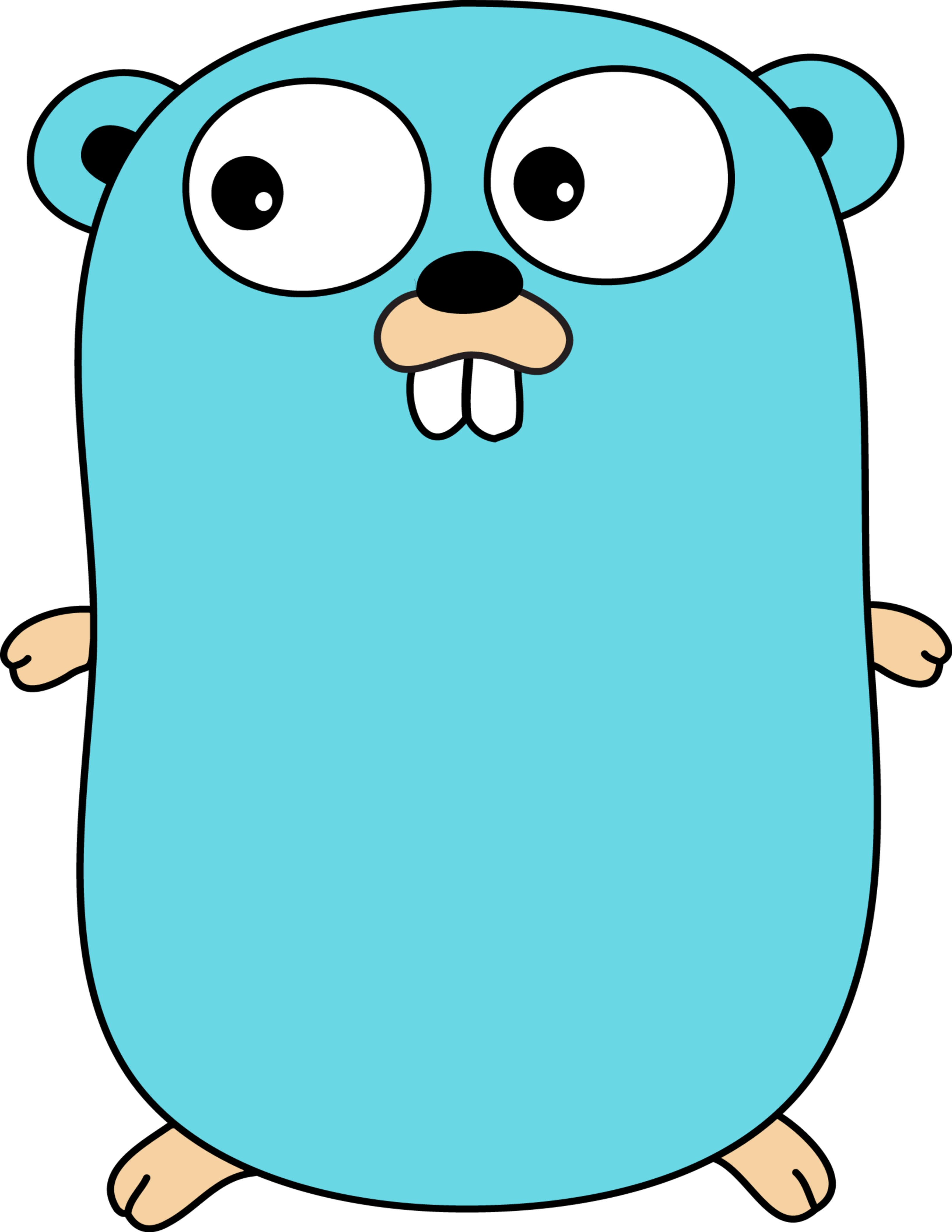


How Do I Write And Deploy Simple Web Apps With Go?

Satish Talim

This book is for sale at http://leanpub.com/howdoibuildawebappwithgo

This version was published on 2017-05-28

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2013 - 2017 Satish Talim

Tweet This Book!

