VS Code with Python and GCP

|  |  |  |
| --- | --- | --- |
| python-tox - Visual Studio Marketplace | CPython Extension Pack - Visual Studio Marketplace | Google Cloud Logo, symbol, meaning, history, PNG, brand |

Contents

[Installing VS Code 3](#_Toc139188816)

[Setting up VS Code for use with python 3](#_Toc139188817)

[Setting up a project folder 3](#_Toc139188818)

[Connecting to the cloud 4](#_Toc139188819)

[Make a project folder for the function 7](#_Toc139188820)

[IMPORTANT – Create an ‘ignore’ file 8](#_Toc139188821)

[Running the program 10](#_Toc139188822)

[Making changes and re-deploying 11](#_Toc139188823)

[Debugging without deploying 14](#_Toc139188824)

[Interacting with the file system 15](#_Toc139188825)

# Installing VS Code

You can get VS Code from the Microsoft Store or search for it in the browser.

Follow the steps. Accept the default selected items during installation.

Also make sure you have installed Python. I recommend one version below the latest version to avoid bugs.

# Setting up VS Code for use with python

Follow the steps here:

<https://www.pythontutorial.net/getting-started/setup-visual-studio-code-for-python/>

In particular, look at the sections on the right-hand side of the page and read the section on virtual environments.

A picture containing text, font, screenshot, guide

Description automatically generated

# Setting up a project folder

In VS Code, select Terminal > New Terminal if the terminal is not already open.

Change the shell to Command Prompt. Some commands won’t work in PowerShell.

A screenshot of a computer program

Description automatically generated with medium confidence

Create an overall folder (e.g. **mkdir gcp\_test**) which will subsequently have subfolders for each function you want to run. Connecting to the GCP function will create a subfolder for it so this is just an ‘overall’ folder.

# Connecting to the cloud

In this guide, we are going to connect it to an existing GCP project.

First, we need to install the Cloud Code extension in the same way we installed the Python extension earlier.

A screenshot of a computer

Description automatically generated with low confidence

Click on the Cloud Code icon on the left side of the window, then click on Cloud Functions.

You may get a message saying the SDK isn’t installed. Follow the prompt to this guide for installing it: <https://cloud.google.com/sdk/docs/install>

Download it and run the installer. It should look like this.

A screenshot of a cloud setup

Description automatically generated with low confidence

Accept the default options until the screen below.

Deselect the Bundled python option as we already have python installed.

A screenshot of a computer

Description automatically generated with medium confidence

On completion, deselect all the options.

A screen shot of a computer

Description automatically generated with medium confidence

In VS Code, go to the terminal. If it is closed, open a new one, making sure you are using Command Prompt.

Type **gcloud init**.

You will then need to log into you google account and select the GCP project from the list by entering the number. Say no to configuring default location etc.

Go back to Cloud Functions on the left. You may need to log in again.

A screen shot of a computer

Description automatically generated with low confidence

Your functions should now be visible.

A screenshot of a computer program

Description automatically generated with medium confidence

# Make a project folder for the function

If you hover over a function (e.g. **list-buckets**), a cloud icon appears to “Download to a new workspace”. If you are certain that the function works in the cloud, then click this cloud icon and select the ‘overall’ folder you created earlier. There should now be a python project subfolder with the name of the function and containing the main.py and requirements.txt files.

I recommend making a virtual environment in this project folder so that you can make sure you’re using the same versions of libraries that are being used in the cloud.

This page describes how and why virtual environments are used.

<https://www.pythontutorial.net/python-basics/python-virtual-environments/>

In the terminal, make sure you’re in your function’s folder.

A black screen with white text

Description automatically generated with low confidence

Make a virtual environment (VE) using this command.



Activate the VE using this command. The name of the VE will be in brackets before the prompt.

A picture containing text, font, screenshot

Description automatically generated

Activating the VE means that any libraries installed will be in this project folder only, not at the global level.

To install the libraries used by the project in the cloud we can use the requirements.txt file that was downloaded when we connected and created this folder.

A screenshot of a computer program

Description automatically generated with medium confidence



Once these have been installed, you should be able to run the python script.

# IMPORTANT – Create an ‘ignore’ file

Before that, we’ll add some files that will be very important should we make changes and deploy the function back to the cloud environment.

In the terminal, initialise the folder as a git repository.



Create a .gitignore file for python by following these steps.

