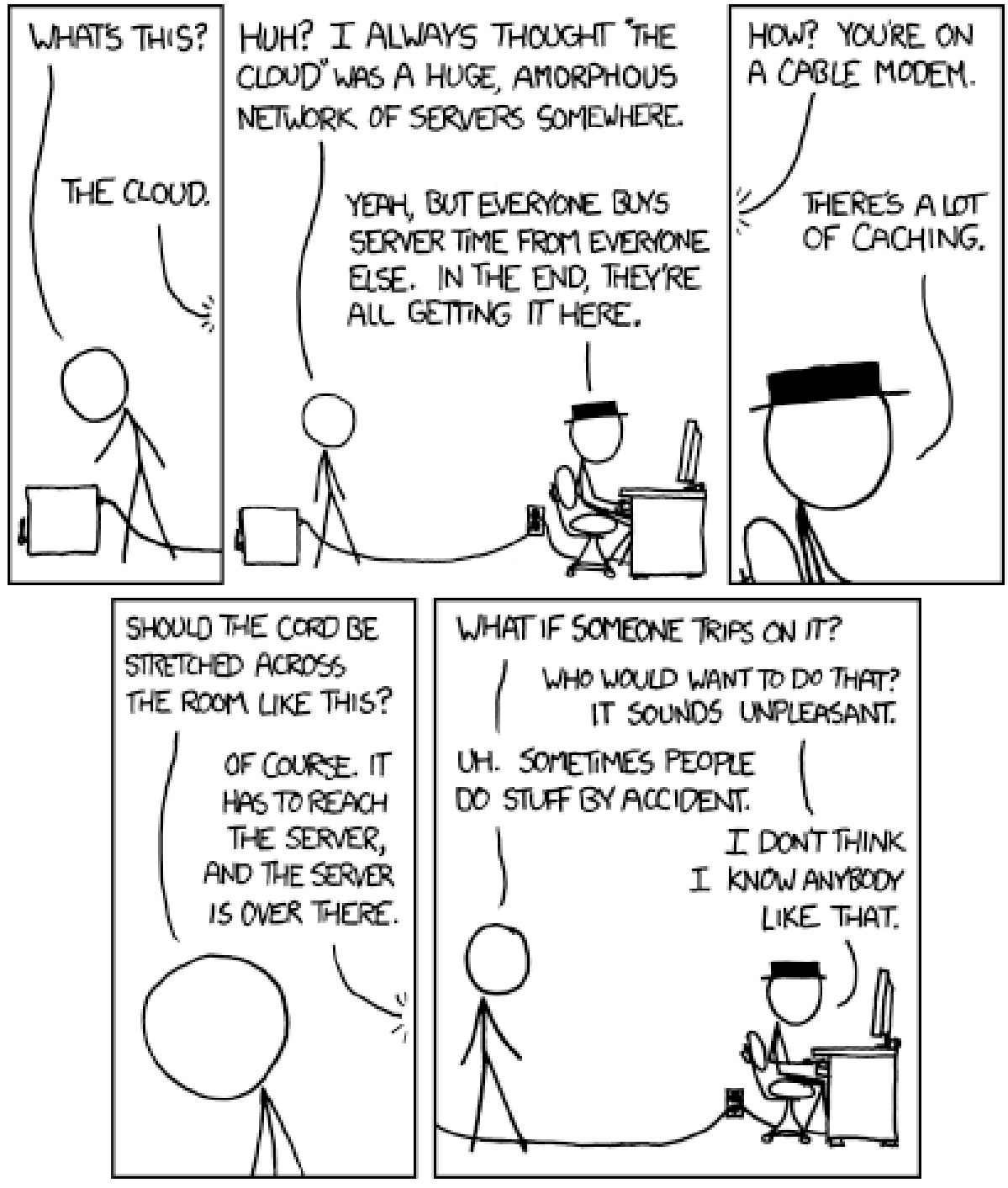
WHAT'S THIS?
THE CLOUD.
HUH? I ALWAYS THOUGHT "THE CLOUD" WAS A HUGE, AMORPHOUS NETWORK OF SERVERS SOMEWHERE.
YEAH, BUT EVERYONE BUYS SERVER TIME FROM EVERYONE ELSE. IN THE END, THEY'RE ALL GETTING IT HERE.
HOW? YOU'RE ON A CABLE MODEM.
THERE'S A LOT OF CACHING.
SHOULD THE CORD BE STRETCHED ACROSS THE ROOM LIKE THIS?
OF COURSE. IT HAS TO REACH THE SERVER, AND THE SERVER IS OVER THERE.
WHAT IF SOMEONE TRIPS ON IT?
WHO WOULD WANT TO DO THAT? IT SOUNDS UNPLEASANT.
UH. SOMETIMES PEOPLE DO STUFF BY ACCIDENT.
I DON'T THINK I KNOW ANYBODY LIKE THAT.

