Chapter 2: Cloud Computing Foundations

This chapter covers some of the core building blocks of the Cloud, including the service models and IaC(Infrastructure as Code). Many hands-on examples are in this chapter, including Elastic Beanstalk, Google App Engine, and AWS Lambda.

Why you should consider using a cloud-based development environment

There is an expression, "use the best tool for the job." When doing development on the Cloud, often the best tool is the native environment. For most of the examples in this book, a cloud-based development environment is a correct approach. For work on AWS, this means AWS Cloud9[87] or the AWS Cloudshell[88]. For work on Google, this means Cloud Shell[89]. The same goes with the Azure environment; the Azure Cloud Shell[90] is a powerful and recommended environment to develop in.

They offer "curated" administrative tools preinstalled like a cloud SDK and Linux development tools. Working on a laptop or workstation running Linux, OS X, or Windows can be made suitable for development work, but each presents a unique challenge. It is recommended you use the "native" cloud development tools as the first option and only expand from these tools when you are an advanced user.

Overview of Cloud Computing

What is Cloud Computing? In a nutshell, cloud computing can use "near-infinite" resources and leverage SaaS platforms built on those resources.
Learn what Cloud Computing is in the screencast.

Video Link: https://www.youtube.com/watch?v=KDWkY0srFpg[91]

[87]https://aws.amazon.com/cloud9/
[88]https://aws.amazon.com/cloudshell/
[89]https://cloud.google.com/shell/
[90]https://docs.microsoft.com/en-us/azure/cloud-shell/overview
[91]https://www.youtube.com/watch?v=KDWkY0srFpg

Economics of Cloud Computing

What is the Economics of Cloud Computing? Several key factors play into the advantages of Cloud Computing. Comparative Advantage means that a company can focus on its strengths instead of building low-level services. Another factor is Economies of Scale; in the case of a large cloud provider, they can generate cost savings that pass down to the customer. Another is the ability to "pay for what you need," much like a utility company versus paying the up-front cost of infrastructure and hardware.

Learn what Cloud Computing economics is in the screencast.

Video Link: https://www.youtube.com/watch?v=22mtNlfGEc8[92]

Cloud Service Model: SaaS, PaaS, IaaS, MaaS, Serverless

A key takeaway of the Cloud is there are many ways to use it. There are many ways to buy food: in bulk, at the grocery store, at a restaurant, or delivery, there are many ways to use Cloud Computing.

Learn what Cloud Computing Service models are in the screencast.

Video Link: https://www.youtube.com/watch?v=7lgy7Cnt72c[93]

Note, there an overview of Cloud Computing available in the book Python for DevOps, Chapter 9: Cloud Computing[94].

SaaS

SaaS (Software as a Service) is a hosted software product. A google example is Google Docs[95] or Office 365[96]. Generally, these software products are hosted on cloud platforms and sold via a subscription model.

Paas

PaaS (Platform as a Service) is a higher-level abstraction for developing software. An excellent example of a PaaS is Heroku[97]. This process allows a developer in a language like Ruby, Python, PHP, or Go to focus mostly on their application's business logic.

A real-world scenario comparison would be a self-service car wash versus a drive-through car wash. In the drive-through car wash, a customer only has to drive through, not use equipment to wash their car.

[92]https://www.youtube.com/watch?v=22mtNlfGEc8
[93]https://www.youtube.com/watch?v=7lgy7Cnt72c
[94]https://learning.oreilly.com/library/view/python-for-devops/9781492057680/ch07.html
[95]https://www.google.com/docs/about/
[96]https://www.office.com/
[97]https://www.heroku.com/

IaaS

IaaS (Infrastructure as a Service) refers to services that provide low-level resources: Compute, Storage, and Networking. Typically these services are low cost to use but require more setup and expertise. On Amazon, these services would be EC2 (compute) and S3 (Storage).