Please help Satish Talim by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just bought ”How Do I Build Web Apps With Go?” ebook #programming #golang #gowebapps

The suggested hashtag for this book is #gowebapps.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#gowebapps

Contents

Preface . . . . **i**
Who is the eBook for? i
Acknowledgements i
How to Use This eBook i
Using Code Examples i
Getting the Code ii
How to Contact Me ii
Thanks ii

1. Deploying Go Web Apps to Heroku **1**
1.1 Cloud Computing Service Levels 1
1.1.1 SaaS 1
1.1.2 IaaS 2
1.1.3 PaaS 2
1.2 Create an account on Heroku 2
1.3 Install the Heroku Toolbelt 3
1.4 Prepare a web app 4
1.5 Use Git 5
1.6 Create a Procfile 5
1.7 Install Godep 5
1.8 Declare app dependencies 5
1.9 Using godep with our project 6
1.10 Add these new files to git 6
1.11 Deploy the app 6
1.12 Program gomongohq.go 7

Preface

Go[1] is an open source programming language that makes it easy to build simple, reliable, and efficient software.

Who is the eBook for?

This short eBook will introduce you to building your own web applications and hosting it on the Internet. It is targetted towards Go programming newbies who have:

- an understanding of basic web technologies HTTP[2],
- an understanding of HTML[3],
- a *working knowledge* of Go[4] programming (to try out the programs in this eBook, you should have a working copy of Go 1.5 on your computer),
- an understanding of JSON[5].

Acknowledgements

There are a good number of people who deserve thanks for their help and support they provided, either while or before this eBook was written, and there are still others whose help will come after the eBook is released. I would like to thank my "gang" of mentors at RubyLearning[6], Doug Sparling and Sanat Gersappa for their help in making this eBook far better than I could have done alone.

How to Use This eBook

I recommend that you go through the entire eBook chapter by chapter, reading the text and running the sample programs. There are no large applications in this eBook – just small, self-contained sample programs. This will give you a much broader understanding of how things are done (and of how you can get things done), and it will reduce the chance of anxiety, confusion, and worse yet, mistakes.

Using Code Examples

All of the code in this eBook can be used pretty much anywhere and anyhow you please.

[1] http://golang.org/
[2] https://github.com/Unknwon/build-web-application-with-golang_EN/blob/master/eBook/03.1.md
[3] http://msdn.microsoft.com/en-us/hh549253.aspx
[4] http://golang.org/
[5] http://en.wikipedia.org/wiki/JSON
[6] http://rubylearning.org/

Getting the Code

You can get a `.zip` or `.tar` archive of the code by going to GitHub Repo[7] and clicking on the “Download” button.

How to Contact Me

I can be reached via e-mail at satish@rubylearning.org. Please contact me if you have any questions, comments, kudos or criticism on the eBook. Constructive criticism is definitely appreciated; I want this eBook to get better through your feedback.

Thanks

Thanks for buying and checking out this eBook. As part of the lean publishing philosophy, you’ll be able to interact with me as the eBook is completed. I’ll be able to change things, reorganize parts, and generally make a better eBook. I hope you enjoy.

