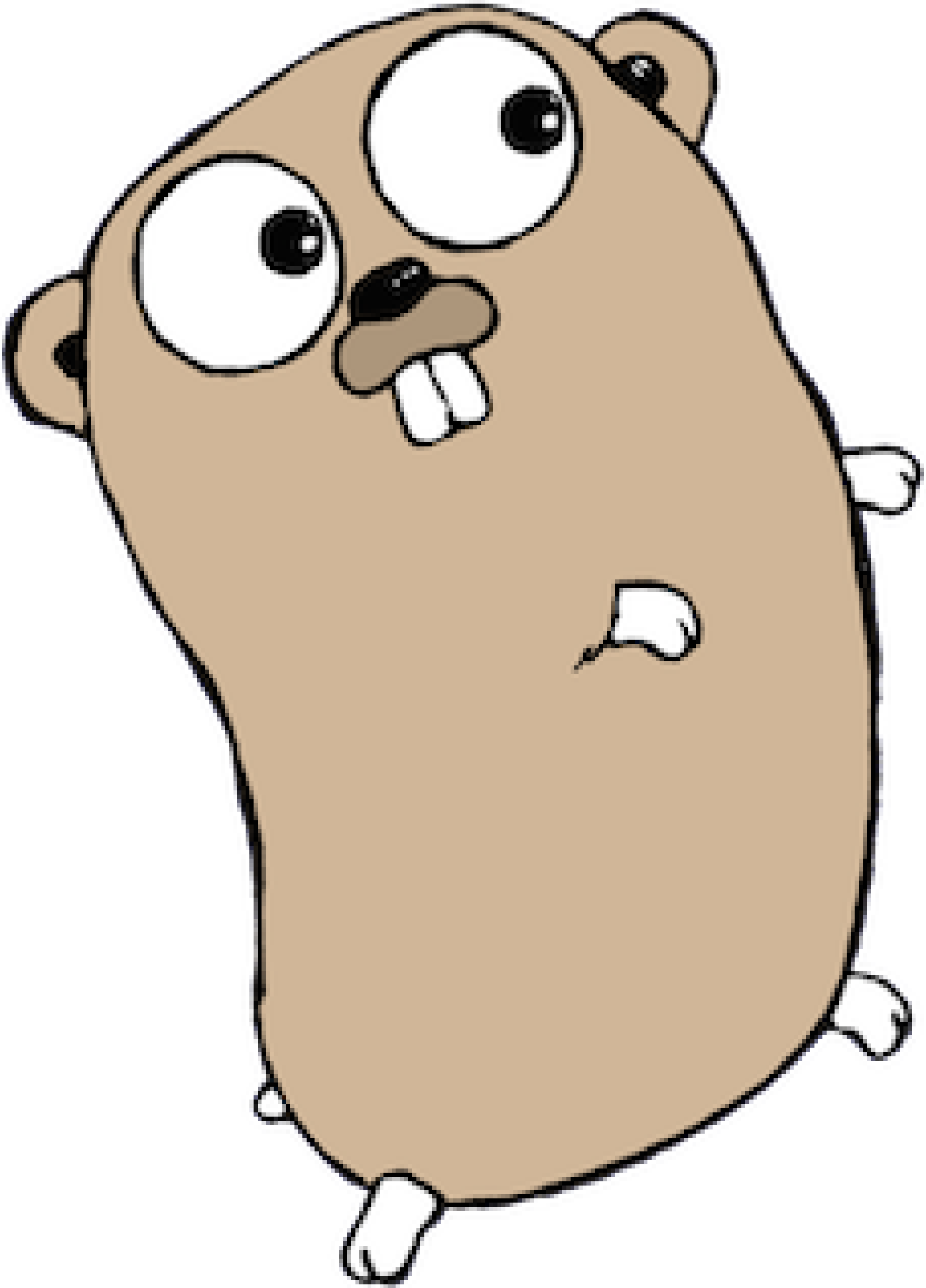


How to Deploy a Go Web App to Heroku 101

Satish Talim

This book is for sale at http://leanpub.com/howtodeployagowebapptoheroku101

This version was published on 2016-09-17

Also By Satish Talim

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

Building a package in Go

How do I use Sourcegraph with Go?

How do I use Sourcegraph with Ruby?

How to Deploy a Go Web App to the Google App Engine 101

How do I use the template package and handle forms?

