Published in datamindedbe

Jonathan Merlevede Follow

Apr 28, 2021 · 7 min read · Listen

AUTOMAGIC AUTHENTICATION ON GOOGLE CLOUD PLATFORM

Authentication on GCP with Docker: Application Default Credentials

How ADCs magically authenticate applications through their environment, and how to make locally running containers magic too.


A
A
A
A

When working with applications that require access to GCP resources to work, you quickly learn the following:

- Applications running locally "magically" start working after you install the Google Cloud SDK and execute `gcloud auth application default-login`. Applications then run using your user credentials.
- Applications running on GCP (a GCE instance, Cloud Run, …) authenticate themselves using the service account that you configure when setting up the service, *even if you are running the application on images where Cloud SDK is not installed*.
- Setting the `GOOGLE_APPLICATION_CREDENTIALS` environment variable authenticates you "manually" and overrides the above. Point the variable to a JSON credentials file on your local filesystem and bam, your application runs using the credentials in the file!

This all just magically… works, for pretty much every application out there! But how? And what if you want to use your user credentials to run a Dockerized application? It won't be able to detect whatever it is that happens when you run `gcloud auth application-default login`. Where to even start?

This story uncovers the engineering behind the magic. It also looks at how to get your Dockerized applications to run using your user credentials.

For writing this story, I looked at Google's authentication library for Python, google-auth, and also briefly at the Google Auth library for Java. If you want more detail, I'd suggest starting there!

Application Default Credentials

The "automagic" authentication described above works because of something called Application Default Credentials (ADC).

The Application Default Credentials (ADC) flow or simply ADC is a Google-defined sequence of steps that applications wanting to interact with GCP resources should follow

The ADC flow

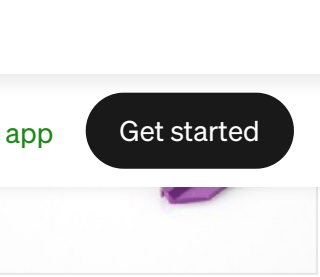
The ADC flow consists of the following steps:

- If the `GOOGLE_APPLICATION_CREDENTIALS` variable is set and points to a credentials file, then the contents of this file are used to obtain an access token. The file can be a user credentials file or a service account credentials file.
- Otherwise, the ADC will try to authenticate using the Cloud SDK. This really means looking for a file called `application_default_credentials.json` inside of the Google Cloud SDK configuration folder. This is where the Google Cloud SDK puts a credential file when you execute `gcloud auth application-default login`. If this file exists, then behavior is as if you pointed `GOOGLE_APPLICATION_CREDENTIALS` to this file.
- Otherwise, an uncredentialled HTTP request is sent to the GCE metadata service. When running on the GCP platform, the result of this request will be an access token.

Detailed notes:

- The Google Cloud SDK does not actually have to be installed for authentication "through the Cloud SDK" to work. All you need to do is to put a valid credentials file in the right place on your filesystem!
- The SDK's just look for the Cloud SDK in its default location, which is `~/.config/gcloud` on UNIX systems.
- For completeness sake: There's also a legacy mechanism for App Engine (the Google App Engine App Identity service) that's sometimes checked *before* the metadata service. Look into this if you're using App Engine and something is not working as expected. You're not likely to be using this.
- I wrote another story that takes a more detailed look at how the steps described above actually work, that is, how to actually use credentials or the metadata service to obtain access tokens. Check it out!


medium.com



The ADC name

> *There are 2 hard problems in computer science: cache invalidation, naming things, and off-by-1 errors.*

The name ADC is somewhat confusing, and Google's communication is a bit pedantic about not calling them credentials, even though they're not themselves consistent. This is my view:

- ADC are *not* credentials in the sense that they may not involve a secure "credential file" or secret. They may simply correspond call to the metadata service. ADC is just a weird name for a sequence of steps or even an authentication library.
- They *are* credentials in the sense that ADC can be "applied" to retrieve an access token, and that they authorize you to interact with GCP resources. Programmatically speaking, that means they implement the credentials interface.

I prefer to use the term "ADC flow" to refer to the sequence of steps involved in applying ADC.