A screenshot of a computer program

Description automatically generated with medium confidence

Choose python and a .gitignore file will be created.

When we deal with git, any files and folders in the .gitignore are ignored.

Similarly, when dealing with GCP we want to ignore files and folders because if they’re not ignored, the whole of the venv folder and other files will be uploaded to the GCP function in the cloud. This will make the space used by the function larger than 512KB which means code from the inline editor will be lost and the code will be stored in a zip file in cloud storage. Basically, it won’t work.

Copy the .gitignore file and call it **.gcloudignore**.

Open .gcloudignore and add the following lines to it.

A screen shot of a computer

Description automatically generated with medium confidence

The venv folder should already be included further down the file.

With this file present, only the main.py and requirements.txt files should be uploaded to the cloud when we deploy in VS Code. These are the only two files GCP requires to deploy a function, anything else is unnecessary.

# Running the program

Before we run the python script, we need to make sure it contains a line that triggers the function in the cloud.

This is done by calling the function used by the entry point and passing in the trigger url as the request.

The trigger url is shown in the Configuration pane when editing the function:

A screenshot of a computer

Description automatically generated with medium confidence

The function used as the entry point is shown in the Code pane:

A screenshot of a computer

Description automatically generated with medium confidence

The lines to add to main.py are shown here:



To run the program in VS Code, just click the play button in the right-hand corner. This can be changed to debug if necessary.

A screenshot of a computer

Description automatically generated with medium confidence

The python command that runs the script appears in the terminal along with any outputs.

A screenshot of a computer

Description automatically generated with medium confidence

As this has an output, the date and time of the output can be checked for verification that it was successful. In this case, we can look at the file in the storage bucket.

# Making changes and re-deploying

I added a try-except block into the code and reduced the number of rows returned in the list to 3.

A picture containing screenshot, font

Description automatically generated

A black screen with yellow text

Description automatically generated with low confidence

These changes ran without any issues, so I want to re-deploy this updated code to the cloud function.

To do this, click on the Cloud Code add-in on the left-hand side of the window.

A screenshot of a computer program

Description automatically generated with low confidence

Click on the CLOUD FUNCTIONS heading and a cloud with an up arrow appears labelled Deploy.

Click on this, select the project, then select Yes to the pop-up.

A screenshot of a computer

Description automatically generated with medium confidence

In the Output window, you should see it attempting to deploy.

A black background with white text

Description automatically generated with low confidence

And hopefully completing it.

A picture containing text, font, screenshot

Description automatically generated

Check the function in the cloud and refresh to see the latest deployment, it should be just now.

A screenshot of a computer

Description automatically generated with low confidence

It should also appear in the log that it was updated.



You can check your code updates have been made by editing the function and looking at the code.

Here you can see my try-except block is now in the cloud code.

A screenshot of a computer

Description automatically generated with medium confidence

In the TESTING tab you can click the button to test it.



Scroll down to see the log.

We get 3 rows listed, reflecting the change we made in VS Code.

A screenshot of a computer

Description automatically generated with low confidence

# Debugging without deploying

Sometimes when you run the script or debug it in VS Code it may try to deploy it automatically.

To run and debug python locally, you’ll need to make sure that VS Code is using the correct configuration.

You should have a launch.json file by this point in the .vscode folder. If not, click on the Run and Debug button on the left and create it.

A screenshot of a computer

Description automatically generated with medium confidence

If you already have a launch.json file you can add a configuration from the Run menu > Add configuration… and select python.

In the Run and Debug pane, select Python: Current File from the drop-down menu.

There may be more than one configuration in the file. Here is an example:

A screen shot of a computer program

Description automatically generated with medium confidence

A screen shot of a computer

Description automatically generated with medium confidence

When you run or debug it will just run the python and ignore the cloud stuff.

# Interacting with the file system

If your python program interacts with your C drive or network, this will only work locally.

Google Cloud Console cannot see your machine and it is not possible to make it do so.

You’ll get an error when running in the cloud console.

A close-up of a text

Description automatically generated with low confidence

See the Stack Overflow responses below.

A screenshot of a computer

Description automatically generated with low confidence

One way of managing this is to test if the file or folder exists before executing any commands. If it’s local, your program will be able to see the file. If it’s in the cloud, it won’t, and you can manage the subsequent statements.

