LEARNING google-cloud-platform

Free unaffiliated eBook created from **Stack Overflow contributors.**

#google-cloud-platform

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: google-cloud-platform

It is an unofficial and free google-cloud-platform ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official google-cloud-platform.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with google-cloud-platform

Remarks

This section provides an overview of what google-cloud-platform is, and why a developer might want to use it.

It should also mention any large subjects within google-cloud-platform, and link out to the related topics. Since the Documentation for google-cloud-platform is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Detailed instructions on getting google-cloud-platform set up or installed.

Read Getting started with google-cloud-platform online: https://riptutorial.com/google-cloud-platform/topic/6864/getting-started-with-google-cloud-platform

Chapter 2: Connect Google Cloud SQL with Apps and Tools?

Examples

How to connect Google Cloud SQL with Apps (like Google App Engine) and Tools like (mySQL workbench)?

In this document we'll see how to create a Google Cloud SQL Instance and connect them in your Google App Engine application and MySQL Workbench admin tool.

Google Cloud SQL:

Google Cloud SQL is a fully-managed database service that makes it easy to set-up, maintain, manage and administer your relational MySQL databases in the cloud.

Google Cloud SQL provides a relational database that you can use with your App Engine application. Cloud SQL is a MySQL database that lives in Google's cloud.

refer:

> https://cloud.google.com/sql/
>
> https://cloud.google.com/sql/docs/

Creating SQL Instances:

A Google Cloud SQL instance is a MySQL database hosted in Google's cloud.

1. Go to the Cloud SQL Instances page in the Google Cloud Platform Console (https://console.cloud.google.com/sql/instances) and Click Create instance.
2. Click Choose First Generation, Enter a name and Choose a tier for the instance and Click Create.
3. After the instance finishes initializing, select the instance to open it.
4. In Access Control > Users, Click Create user account and create a user with name root and specify a password (root_password). This creates the MySQL user 'root'@'%'.
5. In Databases, Click New Database and create a database with a DataBase name (DataBase_Name)

MySQL Workbench:

MySQL Workbench is a unified visual tool for database architects, developers, and DBAs. MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools

for server configuration, user administration, backup, and much more.

refer http://www.mysql.com/products/workbench/

Now we'll see how to connect to your Google Cloud SQL instance database with MySQL Workbench.

Configuring access

1. Go to the Cloud SQL Instances page in the Google Cloud Platform Console and select the instance.

2. In Access Control > IP address, Click Request IPv4 adddress and copy it(Instance_IPv4_address). It is needed to connect your Google Cloud SQL instance database with Admin tools like MySQL Workbench.

   note: You will be charged for IPv4 address @ $0.01 each hour the instance is inactive and $0.1 each hour the instance is active

1. Google 'ip address' to find your public IP address

2. In Access Control > Authorization > Authorized networks, click Add network and enter your IP address.

3. In Access Control > Users, Create a user with username (userName), password (password) and the option 'Allow any host selected'. It is recommended to use a seperate user account to access from WorkBench

Connecting

1. In the MySQL Workbench home view, click New Connection.

1. In the Setup New Connection window, provide a Connection Name, Hostname and Username

1. Click Test Connection. You will be prompted for a password.

1. Once the MySQL connection is made successful, Click OK and click on the saved connection to open SQL Editor

Google App Engine:

Google App Engine is a platform for building scalable web applications and mobile backends. App Engine will scale your application automatically.

refer https://cloud.google.com/appengine

Now we'll see how to set up a connection between an App Engine application and a Cloud SQL instance.

Configuring access

1. Go to the Cloud SQL Instances page in the Google Cloud Platform Console and select the instance.

1. In Access Control > Authorization > Authorized App Engine applications, click Add application ID and enter the application ID. Click Done and Save.

1. In Overview > Properties Copy the 'Instance connection name' (Instance_Connection_Name)

1. In your Google Web Application Project, war/WEB-INF/appengine-web.xml add, true</use-google-connector-j>

Code sample:

An Exaple for Google App Engine - Java Standard Environment

```
public static Connection connect() throws ClassNotFoundException, SQLException {
    String url = null;

    {
        if (SystemProperty.environment.value() == SystemProperty.Environment.Value.Production)
{

            // Connecting from App Engine.
            Class.forName(Messages.getString("com.mysql.jdbc.GoogleDriver"));
            url =
Messages.getString("jdbc:google:mysql://{{Instance_Connection_Name}}/{{DataBase_Name}}?user=root&passwo

        } else {
            // Connecting from an external network or localhost
            Class.forName(Messages.getString("com.mysql.jdbc.Driver"));
            url =
Messages.getString("jdbc:mysql://{{Instance_IPv4_address}}:3306/{{DataBase_Name}}?user={{userName}}&pas

        }

        Connection conn = DriverManager.getConnection(url);

        return conn;
    }
}
```

