# Cloud-Based “Hello, World!”

This lab consists of getting a simple “Hello, World!” application up and running on the cloud. You will use a public cloud provider to accomplish this task, and the tutorial help in the lab notes will assume Google AppEngine Standard as the basis for your application, implemented in Python 3 and Flask.

This is *not required*. You may use any cloud provider with which you are comfortable, and any language that makes sense for you. Do know that if you use something else (e.g., AWS Lambda and JavaScript) I can help you a little, but choosing something like that implies that you don’t **need** as much help, and I will prioritize my time accordingly. Don’t let that stop you if you’re anxious to use your own thing, just know that for the first couple of weeks of the semester I will be spending the bulk of my help time bringing beginners up to speed to get them ready for the remainder of the course.

# Grading

This lab is graded on a pass/fail basis. You can get all or none of the points, since it is relatively simple (not a lot of pieces). Success is measured simply as my (or my CA’s) ability to see your “Hello, World!” site, with your name on it, from somewhere on the Internet.

## Minimum Requirements

Provide me with a URL pointing to your own cloud application. When I navigate to it, I should see (at a minimum, go nuts if you want) something that includes your name. It does not have to say “Hello, World!”, though that is tradition, it merely has to show me your name.

# Background and Tips

Getting up and running with a PaaS (Platform as a Service) or an FaaS (Function as a Service) is not always straightforward. You need to get accounts set up, make sure that your choice includes the appropriate data storage (later on, at least, though storage is not needed for this first lab), find a place to put your code that the provider will serve up, and then figure out how to use the SDK.

Note that more background will be given in the next lab, where things get a little more interesting. Here we just get going on the basics.

The steps you will complete for project setup are basically these:

* Go to <https://console.cloud.google.com> and create a new Project.
* Use the upper-left menu to get to the [AppEngine](https://console.cloud.google.com/appengine) page.
* If you’re in your new project, it will show you a “Create Application” button. Click that.
* Choose a region. We’re in the us-east-1 region, so that will give the best latency from JHU, if you happen to be near the Homewood campus.
* I choose “Python”, “Standard” on the next page, but you can pick another language if you like. I would **not** choose “Flexible” unless you really know what you are doing and are *willing to spend money*.
* Note that by default the documentation it leads you to is for Python 2. The Python 3 documentation can be found here: <https://cloud.google.com/appengine/docs/standard/python3/>
* Follow instructions to download the cloud SDK and get it set up.
* Do a “gcloud auth login” so that deployment can work, and your dev server can hit the database (the local development datastore is apparently deprecated).

Eventually (maybe now!) you will want to go to your project’s “service accounts” and generate a JSON key file for the default appengine service account. You can then point to that file in the [GOOGLE\_APPLICATION\_CREDENTIALS](https://cloud.google.com/docs/authentication/production) environment variable to make things work from your local system. I use a small bash script to point to that when running my local Flask app so that it doesn’t pollute my system with random junk. For example:

|  |
| --- |
| #!/bin/bash  export GOOGLE\_APPLICATION\_CREDENTIALS="$HOME/.config/gcloud/appengine-token.json" export GAE\_ENV="localdev" python main.py |

Once you are up and running, you will need to create your HTML file that you want served. You can do this in a couple of ways:

1. Static: you can set up your app.yaml handler for “/” to be a static directory, and place an “index.html” file in the directory you specify, or
2. Dynamic: you can write a little Python/Flask application that serves up the index.html file and specify that as the script.

There are numerous “Hello, World!” AppEngine tutorials online. I would encourage you to look at these to get started.

## Suggested Additional Work

Whichever approach you chose to use (static or dynamic), try making your lab work the other way, now. That way you’ll be ready for future labs, which use a little bit of both!