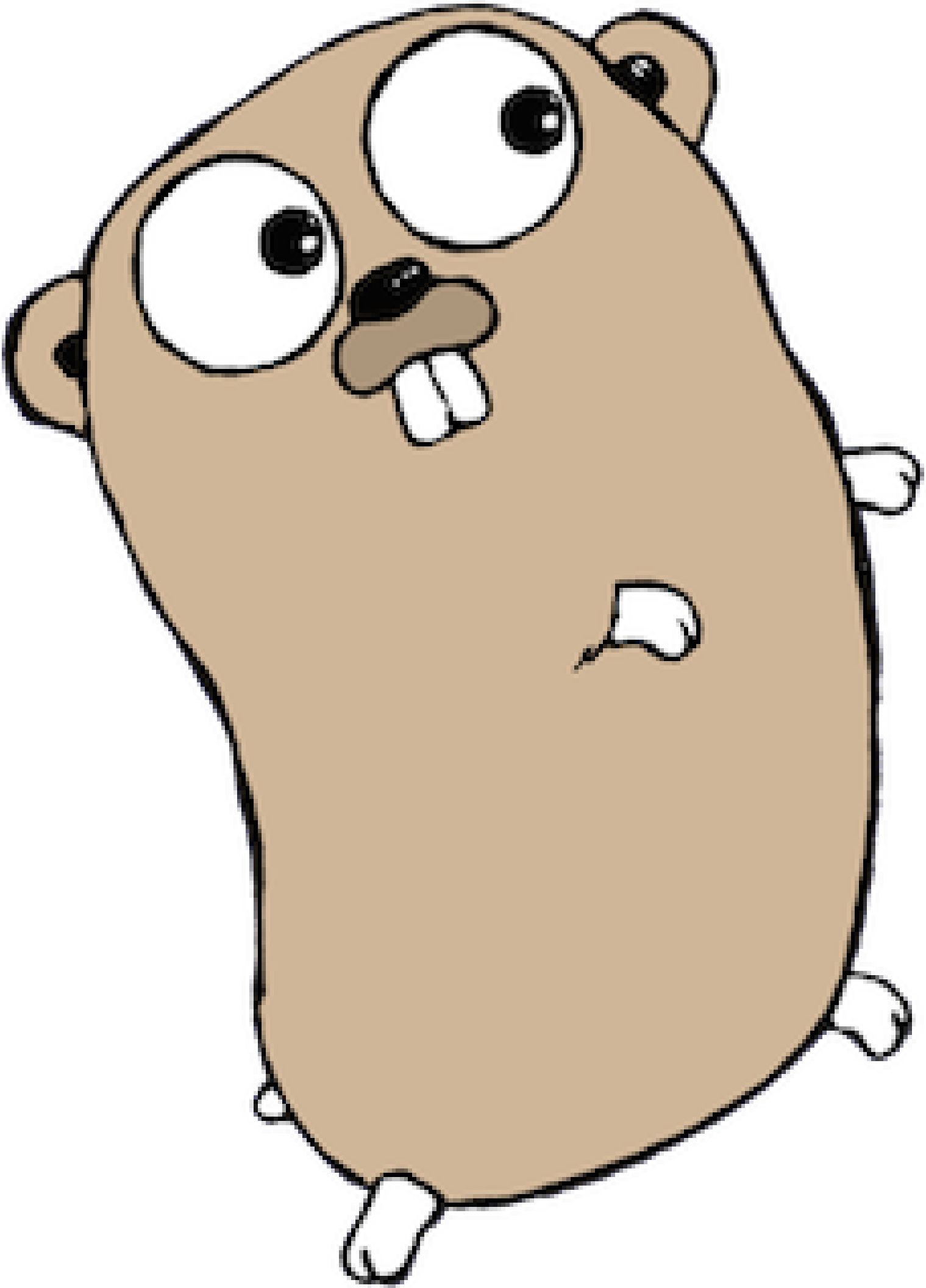

How to Deploy a Go Web App to the Google App Engine 101

Satish Talim

This book is for sale at http://leanpub.com/howtodeployagowebapptothegoogleappengine101

This version was published on 2015-09-27

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 Heroku 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 the Google App Engine? This eBook quickly guides you to do exactly that.