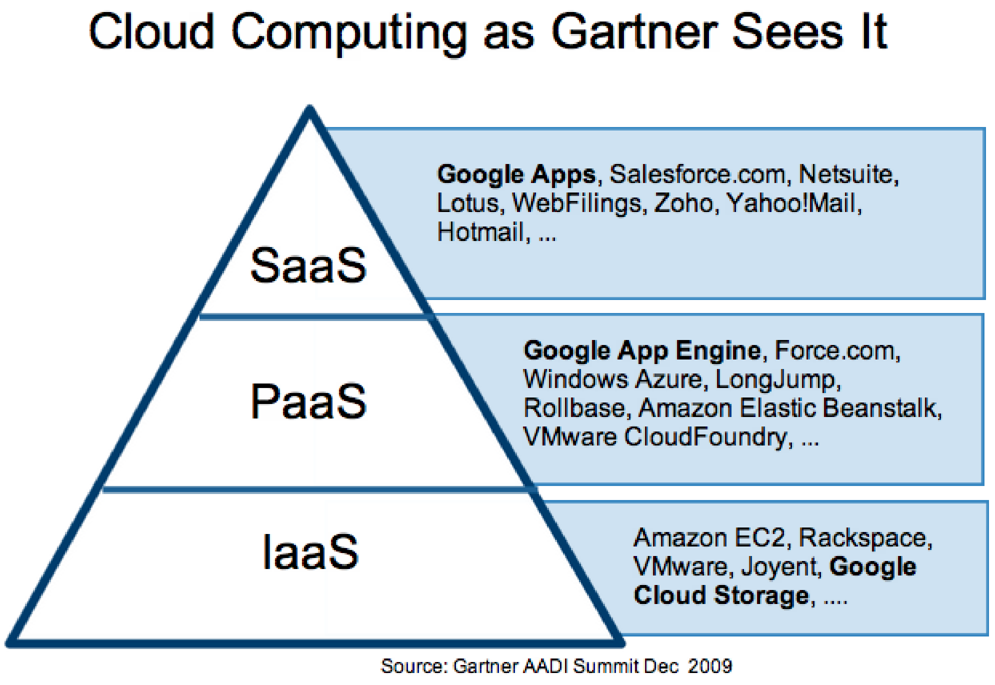
[7] https://github.com/SatishTalim/bwpwg

1. Deploying Go Web Apps to Heroku

There are plenty of definitions for "cloud computing" online, and for the most part, they generally point to the same thing: taking applications and running them on infrastructure other than your own. Companies or individuals who offload or effectively "outsource" their hardware and/or applications are running those apps "in the cloud."

1.1 Cloud Computing Service Levels

In the figure below, you can see how the analyst firm Gartner segregates cloud computing into three distinct classes of service.


Cloud Computing as Gartner Sees It
SaaS
Google Apps, Salesforce.com, Netsuite, Lotus, WebFilings, Zoho, Yahoo!Mail, Hotmail, ...
PaaS
Google App Engine, Force.com, Windows Azure, LongJump, Rollbase, Amazon Elastic Beanstalk, VMware CloudFoundry, ...
IaaS
Amazon EC2, Rackspace, VMware, Joyent, Google Cloud Storage, ...
Source: Gartner AADI Summit Dec 2009

Cloud Computing Service Levels

1.1.1 SaaS

Let's start at the highest level: software applications that are only available online fall into the "Software-as-a-Service" category, also known as "SaaS". The simplest example to understand is e-mail. For personal e-mail, people typically select from a variety of free web-based e-mail servers such as Google's Gmail, Yahoo!Mail, or Microsoft's Hotmail, rather than setting up all of the above through their provider. Not only is it "free" (supported through advertising), but users are freed from any additional server maintenance. Because these applications run (and store their data online), users no longer need to worry about managing, saving, and backing up their files. Of course, now it becomes Google's responsibility to ensure that your data is safe and secure. Other examples of SaaS include Salesforce, IBM's NetSuite, and online games.

1.1.2 IaaS

On the opposite end of the spectrum, we have "Infrastructure-as-a-Service," or "IaaS," where you outsource the hardware. In such cases, it's not just the computing power that you rent; it also includes power, cooling, and networking. Furthermore, it's more than likely that you'll need storage as well. Generally IaaS is this combination of compute and cloud storage.

When you choose to run your applications at this cloud service level, you're responsible for everything on the stack that is required to operate above it. By this, we mean necessities such as the operating system followed by additional (yet optional services) like database servers, web servers, load-balancing, monitoring, reporting, logging, middleware, etc. Furthermore, you're responsible for all hardware and software upgrades, patches, security fixes, and licensing, any of which can affect your application's software stack in a major way.

1.1.3 PaaS

In the middle, we have "Platform-as-a-Service," or "PaaS." At this service level, the vendor takes care of the underlying infrastructure for you, giving you only a platform with which to (build and) host your application(s). Gone are the hardware concerns of IaaS, yet with PaaS, you control the application — it's your code — unlike as the SaaS level where you're dependent on the cloud software vendor. The only thing you have to worry about is your application itself.

