	Workshop Teil 2
12:30 - 13:15	Pause
13:15 - 14:45	Workshop Teil 3
14:45 - 15:15	Pause
15:15 - 16:45	Workshop Teil 4

Workshopformat

Der Workshop besteht aus einer Kombination von Theoriewissen und praktischen Übungen. Rund zwei Drittel der Zeit zeigt der Trainer konzeptionelle Inhalte oder Codebeispiele, anhand derer Go-Basiswissen aufgebaut wird. Ein Drittel der Zeit steht den Teilnehmerinnen und Teilnehmern zur Verfügung, um Go selbst anhand von Aufgabenstellungen auszuprobieren, die der Trainer mitbringen wird.

Systemvoraussetzungen

Teilnehmerinnen und Teilnehmer des Workshops sollten folgende Softwarepakete auf ihren Computern installiert haben, um aktiv teilnehmen zu können:

- Go 1.15[4]
- Aktuelle Version von Git[5]
- Aktuelle Version von Visual Studio Code[6] mit folgenden Extensions: [7]
  - Go for *Visual Studio Code*[8]
  - REST Client[9]
- Docker Desktop[10] (Windows, Mac) oder Docker CE[11] (Linux)
  - Unter Windows wird empfohlen, *Docker Desktop* mit dem *Windows Subsystem for Linux 2*[12] zu verwenden.

[4] https://golang.org/dl/
[5] https://git-scm.com/
[6] https://code.visualstudio.com/
[7] Natürlich können auch andere Entwicklungsumgebungen verwendet werden. Der Trainer wird jedoch *Visual Studio Code* verwenden und kann bei Problemen oder Fragen andere Umgebungen betreffend eventuell nicht helfen.
[8] https://marketplace.visualstudio.com/items?itemName=golang.Go
[9] https://marketplace.visualstudio.com/items?itemName=humao.rest-client
[10] https://www.docker.com/products/docker-desktop
[11] https://docs.docker.com/engine/install/#supported-platforms
[12] https://docs.microsoft.com/en-us/windows/wsl/about

Workshopinhalt

In diesem Kapitel werden die behandelten Themen zusammengefasst und die grobe Gliederung in die Workshopblöcke gezeigt. Es ist ohne weiteres möglich, dass abhängig vom Vorwissen der Teilnehmerinnen und Teilnehmer ad hoc Änderungen am Zeitplan oder Inhalt durchgeführt werden. Der Trainer wird die entsprechenden Schwerpunkte setzen und auf das Zeitmanagement achten.

Block 1

- Einführung in Go
  - Historie
  - Grundidee
  - Philosophie
- Grundlagen der Go-Entwicklung
  - Packages
  - Module System
  - Go Compiler (Windows, Linux, Webassembly)
  - Go in *Visual Studio Code* (Editing, Compiling, Debugging)
  - Go in *Docker* images
  - C Interop
- Grundlagen der Sprache Go
  - Variablen und Konstanten
  - Schleifen
  - Bedingungen
  - Arrays und Slices
  - Functions
  - Pointers
  - Structs, Methods, Interfaces
  - Errors

Block 2

- Goroutines und Channels
  - Grundlagen von Goroutines
  - Grundlagen von Channels

  - Channel Buffering
  - Channel Synchronization
  - Non-blocking Channel operations
  - Select
  - Timeouts
  - Atomic counters, Mutexes
- Panic, defer
- JSON in Go
- HTTP Go
  - Server
  - Client

Block 3

- Kurze Wiederholung der Grundlagen von HTTP-basierenden Web APIs
  - Länge und Tiefe hängt ab vom Vorwissen der Teilnehmerinnen und Teilnehmer
- Entwickeln einer einfachen Web API mit Go
  - HTTP Server
  - API Endpoints, Routes
  - JSON Results
  - Web API in Docker Image, Betrieb in Docker Container
- Konsumieren der Web API mit Go
- Go und gRPC
  - Grundlagen von gRPC
  - gRPC Server
  - gRPC Client

Block 4

- Microservices
  - Wiederholung der Grundkonzepte (Länge und Tiefe hängt ab vom Vorwissen der Teilnehmerinnen und Teilnehmer)
  - Warum Microservices mit Go?
  - Was braucht ein Microservice mehr als eine Web oder gRPC API?