A real-world comparison would be buying grain or beans in bulk from a company like Costco, then using those resources to create a meal. The cost would be much lower than purchasing a full meal from a restaurant but requires time and skill to convert to a meal.

MaaS

MaaS (Metal as a Service) is the ability to rent actual physical servers vs. virtualized servers. One of the advantages of this approach is for specialized scenarios like training deep learning models. A compute operation may require the highest amount of resources available. Virtualization causes some overhead, and eliminating it will allow these specialized operations to fully access the "metal."

Serverless

One way to think about serverless is that "Serverless" refers to running without thinking about servers. An excellent example of this is AWS Lambda[98]. The benefit of using serverless has many benefits. One advantage is the ability to focus on writing functions vs. managing servers. Another benefit is the ability to use new paradigms like event-driven programming against cloud-native resources.

PaaS Continuous Delivery

What follows is an example of using a PaaS platform, Google App Engine, of deploying a Flask web application continuously.

Google App Engine and Cloud Build Continuous Delivery

source code examples in this section are: https://github.com/noahgift/delivery[99] and https://github.com/noahg
hello-ml[100].

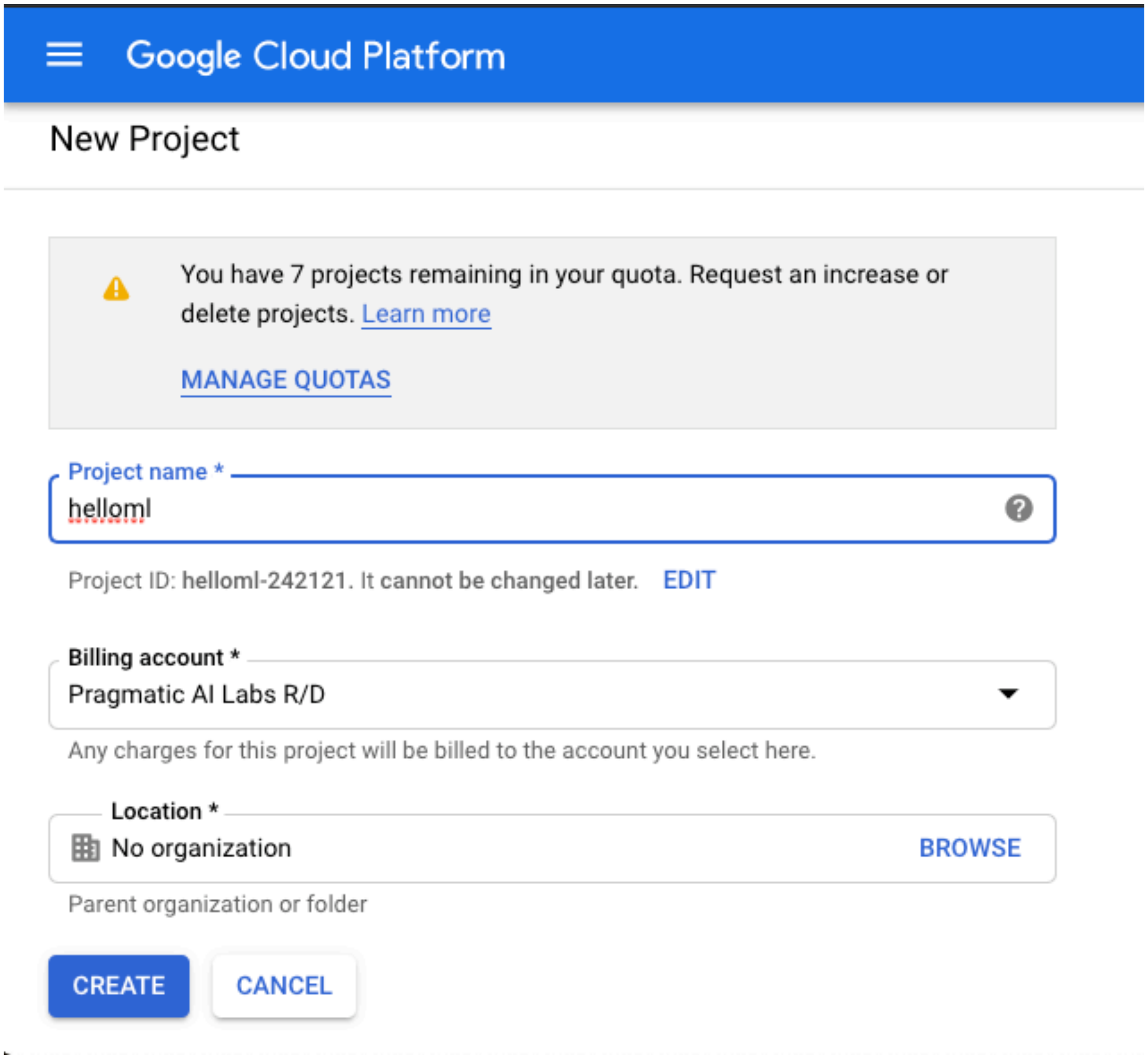
Watch a screencast on deploying Google App Engine.

