COMM034: Cloud Computing

Lab Session 1

Google App Engine (using Python)

*Versions now specified for libraries to prevent breakage*

Contents

[1 INTRODUCTION 1](#_Toc148620409)

[2 OVERVIEW 2](#_Toc148620410)

[2.1 A NOTE ON NOTATION 2](#_Toc148620411)

[3 PREPARATION AND INSTALLATION 3](#_Toc148620412)

[3.1 Signing up for a Google account 3](#_Toc148620413)

[3.2 Set up a project 6](#_Toc148620414)

[3.3 Download, install, and configure Google Cloud SDK 8](#_Toc148620415)

[4 YOUR FIRST APP ENGINE APPLICATION (EXAMPLE 1) 10](#_Toc148620416)

[4.1 Create a local GAE project 10](#_Toc148620417)

[4.2 Test your application locally 13](#_Toc148620418)

[4.3 Deploy your application to GAE 14](#_Toc148620419)

[4.4 Doing slightly more with the ‘gcloud’ command 15](#_Toc148620420)

[5 EXTENDING FROM THE FIRST APPLICATION (EXAMPLE 2) 16](#_Toc148620421)

[6 TEMPLATES, FORMS, AND EXTERNAL SERVICES (CHARTS USING “REST”) (EXAMPLE 3) 22](#_Toc148620422)

[7 Billing management 29](#_Toc148620423)

[7.1 In the dashboard 29](#_Toc148620424)

[7.2 Using the ‘gcloud’ command to manage billing 29](#_Toc148620425)

[8 Further Exercises – Coursework preparation: 31](#_Toc148620426)

# 

# INTRODUCTION

Google App Engine (GAE), part of Google Cloud (GCP), is generally considered as a cloud computing platform (PaaS) for creating and running web applications. GCP, of course, stands for ‘Google Cloud Platform’, yet offers more than just PaaS. [This is a first welcome to the terminological confusion that pervades, and which this module also attempts to unpick].

The ‘Standard Environment’ for GAE has been free and did not previously require credit card signup in order to get going. In the recent past it only had support for Python 2 – which was becoming increasingly difficult to maintain, and many tools risked problems with backward compatibility. In January 2020, use of the ‘Flexible Environment’ was explored but *very quickly discarded* in favour of the ‘Standard Environment’ **due to serious cost problems**.

* Flexible has rather longer deployment times
* Flexible incurs continuous costs even when nothing is being done – it doesn’t offer “scale-to-zero”
* Defaults on deployment can lead to multiples of costs that are a major trap for new and unwary users
* See, for example: <https://stackoverflow.com/questions/47125661/pricing-of-google-app-engine-flexible-env-a-500-lesson> - with some options for cost reductions at: <https://medium.com/google-cloud/three-simple-steps-to-save-costs-when-prototyping-with-app-engine-flexible-environment-104fc6736495>

For a comparison between Flexible and Standard, see: <https://cloud.google.com/appengine/docs/the-appengine-environments#comparing_high-level_features>.

Standard environment supports Python 3, but does require a credit card for signup – Google provides $300 credit, and promises not to touch the credit card for anything unless you agree to it.

**GAE costs generally are something to be careful of – keeping resource use to sensible levels will avoid problems in Standard, and in Section 7 we even avoid dissipating credits.**

**So, probably better to stay away from *Flexible* unless you often set money on fire.**

GAE does support a number of languages. In Standard, **Python** seemed the most usable. For consistency with examples (and with respect to future work we’ll be doing), we’ll stick with that.

**You may need to invest an extra hour or two in addition to the timetabled session to complete this lab – in part because Google has made some things rather more complex to do in recent years.**

# OVERVIEW

**The aim of this lab is to get you up and running on GAE with Python.**

You will be taken through three ‘simple’ applications that will show you some of what GAE offers.

If you are familiar with Linux, software installations and the like, some of the steps listed may seem obvious. Feel free to **work through at whatever pace you are most comfortable with** – but be careful not to skip over steps. Doing so will reduce the likelihood of things working.

**In case you again need a reminder/tutorial on Linux**, see: <https://web.archive.org/web/20210709080930/http://info.ee.surrey.ac.uk/Teaching/Unix/> - parts one and two of this will get you going. **It is vital that you become comfortable with working with Linux quickly!**

**You *may* need access to your mobile phone and credit card to get started with GAE.**

This lab will focus on using Google’s Cloud for building and deploying a small website.

Understanding of Python will also be beneficial this stage, and you should be prepared to explore GAE with Python further.

**By the end of this tutorial, you should be able to create and host a website with Google App Engine.**

Although you’ll not get as far as this today, an example personal website built using App Engine can be seen at: <https://web.archive.org/web/20141218140710/http://binlialfie.appspot.com/>

## A NOTE ON NOTATION

If lines in the code boxes *start* with characters such as “>”, “>>>”, “$” and so on, do not attempt to use these in your own commands.

**A warning**: cut+paste from this document may lead to issues such as incorrect quote characters being inserted into code. If any problems arise, check that the quote characters are the same first – **‘'** **`** are 3 very different characters to ' and will be interpreted differently also.

**Another warning:** line indentation is **very** important to Python; cut+paste can destroy that also!

# PREPARATION AND INSTALLATION

*Instructions for access from other locations and to other locations (if needed) are provided in the ‘remote’ file for Labs in SurreyLearn. If you are not physically in the labs that you are timetabled in for COMM034, you will need to connect remotely.*

1. Login on one of the lab machines (Heron / Otter)
2. Start a Terminal, which you’ll use subsequently:
   1. Either: click the Ubuntu icon in the upper-left, type "terminal", and select the Terminal application from the results that appear.
   2. Or: use the keyboard shortcut by pressing the Ctrl+Alt+T keys together.
   3. *Remember this sequence, as you’ll use if often*

You will need to:

1. Sign up for a Google account (if you don’t have one)
2. Set up one or more projects
3. [Set](http://code.google.com/appengine/downloads.html) the Google Cloud SDK to use a project

## Signing up for a Google account

***If you already have a GMail account, skip forward to step 2.***

You must have a Google account to use education credits. If you don't have a Google account, you’ll need to create one free to register, but you’ll need to accept the various T&Cs provided).