- Go Frameworks für Microservices
- Go kit
  - Why using a framework like Go kit?
  - Key concepts of Go kit
  - Go kit example walkthrough

Übungen

Übung 1: Turmrechnen

Anforderung

Ihre Aufgabe ist das Schreiben einer Kommandozeilenanwendung in Go zum *Turmrechnen*[13].

Ein Benutzer übergibt Ihrer Anwendung zwei Parameter über die Kommandozeile:

- Den Startwert (z.B. *9*); der Startwert muss > 1 sein.
- Die "Höhe" (z.B. *5); die Höhe muss > 2 sein.

Sie geben das Ergebnis in folgendem Format aus:

```
   9 * 2 = 18
  18 * 3 = 54
  54 * 4 = 216
 216 * 5 = 1080
1080 / 2 = 540
 540 / 3 = 180
 180 / 4 = 45
  45 / 5 = 9
```

Sollte der Benutzer falsche Parameter in der Kommandozeile eingegeben haben oder Parameter fehlen, geben Sie eine entsprechende Fehlermeldung am Bildschirm aus.

Overflows können Sie ignorieren.

Tipps

- Zugriff auf Kommandozeilenparameter:
  - Manuell[14]
  - Über *Flags*[15]
- Sie können Ausgaben links mit Leerzeichen auffüllen, indem sie entsprechende Formatangaben bei `Printf` (und Varianten dieser Funktion) angeben. Z.B. gibt `fmt.Printf("|%6d|\n", 42)` vier Leerzeichen und danach *42* aus, um auf sechs Stellen zu kommen.
- Sie können Strings in Integer mit `strconv.Atoi`[16] umwandeln.

[13] http://www.floriangeier.at/schule/kopf/kopf.php
[14] https://gobyexample.com/command-line-arguments
[15] https://gobyexample.com/command-line-flags
[16] https://golang.org/pkg/strconv/

Levels

Je nach Vorwissen wird diese Aufgabe manchen schwerer und manchen leichter fallen. Daher hier Anregungen, wie man Schritt für Schritt das Beispiel lösen könnte. Jeder kann basierend auf der bestehenden Programmierpraxis die Anzahl an Levels abarbeiten, die für sie oder ihn passen.

Level 1 - Rechenlogik

- Treffen Sie Annahmen für Startwert und Höhe und verzichten Sie auf Kommandozeilenparameter.
- Geben Sie nur die Zwischenergebnisse aus. Z.B.:

```
9
18
54
216
1080
540
180
45
9
```

Level 2 - Kommandozeilenparameter

- Fügen Sie die Möglichkeit hinzu, die Parameter über Kommandozeile zu übergeben.
- Denken Sie an die Prüfung der Parameter mit entsprechenden Fehlermeldungen.

Level 3 - Bessere Ausgabe

- Verbessern Sie die Ausgabe, damit das Ergebnis so aussieht, wie am Beginn dieser Beschreibung gefordert.
- Geben Sie wenn möglich den Ausgangswert rechtsbündig aus.

Level 4 - Unit Testing

- Strukturieren Sie Ihren Code so, dass Sie ihn gut testen können.
- Schreiben Sie mindestens einen sinnvollen Unit Test (Kurzanleitung[17])

[17] https://gobyexample.com/testing

Übung 2: PI Monte Carlo

Anforderung

Ihre Aufgabe ist die Näherungsweise Berechnung von PI mit Hilfe der Monte Carlo[18] Methode. Es geht bei diesem Beispiel nicht darum, PI möglichst genau zu berechnen. Ziel ist das Üben von Parallelverarbeitung mit Go. Wenn das Ergebnis in etwa PI entspricht (auf 2-3 Stellen genau) ist das ausreichend.

Verwenden Sie zur Berechnung von PI möglichst alle in Ihrem PC vorhandenen Prozessoren parallel.