Video Link: https://www.youtube.com/watch?v=_TfWdOvQXwU[101]

To get started with Continuous Delivery, do the following.

[98]https://aws.amazon.com/lambda/
[99]https://github.com/noahgift/delivery
[100]https://github.com/noahgift/gcp-hello-ml
[101]https://www.youtube.com/watch?v=_TfWdOvQXwU

1. Create a Github repo
2. Create a project in GCP UI (your project name will be different)
2A Setup API as well[102]


Google Cloud Platform

New Project

You have 7 projects remaining in your quota. Request an increase or delete projects. Learn more

MANAGE QUOTAS

Project name *
helloml

Project ID: helloml-242121. It cannot be changed later. EDIT

Billing account *
Pragmatic AI Labs R/D

Any charges for this project will be billed to the account you select here.

Location *
No organization BROWSE

Parent organization or folder

Project UI

3. Next, activate cloud-shell and add ssh-keys if not already added to Github: i.e. `ssh-keygen -t rsa` than upload key to Github ssh settings.

4. Create an initial project scaffold. You will need the following files, which you can create with the following commands. Note you can copy `app.yaml`, `main.py`, `main_test.py` and `requirements.txt` from this repo from google[103].

[102]https://cloud.google.com/appengine/docs/standard/python3/quickstart
[103]https://github.com/GoogleCloudPlatform/python-docs-samples/tree/master/appengine/standard_python37/hello_world

- Makefile: `touch Makefile`

This scaffolding allows for an easy to remember convention.

- requirements.txt: `touch requirements.txt`

These are the packages we use.

- app.yaml: `touch app.yaml`

The `app.yaml` is part of the IaC (Infrastructure as Code) and configures the PaaS environment for Google App Engine.

- main.py: `touch main.py`

The files are the logic of the Flask application.

5. Finally, run describe using the `gcloud` command-line to verify the project is working.

```
gcloud projects describe $GOOGLE_CLOUD_PROJECT
```

output of command:

```
createTime: '2019-05-29T21:21:10.187Z'
lifecycleState: ACTIVE
name: helloml
projectId: helloml-xxxxx
projectNumber: '881692383648'
```

6. *(optional)* You may want to verify you have the correct project and if not, do this to switch:

```
gcloud config set project $GOOGLE_CLOUD_PROJECT
```

7. Next, create an app engine app in the Cloud:

```
gcloud app create
```

This step will ask for the region. Go ahead and pick us-central [12] or another region if you are an advanced user.

```
Creating App Engine application in project [helloml-xxx] and region [us-central]....\
done.
Success! The app is now created. Please use `gcloud app deploy` to deploy your first\
 app.
```

