freeformz Follow
I am too many things, yet not enough.
Apr 17, 2015 · 5 min read

Go + Heroku : Hello World

NOTE: Heroku has launched official support for Go.

This is a quick start guide to deploying a simple “Hello World” Go web application to Heroku. It is in no way comprehensive, but aims to provide links to supporting documentation. In many ways it is an updated version of the excellent Getting Started With Go On Heroku guide done by Mark McGranaghan.

The guide is command line centric and examples assume basic proficiency with a shell on a UNIX (OS X, Linux, etc) operating system.

What is Go

Go is an open source programming language with built-in concurrency primitives, type safety, fast compile time, automatic memory management and a comprehensive standard library. It’s fun to write and a great fit for web and other network services.

Installing Go

Install Go by following the installation instructions on the Go website for your specific OS.

Once installed you should be able to open a console / terminal / command line prompt and run the following:

```
$ go version
go version go1.4.2 darwin/amd64
```

If you are having trouble installing go, you can find help on the golang-nuts mailing list and/or the Gopher Slack Channel.

Go Workspace

Go code is organized into a workspace rooted at ***$GOPATH***. A workspace contains the source code for multiple go packages (***$GOPATH/src/…***) as well as compiled binaries (***$GOPATH/bin***) and intermediary compiled objects (***$GOPATH/pkg/…***). The go tooling expects your code to be arranged into one or more workspaces.

This is all detailed in the How To Write Go Code document, but here is the TL;DR version:

```
$ mkdir -p $HOME/go
$ export GOPATH=$HOME/go
$ export PATH=$GOPATH/bin:$PATH
```

This sets the workspace to the directory ***$HOME/go*** and adds the workspace's bin directory to your ***$PATH*** for the current shell.

NOTE: From here on we expect ***$GOPATH*** and ***$PATH*** to be set as above.

Hello World Web Application

Start by creating the directory inside of ***$GOPATH*** that will hold the application's source code:

```
mkdir -p $GOPATH/src/helloworld
```

Using your favorite editor create ***$GOPATH/src/helloworld/main.go*** with the following content:

```
package main

import (
  "log"
  "fmt"
  "net/http"
```

```
  "os"
)


func determineListenAddress() (string, error) {
  port := os.Getenv("PORT")
  if port == "" {
    return "", fmt.Errorf("$PORT not set")
  }
  return ":" + port, nil
}


func hello(w http.ResponseWriter, r *http.Request) {
  fmt.Fprintln(w, "Hello World")
}


func main() {
  addr, err := determineListenAddress()
  if err != nil {
    log.Fatal(err)
  }


  http.HandleFunc("/", hello)
  log.Printf("Listening on %s...\n", addr)
  if err := http.ListenAndServe(addr, nil); err != nil {
    panic(err)
  }
}
```

The code above is pretty basic, consisting of just 3 methods:

1. ***main*** is the entry point into the application. It first tries to determine which address to listen on and if it can't exits with an error. Otherwise ***hello*** is registered to handle all requests and listens on the address. If there is any problem listening on the address it panics, otherwise it starts servicing http requests.
2. ***hello*** is the http.Handler that 'says' "Hello World".
3. ***determineListenAddress*** figures out what address to listen on for traffic. It uses the ***$PORT*** environment variable only to determine this. If ***$PORT*** isn't set an error is returned instead.

Compile and install the code with:

```
$ go install ./...
```

Run the application with:

```
$ PORT=5000 $GOPATH/bin/helloworld
Listening on 5000...
```

Make a HTTP request to the application:

```
$ curl -i http://localhost:5000/
HTTP/1.1 200 OK
Date: Wed, 15 Apr 2015 06:32:33 GMT
Content-Length: 12
Content-Type: text/plain; charset=utf-8

Hello World
```

Deploying To Heroku

The application is running locally, so let's deploy it to Heroku.

Heroku Prerequisites

1. Sign up for a Heroku account.
2. Install the Heroku Toolbelt.
3. Login with ***heroku login***.

Preparing the application

Git is installed as part of the Heroku Toolbelt. We'll use it to track and push the source code to Heroku.

```
$ git init
$ git add -A .
$ git commit -m "Hello World"
```

Heroku uses a file named ***Procfile*** to determine what should be run. Let's configure the *web* process type to execute the file named

helloworld:

```
$ echo “web: helloworld” > Procfile
$ git add Procfile
$ git commit –m Procfile
```

Godep

The supported way to manage dependencies for Go applications on Heroku is via Godep. Install godep via:

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

Record and vendor any dependencies:

```
$ godep save –r ./...
$ git add –A Godeps
$ git commit –m Godeps
```

This saves a description of the dependencies to ***Godeps/Godeps.json*** and vendors the source code into ***Godeps/_workspace*** as well as rewrites any import directives to use those copies. This hello world application doesn’t have any dependencies outside of Go’s standard library and the ***Godeps/Godeps.json*** records that:

```
$ cat Godeps/Godeps.json
{
    “ImportPath”: “helloworld”,
    “GoVersion”: “go1.4.2”,
    “Packages”: [
        “./…”
    ],
    “Deps”: []
}
```

Heroku Deploy

Create a new Heroku app using the Go Buildpack:

```
$ heroku create -b https://github.com/heroku/heroku-
buildpack-go.git
Creating sheltered-peak-9514... done, stack is cedar-14
Buildpack set. Next release on sheltered-peak-9514 will use
https://github.com/heroku/heroku-buildpack-go.git.
https://sheltered-peak-9514.herokuapp.com/ |
https://git.heroku.com/sheltered-peak-9514.git
Git remote heroku added
```

The application name is randomly generated, so will differ from above.

Push the code to Heroku:

```
$ git push heroku master
Counting objects: 13, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (13/13), 1.37 KiB | 0 bytes/s, done.
Total 13 (delta 0), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: - - -> Fetching custom git buildpack… done
remote: - - -> Go app detected
remote: - - -> Installing go1.4.2... done
remote: - - -> Running: godep go install -tags heroku ./...
remote: - - -> Discovering process types
remote: Procfile declares types -> web
remote:
remote: - - -> Compressing... done, 1.6MB
remote: - - -> Launching... done, v3
remote: https://sheltered-peak-9514.herokuapp.com/ deployed
to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/sheltered-peak-9514.git
 * [new branch] master -> master
```

The applications is now running on Heroku. Open it in a browser:

```
$ heroku open
```

You can inspect the dyno(s) running the application:

```
$ heroku ps
=== web (1X): `helloworld`
web.1: up 2015/04/14 23:55:51 (~ 6m ago)
```

And you can view requests that come in via the log stream:

```
$ heroku logs --tail
...
2015-04-15T06:59:39.396139+00:00 heroku[router]: at=info
method=GET path="/" host=sheltered-peak-9514.herokuapp.com
request_id=f5a7d634-dbf0-42c2-a278-e1dc37c98178
fwd="76.115.27.201" dyno=web.1 connect=0ms service=1ms
status=200 bytes=148
...
```

Thanks

Thanks to Michelle Peot, Andrew Gwozdziewycz, Cyril David & Mark McGranaghan for reviewing and making suggestions.

. . .

Originally published at icanhazdowntime.org.