Berechnungsmethode

Eine gute Beschreibung der näherungsweisen Berechnung von PI findet man hier[19].

Erstellen Sie viele, zufällige Punkte in einem Quadrat mit Seitenlänge 1.

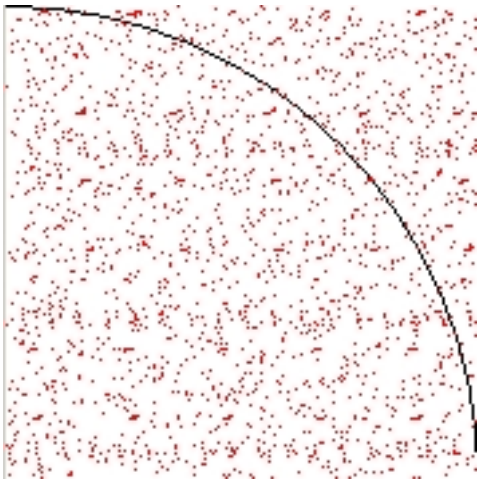

Abbildung 1: Punkte

Zählen Sie die Punkte, die im Einheitskreis (Radius = 1) liegen.

[18] https://en.wikipedia.org/wiki/Monte_Carlo_method
[19] https://www.zum.de/Faecher/Inf/RP/Java/java_1.htm

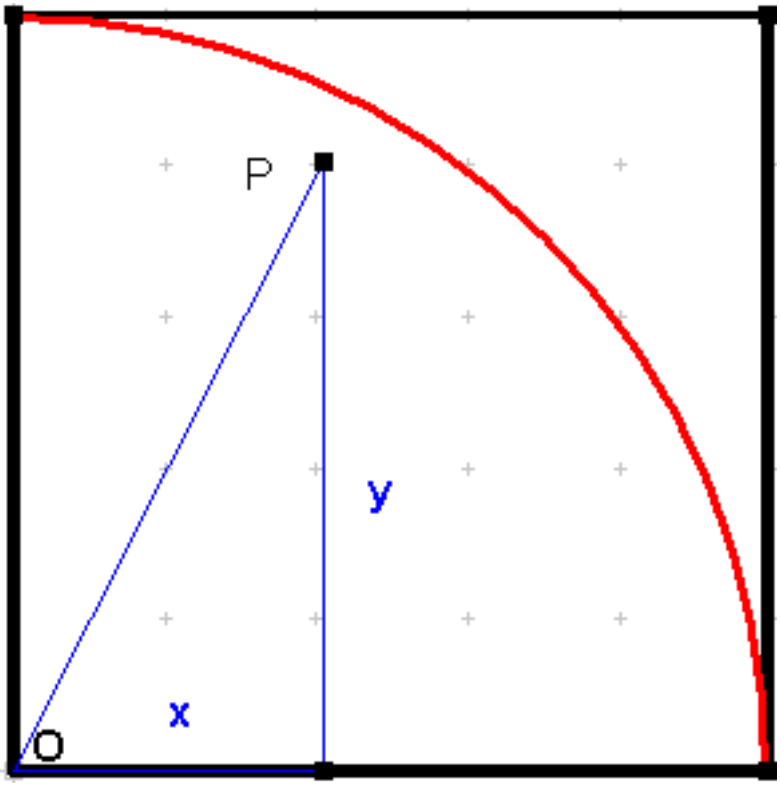

Abbildung 2: Einheitskreis

PI ergibt sich aus dem Verhältnis von Punkten innerhalb des Einheitskreises und der Anzahl an zufälligen Punkten.

$$\pi \approx \frac{4\,v}{g}$$

Abbildung 3: Formel

Hier Beispielcode, der zeigt, wie die näherungsweise Berechnung von PI in Go aussehen könnte:

```
const ITERATIONS = 10000000

// Create a new random number generate. Attention! r is not thread safe.
// You need to call `rand.New` in each goroutine.
r := rand.New(rand.NewSource(time.Now().UTC().UnixNano()))

inside := uint(0)
for i := 0; i < ITERATIONS; i++ {
    a := r.Float32()
```

```
    b := r.Float32()
    c := a*a + b*b
    if c <= float32(1) {
        inside++
    }
}

pi := float32(inside) / float32(ITERATIONS) * float32(4)
fmt.Printf("%.6f", pi)
```