Go and MongoDB on MongoLab and Heroku

Learn Go programming

New to Go? Want to deploy a web app built in Go to Heroku? This eBook quickly guides you to do exactly that.

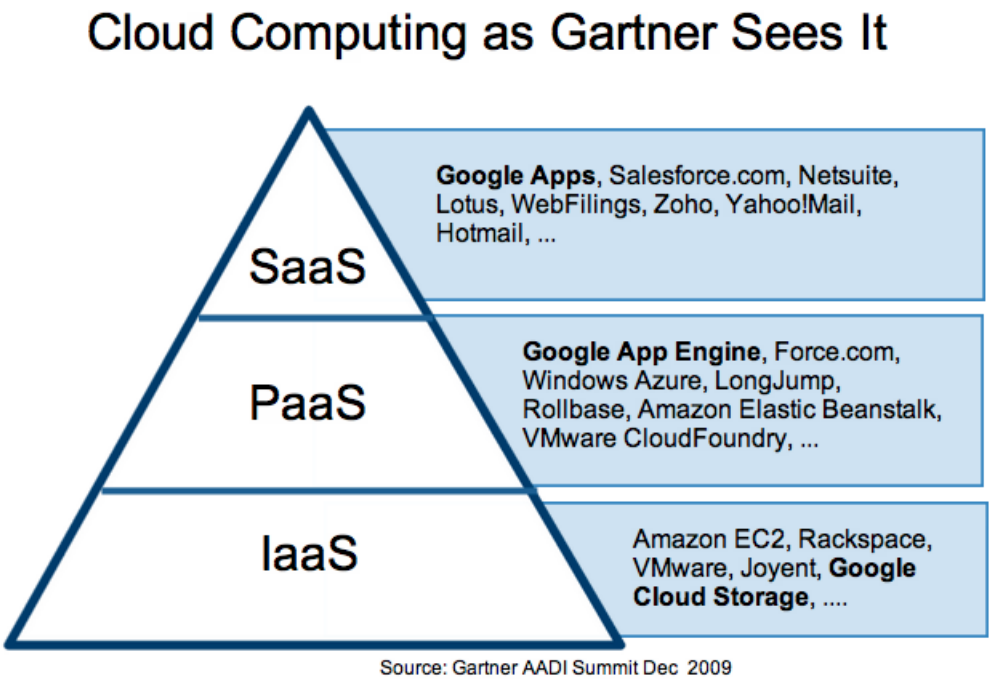
Contents

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."

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 Service Levels

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.

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.

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.

Assumption

I assume that you have:

- Go 1.6+ or 1.7+ installed.
- $GOPATH/bin has been added to your $PATH.

[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

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**.

Install the Heroku Command Line Interface (CLI)

The Heroku CLI[5] provides you access to the Heroku Command Line Interface (CLI). Once installed, you'll have access to the `heroku` and `git` command from your command window.

Log in to heroku

Open a command window and create a folder `webapphr` under the folder `$GOPATH/src/github.com/SatishTalim/`. Change your folder to `$GOPATH/src/github.com/SatishTalim/webapphr`.

Now log in to Heroku using the email address and password you used when creating your Heroku account:

```
$ heroku login
Enter your Heroku credentials.
Email: satish.talim@gmail.com
Password (typing will be hidden):
Logged in as satish.talim@gmail.com
```

Authenticating is required to allow both the heroku and git commands to operate.

Introduce yourself to Git

On Windows, start the Command Prompt (cmd.exe) to access the command shell.

For all operating users, you now need to identify yourself to Git (you need to do this only once) so that it can properly label the commits you make later on. I am using SatishTalim and satish.talim@gmail.com below:

```
$ git config --global user.name "Satish Talim"
$ git config --global user.email satish.talim@gmail.com
```

Substitute in your own user name and email id.

