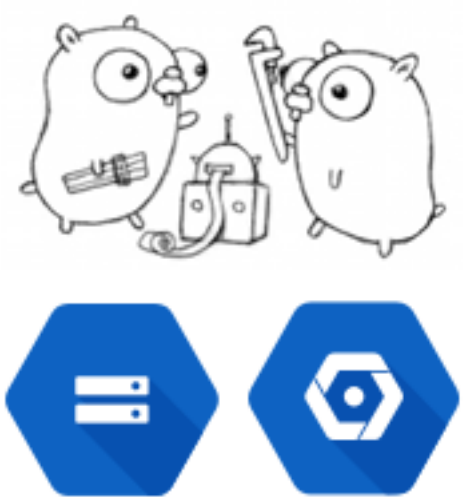
How to use Google Cloud Storage with Golang



I recently had the “pleasure” of getting started with Google Cloud Storage using Go. It was a hassle, but all’s well that ends well.

Google Cloud Storage, Storage or GCS for short, is a S3 like service from Google. It lets you store files and other “blob” data cheaply and easily. These files can be private or something you share with the world. With public files, Google will cache them automatically to make fetching them faster around the world. See the official Storage page for a longer salespitch.

Storage is a part of the overall Google Cloud Platform that also includes things like Compute Engine and App Engine (comes in two flavours: Standard and Flexible). See the Google Cloud Platform homepage for more.

Go (or golang if you want) is an open source programming language started by some guys at Google and now used by many people for many different things. It’s seen most success within newer, devops like, serverside tools such as Docker, Kubernetes and Consul. This is not the only usecase though, as it’s a general purpose language with many strong points. I currently use it to build back-end systems that talk http, http/2 and websockets with mobile app’s, JS frontends, special hardware etc.

1 Digression: A short Google Cloud Platform rant

Before we get to the point, I have to rant a little about the Google Cloud Platform (GCP). In many ways I’m a fan. It seems like they understand “no ops” better than almost anybody, and in my opinion App Engine could be the only real PaaS out there. Their pricing model is very nice, with a linear increase in costs as the traffic increases (quite unlike Heroku). With Compute Engine, their EC2-like IaaS, you get automatic discounts when running a server for more than 25% of a month (quite unlike Amazon’s reserved instances). This means there’s no need to “reserve instances” up front. Just run a normal instance for enough time and get automatic discounts. Great!

So, there are many things to like about GCP, but obviously things are not perfect. While the platform itself seems well-engineered, the developer experience can be horrible at times. The learning curve is a roller coaster, usually for stupid reasons.

If you’re in a hurry and you don’t have anyone with GCP experience I would probably recommend something else. That being said, developing on GCP is fast and easy when you know how to do it. If you can avoid the quirks and gotcha’s, the platform itself in incredibly powerful.

There’s three main problems I’ve seen so far:

- The documentation falls into three categories: Correct, Outdated and Just Plain Wrong. It sometimes feels like they are distributed evenly, and therefore only correct 33% of the time, but that’s probably just me being negative.
- Related, there’s usually 3 or 4 libraries to do something:

  - 1-2 legacy libraries that you are discouraged to use
  - 1 new, but still in beta and missing some feature, library,
  - 1 lib that's auto generated and probably works most of the time kind of.
  - (The documentation, of course, refers to one or all of these seemingly at random).
- Quirks and gotchas in the different products. These are actually, for the most part, documented and well known. They can still bite you though if you're not careful.

We'll see all these 3 in action here.

2 Cloud Storage with Go

I did not start completely from scratch. I already knew Storage pretty well having done something similar with Storage, App Engine Standard and Python.

The question therefore, when starting out, where something like:

- When developing on App Engine Standard (AE) you usually run it on your own machine first. The AE tooling helps to simulate some of the AE and Google Cloud Platform services locally. Can these tools simulate Storage locally? This would be useful when writing and testing new code.
- Which of the many libraries should I use? Do I need more than one?
- I'm working with private (not public) files. How is authentication handled?

These questions answered, in order:

2.1 Can you simulate Storage locally?

The short answer: No you can not. Use one or more buckets just for testing.

What most people end up doing is to have 1 or more Storage buckets just for development. This actually works pretty well when you get authentication working (quite easy, more on that later). You have to pay for the traffic you upload and download when testing, but the price is so low that it's just a rounding error for most usecases. Of course, if your files are very large uploading them could take some time. On the other hand, you probably have to upload them sooner or later anyway. And having them in the real Storage makes them accessible to the rest of the Google Cloud Platform, most of which integrates nicely with Storage.

The App Engine Standard tooling includes support for something called "Blobstore" which is the AE specific predecessor to Storage. It has some limited interoperability with Storage: You can tell it to actually store the files in Storage instead of in the old blobstore. I haven't tried this as I can't really see any of my projects staying with just the Blobstore for very long. Storage integrates with the rest of the Google Cloud, has many extra features and keeps evolving.

2.2 Which Go library should you use with Storage?

The short answer: Use the new gcloud Go libs: Main page / Storage client docs

This is the newest library and even though it's marked with "beta" it seems stable enough that people use it in production.

Here are the different libraries I've found so far:

- cloud.google.com/go/storage
- google.golang.org/cloud/storage
- code.googlesource.com/gocloud-legacy
- github.com/google/google-api-go-client

See what I meant in my rant? I probably even forgot a couple..

2.3 How do you authenticate to the “real” Storage when running locally?

The short answer: Use the gcloud lib mentioned above. Install the gcloud client program if you haven’t already (not related to the gcloud lib, confusing eh? =)

Run:

```
gcloud beta auth application-default login
```

on the command line and log in with your Google account.

After reading and googling this stuff for a while I was under the impression that this was more complicated. The above methods uses the developers Google account to authenticate with Storage. This is useful when running locally, but is a bad fit in some situations. You may want the program or machine to authenticate instead of the developer, for example when running automated tests. Therefore there are other ways of authenticating, mainly using Key files and something called Service Account’s.

I haven’t tried this in Go, but the process seems to be:

- Generate and download a key from the Cloud Console. You can use an existing Service Account or create a new one. You find this under IAM & Admin ⇒ Service Accounts.
- Pass this keyfile to the gcloud library like this:

```
client, err := storage.NewClient(ctx, option.WithServiceAccountFile("path/to/keyfile.json" ↩
    ))
```

All that is not necessary to just run code locally against the real Storage. Contrary to the common wisdom (at the time of writing), the helper gcloud command above (gcloud beta auth application-default login) actually works. This makes using the gcloud library a breeze, just do:

```
client, err := storage.NewClient(ctx).
```

When running on the Google Cloud Platform the same code just works and you should not have to think too much about authentication and authorization in common scenarios.

One last tip: I could not get auto-detecting of the bucketname, as seen in the docs, to work. This confused me for a long time since it doesn’t throw an error, it just returns an invalid bucketname.

Hopefully this is useful for someone. It took me hours to figure out, but it’s pretty simple and straight-forward when you know how to do it =).