The logs below show that the function runs to completion in the cloud but manages the interactions with the local file system and reports it in the log.

A screenshot of a computer

Description automatically generated with medium confidence

Alternatively, you can make your function into a web page that browses and uploads a selected file manually.

# Appendix 1: Working python code

import functions\_framework

from google.cloud import storage

import pandas as pd

import fastparquet

import os

from os.path import exists

def download\_blob(bucket\_name, source\_blob\_name, destination\_file\_name):

    """Downloads a blob from the bucket."""

    # The ID of your GCS bucket

    # bucket\_name = "your-bucket-name"

    # The ID of your GCS object

    # source\_blob\_name = "storage-object-name"

    # The path to which the file should be downloaded

    # destination\_file\_name = "local/path/to/file"

    storage\_client = storage.Client()

    bucket = storage\_client.bucket(bucket\_name)

    # Construct a client side representation of a blob.

    # Note `Bucket.blob` differs from `Bucket.get\_blob` as it doesn't retrieve

    # any content from Google Cloud Storage. As we don't need additional data,

    # using `Bucket.blob` is preferred here.

    blob = bucket.blob(source\_blob\_name)

    if os.path.exists('C:\\'):

        blob.download\_to\_filename(destination\_file\_name)

        print(

            "Downloaded storage object {} from bucket {} to local file {}.".format(

                source\_blob\_name, bucket\_name, destination\_file\_name

            )

        )

    else:

        print("ERROR DOWNLOADING: Cannot find local machine in this environment. Cloud console cannot interact with the local file system. ")

def upload\_blob(bucket\_name, source\_file\_name, destination\_blob\_name):

    """Uploads a file to the bucket."""

    # The ID of your GCS bucket

    # bucket\_name = "your-bucket-name"

    # The path to your file to upload

    # source\_file\_name = "local/path/to/file"

    # The ID of your GCS object

    # destination\_blob\_name = "storage-object-name"

    storage\_client = storage.Client()

    bucket = storage\_client.bucket(bucket\_name)

    blob = bucket.blob(destination\_blob\_name)

    # Optional: set a generation-match precondition to avoid potential race conditions

    # and data corruptions. The request to upload is aborted if the object's

    # generation number does not match your precondition. For a destination

    # object that does not yet exist, set the if\_generation\_match precondition to 0.

    # If the destination object already exists in your bucket, set instead a

    # generation-match precondition using its generation number.

    # generation\_match\_precondition = 0

    if os.path.exists(source\_file\_name):

        # blob.upload\_from\_filename(source\_file\_name, if\_generation\_match=generation\_match\_precondition)

        blob.upload\_from\_filename(source\_file\_name)

        print(

            f"File {source\_file\_name} uploaded to {destination\_blob\_name}."

        )

    else:

        print(f"ERROR UPLOADING: Cannot find {source\_file\_name} in this environment. Cloud console cannot interact with the local file system. ")

@functions\_framework.http

def pandas\_processing(request):

    # it is mandatory initialize the storage client

    client = storage.Client()

    # Upload file from local machine or network

    upload\_blob('gcp-python-files', 'C:\dev\drinks.csv', 'drinks.csv')

    #please change the file's URL

    try:

        df = pd.read\_csv('gs://gcp-python-files/drinks.csv', encoding='utf-8')

    except Exception as e:

        print("ERROR: " + str(e))

    print("List csv file header\n")

    print (df.head(3))

    df.to\_parquet('gs://gcp-python-files/drinks.parquet',engine='fastparquet')

    print("Parquet exported to bucket")

    # Download file from gcp bucket to local machine or network

    download\_blob('gcp-python-files', 'drinks.parquet', 'c:\dev\drinks.parquet')

    return f'end of function'

if \_\_name\_\_ == '\_\_main\_\_':

    print("\_\_\n")

    pandas\_processing("https://europe-west2-gcp-python-test-380311.cloudfunctions.net/list-buckets")

Create a folder for your project and activate it by following the instructions in the guide.

<https://www.pythontutorial.net/python-basics/python-virtual-environments/>

The folder is now ready. In VS Code, open the folder by clicking the folder icon on the left-hand side.

A screenshot of a computer

Description automatically generated

The project can be used for a fresh python project from this point. See OTHER DOCUMENT YET TO BE WRITTEN.

C:\Users\mattr\AppData\Roaming\gcloud\application\_default\_credentials.json]