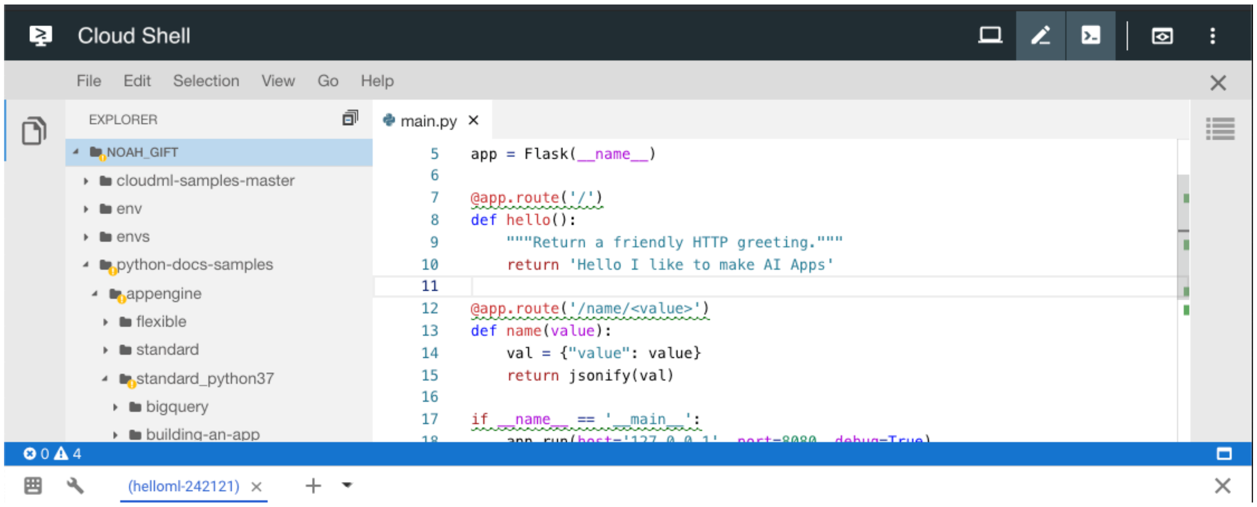
10. Create and source the virtual environment.

```
virtualenv --python $(which python) venv
source venv/bin/activate
```

Now, double-check it works.

```
which python
/home/noah_gift/python-docs-samples/appengine/standard_python37/hello_world/venv/bin\
/python
```

10. Next, activate the cloud shell editor, or use a terminal editor like `vim`.



```
app = Flask(__name__)

@app.route('/')
def hello():
    """Return a friendly HTTP greeting."""
    return 'Hello I like to make AI Apps'

@app.route('/name/<value>')
def name(value):
    val = {"value": value}
    return jsonify(val)

if __name__ == '__main__':
    app.run(host='127.0.0.1', port=8080, debug=True)
```

code editor

11. Now install packages.

```
make install
```

This step should install flask and other packages you have created.