Read Connect Google Cloud SQL with Apps and Tools? online: https://riptutorial.com/google-cloud-platform/topic/10772/connect-google-cloud-sql-with-apps-and-tools-

Chapter 3: Google App Engine

Introduction

Google App Engine(GAE) is a Platform as a Service(PaaS) offering on Google Cloud Platform, which abstracts away the infrastructure so that you focus on your web application code. App Engine handles both automatic scaling up and down of instances on-demand for your web application based on the number of requests.

App Engine is available in 2 types of environments - Standard and Flexible and supports the following programming languages: Go, Java, Python, PHP, Node.JS, Ruby, and .NET.

Examples

app.yaml for Php app on Flexible environment

```
runtime: php
vm: true
api_version: 1

runtime_config:
  document_root: web
```

Installing gcloud cli

This is how you install the `gcloud` command-line tool, which is a part of the Google Cloud SDK.

```
$ curl https://sdk.cloud.google.com | bash
```

Login and Initialize

Authorize yourself, you will be navigated to Google Sign In page.

```
$ gcloud auth login
```

Initialize or reinitialize the `gcloud`, you can set your configurations here.

```
$ gcloud init
```

Display information about the current `gcloud` environment.

```
$ gcloud info
```

Display the list of active SDK configurations.

```
$ gcloud config list
```

Display the list of accounts whoes credentials are stored on the local system.

```
$ gcloud auth list
```

Display the help options.

```
$ gcloud help
```

Connect to Cloud SQL instance using Php

Create a Cloud SQL instance

```
$dsn = "/cloudsql/PROJECT:REGION:INSTANCE;dbname=DATABASE";
$user = "USER";
$password = "PASSWORD";
$db = new PDO($dsn, $user, $password); //Whatever is your favorite MySQL connection method
```

The important part here is `/cloudsql/PROJECT:REGION:INSTANCE;`. Here we are connecting to the Cloud SQL instance via a Unix socket. You can find this in instance properties as `Instance connection name`.

Write to Cloud Storage

App Engine does not allow writing to files, instead write to and read from Cloud Storage.

Write to Cloud Storage

```
$file = 'gs://<your-bucket>/hello.txt';
file_put_contents($file, 'hello world');
```

Read from Cloud Storage

```
$contents = file_get_contents($file);
var_dump($contents);
```

Logging and Log Monitoring

Simply use PHP's standard syslog function to write logs

```
syslog(LOG_INFO, "Authorized access");
syslog(LOG_WARNING, "Unauthorized access");
```

You can see the logs from "Stackdriver Logging" (https://console.cloud.google.com/logs)

Read Google App Engine online: https://riptutorial.com/google-cloud-platform/topic/9944/google-app-engine

Chapter 4: New Project with Cloud Resource Manager API Client for .NET

Introduction

We will use Google API Client Libraries for .NET for this sample.

There are other libraries. Please see Google Client Libraries Explained for details.

We will use the Cloud Resource Manager API for Creating and Managing Projects.

Let's get started.

Remarks

Putting it all together...

You should have two files. The first file is either called `packages.config` or `project.json`.

Let's name the second file `Program.cs`. All the code that was included in the sections above may be pasted into a single main method:

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

using Google.Apis.Auth.OAuth2;
using Google.Apis.CloudResourceManager.v1;
using Google.Apis.Services;

using Data = Google.Apis.CloudResourceManager.v1.Data;

namespace OurFirstProject
{
    public class Program
    {
        private const string projectId = [YOUR-PROJECT-ID];
        private const string applicationName = "Test";

        public static void Main(string[] args)
        {

            var scopes = new String[] {
                CloudResourceManagerService.Scope.CloudPlatform
            };

            GoogleCredential credential = Task.Run(
                () => GoogleCredential.GetApplicationDefaultAsync()
            ).Result;

            if (credential.IsCreateScopedRequired)
            {
```

```
                credential = credential.CreateScoped(scopes);
            }

            CloudResourceManagerService service = new CloudResourceManagerService(
                new BaseClientService.Initializer()
                {
                    HttpClientInitializer = credential,
                    ApplicationName = applicationName
                }
            );

            Console.WriteLine("1. Create Project");
            Data.Operation operation1 = service.Projects.Create(
                new Data.Project()
                {
                    ProjectId = projectId,
                }
            ).Execute();