Systems like **Google App Engine**, Salesforce's **Heroku** and force.com, Microsoft Azure, and VMwares Cloud Foundry, all fall under the PaaS umbrella.

A number of Platform-as-a-Service (PaaS) providers[1] allow you to use Go applications on their clouds.

Heroku[2] is a new approach to deploying web applications[3]. Forget about servers; the fundamental unit is the app. Develop locally on your machine just like you always do. When you're ready to deploy, use the Heroku client gem to create your application in their cloud, then deploy with a single git push. Heroku has full support for Go applications.

We shall soon see how we can deploy an app to Heroku.

1.2 Create an account on Heroku

Please ensure that you are connected to the internet and then create an account on Heroku (obviously do this only once). If you don't have one, then signup[4]. It's free and instant. A free account can have up to 5 apps without registering your credit card.

[1] https://code.google.com/p/go-wiki/wiki/ProviderIntegration
[2] http://heroku.com/
[3] https://devcenter.heroku.com/articles/getting-started-with-go#introduction
[4] http://heroku.com/signup

1.3 Install the Heroku Toolbelt

The Heroku Toolbelt[5] provides you access to the Heroku Command Line Interface (CLI). Once installed, you'll have access to the `heroku` command from your command window. Log in using the email address and password you used when creating your Heroku account:

```
$ heroku login
Installing Heroku Toolbelt v4... done.
For more information on Toolbelt v4: https://github.com/heroku/heroku-cli
Setting up node-v4.1.1...done
Installing core plugins heroku-apps, heroku-fork, heroku-git, heroku-local, hero
ku-run, heroku-status... done
Enter your Heroku credentials.
Email: satish@rubyconfindia.org
Password (typing will be hidden):
Logged in as satish@rubyconfindia.org
```

[5] https://toolbelt.heroku.com

1.4 Prepare a web app

In this step, you will prepare a simple Go application that can be deployed.

Create a folder `webapphr` under the folder `$GOPATH/src/github.com/SatishTalim/` and write the program `webapphr.go` in the folder `webapphr` as follows:

Program webapphr.go

```go
package main

import (
	"fmt"
	"log"
	"net/http"
	"os"
)

func main() {
	http.HandleFunc("/", handler)
	fmt.Println("listening...")
	err := http.ListenAndServe(GetPort(), nil)
	if err != nil {
		log.Fatal("ListenAndServe: ", err)
	}
}

func handler(w http.ResponseWriter, r *http.Request) {
	fmt.Fprintf(w, "Hello. This is our first Go web app on Heroku!")
}

// Get the Port from the environment so we can run on Heroku
func GetPort() string {
	var port = os.Getenv("PORT")
	// Set a default port if there is nothing in the environment
	if port == "" {
		port = "4747"
		fmt.Println("INFO: No PORT environment variable detected, defaulting to " + port)
	}
	return ":" + port
}
```

1.5 Use Git

In order to deploy to Heroku we'll need the app stored in Git. In the same folder i.e. `$GOPATH/src/github.com/SatishTalim/webapphr` type:

```
$ git init
$ git add -A .
$ git commit -m "code"
```

1.6 Create a Procfile

Use a `Procfile`, a text file in the root directory of your application (`$GOPATH/src/github.com/SatishTalim/webapphr`), to explicitly declare what command should be executed to start your app.

The `Procfile` looks like this:

```
web: webapphr
```

This declares a single process type, `web`, and the command needed to run it. The name `web` is important here. It declares that this process type will be attached to the HTTP routing stack of Heroku, and receive web traffic when deployed.

1.7 Install Godep

The recommended way to manage Go package dependencies on Heroku is with Godep[6], which helps build applications reproducibly by fixing their dependencies.

Let us install `Godep`:

```
$ go get github.com/tools/godep
```

1.8 Declare app dependencies

Heroku recognizes an app as a Go app by the existence of a `Godeps.json` file in the `Godeps` directory located in your application's root directory (`$GOPATH/src/github.com/SatishTalim/webapphr`).