*Note that instructions for signup up may vary from those shown here.*

1. If you need a Google Account, sign up via: <https://account.google.com> (“Create an account” - NB: Google interfaces are subject to abrupt change and what you see may differ from the image below)

A screenshot of a computer

Description automatically generated

Figure 1: Creating an account with Google, if you don’t already have one.

1. You should now be able to obtain **Google Cloud credits that last for 12 months** - the module leader has obtained these on your behalf from Google.

***This is different to the free trial Google offers*** *where you’ll get more credit but only for 90 days. You can have both, but it’s easier not to so that by default Education credits are being used. See the next section if you do want both.*

*The following instructions may change!*

To avoid them being lost in email, instructions are provided here – visit the Link to obtain the coupon (via email):

|  |
| --- |
| Here is the URL you will need to access in order to request a Google Cloud coupon. You will be asked to provide your school email address and name. An email will be sent to you to confirm these details before a coupon is sent to you.  [Student Coupon Retrieval Link](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fgcp.secure.force.com%2FGCPEDU%3Fcid%3Du2t4geFpZJH3uRi86jD0pnvDEKeaNyfh6CYUJEWqXfOrGgpRsZR1gOfziYmxweJX%2F&data=05%7C02%7Cl.gillam%40surrey.ac.uk%7Cc7539f2f6504436764ab08dc00ac23b4%7C6b902693107440aa9e21d89446a2ebb5%7C0%7C0%7C638385986533228067%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=1AM0nWXdW49htum0T%2BHsdPr19%2BWgcdolmyfcFRG200E%3D&reserved=0)   * You will be asked for a name and email address, which needs to match the **surrey.ac.uk** domain. A confirmation email will be sent to you with a coupon code. * You can request a coupon from the URL and redeem it until: 5/29/2024 * Coupon valid through: 1/29/2025 * You can only request ONE code per unique email address. |

1. Redeem the coupon per the instructions that will be provided alongside the coupon code in the email.

Projects should all be associated to these credits – unless you additionally go for Google’s free trial, in which case you’ll want to pay close attention to where the bills are going! However, **being sensitive as to what is causing cost even to credits can be beneficial for the future.**

The Google credit amount should be more than enough for what you need in the standard environment - when used in line with this module.

## Full Google Cloud - optional

You should be able to get the 90 day Free Trial as well – indeed, it may be promoted to you in various ways. [Google offers some advice about this](https://services.google.com/fh/files/helpcenter/cloud_edu_free_trial_warning.pdf). You don’t need this for the module.

Note that this promises not to charge you for **90 days** and unless you *explicitly enable billing* – certain things are free beyond this, up to some reasonable limits; billing avoidance is addressed in section 7. It’s worth putting a calendar reminder for about 85 days in future that anything from Google about credits having ended relates to this, not the Education credits – and you’ll not then be concerned at the arrival of an email telling you this.

Your credit card details are needed for verification only - as the screen image shows, to “make sure you are not a robot” (robots, apparently, can’t get credit cards).

“Free tier” limits, and $300 for 90 days both discussed in more detail at: <https://cloud.google.com/free/>

Graphical user interface, application

Description automatically generated

The previously more generous *12 month* explanation used to look like this:

A white background with black text

Description automatically generated

***Note:*** *You may need to be logged in to GMail to verify your Google Cloud registration.*

## Set up a project

1. In Google Cloud (<https://console.cloud.google.com>), click ‘Select from’ (or similar) to the right of “Google Cloud” in the top blue bar.
2. From here, you should be able to create a **New Project** (top right of pop-up screen).
3. Provide a project name if you wish to – a **Project ID** will be assigned at random (‘universal-fort-265023’ in the screenshot above). If you want your own Project ID as well, click **EDIT** next to this name – you will then need to provide an ID that is globally unique (usually found by trial-and-error).

The **Project ID** (below Project name) will be the appspot endpoint for your application – so we will refer to this interchangeably as the ***application*** from here on.

If it is possible to set the **Billing account** make sure that you associate the project’s to “Billing Account for Education”, or the name that came from the coupon being associated (not *My Billing Account*) – **this is particularly important if you’ve opened a Google Cloud account using your credit card**.

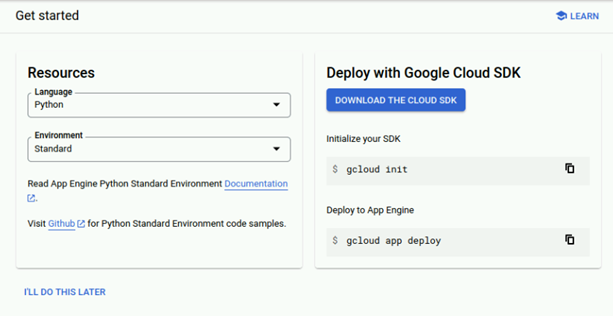
*You could create 3 applications here for what the rest of this lab will do – but you don’t need to, as we will reuse one. The Project ID for each new project will need to be globally unique.*

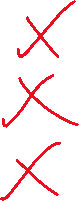
Click CREATE, and wait for the Notification popup to offer SELECT PROJECT and then click that. Or select the project next to ‘Google Cloud Platform’ (blue bar across the window).

|  |
| --- |
| ***Note:*** *For your first application, you may need your mobile phone in order to receive an SMS message containing a code (activation only needed once), and only one number is allowed to register to one account.*  *UK country code is +44, so a mobile number would be something like +44 7123 456789* |

1. From the left menu, click on **App Engine** 🡪 **Dashboard**, and in the App Engine screen click **Create Application**.
2. Select “europe-west2” as the region and click **Next**.
3. After a short time, make sure **Python** is selected from the Language dropdown, and the **Standard** environment (note that this may require billing to be enabled at some point but not now).

The screen should look something like:

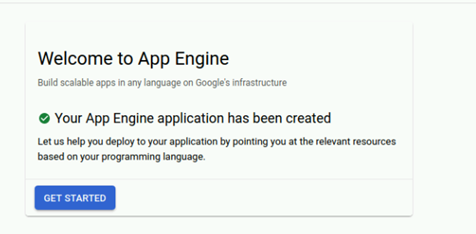




1. **DON’T** click to Download the Cloud SDK, and ignore what’s written below that.
2. Click



1. You should see a screen showing that your application has been created. You need do no more in the dashboard at this point – and you can avoid going through any offered Tutorials also.