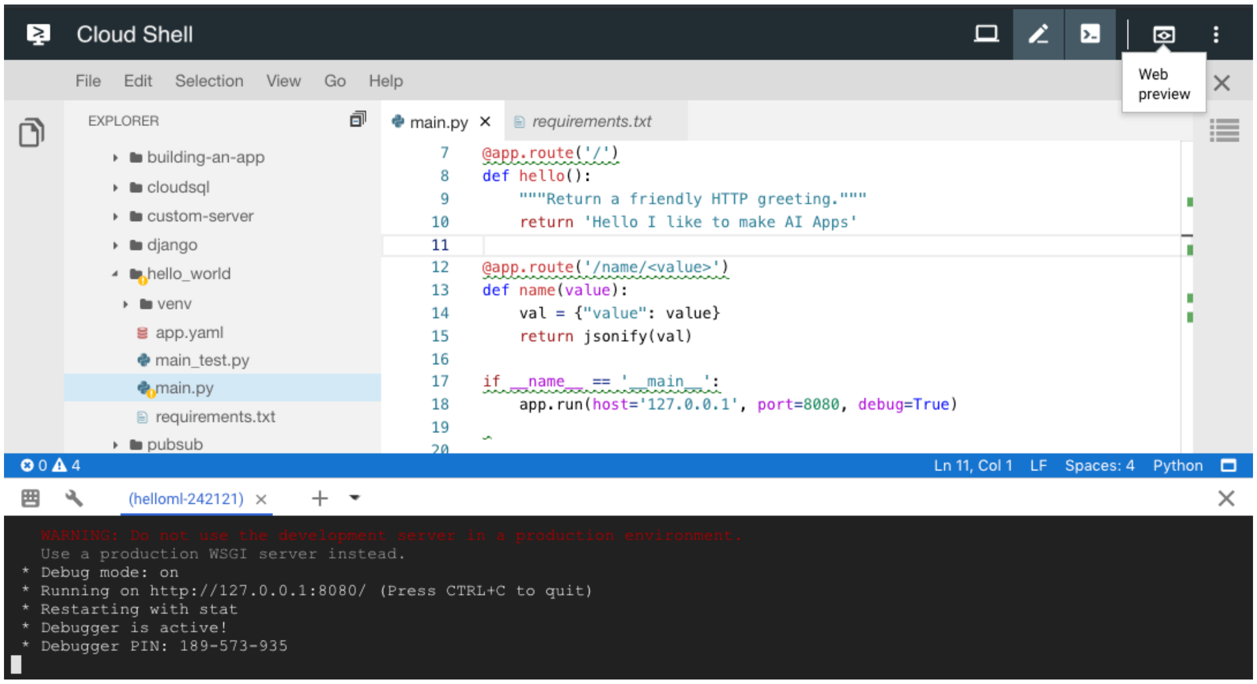
```
Flask==1.x.x
```

12. Now, run `flask` locally.

This command runs flask locally in the GCP shell.

```
python main.py
```

13. Now preview the running application.



```
@app.route('/')
def hello():
    """Return a friendly HTTP greeting."""
    return 'Hello I like to make AI Apps'

@app.route('/name/<value>')
def name(value):
    val = {"value": value}
    return jsonify(val)

if __name__ == '__main__':
    app.run(host='127.0.0.1', port=8080, debug=True)
```

```
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 189-573-935
```

preview

14. Now that the application is running, try for an update of the `main.py`.

```python
from flask import Flask
from flask import jsonify

app = Flask(__name__)

@app.route('/')
def hello():
    """Return a friendly HTTP greeting."""
    return 'Hello I like to make AI Apps'

@app.route('/name/<value>')
def name(value):
    val = {"value": value}
    return jsonify(val)

if __name__ == '__main__':
    app.run(host='127.0.0.1', port=8080, debug=True)
```

15. You can test out passing in parameters to exercise this function:

```
@app.route('/name/<value>')
def name(value):
    val = {"value": value}
    return jsonify(val)
```

For example, calling this route will take the word lion and pass into the Flask application's name function.

```
https://8080-dot-3104625-dot-devshell.appspot.com/name/lion
```

This step returns the value in a web browser:

```
{
value: "lion"
}
```

16. Now deploy the app

```
gcloud app deploy
```

Warning the first deployment could take about 10 minutes
FYI!!! You may also need to enable cloud build API.

```
Do you want to continue (Y/n)?  y
Beginning deployment of service [default]
...Uploading 934 files to Google Cloud Storage...
```