[4] http://heroku.com/signup

[5] https://s3.amazonaws.com/assets.heroku.com/heroku-toolbelt/heroku-toolbelt.exe

Prepare a web app

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

In the folder `webapphr` under the folder `$GOPATH/src/github.com/SatishTalim/` write the program `webapphr.go` as follows:

Program webapphr.go

```
1  package main
2
3  import (
4          "fmt"
5          "log"
6          "net/http"
7          "os"
8  )
9
10 func main() {
11         http.HandleFunc("/", handler)
12         fmt.Println("listening...")
13         err := http.ListenAndServe(GetPort(), nil)
14         if err != nil {
15                 log.Fatal("ListenAndServe: ", err)
16         }
17 }
18
19 func handler(w http.ResponseWriter, r *http.Request) {
20         fmt.Fprintf(w, "Hello. This is our first Go web app on Heroku!")
21 }
22
23 // Get the Port from the environment so we can run on Heroku
24 func GetPort() string {
25         var port = os.Getenv("PORT")
26         // Set a default port if there is nothing in the environment
27         if port == "" {
28                 port = "4747"
29                 fmt.Println("INFO: No PORT environment variable detected, defaulting to " + port)
30         }
31         return ":" + port
32 }
```

Use Git to deploy our app to Heroku

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"
```

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.

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
```

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

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`.

Add these new files to git

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

Now we're ready to ship this to Heroku.

Create and Deploy the app

In this step, you will first create and then deploy the app to Heroku.

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

```
$ heroku create
Creating app... done, stormy-lake-46504
https://stormy-lake-46504.herokuapp.com/ | https://git.heroku.com/stormy-lake-46504.git
```

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 `stormy-lake-46504`) 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% (8/8), done.
Writing objects: 100% (11/11), 1.30 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 go1.6.2... done
remote:  !!    Installing package '.' (default)
remote: -----> Running: go install -v -tags heroku .
remote: github.com/SatishTalim/webapphr
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 2.2M
remote: -----> Launching...
remote:        Released v3
remote:        https://stormy-lake-46504.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/stormy-lake-46504.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!

A brief note by Gunnar Aasen

Troubleshooting

A number of bugs can potentially be encountered in the process of deploying a Go application to Heroku. Many times, a bug can be traced to a misunderstanding of Heroku's architecture and how it works.

Heroku's Architecture

Overall, Heroku applications are expected to follow the "process model"[7] architecture. The process model allows applications, and web applications in particular, to be easily scaled out and managed with little administrative overhead. In exchange, the architecture requires applications to conform to certain rules and restrictions.

There are two fundamental parts to a Heroku application: Dynos, which follow the proccess model; and Add-ons, which are provided through third-party integrations.

Dynos

The main unit of a Heroku application is the Dyno.

Dynos are ephemeral servers which run a single process. Each process started by a dyno must be assigned a "process type". A dyno's type is defined by the starting process type.

Even though a dyno can only start a single process, once started that process may spawn additional processes within the dyno within limits[8].

Keep in mind, dynos do not retain data. Data does not persist between Dyno restarts and there are no shared volumes between dynos. Practically speaking, you cannot retain any data on a dyno. Add-ons[9] should be used for persistence and are explained below.

There are three types[10] of Dynos tailored for different workloads: web dynos, worker dynos, and one-off dynos.

Only web dynos receive HTTP traffic via the Heroku router. Heroku assigns a single `$PORT` environment variable to all dynos running a `web` process type. All inbound HTTP traffic to a `web` process goes through the single assigned `$PORT` variable. The Heroku router attaches headers[11] to HTTP requests to describe the original request as received by Heroku.

Web Dynos also have the following restrictions not discussed above: - They allow 50 active requests per `web` dyno with a 50 request queue. - They must connect to the Heroku provided `$PORT` environment variable within 60 seconds[12] or the dyno will automatically be shut down.

Add-ons

Persistence and other functionality needed by applications which cannot be provided by Dynos are available through Add-ons[13]. Many add-ons are integrated with Dynos through environment variables.

[7] https://devcenter.heroku.com/articles/process-model
[8] https://devcenter.heroku.com/articles/dynos#process-thread-limits
[9] https://addons.heroku.com/
[10] https://devcenter.heroku.com/articles/dynos#types-of-dynos
[11] https://devcenter.heroku.com/articles/http-routing#heroku-headers
[12] https://devcenter.heroku.com/articles/dynos#web-dynos
[13] https://addons.heroku.com/

Slugs

In addition to the process described above, Go applications can also be cross-compiled into a binary and deployed to Heroku through the use of a custom slug. Slugs are containers (similar to Docker images) which hold everything needed to run a program on Heroku's infrastructure. To deploy a Go application this way, a slug needs to be created, published, and released on Heroku. Heroku has published an article specifically about deploying Go slugs[14].

[14] https://devcenter.heroku.com/articles/platform-api-deploying-slugs#go