* You should now have a Project ready for use. But you will not now use it until section 4.3 after you have set up the SDK and tested an application.

## Check project billing (optional)

1. Visit <https://console.cloud.google.com/billing>. Projects should automatically associate to the “Billing Account for Education” or similar.

However, if you took the full free trial as well, or if you ever enable billing, you might see two accounts – probably something like “Billing Account for Education” and “My Billing Account”. At which point you would have the option to choose where credits are deducted from or to give Google money (if you’ve upgraded).

If no choice is available, this isn’t something to worry about yet.

## Download, install, and configure Google Cloud SDK

You should have started the degree programme with a good allocation of home directory, but you’ll want to know how much you've consumed before proceeding as this will influence where you put things (on Linux).

~~In the Terminal:~~

|  |
| --- |
| ~~quota | tail -1 | awk '{print "Disk is " $1/$2\*100 "% full"}'~~ |

~~If you're 50% or above, and started in September/October, you already risk running out. If you go to 100%, you may become unable to login. Due to the risk of this, we’ll put the Google Cloud SDK somewhere that it will not take up lots of this space (needs about~~ **~~2GB~~**~~) –~~ **~~with the added difficulty that if you move to a different lab machine it will not be there!~~** ~~If you have to move because the machine you used is unresponsive, you’ll need to reinstall using the instructions~~ **~~only~~** ~~in the remainder of this subsection.~~

In the recent past, the “quota” command would help us to know how much space remained in home directories. This seems presently not to work, so we can’t rely on it. Instead, we’ll look elsewhere.

In the Terminal from above, run commands **EXACTLY as they are shown**.

1. Check there’s available space for the install:

|  |
| --- |
| df -h /scratch |

Make sure that under “Avail” there is a number to the left of the letter ‘G’ that is greater than 8. If not, the disk risks becoming too full to work on – **move to another machine** and start again from the above command.

Provided that there’s enough space on the machine you’re using:

|  |
| --- |
| mkdir -m700 /scratch/COMM034-$USER |

This will create a new folder that has your username appended. This folder will then be available ON THIS MACHINE ONLY.

If you move to a different machine, **you will need to take it with you** - THE FOLDER WILL NOT FOLLOW YOU BY ITSELF – but it could be copied across the lab reasonably quickly.

1. With suitable available space, create your folder (run the command below **EXACTLY** as shown):

|  |
| --- |
| cd /scratch/COMM034-$USER |

1. Download the Google Cloud SDK:

|  |
| --- |
| wget https://dl.google.com/dl/cloudsdk/channels/rapid/downloads/google-cloud-cli-457.0.0-linux-x86\_64.tar.gz |

1. Unpack it:

|  |
| --- |
| tar -xf google-cloud-cli-457.0.0-linux-x86\_64.tar.gz |

1. Install it (-q means ‘quietly’ – otherwise you’ll have 3 questions to answer but would accept the defaults anyway):

|  |
| --- |
| ./google-cloud-sdk/install.sh -q **--path-update true** |

1. To add useful settings to your user profile:

|  |
| --- |
| source /scratch/COMM034-$USER/google-cloud-sdk/path.bash.inc  source ~/.bashrc |

1. Add the Python App Engine components to the Google Cloud SDK install:

|  |
| --- |
| gcloud components install app-engine-python -q |

1. The gcloud setup should now be usable – to see what the range of options for this command is:

|  |
| --- |
| gcloud |

All being well, and with messages that you’ve not told it what to do, you should see the screen below:

*Hit <space> until you see ‘END’ at the bottom of the window, then press Q.*

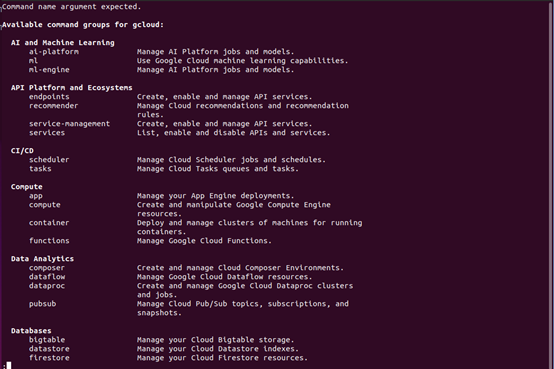


Figure 2: Response expected from gcloud

1. Use the command below to specify which of your App Engine projects to deploy code to:

|  |
| --- |
| gcloud init |

As part of this, you will need accept to login (‘Y’) using your Google account, via a browser. If you have just the one Project ID, it will be selected for you; if you created others, you’ll have a choice to make of which to use for the next steps.

*Your credentials will be being stored in your Home Directory under ~/.config/gcloud rather than in the /scratch folder.*

With 2 step verification, there will likely be challenges to respond to using whichever device(s) you have associated to your Google account. Clicking **Allow** in the resulting browser page should complete the process. Finally, back in the terminal, make sure that your newly created project has been chosen. You should now be ready to work with your project(s).

Having done this, your setup should be ready to use.

**You only need to set this up again if you move to a different lab machine – but would not need to repeat steps 6, 8 and 9.**

* Use CUT+PASTE from this document to try to avoid *most* syntax errors in typing in commands.

* Incorrect commands, file names or file locations will slow progress. AND try to understand what each Linux command does (you will need more like these later in the module).

# YOUR FIRST APP ENGINE APPLICATION (EXAMPLE 1)

The following involves use of Linux. The same approach should work in Windows, but equivalent commands will need to be used.

In this section, you will:

1. Create a *local* GAE project
2. Test your application locally
3. Deploy the project to the Cloud – (appspot.com)

Visit <https://versatile-nomad-342016.ew.r.appspot.com/hello> to see a finished version of this first application (“Hello World” for a cloud service, though with an address slightly different to what you will start with), and <https://versatile-nomad-342016.ew.r.appspot.com/> to see a finished version of the 3rd application.

