Cincinnati Go Meetup

An Introduction of the Go programming language

Who am I?

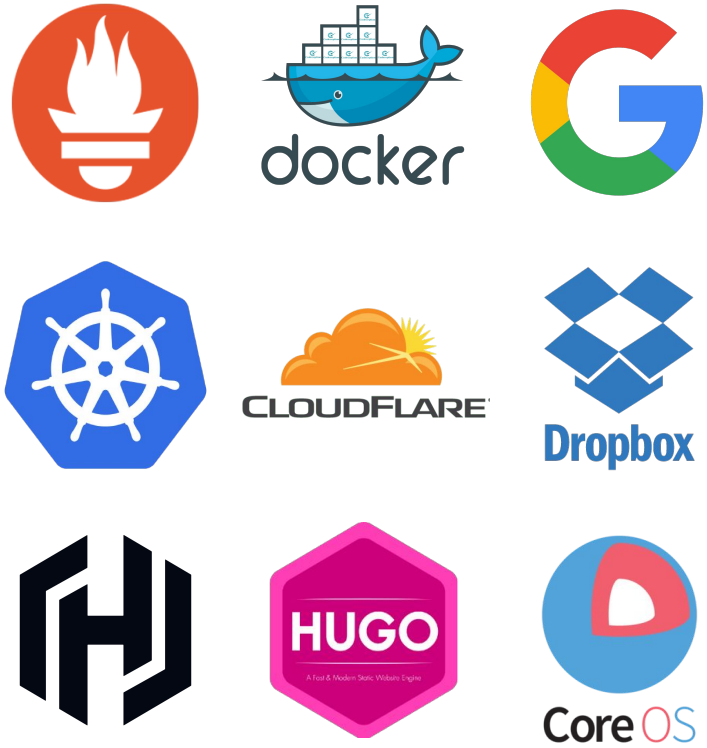
Shad Beard
Infrastructure Engineer @ Losant
Twitter: dahs81

The Business Case

Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.

- Cross Platform
- Designed for Concurrency
- Rapid Growth (TIOBE)
- Fast/Efficient (Cost effective)
- Low Level, but Simple
- Developer Friendly

Go in Production



… And many more

Quick Start

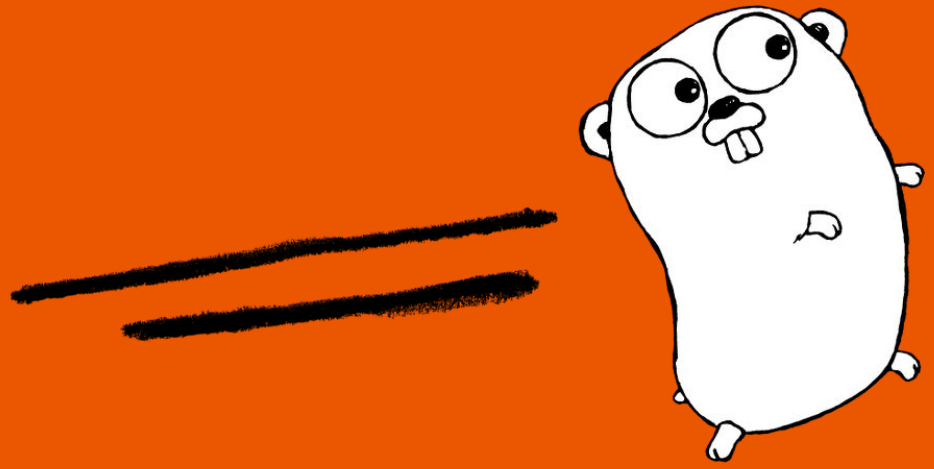
- Download Go from https://golang.org/dl/
- Click the installer and follow the directions
- *mkdir $HOME/go && export GOPATH=$HOME/go*
- Create bin, pkg, src directories in your GOPATH
- Start Coding!

```go
package main

import "fmt"

func main() {
    city := "Cincinnati"

    fmt.Printf("Hello, %s!\n", city)
}
```

```
root@e8c9bc07128b:/# go run main.go
```

5 Features I Love

1. Simplicity
2. Single Binary
3. Tools
4. Interfaces
5. Concurrency



Simplicity

01

Proverb:

Clear is better than Clever.

1. Designed with Readability in Mind
2. Less Features - Less Complexity
3. Built for Teams

Simplicity Is Complicated

Single Binary

02

Build for any operating system on any architecture.

How to create a binary for any system.