Tipps

- Die Anzahl an logischen CPUs erhalten Sie mit `runtime.NumCPU()`

Levels

Je nach Vorwissen wird diese Aufgabe manchen schwerer und manchen leichter fallen. Daher hier Anregungen, wie man Schritt für Schritt das Beispiel lösen könnte. Jeder kann basierend auf der bestehenden Programmierpraxis die Anzahl an Levels abarbeiten, die für sie oder ihn passen.

Level 1 - Rechenlogik

- Nehmen Sie den Beispielcode von oben und wandeln Sie ihn in ein lauffähiges Go-Programm um.
- Kontrollieren Sie, ob ein Ergebnis in der Nähe von PI geliefert wird.

Level 2 - Parallelverarbeitung

- Ändern Sie den Code so, dass er parallel auf jeder CPU ausgeführt wird.

Level 3 - Hilfsmethode

- Gliedern Sie den Algorithmus, der eine Funktion auf allen CPUs parallel ausführt, in eine eigene Funktion aus. Der Code zur Parallelisierung und der zur Berechnung von PI sollte logisch getrennt sein. Hier ein Beispiel, wie das aussehen könnte:

```
func runInParallel(body func(...), goroutines int, ...) {
    ...
    for i := 0; i < goroutines; i++ {
        go body(...)
```

```
    }
    ...
}

func main2() {
    cpus := runtime.NumCPU()

    go runInParallel(func(...) {
        ...
    }, ...)

    ...
    fmt.Printf("%.6f", pi)
}
```

Level 4 - Unit Testing

- Schreiben Sie mindestens einen sinnvollen Unit Test (Kurzanleitung[20]) für die Methode zur Parallelisierung von Funktionen (im obigen Beispiel `runInParallel`).

[20] https://gobyexample.com/testing

Übung 3: ToDo List API

Anforderung

Ihre Aufgabe ist die Implementierung einer RESTful Web API **mit Go**.

Die technische Spezifikation der Web API finden Sie im *Open API* Format in *api-spec.yaml*[21]. Hinweis: Sie können die Spezifikation in einem angenehm lesbaren Format ansehen, indem Sie https://editor.swagger.io/ im Browser öffnen und den Inhalt der Datei *api-spec.yaml*[22] in den Eingabebereich auf der linken Seite kopieren.

Lesen Sie die API Spezifikation genau. Auch scheinbare Kleinigkeiten wie zum Beispiel geforderte *Response Codes* (z.B. *404* für *Not Found*, *201* für *Created*) sind wichtig und sollten beachtet werden.

Levels

Je nach Vorwissen wird diese Aufgabe manchen schwerer und manchen leichter fallen. Daher hier Anregungen, wie man Schritt für Schritt das Beispiel lösen könnte. Jeder kann basierend auf der bestehenden Programmierpraxis die Anzahl an Levels abarbeiten, die für sie oder ihn passen.

Level 1 - *Get Items*

- Befüllen Sie die Todo-Liste im Hauptspeicher im Programm mit ein paar Beispieldatensätzen
- Implementieren Sie den Basis-Webserver und die Operation *getItems* (`GET /api/toDoItems`)

Level 2 - *Get Single Item*

- Erweitern Sie die API um eine Funktion zum Abfragen eines einzelnen Items (*getItem* (`GET /api/toDoItems/{id}`))

Level 3 - *Add Items*

- Erweitern Sie die API um eine Funktion zum Hinzufügen von Items (*addItem* (`POST /api/toDoItems`))

[21] https://github.com/rstropek/golang-samples/blob/master/labs/todo-api/api-spec.yaml
[22] https://github.com/rstropek/golang-samples/blob/master/labs/todo-api/api-spec.yaml

Level 4 - *Nested Package*

- Gliedern Sie die Geschäftslogik zum Verwalten der Todo-Liste in ein eigenes *nested package* aus und trennen Sie diesen Code damit vom Code der Web API.
- Schreiben Sie zumindest drei sinnvolle Unit Tests für das Package.