Contents

Google App Engine

Google App Engine[1] is different from most other cloud systems because it is neither IaaS (Infrastructure-as-a-Service, e.g., Amazon EC2) nor SaaS (Software-as-a-Service, e.g., Salesforce). It is something in-between - PaaS (Platform-as-a-Service). Instead of a fixed application (SaaS) or raw hardware (IaaS), App Engine manages your infrastructure for users. Furthermore, it provides a development platform… users get to create apps, not use the ones provided by the cloud vendor, and it leverages the infrastructure as a hosting platform.

Google App Engine lets you run web applications on Google's infrastructure. With App Engine, there are no servers to maintain: You just upload your application, and it is ready to serve your users. App Engine costs nothing to get started. All applications can use up to *1 GB of storage* and enough CPU and bandwidth to support an efficient app serving around 5 million page views a month, absolutely *free*.

Creating an App Engine application is easy, and only takes a few minutes. And it is free to start: upload your app and share it with users right away, at no charge and with no commitment required.

Sandbox

Developers would not be interested in letting other applications/users get any kind of access to their application code or data. To ensure this, all App Engine applications run in a restricted environment known as a sandbox.

This is a Warning

Because of the sandbox, applications can't execute certain actions. These include: open a local file for writing, open a socket connection, and make operating system calls.

Services

The App Engine team has created a set of higher-level APIs/services for developers to use. Want your app to send and receive e-mail or instant messages? That's what the e-mail and XMPP APIs are for! Want to reach out to other web applications? Use the `URLfetch` service! Need Memcache? Google has a global Memcache API. Need a database? Google provides both its traditional NoSQL scalable datastore and access to the relational MySQL-compatible Google Cloud SQL service.

The list of all the services that are available to users changes quite often as new APIs are created.

The Administration Console

The Google App Engine Administration Console gives you complete access to the public version of your application. Access the Console by visiting this link[2] in your web browser. Google recommends that you use

[1] https://developers.google.com/appengine/docs/whatisgoogleappengine
[2] https://appengine.google.com/

the Google Developers Console[3] instead. The Developers Console supports all the Cloud Platform products, including App Engine, as well as other Google developer APIs.

Applications (web and non-web)

While many applications running on Google App Engine are web-based apps, they are certainly not limited to those. App Engine is also a popular backend system for mobile apps. When developing such apps, it's much safer to store data in a distributed manner and not solely on devices which could get lost, stolen, or destroyed. Putting data in the cloud improves the user experience because recovery is simplified and users have more access to their data.

DataStore

App Engine Datastore[4] is a schemaless object datastore providing robust, scalable storage for your web application. *This means that you can't run MongoDB, for example, on the Google App Engine (GAE). You need to use the Google datastore*. However, GAE now supports Google Cloud SQL[5] a fully managed MySQL service hosted on Google Cloud Platform.

The Go runtime environment

With the Google App Engine for Go, you can build web applications using the Go Programming Language. Your Go application runs on Google's scalable infrastructure and uses large-scale persistent storage and services.

App Engine builds and executes Go application code using a safe "sandboxed" environment. Your app receives web requests, performs work, and sends responses by interacting with this environment.

The Go runtime environment uses the latest version of Go version 1.4. The SDK includes the Go compiler and standard library, so it has no additional dependencies. As with the other runtimes, not all the standard library's functionality is available inside the sandbox. For example, attempts to open a socket or write to a file will return an `os.ErrPermission` error.

The SDK includes an automated build service to compile your app, so you'll never need to invoke the compiler yourself. And your app will be automatically re-built whenever you change the source.

Go apps run inside a secure "sandbox" environment with a reduced set of libraries. For instance, *an app cannot write data to the local file system* or make arbitrary network connections. Instead, apps use scalable services provided by App Engine to store data and communicate over the Internet.

Download and Install the App Engine SDK

To start developing Google App Engine applications in Go, you first download and set up the App Engine Go software development kit (SDK).

[3] https://developers.google.com/console/help/console?_ga=1.214591252.248597932.1423453183
[4] https://developers.google.com/appengine/docs/go/datastore/
[5] https://developers.google.com/cloud-sql/

The Go SDK includes a web server application that simulates the App Engine environment, including a local version of the datastore, Google Accounts, and the ability to fetch URLs and send email directly from your computer using the App Engine APIs.

The Go SDK will run on any Intel-based Mac OS X, Linux or Windows computer with *Python 2.7*. If necessary, download and install Python 2.7 for your platform from the Python web site[6]. Most Mac OS X users already have Python 2.7 installed. If you have issues with the Python tools, please ensure you have Python 2.7 installed.