## Create a local GAE project

1. On the lab machine (Heron, or Otter) in a Terminal, create a working folder for GAE:

|  |
| --- |
| mkdir -p ~/COMM034/lab1  cd ~/COMM034/lab1 |

1. Create a file defining what libraries the Python application is going to use - **requirements.txt**. You can use any text editor you like, here we use **gedit –** but you could use **nano** or **vi** or ….

|  |
| --- |
| gedit requirements.txt |

1. Include the library (and version) specifications shown below, and Save – this provides for Flask, a web framework, Gunicorn, a lightweight WSGI server, and to ease life for future labs, boto3 which we’ll subsequently use to integrate with AWS.

*During the module run in 2022, but never previously, we also had to specify versions of* jinja2*,* werkzeug*, and* itsdangerous*. App Engine defaults to latest versions if they have not been pinned by version numbers, and newer versions had not been properly built into App Engine, leading to a somewhat broken setup, with error messages for usually starting: “ImportError: cannot import name”. Having happened once (in ~15 years) that could repeat so we’re adding known-working versions here.*

|  |
| --- |
| Flask==3.0.0  gunicorn==21.2.0  boto3==1.29.0 |

1. Install these requirements:

|  |
| --- |
| pip3 install -r requirements.txt --user |

1. Create an application definition file: **app.yaml**. (The final character is a lowercase “L”, rather than number “1” – small, but utterly vital details).

|  |
| --- |
| gedit app.yaml |

1. Include the following in app.yaml, and Save. The default entry point for an application involves looking for main.py, and the variable ‘app’ inside it. Here, we’re overriding to use **index**.py but still look for ‘**app**’ (so, **index:app**)

|  |
| --- |
| runtime: python312  entrypoint: gunicorn -b :$PORT index:app |

For more information about app.yaml, see: <https://cloud.google.com/appengine/docs/flexible/python/reference/app-yaml>

1. Now create the code for the application: **index.py**, with the content below:

|  |
| --- |
| import os  import logging    from flask import Flask    app = Flask(\_\_name\_\_)    # various Flask explanations available at:  # https://flask.palletsprojects.com/en/1.1.x/quickstart/    @app.route('**/**')  # A Hello World message to show that at least something is working  def hello():  **return 'Hello World!'**    @app.errorhandler(500)  # A small bit of error handling  def server\_error(e):  logging.exception('ERROR!')  return """  An error occurred: <pre>{}</pre>  """.format(e), 500    if \_\_name\_\_ == '\_\_main\_\_':  # Entry point for running on the local machine  # On GAE, endpoints (e.g. /) would be called.  # Called as: gunicorn -b :$PORT index:app,  # host is localhost; port is 8080; this file is index (.py)  **app.run**(host='127.0.0.1', port=8080, debug=True) |

1. Save the file!

If you copy and paste the code, you may need to check if single quote characters copy correctly.

The entry point for the application running locally is **app.run**. This runs as a lightweight HTTP server, with route callbacks – so when a call is made to ‘**/**’ the function immediately following the route line is executed. Here, it just returns a text string.

* **Statements must be indented (align) as shown**; pasting this from the PDF may not always work directly – if you were to cut+paste, and lose all the layout, you’d need to fix it!.
* Your application is nearly ready to be run – first, test locally; then deploy!

With installed requirements, an application definition in app.yaml, and code in index.py, you should be ready to run on the local machine.

## Test your application locally

1. Run the application using the following

|  |
| --- |
| python3 index.py |

The Terminal is now unresponsive – so **open another Terminal** to do the next step.

1. Check that you can get the Hello World message (it will appear immediately before the command prompt) – we’ll be making calls as if to APIs using curl in future:

|  |
| --- |
| curl "http://localhost:8080" |

1. If the response is there, and no errors appear, the same should work in Firefox:

|  |
| --- |
| firefox <http://localhost:8080> |

*The ‘&’ character runs Firefox in the background, keeping the terminal usable for other purposes.*

* If you can see text saying Hello World! (or anything you might change it to), congratulations, your first Python application *suitable for* Google App Engine is running.

**Note:** *You can make changes to the code while the app is running – e.g. to change the message printed on the webpage. Just save the changes to the file, and then refresh the webpage.*

1. **If you cannot get this to work…..** the lab code can be found in a Zip file under Labs in SurreyLearn’s module content. The code for this can be found there in lab1, and in folder **ex1** for this first application, and ex2/ex3 for the next 2.

It is certainly worth checking what you’ve got that differs from what is provided - that you have the same files, and that the contents of each file is right. The Linux application ‘diff’ can be used to compare 2 files.

1. Terminate the web application - in Linux or MacOS, press **Ctrl+c** in the Terminal from which you launched it; in Windows, you may need to close the cmd window directly if Ctrl+c does not work.

Once the application is terminated, you will no longer be able to access [localhost:8080](http://localhost:8080).

## Deploy your application to GAE

Assuming all went well before this point, you have a working Python-based app and everything else needed to deploy to GAE. Relatively recently, Google added yet another step to this - you first need to Enable GAE to build things in your account – at this point, just for App Engine.

1. In the Google Cloud console, <https://console.cloud.google.com/> use the search box in the blue bar to locate ***Cloud Build Settings***. There, **Enable** the service. If you can then ‘VIEW API’ and see ‘Manage’ you have done all you need to. Google might take you on a short tour of other things that need to be enabled – these are side-effects of increasing complexity. *Just don’t go into creating any* Credentials *unless otherwise forced.*
2. Back in the Terminal you should now need just one command (and lots of patience):

|  |
| --- |
| gcloud app deploy |

After some time…(5 minutes?) you should see:

|  |
| --- |
| *File upload done.*  *Updating service [default]...done.*  *Setting traffic split for service [default]...done.*  *Deployed service [default] to [……appspot.com]*  *You can stream logs from the command line by running:*  *$ gcloud app logs tail -s default*  *To view your application in the web browser run:*  *$ gcloud app browse* |

* Check the response from (your project’s equivalent of) [https://*myappidentifier*.appspot.com](https://myappidentifier.appspot.com) (i.e. using your Project ID within the address) using *curl*.
* Go to (your project’s equivalent of) [https://*myappidentifier*.appspot.com](https://myappidentifier.appspot.com) (i.e. using your Project ID within the address) using the web browser.
* If all went well your application produces a very simple output – and is being **run in Google’s Cloud**

## Doing slightly more with the ‘gcloud’ command

The ‘gcloud’ command can be used for various other tasks (section 7 will demonstrate this). For example to see current project settings, and then all the projects you have:

|  |
| --- |
| gcloud config list  gcloud projects list |

And if you wanted to change to another of your listed projects (if you have more than one:

|  |
| --- |
| gcloud config set project OTHERPROJECT |

Replacing OTHERPROJECT with the PROJECT\_ID provided by a project list.