Level 5 - Docker

- Schreiben Sie ein Dockerfile zum Erstellen eines Docker Image für die API (`docker build`)
- Testen Sie Ihre API in einem Docker Container (`docker run`)
- Veröffentlichen Sie das Docker Image am *Docker Hub* (`docker push`)

Testfälle

ToDo Items abfragen Request:

```
GET http://localhost:8080/api/toDoItems
```

ToDo Item einfügen Request:

```
POST http://localhost:8080/api/toDoItems
Content-Type: text/plain

Einkaufen
```

Einzelnes ToDo Item abfragen Request (ID muss durch eigenen Wert ersetzt werden):

```
GET http://localhost:8080/api/toDoItems/MXdzY2yO5rnZoZeg
```

ToDo Item auf erledigt setzen Request (ID muss durch eigenen Wert ersetzt werden):

```
POST http://localhost:8080/api/toDoItems/MXdzY2yO5rnZoZeg/setDone
```

Übung 4: ToDo List API mit Go Kit

Anforderung

Auf GitHub[23] finden Sie ein einfaches Beispiel für eine Web API zum Verwalten von Personen:

- Go Sourcecode[24]
- Beispielanfragen[25]

Ihre Aufgabe ist es, diese Web API mit Go Kit umzusetzen. Als Grundlage können Sie dafür das besprochene Go Kit[26] Beispiel verwenden.

Levels

Je nach Vorwissen wird diese Aufgabe manchen schwerer und manchen leichter fallen. Daher hier Anregungen, wie man Schritt für Schritt das Beispiel lösen könnte. Jeder kann basierend auf der bestehenden Programmierpraxis die Anzahl an Levels abarbeiten, die für sie oder ihn passen.

Level 1 - *Service für Geschäftslogik*

- Extrahieren Sie die Geschäftslogik zum Verwalten der Personen in ein Go Kit Service[27].
- Wenn Sie sich den Code des Ausgangsservice ansehen, werden Sie sehen, dass er keine Mutexe zur Steuerung des gleichzeitigen Zugriffs enthält. Korrigieren Sie das im Rahmen der Serviceentwicklung.

Level 2 - *Endpoints* und *Transports* für `GetPerson`

- Entwickeln Sie einen Go Kit Endpoint[28] und Transport[29] für die Operation *Get Person*.
- Schreiben Sie eine *Main* Konsolenanwendung zum Betrieb des Go Kit Microservice.

Level 3 - *Endpoints* und *Transports* vervollständigen

- Fügen Sie Endpoints und Transports für die restlichen Operationen (*Get People*, *Create Person*, *Delete Person*) hinzu.

[23] https://github.com/rstropek/golang-samples/tree/master/go-microservices/05-simple-web-api
[24] https://github.com/rstropek/golang-samples/blob/master/go-microservices/05-simple-web-api/web.go
[25] https://github.com/rstropek/golang-samples/blob/master/go-microservices/05-simple-web-api/demo.http
[26] https://github.com/rstropek/golang-samples/tree/master/go-kit
[27] https://gokit.io/faq/#services-mdash-what-is-a-go-kit-service
[28] https://gokit.io/faq/#endpoints-mdash-what-are-go-kit-endpoints
[29] https://gokit.io/faq/#transports-mdash-what-are-go-kit-transports

Level 4 - *Logging Middleware*

- Fügen Sie eine *Logging Middleware*[30] hinzu.

Level 5 - Docker

- Schreiben Sie ein Dockerfile zum Erstellen eines Docker Image für die API (`docker build`)
- Testen Sie Ihre API in einem Docker Container (`docker run`)
- Veröffentlichen Sie das Docker Image am *Docker Hub* (`docker push`)

[30] https://gokit.io/faq/#middlewares-mdash-what-are-middlewares-in-go-kit