Let us now download the App Engine SDK[7]. Next follow the instructions on the download page to install the SDK on your computer.

I installed the App Engine SDK to `C:\go_appengine` on my Windows 7 desktop.

Later on in this article, we will use the following two commands from the SDK:

- goapp serve[8] - for running a local development server
- goapp deploy[9] - for uploading your app to App Engine

You can find these commands in the `C:\go_appengine` directory. To simplify development and deployment, consider adding this directory to your `PATH` environment variable.

Tip

While setting the `PATH` ensure that `C:\go_appengine` comes after `C:\go\bin` i.e. it should be like `PATH=C:\go\bin;C:\go_appengine;...` This ensures that we use the `go` command from the original Go installation and not the `go` command from the App Engine.

Let us build a small app (mytext.go) locally

The local development environment lets you develop and test complete App Engine applications before showing them to the world. Let us write some code.

Go App Engine applications communicate with the outside world via a web server compatible with Go's http package[10]. This makes writing Go App Engine applications very similar to writing stand-alone Go web applications.

Let us begin by implementing a tiny application that displays a short message to a user.

[6] http://www.python.org/download/
[7] https://developers.google.com/appengine/downloads#Google_App_Engine_SDK_for_Go
[8] https://developers.google.com/appengine/docs/go/tools/devserver
[9] https://developers.google.com/appengine/docs/go/tools/uploadinganapp
[10] http://golang.org/pkg/net/http/

Program mytext.go

Inside the folder `$GOPATH/src/github.com/SatishTalim` create the folder `mytext`.

Next inside the `mytext` folder, create a file named `mytext.go`, and give it the following contents:

Program mytext.go

```
1  package mytext
2
3  import (
4          "fmt"
5          "net/http"
6  )
7
8  func init() {
9          http.HandleFunc("/", handler)
10 }
11
12 func handler(w http.ResponseWriter, r *http.Request) {
13         fmt.Fprint(w, "Hello. This is our first Go web app for Google App Engine!")
14 }
```

This Go package responds to any request by sending a response containing the message: `Hello. This is our first Go web app for the Google App Engine!`

Note:

- when writing a stand-alone Go program we would place this code in package `main`. The Go App Engine Runtime provides a special `main` package, so you should put HTTP handler code in a package of your choice (in this case, `mytext`).
- to work with some printing functions, we import the package fmt[11].
- the App Engine Go API uses the standard `http` package as an interface between your Go program and the App Engine servers. Thus for web related `http` functionality, we import the package http[12]. Any functions within that we refer as `http.function_name`.
- within the `init` program, we redirect any incoming requests to the `handler` function. We do this by calling http.HandleFunc[13] and passing it two parameters - the first one is a part of the incoming url, and the second is the method capable of handling it.
- the function `handler` takes an `http.ResponseWriter` and an `http.Request` as its arguments.
- when a user connects, the programs responds with a text that is sent back to the browser. The `http.ResponseWriter` value assembles the HTTP server's response; by writing to it, we send data to the HTTP client.
- an `http.Request` is a data structure that represents the client HTTP request.
- all the parameters of a request can be received via the parameter `http.Request` in the `handler`. You can get the URL, the input values and other details.

Create the Configuration File

An App Engine application has a configuration file called `app.yaml`. Among other things, this file tells the App Engine service which runtime to use and which URLs should be handled by our Go program.

Inside the `$GOPATH/src/github.com/SatishTalim/mytext` directory, create a file named `app.yaml` with the following contents:

[11] http://golang.org/pkg/fmt/
[12] http://golang.org/pkg/net/http/
[13] http://golang.org/pkg/net/http/#HandleFunc

File app.yaml

```
# This is a comment

# application is mandatory (on web gochgmsg)
application: helloworld

# version is mandatory
version: 1-0

# runtime is mandatory
runtime: go

# api_version is mandatory
api_version: go1

# handlers is mandatory
handlers:
- url: /.*
  script: _go_app
```

From top to bottom, this configuration file says the following about this application:

- The application identifier is `helloworld`. When you register your application with App Engine later on, you will select a unique identifier, and update this value (later on we will update the value to `gochgmsg`). This value can be anything during development. For now, leave it set to `helloworld`.
- This is version number `1-0` of this application's code. Your application versioning information can contain alphanumeric characters, and hyphens. If you adjust this before uploading new versions of your application software, App Engine will retain previous versions, and let you roll back to a previous version using the administrative console.
- This code runs in the `go` runtime environment, with API version `go1`.
- There are two kinds of handlers: *script handlers*, and *static file handlers*. A script handler runs a Go script in your application to determine the response for the given URL. A static file handler returns the contents of a file, such as an image, as the response.
- Static files are files to be served directly to the user for a given URL, such as images, CSS stylesheets, or JavaScript source files. Static file handlers describe which files in the application directory are static files, and which URLs serve them.
- `url` is a URL prefix. This value uses regular expression syntax (and so regexp special characters must be escaped \). All URLs that begin with this prefix are handled by this handler, using the portion of the URL after the prefix as part of the file path.
- Every request to a URL whose path matches the regular expression `/.*` (all URLs) should be handled by the Go program.
- For Go apps, `script` should always have a value of `_go_app`.

Note: All Go packages for a given app are built into a single executable, and request dispatch is handled by the Go program itself. This is why we call `http.HandleFunc` inside the `init` function to associate our handler with the web root (`"/"`). However, you may still use the `app.yaml` file to configure paths that serve static files or require special permissions.

For a complete list of configuration options, see the Go Application Configuration page[14].

Test the App

You can now test your app with the web server included with the App Engine SDK.

The application's directory should contain the files `mytext.go` and `app.yaml`.

From the `$GOPATH/src/github.com/SatishTalim` directory run the following command, to compile your app and start the development web server:

```
goapp serve mytext/
```

The web server is now running, listening for requests on port 8080. Test the application by visiting the following URL in your web browser: http://localhost:8080/[15]. For more information about running the development web server, including how to change which port it uses, see the Development Server reference[16].

Iterative Development

The development app server knows to watch for changes in your file. As you update your source, it re-compiles them and relaunches your local app. There's no need to restart `goapp serve`.

Try it now: leave the web server running, then edit `mytext.go` to change `Hello. This is our first Go web app for the Google App Engine!` to something else. Reload http://localhost:8080/[17] to see the change.

To shut down the web server, make sure the terminal window is active, then press `Control-C` (or the appropriate "break" key for your console).

You now have a complete App Engine application! You could deploy this simple program right now and share it with users worldwide.

Uploading Your App to Google's App Engine

Registering the App

You will now need to have a Google account. If you do not have a Google account, you can create a Google account[18] with an email address and password.

You create and manage App Engine web applications from the Developer's Console[19]. Sign in to App Engine using your Google account.

[14] https://developers.google.com/appengine/docs/go/config/appconfig
[15] http://localhost:8080/
[16] https://developers.google.com/appengine/docs/go/tools/devserver
[17] http://localhost:8080/
[18] https://www.google.com/accounts/
[19] https://console.developers.google.com/

To create a new application, click the "Create a Project" button. A screen pops up as shown below:

Create a project

Enter `gochgmsg` for the Project name. It creates a unique project ID which in our case is `gochgmsg-1057`.

We have elected to use the free "appspot.com" domain name. With that, the full URL for the application will be `http://gochgmsg-1057.appspot.com/`.

Edit the `app.yaml` file, then change the value of the application: setting from `helloworld` to `gochgmsg-1057`.

Upload and Access the app

From the folder: `$GOPATH/src/github.com/SatishTalim` folder, type:

```
goapp deploy mytext/
```

If you see compilation errors, fix the source and re-run `goapp deploy`; it won't launch (or update) your app until compilation is successful.

Note: One issue that users could face while deploying the app—if you have multiple Google Accounts and the default one in which you are logged in currently is not the one that is containing the App Engine project, then the `goapp deploy` will fail with the error "The application does not exist ..

Access the app

You can now see your application running on App Engine. We have our message app[20] running, that you can access and check out.

Congrats you have just successfully launched your first Go web app for the world to see!!

App Engine determines that an incoming request is intended for your application using the domain name of the request. A request whose domain name is `http://your_app_id.appspot.com` is routed to the application whose ID is `your_app_id`. Every application gets an `appspot.com` domain name for free.

Requests for these URLs all go to the version of your application that you have selected as the default version in the App Engine Administration Console. Each version of your application also has its own URL, so you can

[20] http://gochgmsg-1057.appspot.com/

deploy and test a new version before making it the default version. The version-specific URL uses the version identifier from your app's configuration file in addition to the `appspot.com` domain name, in this pattern: `http://version_id-dot-latest-dot-your_app_id.appspot.com`.