17. Now stream the log files.

```
gcloud app logs tail -s default
```

18. The production app is deployed and should like the following output.

```
Setting traffic split for service [default]...done.
Deployed service [default] to [https://helloml-xxx.appspot.com]
You can stream logs from the command line by running:
  $ gcloud app logs tail -s default

  $ gcloud app browse
(venv) noah_gift@cloudshell:~/python-docs-samples/appengine/standard_python37/hello_\
world (helloml-242121)$ gcloud app
 logs tail -s default
Waiting for new log entries...
2019-05-29 22:45:02 default[20190529t150420]  [2019-05-29 22:45:02 +0000] [8] [INFO]\
 Starting gunicorn 19.9.0
2019-05-29 22:45:02 default[20190529t150420]  [2019-05-29 22:45:02 +0000] [8] [INFO]\
 Listening at: http://0.0.0.0:8081
 (8)
2019-05-29 22:45:02 default[20190529t150420]  [2019-05-29 22:45:02 +0000] [8] [INFO]\
 Using worker: threads
2019-05-29 22:45:02 default[20190529t150420]  [2019-05-29 22:45:02 +0000] [25] [INFO\
] Booting worker with pid: 25
2019-05-29 22:45:02 default[20190529t150420]  [2019-05-29 22:45:02 +0000] [27] [INFO\
] Booting worker with pid: 27
2019-05-29 22:45:04 default[20190529t150420]  "GET /favicon.ico HTTP/1.1" 404
2019-05-29 22:46:25 default[20190529t150420]  "GET /name/usf HTTP/1.1" 200
```

19. Add a new route and test it out

```
@app.route('/html')
def html():
    """Returns some custom HTML"""
    return """
    <title>This is a Hello World World Page</title>
    <p>Hello</p>
    <p><b>World</b></p>
    """
```

20. Install pandas and return `json` results. At this point, you may want to consider creating a Makefile and do this:

```
touch Makefile
#this goes inside that file
install:
        pip install -r requirements.txt
```

you also may want to setup lint:

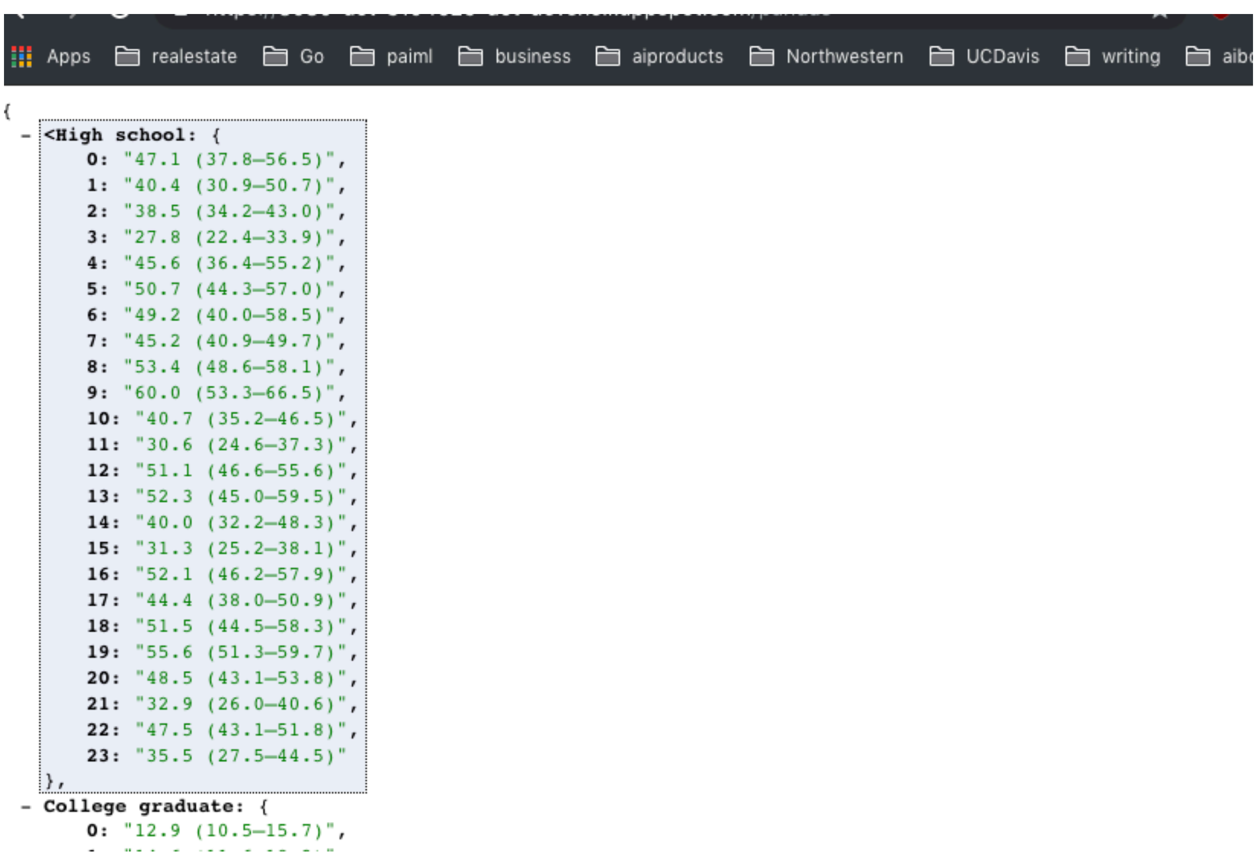
```
pylint --disable=R,C main.py
------------------------------------
Your code has been rated at 10.00/10
```