# EXTENDING FROM THE FIRST APPLICATION (EXAMPLE 2)

The first application, despite running in Google’s Cloud, is incredibly basic.

Using Flask, we can build a bit more capability – things will begin to become a bit more complicated, but you should see some value of this subsequently.

For more about Flask see, e.g. <https://flask.palletsprojects.com/en/1.1.x/quickstart/>

In this section, you will:

1. Construct a slightly more complex Cloud application that uses the GAE framework
2. Split functions by **route** (these relate code to folder/page requests)
3. Work with some relatively simple webpages –

Note that you will not need to learn **the entirety of** HTML/CSS for this, and are not expected to either. In fact, you should be able to ignore CSS almost entirely – it’s largely just to offer up visual examples and keep formatting away from HTML to keep the HTML cleaner.

If you want/need to work through a tutorial on HTML, try <https://www.w3schools.com/html/html_intro.asp> - what you’ll likely find to be useful later in the module just involves being aware of forms and tables:

* 1. <https://www.w3schools.com/html/html_forms.asp>
  2. <https://www.w3schools.com/html/html_tables.asp>

1. First, make a change of route to that already in index.py:

|  |
| --- |
| # change from @app.route('/') to  @app.route('/hello') |

If you run as before (using python3), you would have to call localhost:8080/hello to get the response (something that should at least show things are working.

We don’t need to change **app.yaml**, but we’ll add some things that are useful: folders for **templates**, and **static** files, and then we can get on with editing **index.py** to do the work. (the **index.pyc** file contains Python bytecode, akin to .class files in Java)

1. In the same folder as index.py:

|  |
| --- |
| mkdir templates static |

We use the **templates** folder to keep HTML files, which will be rendered (processed, and printed), and a **static** folder to keep files which are simply delivered without any intervention, such as css files, image files, and documents.

*If you want to create another project in Google Cloud to deploy this to, do so - you can switch between them (see Section 4.4) using:*

*gcloud config set project [PROJECT\_ID]*

1. Create **templates/index.htm** with the following content:

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />  <link href="**/static/def.css**" rel="stylesheet" type="text/css" />  <title>first page</title>  </head>  <body>  <div id="page">  <h1>I am student from University of Surrey. <br/><br/></h1>  <h1>Here is my First Page.<br/></h1>  <h3> if you like, you can switch to my <a href="page2.htm">Second Page</a>.</h3>  </div>  </body>  </html> |

1. **templates/page2.htm** needs tocontain the following:

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />  <link href="**/static/def.css**" rel="stylesheet" type="text/css" />  <title>second page</title>  </head>  <body>    <div id="page">  <h1>Here is my Second Page.<br/></h1>  <h3> if you like, you can switch to my <a href="index.htm">First Page</a>.</h3>  </div>  </body>  </html> |

1. Both of those files need the **def.css** file in the **static** folder: This applies the same formatting to various HTML elements, offering look and feel consistency (though somewhat ‘grey’ here).

|  |
| --- |
| /\* CSS layout \*/  body {  padding: 0;  text-align: justify;  font: 15px Arial, Helvetica, sans-serif;  color: #626262;  }  h1, h1 a, h2, h2 a, h3, {  text-decoration: none;  font-family: Tahoma, Georgia, "Times New Roman", Times, serif;  font-weight: normal;  color: #111111;  }  h1 {  letter-spacing: -1px;  font-size: 1.8em;  font-family: Verdana, Arial, Helvetica, sans-serif;  }  h2 {  letter-spacing: -1px;  font-size: 1.5em;  }  h3 {  font-size: 1.2em;  }  #page {  margin-left: 80px;  background: #AAAAAA;  } |

**Note:** *Both HTML pages will use def.css, so it is vital that all pages exist in the correct locations and references to CSS (****link href="/static/def.css"****) can be fully resolved.*

1. Now, edit **index.py** to show the following (new parts in bold):

|  |
| --- |
| import os  import logging  from flask import Flask**, request, render\_template**  app = Flask(\_\_name\_\_)  # various Flask explanations available at: https://flask.palletsprojects.com/en/1.1.x/quickstart/  **def doRender(tname, values={}):**  **if not os.path.isfile( os.path.join(os.getcwd(), 'templates/'+tname) ): #No such file**  **return render\_template('index.htm')**  **return render\_template(tname, \*\*values)**  @app.route('/hello')  # Keep a Hello World message to show that at least something is working  def hello():  return 'Hello World!'  **# catch all other page requests - doRender checks if a page is available (shows it) or not (index)**  **@app.route('/', defaults={'path': ''})**  **@app.route('/<path:path>')**  **def mainPage(path):**  **return doRender(path)**  @app.errorhandler(500)  # A small bit of error handling  def server\_error(e):  logging.exception('ERROR!')  return """  An error occurred: <pre>{}</pre>  """.format(e), 500  if \_\_name\_\_ == '\_\_main\_\_':  # Entry point for running on the local machine  # On GAE, endpoints (e.g. /) would be called.  # Called as: gunicorn -b :$PORT index:app,  # host is localhost; port is 8080; this file is index (.py)  app.run(host='127.0.0.1', port=8080, debug=True) |

1. *Try to determine what this is doing in the flow from a route through to a resulting page, although we’ll see a bit more about this later as well when it’s re-used. It is not important to understand every single line here though.*
2. *The entry point to the application is the same, but this now catches all possible page requests except for those involving ‘static’ pages (see if you can point the browser at /static/def.css)*
3. *The* ***doRender*** *function works with page templates – if no specific page name is provided (‘/’) it renders* ***index.htm.*** *It checks first if a template, e.g. page2.htm, can be found in* ***templates/*** *for this. If it’s there, it works on any data given with respect to it; if it’s not found, it does ‘something else’.*
4. *In terms of the ‘something else’, ask it for a page that does not exist? For example page3.htm*
5. Run the application locally:

|  |
| --- |
| python3 index.py |

If there is no error, visit [localhost:8080](http://localhost:8080) in a web browser and you should be able to switch between the 2 pages (Figure 3):