The `Godeps/Godeps.json` file is used by `Godep` and specifies both the dependencies that are vendored with your application and the version of Go that should be used to compile the application.

When an app is deployed, Heroku reads this file, installs the appropriate Go version and compiles your code using `godep go install ./…`.

[6]https://github.com/tools/godep

1.9 Using godep with our project

In the folder `$GOPATH/src/github.com/SatishTalim/webapphr` type:

```
$ godep save -r
```

This will save a list of dependencies to the file `Godeps/Godeps.json`.

Note: Read the contents of `Godeps/_workspace` and make sure it looks reasonable. `Godep` does **not copy** files from source repositories that are not tracked in version control. Then commit the whole `Godeps` directory to version control, including `Godeps/_workspace`.

1.10 Add these new files to git

```
$ git add -A .
$ git commit -m "dependencies"
```

Now we're ready to ship this to Heroku.

1.11 Deploy the app

In this step, you will deploy the app to Heroku.

Create an app on Heroku, which prepares Heroku to receive your source code.

```
$ heroku create
Creating thawing-harbor-9085… done, stack is cedar-14
https://thawing-harbor-9085.herokuapp.com/ | https://git.heroku.com/thawing-harbor-9085.git
Git remote heroku added
```

When you create an app, a git remote (called `heroku`) is also created and associated with your local git repository.

Heroku generates a random name (in this case `thawing-harbor-9085`) for your app, or you can pass a parameter to specify your own app name.

Now deploy your code:

```
$ git push heroku master
Counting objects: 11, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (11/11), 1.28 KiB | 0 bytes/s, done.
Total 11 (delta 0), reused 0 (delta 0)
remote: Compressing source files… done.
remote: Building source:
remote:
remote: – – -> Go app detected
remote: – – -> Checking Godeps/Godeps.json file.
remote: – – -> Installing go1.5.1… done
remote: – – -> Running: godep go install -tags heroku ./…
remote: – – -> Discovering process types
remote: Procfile declares types -> web
remote:
remote: – – -> Compressing… done, 1.9MB
remote: – – -> Launching… done, v3
remote: https://thawing-harbor-9085.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy…. done.
To https://git.heroku.com/thawing-harbor-9085.git
 * [new branch] master -> master
```

The application is now deployed.

Visit the app at the URL generated by its app name. As a handy shortcut, you can open the website as follows:

```
$ heroku open
```

That's it—you now have a running Go app on Heroku!

Exercises

Deploy the apps `webtime.go`, `dosasite.go`, `ipweb.go`, `stringupper.go`, `geoweb.go` and `trails.go` that you had written previously to Heroku.

1.12 Program gomongohq.go

Previously we had written the program `mongohqconnect.go`. We shall modify that program and store the modified version in the file `gomongohql.go`. We shall host `gomongohql.go` on Heroku; connect it to our database `godata` hosted on MongoLab and fetch information (the email id of say user Stefan Klaste) from the collection `user`.