            Console.Write("2. Awaiting Operation Completion");
            Data.Operation operation2;
            do
            {
                operation2 = service.Operations.Get(operation1.Name).Execute();
                Console.WriteLine(operation2.Done.ToString());
                System.Threading.Thread.Sleep(1000);
            } while (operation2.Done != true);

            Console.WriteLine();
            Console.WriteLine("Enter to continue");
            Console.ReadLine();

            Console.WriteLine("3. Deleting Project");
            var operation3 = service.Projects.Delete(projectId).Execute();
        }
    }
}
```

If you're using Windows and Visual Studio, "Start"

If you're using Linux, you should first restore the packages, then run the app

```
dotnet restore
dotnet run
```

The output should be similar to:

```
Compiling Projects for .NETCoreApp,Version=v1.1

Compilation succeeded.
    0 Warning(s)
    0 Error(s)

Time elapsed 00:00:01.4161926


1. Create Project
```

```
2. Awaiting Operation Completion



True

Enter to continue

3. Deleting Project
```

The "Awaiting" step will contain blank lines (hopefully) ending in "True".

Examples

Getting Started

The code will work on Windows and Linux.

It runs on .NET Core or .NET Standard. The only difference is that, if you wish to run on .NET Core, you should use `project.json`. If you wish to run on .NET Standard, you should use `packages.config`.

The API Client Library for Cloud Resource Manager API is available on nuget:

https://www.nuget.org/packages/Google.Apis.CloudResourceManager.v1/

The version as of writing is: 1.22.0.809

We'll do this 2 ways:

- Windows with .NET Standard; and
- Linux with .NET Core.

Windows

If you are using Visual Studio, create a new "Visual C#" "Console Application". Otherwise, create a directory for the project and create a file in it called `packages.config`. packages.config is for .NET Standard but we're using .NET Standard with Windows. You may only run .NET Standard on Windows. Replace the contents of `packages.config` with:

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Google.Apis" version="1.22.0"
    targetFramework="net452" />
  <package id="Google.Apis.Auth" version="1.22.0"
    targetFramework="net452" />
  <package id="Google.Apis.CloudResourceManager.v1" version="1.22.0.809"
    targetFramework="net452" />
  <package id="Google.Apis.Core" version="1.22.0"
    targetFramework="net452" />
</packages>
```

Linux

Create a directory for the project and create a file in it called `project.json`. `project.json` is for .NET Core but we're using .NET Core with Linux in this example. You may run .NET Core on Linux or on Windows. Replace the contents of `project.json` with:

```
{
  "version": "1.0.0-*",
  "buildOptions": {
    "debugType": "portable",
    "emitEntryPoint": true
  },
  "dependencies": {},
  "frameworks": {
    "netcoreapp1.1": {
      "dependencies": {
        "Microsoft.NETCore.App": {
          "type": "platform",
          "version": "1.1.0"
        },
        "Google.Apis.CloudResourceManager.v1":"1.22.0.809"
      },
      "imports": "dnxcore50"
    }
  }
}
```

Application Default Credentials

I won't repeat it all here: "Application Default Credentials provide a simple way to get authorization credentials for use calling Google APIs."

If you can use Application Default Credentials, do.

There is an extra step you will need to perform the before first using Application Default Credentials as your identity when calling APIs from your machine:

```
gcloud auth application-default login [yourname@gmail.com]
```

Here's why you'll prefer to use Application Default Credentials:

```
var scopes = new String[] {
    CloudResourceManagerService.Scope.CloudPlatform
};

GoogleCredential credential = Task.Run(
    () => GoogleCredential.GetApplicationDefaultAsync()
).Result;

if (credential.IsCreateScopedRequired)
{
    credential = credential.CreateScoped(scopes);
}
```

...That's all the code you need to authorize calls to (any) Google Cloud API!

We'll use the `credential` object in the next step to make calls against the Google service...

Calling any method (!) on any Google service (!)

Once you become familiar with the code to call one method on one Google service, you will be able to infer how to call *any* method on *any* Google service.

First, we make a connection to the service using the `credential` object instantiated in the previous example:

```
CloudResourceManagerService service = new CloudResourceManagerService(
    new BaseClientService.Initializer()
    {
        HttpClientInitializer = credential,
        ApplicationName = "Our First Google API Client"
    }
);
```

Then we can call methods provided by the service. What methods are available?

https://cloud.google.com/resource-manager/docs/apis

What is the underlying REST request for `Projects.Create`?

https://cloud.google.com/resource-manager/reference/rest/v1/projects/create

OK... Let's write the code.