ADC in Docker

Knowing the above, getting the ADC flow to work in Docker is quite easy! ADCs will work if you simply volume-mount your application default credentials file inside of the container, and point the variable `GOOGLE_APPLICATION_CREDENTIALS` to it. Assuming you're on a Unix system and that the Google Cloud SDK configuration folder is in the standard location, the command looks as follows:

```
docker run -v "$HOME/.config/gcloud/application_default_credentials.json":/gcp/creds.json:ro \
  --env GOOGLE_APPLICATION_CREDENTIALS=/gcp/creds.json \
  ...
```


- Running applications using your user credentials can be fine for development, but is not something you should do in production. This is probably one of the reasons why Google seems to like hiding the `application_default_credentials.json` file.
- You can of course also use a service account file.

The Google Cloud SDK

So, that's it then? Maybe yes, maybe no. Although technically, the solution above does make ADC available in the container that you run, there's one important set of applications not adhering to the ADC flow…. those part of the Google Cloud SDK itself! This includes `gcloud`, but also tools that accompany (and predate) it, like `bq` or `gsutil`. If you want to use these tools inside a Docker container, the solution above will not work.

The Cloud SDK tools authenticate themselves in the same way as other applications: using either user credentials, service account credentials or the metadata service. However, they do not look for them in the usual place, that is, they do not use ADC. Instead, the SDK tools store credentials for logged-in users and service accounts in a local database. I find this a little unfortunate, although this does allow you to use the `gcloud` command to easily switch between user profiles.

Detailed notes:

- The Google identity used in the ADC does not have to be the same identity as the one used to interact with the SDK tools. If you log in using different accounts using `gcloud auth login` (or `gcloud auth activate-service-account`) and `gcloud auth application-default login` then these identities will be different.
- If you want to know more about where your Cloud SDK configuration is stored, how to access your credentials database and how you can manipulate its configuration through environment variables, check out another story that I wrote below.

Getting ADC and the Cloud SDK to work in Docker

To get both the Cloud SDK and the ADC to work, a somewhat heavy-handed but powerful solution is to volume-mount your entire Cloud SDK configuration directory inside of the container. You can then tell the Cloud SDK to use the mounted volume as the configuration directory:

```
1  docker run -v "$HOME/.config/gcloud:/gcp/config:ro" \
```

The snippet works as follows:

- I mount the configuration as read only, because I generally do not want it to be modified from within containers. Remove the `:ro` as you please.
- The `CLOUDSDK_CONFIG` variable allows you to store your `gcloud` configuration in a non-standard location (the default location is `~/.config/gcloud` on UNIX systems).
- I volume-mount the logs directory (`-v /gcp/config/logs`); this makes the logs directory writeable again and increases performance compared to writing into a Docker layer.
- Explicitly set `GOOGLE_APPLICATION_CREDENTIALS` for good measure, as some applications using partial implementations of ADC may not pick up on the value of the `CLOUDSDK_CONFIG` variable.
- There is a variable called `CLOUDSDK_AUTH_CREDENTIAL_FILE_OVERRIDE` that should save us from having to mount our entire configuration. Although pointing this variable to a credentials file causes `gcloud` to work, programs like `bq` and `gsutil` unfortunately do not currently respect it.
- I configure the configuration location using the `CLOUDSDK_CONFIG` variable instead of mounting it inside of the Docker user's home directory, because the location of this home directory is not the same for all Docker containers you'll be running.
- I define the snippet above as an alias that I call `docker_run_gcp`, and add it to my `~/.bash_aliases` file. That way, if I want to do a "credentialed run", I just have type `docker_run_gcp` instead of `docker run`.
- You can use this idea to get to a system onto which you don't even have the Google Cloud SDK installed, but only run it through Docker (~ `alias gcloud=docker_run_gcp google/cloud-sdk`; do leave out the `:ro` flag).

That's it! Everything you should know about default application credentials, how the Cloud SDK authenticates itself and how to get rid of authentication problems and achieve a smooth dev experience with Docker!

. . .

> *I work at Data Minded, an independent data engineering and data analytics consultancy based in Leuven, Belgium. If GCP* causes you grief, free to contact us! (* We're also into AWS, Azure, Terraform, Spark, …)*