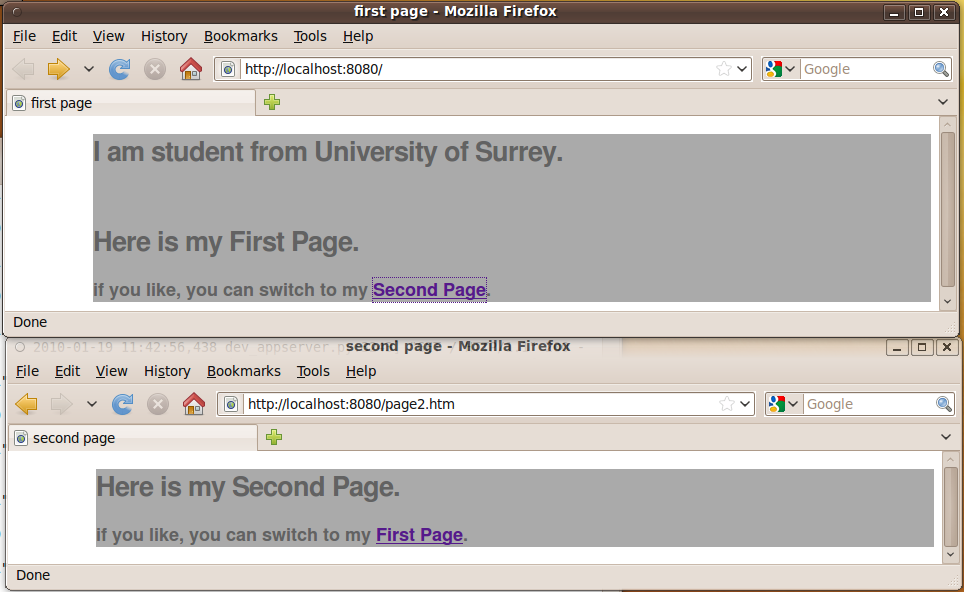


Figure 3: The local application view

**Note:** Again you can make changes to the code while the application is running; remember to save the changes and refresh the webpage.

Recall from 4.2 that the Terminal is now unresponsive – so you’ll need to **open another Terminal** and then you can try this also, from the command line (what happens with the final ‘curl’?):

|  |
| --- |
| curl "http://localhost:8080"  curl "http://localhost:8080/hello"  curl "http://localhost:8080/page2.htm"  curl "http://localhost:8080/page3.htm" |

1. **If you cannot get this to work…..** again, look to the lab code Zip file on SurreyLearn – lab1/**ex2** for this application. Try that instead, and then compare with what you have – where are the differences?

* Refer to section 4.3 for the GAE deployment, and you should be able to deploy your application “to the Cloud”

**Note:**

1. *If you ask via the browser for a page that does not exist - for example page3.htm,* ***could you explain how the code handles this?***

# TEMPLATES, FORMS, AND EXTERNAL SERVICES (CHARTS USING “REST”) (EXAMPLE 3)

In the previous section, there was still a reasonable amount of repetition which would lead to maintenance issues. Using templates in the GAE framework simplifies this.

In this section, you will:

1. Use templates to simplify your Cloud application
2. Add interaction using forms
3. Add a software (charts) service – possibly 2, depending on whether Google Image Charts have become fully switched off or not.

If you want to keep the previous version, make a copy of the entire folder and make changes to the copied files; it can be faster to just change the files that you have.

1. Edit **templates/\_base.htm** to contain the following (either make changes so the file matches, or delete the content and add what is below):

|  |
| --- |
| <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />  <link href="**/static/def.css**" rel="stylesheet" type="text/css" />  <title>  {% block **title** %} replace title {% **endblock** %}  </title>  </head>  <body>    <div id="page">  {% **block pageblock**%}  replace page  {% **endblock**%}  </div>  <div id="footer">  <p>&copy; 2021 University of Surrey</p>  </div>    </body>  </html> |

This is the basis for other pages – which will need to provide a **title** and a **pageblock** – and so the initial lines, dealing with items from elsewhere, and the copyright footer, need not be replicated.

*In* ***<title>****, the block* ***{% block title %} replace title {% endblock %}*** *would be replaced by the specific title for each page, and a similar block for page content*

1. Edit **templates/index.htm** so that it will use **\_base.htm** and makes a call to include a small but simple form in the pageblock – the complete code that does this is:

|  |
| --- |
| {% extends "\_base.htm" %}  {% **block title** %}  Page 1  {% endblock %}  {% **block pageblock** %}  <div id="up">  <h1>I am a student from University of Surrey. <br/><br/></h1>  <h1>Here is my First Page.<br/></h1>  <h3>if you like, you can switch to my <a href="page2.htm">Second Page</a>.</h3>  </div>    **{% include 'form.htm' %}**  {% endblock %} |

1. Create **templates/form.htm** which will be called in by the above ***include*** statement:

|  |
| --- |
| <div id="down">  <h3>Numbers of MPs in Parliament :&nbsp;</h3>  <form id="stats" method="post" action="/calculate">  <h4>Labour: <input type="text" name="labour" /><br/></h4  >  <h4>Conservative: <input type="text" name="conservative"  /><br/></h4>  <h4><input type="submit" name="draw" value="Draw Chart"  />  <input type="submit" name='cancel' value="Cancel"  onclick="window.location='/'; return false;" /><br/></h4  >  </form>    {% if note %}  <h3 style="color: #11FFFF">  {{ note }}  </h3>  {% endif %}    </div> |

**Note:** *We now only need replace* ***blocks*** *for a page – we can* include*, for example, a different* form *in the same page just by creating a new HTML fragment that contains it and including that file instead. Here, we have extended* ***index.htm****, to* ***include*** *a form that has two fields to input numbers: of Labour and Conservative MP numbers – we are separating out the form from other content. We will use this to generate simple pie charts (possibly 2, depending on Google Charts). This will be shown in a final page,* ***chart.htm****, which also extends from* ***\_base.htm****.*