The code expects a string value for `projectId`. Project IDs are unique identifiers. I recommend you use a system for naming your projects to help you identify them.

`Projects.Create` expects a `Data.Project` object. This object one mandatory property, the Project ID which is all we'll provide but we could also provide a Project Name, Labels, details of the Project's parent etc.

```
Data.Operation operation1 = service.Projects.Create(
    new Data.Project()
    {
        ProjectId = projectId,
    }
).Execute();
```

Project creation is handled asynchronously. We are given an `Operation` object that we must poll to determine when the Project is created. Operations have a Name property that uniquely identifies the operation. The next section of code polls the platform "Are we done yet?". The project will be created when our new operation includes a `Done` property that is `True`.

```
Data.Operation operation2;
do
```

```
{
    operation2 = service.Operations.Get(operation1.Name).Execute();
    System.Threading.Thread.Sleep(1000);
} while (operation2.Done != true);
```

For completeness, and hopefully many years from now after much happy use of your project, you may need to delete your project. We simply call Projects.Delete and provide our Project ID. This also returns an operation and we ought really to poll this operation too until it completes definitively. Our project will then be deleted.

```
var operation3 = service.Projects.Delete(projectId).Execute();
```

That's it!

Read New Project with Cloud Resource Manager API Client for .NET online: https://riptutorial.com/google-cloud-platform/topic/9523/new-project-with-cloud-resource-manager-api-client-for--net

Chapter 5: New Project with Cloud Resource Manager API Client for Python

Introduction

We will use Google API Client Libraries for Python for this sample.

There are other libraries. Please see Google Client Libraries Explained for details.

We will use the Cloud Resource Manager API for Creating and Managing Projects.

Let's get started.

Remarks

Putting it all together...

If you followed the examples above, you should in a directory called `my_project_folder` and it hopefully contains a subdirectory called `venv`.

Ensure your virtualenv is activated.

All the code can be placed in one file, let's call it `create_project.py`.

Create the file `create_project.py` in `my_project_folder`.

```
import json
import time

from apiclient.discovery import build
from oauth2client.client import GoogleCredentials

SERVICE_NAME = "cloudresourcemanager"
SERVICE_VERSION = "v1"

# Don't forget to replace YOUR-PROJECT-ID with your choice of Project ID
# Even though the code deletes the Project, change this value each time
PROJECT_ID = "YOUR-PROJECT-ID"

credentials = GoogleCredentials.get_application_default()

service = build(
    SERVICE_NAME,
    SERVICE_VERSION,
    credentials=credentials)

operation1 = service.projects().create(
    body={
        "project_id": PROJECT_ID
    }
).execute()
```

```
print(json.dumps(
    operation1,
    sort_keys=True,
    indent=3))

name = operation1["name"]
while True:
    operation2 = service.operations().get(
        name=name
    ).execute()
    print(json.dumps(
        operation2,
        sort_keys=True,
        indent=3))
    if "done" in operation2:
        if (operation2["done"]):
            break
    time.sleep(1)

raw_input("Press Enter to delete the Project...")

operation3 = service.projects().delete(
    projectId=PROJECT_ID
).execute()
```

Save the file and, from the command-prompt type:

```
python create_project.py
```

The output should be similar to:

```
{
   "metadata": {
      "@type": "type.googleapis.com/google.cloudresourcemanager.v1.ProjectCreationStatus",
      "createTime": "2017-12-31T00:00:00.000Z"
   },
   "name": "operations/pc.1234567890123456789"
}
...
{
   "done": true,
   "metadata": {
      "@type": "type.googleapis.com/google.cloudresourcemanager.v1.ProjectCreationStatus",
      "createTime": "2017-12-31T00:00:00.000Z"
      "gettable": true,
      "ready": true
   },
   "name": "operations/pc.1234567890123456789",
   "response": {
      "@type": "type.googleapis.com/google.cloudresourcemanager.v1.Project",
      "createTime": "2017-12-31T00:00:00.000Z",
      "lifecycleState": "ACTIVE",
      "projectId": "your-project-id",
      "projectNumber": "123456789012"
   }
}
...
```

```
Press Enter to delete the Project...
```

Examples

Application Default Credentials

I won't repeat it all here: "Application Default Credentials provide a simple way to get authorization credentials for use calling Google APIs."

If you can use Application Default Credentials, do.