Program gomongohql.go

```go
package main

import (
	"fmt"
	"html/template"
	"labix.org/v2/mgo"
	"labix.org/v2/mgo/bson"
	"log"
	"net/http"
	"os"
)

type Person struct {
	Name string
	Email string
}

func main() {
	http.HandleFunc("/", root)
	http.HandleFunc("/display", display)
	fmt.Println("listening...")
	err := http.ListenAndServe(GetPort(), nil)
	if err != nil {
		log.Fatal("ListenAndServe: ", err)
		return
	}
}

func root(w http.ResponseWriter, r *http.Request) {
	fmt.Fprint(w, rootForm)
}

const rootForm = `
  <!DOCTYPE html>
    <html>
      <head>
        <meta charset="utf-8">
        <title>Your details</title>
        <link rel="stylesheet" href="http://yui.yahooapis.com/pure/0.4.2/pure-min.css">
      </head>
      <body style="margin: 20px;">
        <h2>A Fun Go App on Heroku to access MongoDB on MongoLab</h2>
        <p>This simple app will fetch the email id of a person, if it's already there in t\
he MongoDB database.</p>
```

```
        <p>Please enter a name (example: Stefan Klaste)</p>
        <form action="/display" method="post" accept-charset="utf-8" class="pure-form">
          <input type="text" name="name" placeholder="name" />
          <input type="submit" value=".. and query database!" class="pure-button pure-butt\
on-primary"/>
        </form>
        <div>
          <p><b>&copy; 2015 Go Challenge. All rights reserved.</b></p>
        </div>
      </body>
    </html>
`

var displayTemplate = template.Must(template.New("display").Parse(displayTemplateHTML))

func display(w http.ResponseWriter, r *http.Request) {
        // In the open command window set the following for Heroku.
        // Remember to use your login/password in the string below
        // heroku config:set MONGOLAB_URL=mongodb://IndianGuru:dbpassword@ds051523.mongola\
b.com:51523/godata
        uri := os.Getenv("MONGOLAB_URL")
        if uri == "" {
                fmt.Println("no connection string provided")
                os.Exit(1)
        }

        sess, err := mgo.Dial(uri)
        if err != nil {
                fmt.Printf("Can't connect to mongo, go error %v\n", err)
                os.Exit(1)
        }
        defer sess.Close()

        sess.SetSafe(&mgo.Safe{})

        collection := sess.DB("godata").C("user")

        result := Person{}

        collection.Find(bson.M{"name": r.FormValue("name")}).One(&result)

        if result.Email != "" {
                errn := displayTemplate.Execute(w, "The email id you wanted is: " + result\
.Email)
                if errn != nil {
                        http.Error(w, errn.Error(), http.StatusInternalServerError)
                }
        } else {
                displayTemplate.Execute(w, "Sorry... The email id you wanted does not exis\
t.")
        }
}

const displayTemplateHTML = `
<!DOCTYPE html>
  <html>
    <head>
      <meta charset="utf-8">
      <title>Results</title>
      <link rel="stylesheet" href="http://yui.yahooapis.com/pure/0.4.2/pure-min.css">
    </head>
    <body>
      <h2>A Fun Go App on Heroku to access MongoDB on MongoLab</h2>
      <p><b>{{html .}}</b></p>
      <p><a href="/">Start again!</a></p>
      <div>
        <p><b>&copy; 2015 Go Challenge. All rights reserved.</b></p>
      </div>
    </body>
  </html>
`

// Get the Port from the environment so we can run on Heroku
func GetPort() string {
        var port = os.Getenv("PORT")
        // Set a default port if there is nothing in the environment
        if port == "" {
                port = "4747"
                fmt.Println("INFO: No PORT environment variable detected, defaulting to " + port)
        }
        return ":" + port
}
```

The program by now should be self-explanatory.

- The `rootForm` uses Pure[7] – a set of small, responsive CSS modules that you can use in every web project.
- The function `display` uses the `html/template` package and the `mgo` driver to access the database on MongoLab. If the name is found in the database the function throws a page to the user with the email id for that name.

[7] http://purecss.io/

Note: Before you deploy `gomongohql.go` to Heroku. Remember to:

- Read the contents of `Godeps/_workspace`. You will observe that you need to add the folder `src/labix.org` with all its contents under `Godeps/_workspace`. Then commit the whole `Godeps` directory to version control, including `Godeps/_workspace`.
- After you have deployed the app to Heroku, remember to set the `heroku config:set MONGOLAB_URL=mongodb://IndianGuru:dbpassword@ds051523.mongolab.com:51523/godata` and then `heroku open`.

Exercise

Write a simple Go program (`getcapital.go`) and host it on Heroku. This program when accessed shows a simple form to the user, where he/she enters a country name. The Go app uses this information to access a MongoDB database on MongoLab and fetches the capital of that country. It then either displays the name of the capital or an error message to the user.

Exercise

The Gopher Academy Blog has an excellent article titled "Build a Christmas List with Martini[8]". As an exercise build this app on Heroku and which accesses the MongoDB database on MongoLab.

[8]http://blog.gopheracademy.com/day-11-martini