1. Create **templates/chart.htm**:

|  |
| --- |
| {% extends "\_base.htm" %}  {% block title %}  Page 1  {% endblock %}  {% block pageblock %}  <div id="up" style="text-align: right;">  <h3><a href="index.htm">back to first page</a><br/></h3>  </div>  <div id="up">  <h2>MPs in Parliament : <br/><br/></h2>  <h3>Using Image Charts - 3d (p3) falls back to 2d</h3>  <img src="https://image-charts.com/chart?cht=p3&chd=a:{{ **data** }}&chs=250x100&chl=Lab|Con&chco=FF2027,FFFF10"  classed="displayed" alt="Graph: no internet connection." align="center" /><br/><br/>  <h3>Using deprecated Google Charts</h3>  <img src="http://chart.apis.google.com/chart?cht=p3&chd=t:{{ **data** }}&chs=250x100&chl=Lab|Con"  classed="displayed" alt="Graph: no internet connection." align="center" /><br/><br/>    </div>  {% endblock %} |

***Note:*** *The simple, one-call, Chart API from Google has been deprecated for a time already and was apparently switched off. “*This API is deprecated in 2012 and was turned off on March 18, 2019”:.*<https://developers.google.com/chart/image/>.*

*A newer Google Chart API exists but requires more effort for usage (JavaScript).*

*Image Charts is provided as a similarly simple, though not as visually pleasing, alternative. For more information on the newer version of Google Charts, see:* <https://developers.google.com/chart/>

1. Edit **templates/page2.htm** to use the template:

|  |
| --- |
| {% extends "\_base.htm" %}  {% block title %}  Page 2  {% endblock %}  {% block pageblock %}  <div id="up">  <h1>Here is my Second Page.<br/></h1>  <h3>if you like, you can switch to my <a href="index.htm">First Page</a>.</h3>  </div>    <div id="down">  <h3><a href="http://www.surrey.ac.uk">Go to University Home Page</a></h3>  </div>  {% endblock %} |

1. And, finally, **def.css** (**static** folder) has some additions:

|  |
| --- |
| /\* CSS layout \*/  body {  text-align: justify;  font: 15px Arial, Helvetica, sans-serif;  color: #626262;  }  h1, h1 a, h2, h2 a, h3, {  text-decoration: none;  font-family: Tahoma, Georgia, "Times New Roman", Times, serif;  font-weight: normal;  color: #111111;  }  h1 {  letter-spacing: -1px;  font-size: 1.8em;  font-family: Verdana, Arial, Helvetica, sans-serif;  }  h2 {  letter-spacing: -1px;  font-size: 1.5em;  }  h3 {  font-size: 1.2em;  }  IMG.displayed{  display: block;  margin-left: 200px;  }  #page {  background: #AAAAAA;    }  #up {  margin-left: 10px;  }  #down{  margin-left: 300px;  }  #footer p {  clear: both;  padding: 10px 0 10px 0;  background: #757575;  text-align: center;  font-size:1em;  color: #BABABA;  height: 20px;  }  #footer a{  color: #BABABA;  } |

1. Now we need **index.py** to account for this, by adding a route that handles the form – add the following BEFORE the “catch all” [*stated by the comment*] routing code:

|  |
| --- |
| # Defines a POST supporting calculate route  @app.route('/calculate', methods=['POST'])  def calculateHandler():  if request.method == 'POST':  l = request.form.get('labour')  c = request.form.get('conservative')  if l == '' or c == '':  return doRender('index.htm',  {'note': 'Please specify a number for each group!'})  else:  total = float(l) + float(c)  lP = int(float(l)/total\*100)  cP = int(float(c)/total\*100)  return doRender('chart.htm', {'**data**': str(lP) + ',' + str(cP)})  return 'Should not ever get here' |

We added a **CalculateHandler** class to handle the form submission, which, given data **POST**ed to the form will take the user input that was provided in the two text areas and use them. If any input is null, we inform the user to input again. When both inputs are provided, we obtain the values, determine percentages, then pass a list of 2 values (as the 3rd argument to doRender) to the chart.htm page as **data**.

In **chart.htm**, we take this result and represent it with Google Chart – they are substituted into chart calls:

http://chart.apis.google.com/chart?cht=p3&chd=t:**{{ data }}** …

https://image-charts.com/chart?cht=p3&chd=a:**{{ data }} …**

e.g.

<http://chart.apis.google.com/chart?cht=p3&chd=t:40,25&chs=250x100&chl=Lab|Con>

<https://image-charts.com/chart?cht=p3&chd=a:40,25&chs=250x100&chl=Lab|Con&chco=FF2027,FFFF10>

1. Test locally: If there is no error, test using <http://localhost:8080> in your web browser. You should see the page as follows (Figure 4, Figure 5).