There is an extra step you will need to perform the before first using Application Default Credentials as your identity when calling APIs from your machine:

```
gcloud auth application-default login [yourname@gmail.com]
```

Here's why you'll prefer to use Application Default Credentials:

```
scopes = [
    "https://www.googleapis.com/auth/cloud-platform"
]
credentials = GoogleCredentials.get_application_default()
if credentials.create_scoped_required():
    credentials = credentials.create_scoped(scopes)
```

In truth, you can generally get away with just:

```
credentials = GoogleCredentials.get_application_default()
```

...That's all the code you need to authorize calls to (any) Google Cloud API!

We'll use the `credential` object in the next step to make calls against the Google service...

Calling any method (!) on any Google service (!)

Once you become familiar with the code to call one method on one Google service, you will be able to infer how to call **any** method on **any** Google service.

First, we make a connection to the service using the `credential` object instantiated in the previous example:

```
service = build(
    SERVICE_NAME,
    SERVICE_VERSION,
    credentials=credentials)
```

Then, we can call methods provided by the service. What methods are available?

https://cloud.google.com/resource-manager/docs/apis

What is the underlying REST request for Projects.Create?

https://cloud.google.com/resource-manager/reference/rest/v1/projects/create

OK... Let's write the code.

The `create` method expects a body minimally containing the Project ID. Project IDs are unique identifiers. I recommend that you use a system for naming your projects to help you identify them. The method also accepts a Project Name, Labels, details of the Project's parents etc.

```
operation1 = service.projects().create(
  body={
    "project_id": PROJECT_ID
  }
).execute()
```

Project creation is handled asynchronously. We are given an Operation object that we must poll to determine when the Project is created. Operations have a Name property that uniquely identifies the operation. The next section of code polls the platform "Are we done yet?". The project will be created when our new operation includes a `Done` property that is `True`.

```
name = operation1["name"]
while True:
    operation2 = service.operations().get(
        name=name
    ).execute()
    if "done" in operation2:
        if (operation2["done"]):
            break
    time.sleep(1)
```

For completeness, and hopefully many years from now after much happy use of your project, you may need to delete your project. We simply call the delete method and provide our Project ID. This also returns an operation but I'll leave it to you to poll the operation until it completes

```
operation3 = service.projects().delete(
    projectId=PROJECT_ID
).execute()
```

That's it!

Getting Started

You will need to be able to run Python.

Python is available for Linux, Mac OS X and Windows.

I recommend pip and virtualenv.

Pip is the recommend tool for installing Python packages.

The Google Cloud API Libraries are available as pip packages.

A Virtual Environment (aka "virtualenv") is a "tool to keep dependencies required by different projects in separate places".

Create a directory for your project and then:

```
cd my_project_folder
virtualenv venv
```

You may name your virtualenv folder other than "venv" but this name is a useful reminder of the directory's purpose. virtualenv should display something similar to:

```
New python executable in venv/bin/python
Installing setuptools, pip, wheel...done.
```

The virtualenv is created but, to use it, we must `activate` it. When we're done using it, it's a good practice (although not mandatory) to `deactivate` it too.

```
source venv/bin/activate
```

To indicate that we're in the virtualenv called venv, the command-prompt on my Linux machine changes. You will remain in your project directory. Your mileage may vary, but:

```
(venv) user@host
```

Now let's upgrade pip and install the Google API Client Libraries

```
pip install --upgrade pip
pip install --upgrade google-api-python-client
```

All being well if you type `pip freeze`, you should see something similar to this list. Your versions may be higher:

```
pip freeze
google-api-python-client==1.6.2
httplib2==0.10.3
oauth2client==4.0.0
pyasn1==0.2.3
pyasn1-modules==0.0.8
rsa==3.4.2
six==1.10.0
uritemplate==3.0.0
```

Your Python environment is now ready, the Google API Client Libraries are installed. We can write our Python code. Please continue to the next example.

When you're finished with the project, it's a good practice to `deactivate` the virtualenv. Please don't do this now as we're going to write some code. But, when you're done, please return here and:

```
deactivate
```

If you wish to return to the virtualenv, just re-run the `activate` command

Read New Project with Cloud Resource Manager API Client for Python online: https://riptutorial.com/google-cloud-platform/topic/9536/new-project-with-cloud-resource-manager-api-client-for-python

Credits

S. No	Chapters	Contributors
1	Getting started with google-cloud-platform	Community
2	Connect Google Cloud SQL with Apps and Tools?	Newton Joshua
3	Google App Engine	Milindu Sanoj Kumarage, Tuxdude
4	New Project with Cloud Resource Manager API Client for .NET	DazWilkin
5	New Project with Cloud Resource Manager API Client for Python	DazWilkin