Authentication Components

Classes

class	**google::cloud::Credentials** An opaque representation of the authentication configuration. More...
struct	**google::cloud::UnifiedCredentialsOption** A wrapper to store credentials into an options. More...
struct	**google::cloud::DelegatesOption** Configure the delegates for `MakeImpersonateServiceAccountCredentials()` More...
struct	**google::cloud::ScopesOption** Configure the scopes for `MakeImpersonateServiceAccountCredentials()` More...
struct	**google::cloud::AccessTokenLifetimeOption** Configure the access token lifetime. More...
struct	**google::cloud::CARootsFilePathOption** Configures a custom CA (Certificates Authority) certificates file. More...

Functions

std::shared_ptr< **Credentials** >	**google::cloud::MakeInsecureCredentials** () Create insecure (aka anonymous, aka unauthenticated) credentials. More...
std::shared_ptr< **Credentials** >	**google::cloud::MakeGoogleDefaultCredentials** () Creates the default credentials. More...
std::shared_ptr< **Credentials** >	**google::cloud::MakeAccessTokenCredentials** (std::string const &access_token, std::chrono::system_clock::time_point expiration) Creates credentials with a fixed access token. More...
std::shared_ptr< **Credentials** >	**google::cloud::MakeImpersonateServiceAccountCredentials** (std::shared_ptr< **Credentials** > base_credentials, std::string target_service_account, **Options** opts={}) Creates credentials for service account impersonation. More...
std::shared_ptr< **Credentials** >	**google::cloud::MakeServiceAccountCredentials** (std::string json_object) Creates service account credentials from a service account key. More...

Detailed Description

Most services in Google Cloud Platform requires the client to authenticate the requests. Notable exceptions include public buckets in GCS and public data sets in BigQuery. The C++ client libraries are automatically configured to use "Google Default Credentials", some applications may need to override this default. The functions and classes related to change the authentication configuration are documented here.

This document is not a general introduction to authentication for Google Cloud Platform. For readers seeking such an introduction we recommend Authentication at Google as a good starting point. Covering authorization in any detail is also out of scope. We recommend reading the IAM overview if that is of interest.

In most cases applications can control the principal used by the client libraries without having to change any code. By default the client libraries use Application Default Credentials which can be configured via environment variables, the `gcloud` CLI, or by changing the service account associated with your deployment environment (GCE, Cloud Run, GKE, etc.)

General Concepts

As mentioned complete overview of authentication and authorization for Google Cloud is outside the scope of this document. The following brief introduction may help as you read the reference documentation for components related to authentication.

Google Cloud Platform largely uses OAuth2 access tokens for authentication. There are multiple ways to create such tokens. For example, when running on GCE the VM has access to a metadata server that can create these tokens for any application running on the VM. As another example, you can download a service account keyfile and the C++ client libraries will create access tokens using the contents of this file.

Access tokens usually expire in about an hour. The client libraries automatically refresh these tokens when needed. The only exception is **`MakeAccessTokenCredentials()`** where the application provides the access token.

Development Workstations

During development the most common configuration to use Application Default Credentials are:

1. Use the `gcloud auth application-default` to authentication using the developer's account for authentication.

1. Set `GOOGLE_APPLICATION_CREDENTIALS` environment variable to load a service account key. The value of this environment variable is the full path of a file which contains the service account key.

1. If you are using a GCE instance as your development environment, simply use the service account of the GCE machine to access GCP services.

Limitations

The C++ authentication components do not allow applications to create their own credential types. It is not possible to extend the C++ libraries without changing internal components. If you need additional functionality please file a feature request on GitHub. Likewise, creating the components that implement (as opposed to *describing*) authentication flows are also considered implementation details. If you would like to use them in your own libraries please file a feature request. We cannot promise that we will be able to satisfy these requests, but we will give them full consideration.

Function Documentation

◆ MakeAccessTokenCredentials()

```
std::shared_ptr<Credentials>
google::cloud::MakeAccessTokenCredentials ( std::string const &                    access_token,
                                            std::chrono::system_clock::time_point expiration
                                          )
```

Creates credentials with a fixed access token.

These credentials are useful when using an out-of-band mechanism to fetch access tokens. Note that access tokens are time limited, you will need to manually refresh the tokens created by the

See also
https://cloud.google.com/docs/authentication for more information on authentication in GCP.