The route looks like the following, so add pandas import at the top.

```
import pandas as pd
```

```
@app.route('/pandas')
def pandas_sugar():
    df = pd.read_csv("https://raw.githubusercontent.com/noahgift/sugar/master/data/e\
ducation_sugar_cdc_2003.csv")
    return jsonify(df.to_dict())
```

When you call the route `https://<yourapp>.appspot.com/pandas`, you should get something like the following ouput.



```
{
  - <High school: {
        0: "47.1 (37.8–56.5)",
        1: "40.4 (30.9–50.7)",
        2: "38.5 (34.2–43.0)",
        3: "27.8 (22.4–33.9)",
        4: "45.6 (36.4–55.2)",
        5: "50.7 (44.3–57.0)",
        6: "49.2 (40.0–58.5)",
        7: "45.2 (40.9–49.7)",
        8: "53.4 (48.6–58.1)",
        9: "60.0 (53.3–66.5)",
        10: "40.7 (35.2–46.5)",
        11: "30.6 (24.6–37.3)",
        12: "51.1 (46.6–55.6)",
        13: "52.3 (45.0–59.5)",
        14: "40.0 (32.2–48.3)",
        15: "31.3 (25.2–38.1)",
        16: "52.1 (46.2–57.9)",
        17: "44.4 (38.0–50.9)",
        18: "51.5 (44.5–58.3)",
        19: "55.6 (51.3–59.7)",
        20: "48.5 (43.1–53.8)",
        21: "32.9 (26.0–40.6)",
        22: "47.5 (43.1–51.8)",
        23: "35.5 (27.5–44.5)"
    },
  - College graduate: {
        0: "12.9 (10.5–15.7)",
```

example out

Cloud Build Continuous Deploy

Finally, set up Cloud Build Continuous Deploy you can follow the guide here[104].

- Create a `cloudbuild.yaml` file
- Add to the repo and push `git add cloudbuild.yaml`, `git commit -m "add cloudbuild config"`, `git push origin master`.
- Create a build trigger
- Push a simple change
- View progress in build triggers page[105]

References

These are additional references that are helpful for GAE Continuous Delivery.

- Enable Trigger on Github[106]

[104]https://cloud.google.com/source-repositories/docs/quickstart-triggering-builds-with-source-repositories
[105]https://console.cloud.google.com/cloud-build/triggers
[106]https://cloud.google.com/cloud-build/docs/create-github-app-triggers

- GAE Quickstart[107]
- Cloud Build Continuous Deploy[108]
- Cloud Build GCP Hello ML[109]