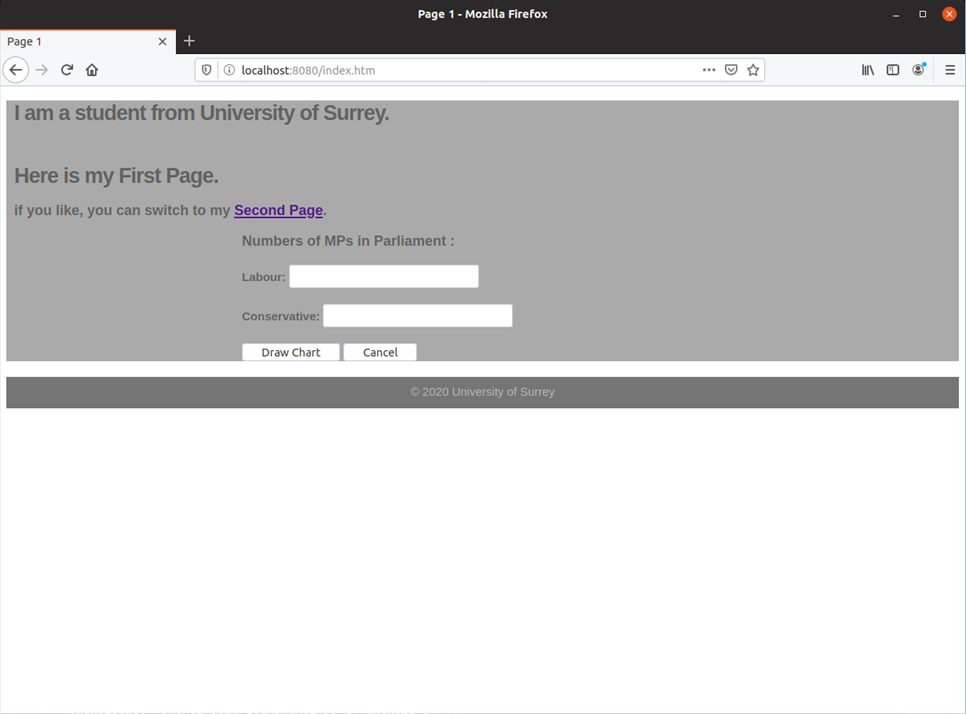


Figure 4: A dynamic local application – index page

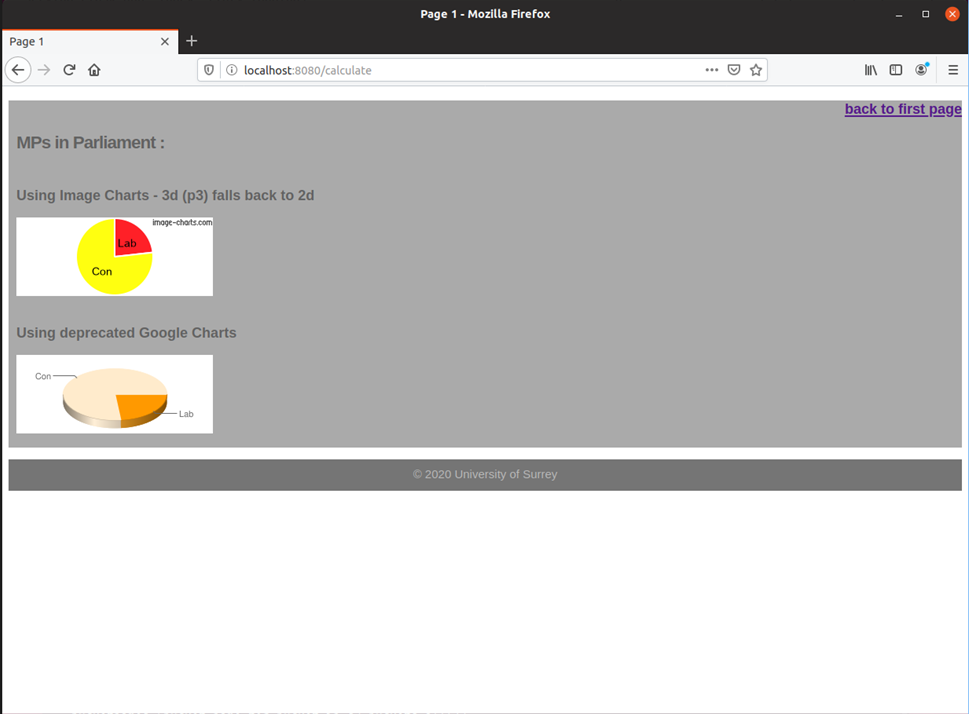


Figure 5: A dynamic local application, displaying data using 2 Chart services

1. **If you cannot get this to work…..** see lab code on SurreyLearn, lab1/**ex3**.

* Refer to the relevant section for GAE deployment, and you should be able to deploy your application “to Google Cloud”

# Billing management

## In the dashboard

You can turn off any possible GAE project billing at:

<https://console.cloud.google.com/billing/projects>

For each project, click Actions 🡪 Disable Billing (if billing is enabled), and the billing account information should change to: Billing is disabled.

**The application will stop responding to requests.**

**You’ll need to remember to Enable Billing again (at the same page) in order to use it!**

## Using the ‘gcloud’ command to manage billing

The ‘alpha’ tools need to be installed (lab machine Terminal) first:

|  |
| --- |
| gcloud components install alpha |

*Default response is Y, so just hit <Enter>*

Now you will need to identify things like the billing account (first command below – identify the **ACCOUNT\_ID**) and the project id (from the second command – identify the **PROJECT\_ID**):

|  |
| --- |
| gcloud alpha billing accounts list  gcloud alpha projects list |

With this information, you can identify projects associated to a specific billing account, stop a project being linked (the equivalent to Disable Billing in the previous subsection) or add/change the billing account ofr a project.

Projects linked to a billing account (substituting the ACCOUNT\_ID identified above):

|  |
| --- |
| gcloud alpha billing projects list --billing-account **ACCOUNT\_ID** |

Remove billing from a project:

|  |
| --- |
| gcloud alpha billing projects unlink **PROJECT\_ID** |

**The application will stop responding to requests.**

**You’ll need to remember to Enable Billing again in order to use it!**

To add a project to a billing account (if you ever want it to work again):

|  |
| --- |
| gcloud alpha billing projects link **PROJECT\_ID** --billing-account **ACCOUNT\_ID** |

Recall that to see which project is current:

|  |
| --- |
| gcloud config list |

At the end of the module, it may be worth removing your App Engine projects, and storage usage, to help retain remaining credits – which will expire eventually anyway. App Engine may even be replaced or removed in due course (it’s Google, so who knows: <https://killedbygoogle.com/>)

# Further Exercises – Coursework preparation:

1. Add a ‘global’ counter to track how many calls the app has received whilst running – determine when the counter needs to be incremented, and create a page (access\_count.htm?) that displays this number so that it can be obtained using curl.
2. Create a further app.route with suitable code and HTML to generate a table for, for example, some data you may have from another module. Consider <https://www.geeksforgeeks.org/python-using-for-loop-in-flask/> as a starting point (you can remove the CSS that they have added.

With Python2, Google’s DataStore was readily usable. Now it requires a number of additional steps. You could look at what these involve, in preparation for coursework, but you might want to wait until after Lab 5 when we’ve looked a bit more at both credentials and other storage possibilities. Note, also, that there is a temporarily writable filesystem – however, when the application is taken offline after a period without being used anything that had been written there would be lost.

<https://cloud.google.com/appengine/docs/standard/python3/using-python3-libraries>

<https://cloud.google.com/appengine/docs/standard/python3/using-temp-files>

* **You have completed this Lab. Please make a note of, and ask, any questions you have. If you finished quickly, investigate what is available in the rest of the Dashboard, in particular ‘Logs’ (where any errors will be found).**
* **We’ve done much of what we intend to with Google Cloud for now but will reuse some of this from Lab 3 onwards.**