- GOOS
- GOARCH
- go build
- go install

Really great for CLI's

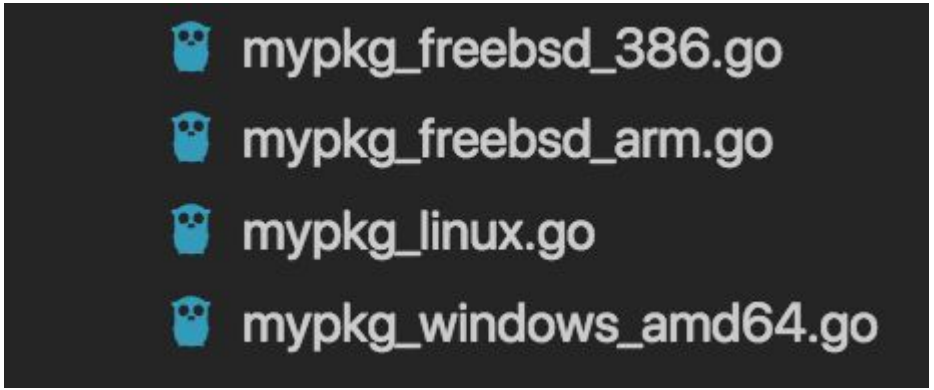
Build from command line:

```
t@e8c9bc07128b:/# GOOS=windows GOARCH=amd64 go build
```

Build using +build at top of file - (Linux and 386) OR (darwin and NOT cgo):

```
// +build linux,386 darwin,!cgo
```

Building using filename extensions:



Tooling

03

- gofmt
- go tool <option>
- go run -race
- pprof
- go-wrk
- godoc
- go vet
- go install
- go-torch
- go imports
- go list -f
- go guru
- go test

godoc/go list

> go doc

> go doc fmt

> go list -f ‘{{ .Doc }}’ fmt

> go list -f ‘{{ join .Imports “\n” }}’ fmt

> godoc -http :6060 (Go to http://localhost:6060 -> Click on Packages)

Interfaces

04

Proverb:

The bigger the interface, the weaker the abstraction.

```go
package main

import "fmt"

func main() {
    goEng := GolangEngineer{
        Name:                "Rob Pike",
        IsWearingHeadphones: true,
    }

    if goEng.IsCoding() == true {
        fmt.Printf("%s is currently writing %s code. Leave them be!", goEng.Name, goEng.PreferredLanguage())
    } else {
        fmt.Printf("It's ok to talk to %s now.", goEng.Name)
    }
}

type SoftwareEngineer interface {
    IsCoding() bool
    PreferredLanguage() string
}

type GolangEngineer struct {
    Name                string
    IsWearingHeadphones bool
}

func (ge *GolangEngineer) IsCoding() bool {
    if ge.IsWearingHeadphones == true {
        return true
    }
    return false
}

func (ge *GolangEngineer) PreferredLanguage() string {
    return "Golang"
}
```

Interface Composition

Interfaces can be composed of other interfaces.

This allows you to extend functionality without breaking existing code.

```go
type Writer interface {
    Write([]byte) (int, error)
}

type Closer interface {
    Close() error
}

type WriterCloser interface {
    Writer
    Closer
}
```

Custom io.Writer

This creates a type ReverseWriter that implements the Writer interface and reverses a []bytes, then outputs it to Stdout.

```go
type ReverseWriter struct{}

func (rw ReverseWriter) Write(b []byte) (int, error) {
    s := string(b)
    s = Reverse(s)

    n, err := os.Stdout.Write([]byte(s))

    return n, err
}

func Reverse(s string) (result string) {
    for _, v := range s {
        result = string(v) + result
    }
    return
}
```

Use Custom Writer

Create the custom Writer. Open a file on disk and copy the file to the ReverseWriter. This will output the reversed string to Stdout.

```go
func main() {
    var w Writer = ReverseWriter{}

    file, err := os.Open("./normal.txt")
    if err != nil {
        log.Fatal("Cannot create file", err)
    }
    defer file.Close()

    // io.Copy takes an io.Writer and io.Reader, so when w is passed to
    // to Copy, the string is reversed.
    // Because Copy accepts a Writer and Reader, we can replace either
    // with anything that implements io.Writer and io.Reader (ex: os.Stdout)
    // _, err = io.Copy(os.Stdout, file)
    _, err = io.Copy(w, file)
    if err != nil {
        log.Fatal(err)
    }
}
```

Concurrency

05

Proverb:

Don’t communicate by sharing memory, share memory by communicating.

Goroutines
Channels

goroutines

A lightweight abstraction over unix threads. Sometimes called green threads in other languages.