Google App Engine

und

Go

Michael Stapelberg

2012-05-25

powered by LaTeX

Inhalt

- Übersicht: Google App Engine
- Einführung in Go
- Einführung in Google App Engine

Google App Engine

- Platform as a Service (PaaS), genauer: Plattform für Webapps
- Skaliert automatisch mit der Anzahl an Requests
- 99.95% Uptime SLA
  (4 Stunden Ausfall pro Jahr)
- Apps in Python, Java oder Go;
  eigene APIs

Vorteile

- Konsistente APIs, gute Apps schnell umsetzbar
- Keine Gedanken mehr an Hosting, Scaling, Capacity Planning, Monitoring, Security* verschwenden
- (Nur?) soviel zahlen wie man wirklich nutzt

Nachteile

- Vendor lock-in (FOSS-Alternativen existieren, aber…)
- Nicht für alle Apps (gut) geeignet

Go

- Programmiersprache, welche die Effizienz kompilierter Sprachen mit der Leichtigkeit dynamischer Sprachen vereinen will
- Besonders gute Unterstützung für Concurrency
- Schnelles Kompilieren, keine Makefiles
- Garbage Collection
- Unicode-Unterstützung, Arrays/Maps, HTTP/JSON/Crypto/… in der stdlib

Hello World

```
package main

import "fmt"

func main() {
    fmt.Println("Hallöchen, SEM…")
}
```

Variablen/Typen

```
// Äquivalent (wegen type inference)
var foo string = "ohai"
foo := "ohai"

// Array
weekdays := []string{"Mo", "Di"}

// Map
klausurpunkte := make(map[string]int)
klausurpunkte["Michael Stapelberg"] = 0
klausurpunkte["Sven Schönung"] = 15
```

Zuweisungen

```
weekdays := []string{"Mo", "Di"}

for index, value := range(weekdays) {
    fmt.Printf("Der %d. Wochentag ist %s\n", index, value)
}

for _, value := range(weekdays) {
    fmt.Printf("%s ist ein Wochentag\n", value)
}
```

Fehlerbehandlung

```
// Fehler prüfen und reagieren
file, err := os.Open("funnycat.jpg")
if err != nil {
    fmt.Printf("Konnte Bild nicht öffnen: %v\n", err)
    os.Exit(1)
}

// Fehler ignorieren (gelegentlich sinnvoll)
file, _ := os.Open("funnycat.jpg")
```

Typen (Deklaration)

```
type SizeIndex struct {
    filename string
    Index map[string]int64
}

var idx SizeIndex
idx.Index = make(map[string][]byte)
idx.Index["/home/michael/sem.pdf"] = 934821
```

Typen (Methoden)

```
func (idx *SizeIndex) Save() error {
    file, err := os.Open(idx.filename)
    if err != nil {
        return err
    }
    defer file.Close()

    encoder := gob.NewEncoder(file)
    if err := encoder.Encode(idx); err != nil {
        return err
    }

    return nil
}
```

Goroutinen

```
func IsReady(what string, duration time.Duration) {
    time.Sleep(duration)
    fmt.Printf("%s is ready!\n", what)
}

func main() {
    go IsReady("tea", 6 * time.Second)
    go IsReady("coffee", 2 * time.Second)
    fmt.Println("waiting…")
    time.Sleep(10 * time.Second)
}
```

Channels

```
func cacheFlusher(cacheChan chan string) {
    for {
        select {
            case url := <-cacheChan:
            fmt.Printf("read URL %s\n", url)
            case <-time.After(10 * time.Second):
            fmt.Println("writing cache to disk")
        }
    }
}

cacheChan := make(chan string)
go cacheFlusher(cacheChan)
for {
    fmt.Scanf("%s", &input)
    cacheChan <- input
}
```

Beispiel: Website-Download mit Timeout

```
result := make(chan *http.Response, 1)

go func() {
    resp, _ := http.Get("http://www.hs-mannheim.de/")
    result <- resp
}()

go func() {
    time.Sleep(5 * time.Second)
    result <- nil
}()

if resp := <-result; resp != nil {
    fmt.Printf("HTTP %d\n", resp.StatusCode)
}
```

Google App Engine

Quellen/Links

- http://en.wikipedia.org/wiki/Google_AppEngine
- http://en.wikipedia.org/wiki/Go_(programming_language)
- http://www.golang.org/
- http://tour.golang.org/ (bzw. http://play.golang.org/