◆ MakeGoogleDefaultCredentials()

```
std::shared_ptr<Credentials> google::cloud::MakeGoogleDefaultCredentials ( )
```

Creates the default credentials.

These are the most commonly used credentials, and are expected to meet the needs of most applications. The Google Default **Credentials** conform to aip/4110. Consider using these credentials when:

- Your application is deployed to a GCP environment such as GCE, GKE, or Cloud Run. Each of these deployment environments provides a default service account to the application, and offers mechanisms to change the default credentials without any code changes to your application.
- You are testing or developing the application on a workstation (physical or virtual). These credentials will use your preferences as set with gcloud auth application-default. These preferences can be your own GCP user credentials, or some service account.
- Regardless of where your application is running, you can use the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to override the defaults. This environment variable should point to a file containing a service account key file, or a JSON object describing your user credentials.

See also
https://cloud.google.com/docs/authentication for more information on authentication in GCP.

◆ MakeImpersonateServiceAccountCredentials()

```
std::shared_ptr<Credentials>
google::cloud::MakeImpersonateServiceAccountCredentials ( std::shared_ptr< Credentials >  base_credentials,
                                                          std::string                     target_service_account,
                                                          Options                         opts = {}
                                                        )
```

Creates credentials for service account impersonation.

Service account impersonation allows one account (user or service account) to *act as* a second account. This can be useful in multi-tenant services, where the service may perform some actions with an specific account associated with a tenant. The tenant can grant or restrict permissions to this tenant account.

When using service account impersonation is important to distinguish between the credentials used to *obtain* the target account credentials (the `base_credentials`) parameter, and the credentials representing the `target_service_account`.

Use `AccessTokenLifetimeOption` to configure the maximum lifetime of the obtained credentials. The default is 1h (3600s), see IAM quotas for the limits set by the platform and how to override them.

Use `DelegatesOption` to configure a sequence of intermediate service account, each of which has permissions to impersonate the next and the last one has permissions to impersonate `target_service_account`.

Use `ScopesOption` to restrict the authentication scope for the obtained credentials. See below for possible values.

See also

https://cloud.google.com/docs/authentication for more information on authentication in GCP.

https://cloud.google.com/iam/docs/impersonating-service-accounts for information on managing service account impersonation.

https://developers.google.com/identity/protocols/oauth2/scopes for authentication scopes in Google Cloud Platform.

◆ MakeInsecureCredentials()

std::shared_ptr<**Credentials**> google::cloud::MakeInsecureCredentials ( )

Create insecure (aka anonymous, aka unauthenticated) credentials.

These credentials are mostly intended for testing. Integration tests running against an emulator do not need to authenticate. In fact, it may be impossible to connect to an emulator using SSL/TLS because the emulators typically run without secure communication.

In addition, unit tests may benefit from using these credentials: loading the default credentials unnecessarily slows down the unit tests, and in some CI environments the credentials may fail to load, creating confusing warnings and sometimes even errors.

◆ MakeServiceAccountCredentials()

std::shared_ptr<**Credentials**> google::cloud::MakeServiceAccountCredentials (std::string json_object)

Creates service account credentials from a service account key.

A service account is an account for an application or compute workload instead of an individual end user. The recommended practice is to use Google Default **Credentials**, which relies on the configuration of the Google Cloud system hosting your application (GCE, GKE, Cloud Run) to authenticate your workload or application. But sometimes you may need to create and download a service account key, for example, to use a service account when running your application on a system that is not part of Google Cloud.

Service account credentials are used in this latter case.

You can create multiple service account keys for a single service account. When you create a service account key, the key is returned as string, in the format described by aip/4112. This string contains an id for the service account, as well as the cryptographical materials (a RSA private key) required to authenticate the caller.

Therefore, services account keys should be treated as any other secret with security implications. Think of them as unencrypted passwords. Do not store them where unauthorized persons or programs may read them.

As stated above, most applications should probably use default credentials, maybe pointing them to a file with these contents. Using this function may be useful when the service account key is obtained from Cloud Secret Manager or a similar service.

Generated on Thu Oct 27 2022 19:29:21 for Google Cloud C++ Client by doxygen 1.9.1