Uses a scheduler to map goroutines to unix threads.

```go
func main() {
    wg := sync.WaitGroup{}

    wg.Add(1)

    go func() {
        fmt.Println("Hello, Cincinnati!")
        wg.Done()
    }()

    wg.Wait()
}
```

channel basics

Channels allow you to synchronize data between different goroutines.

Both, sender and receiver blocks until the other is ready.

```go
func main() {
    intChan := make(chan int)
    wg := sync.WaitGroup{}
    wg.Add(2)

    go func() {
        intChan <- 7
        wg.Done()
    }()

    go func() {
        value := <-intChan
        fmt.Printf("Value received on intChan: %v\n", value)
        wg.Done()
    }()

    wg.Wait()
}
```

Without channels

In this example, we loop through the slice of urls one at a time.

```go
func main() {
    urls := []string{
        "https://jsonplaceholder.typicode.com/posts/1",
        "https://jsonplaceholder.typicode.com/posts/1/comments",
    }

    for _, url := range urls {
        httpResponse, _ := http.Get(url)
        fmt.Println(httpResponse.Status)
    }
}
```

With channels

In this example, we are using channels with goroutines to fetch 2 urls in parallel.

```go
func main() {
    var ch chan HTTPResponse = make(chan HTTPResponse)
    urls := []string{
        "https://jsonplaceholder.typicode.com/posts/1",
        "https://jsonplaceholder.typicode.com/posts/1/comments",
    }
    for _, url := range urls {
        go DoHTTPGet(url, ch)
    }

    for range urls {
        fmt.Println((<-ch).status)
    }
}

func DoHTTPGet(url string, ch chan<- HTTPResponse) {
    httpResponse, _ := http.Get(url)
    httpBody, _ := ioutil.ReadAll(httpResponse.Body)
    ch <- HTTPResponse{httpResponse.Status, httpBody}
}

type HTTPResponse struct {
    status string
    body   []byte
}
```

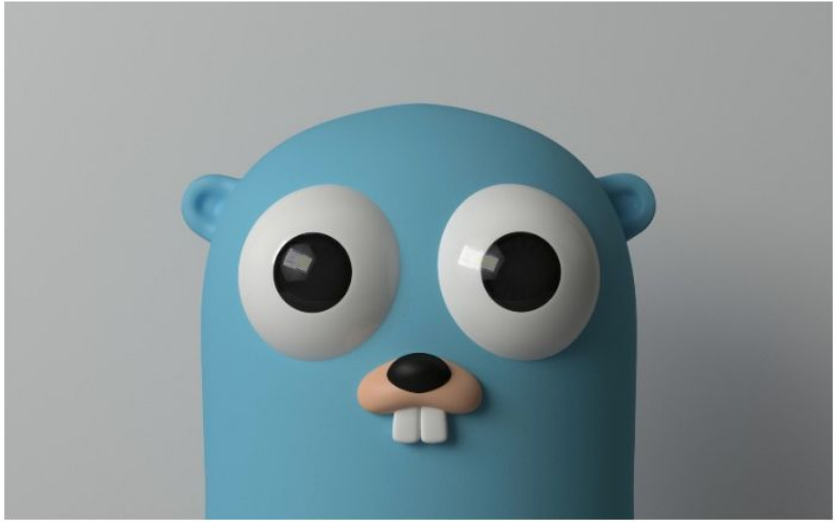
This was using only 4 ec2.c4.large instances. With reserved instances, this would cost less than $200/month.

~43.8B events/month for $200

Marcio Castilho Follow

CEO of SMS Junk Filter and Chief Architect Officer of KnowBe4.

Aug 30, 2017 · 7 min read



Handling 1 Million Requests per Minute with Golang

Demo

The code for this demo can be found here:

https://github.com/dahs81/QPC

Other Features

- First-Class Functions
- Closures
- Strongly Typed
- Compiled
- Powerful stdlib
- Testing support

Resources

https://dave.cheney.net/

https://www.ardanlabs.com/blog/

https://github.com/ardanlabs/gotraining

A Tour of Go

https://play.golang.org/

https://medium.com/smsjunk/handling-1-million-requests-per-minute-with-golang-f70ac505fcaa

Falling Forward (Youtube)

Just For Func

Understanding Interfaces

Simplicity Is Complicated

https://www.ardanlabs.com/blog/2015/09/composition-with-go.html

https://github.com/uber/go-torch

https://www.youtube.com/watch?v=ySy3sR1LFCQ&index=8&list=PL64wiCrrxh4Jisi7OcCJIUpguV_f5jGnZ&t=0s

https://www.youtube.com/watch?v=uBjoTxosSys

Go Proverbs

https://www.youtube.com/watch?v=29LLRKIL_TI

Questions?