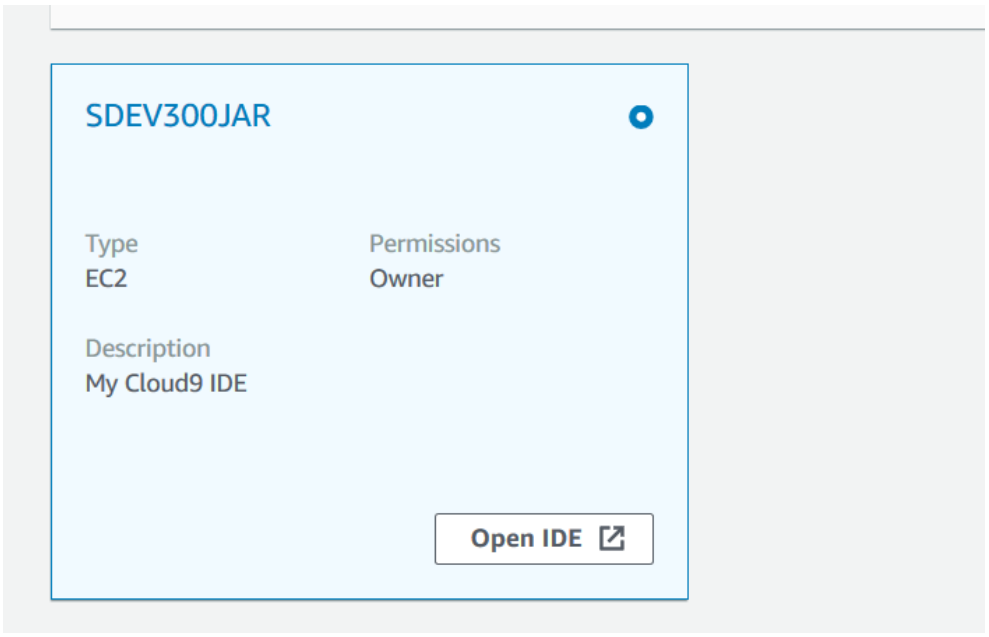
Using Python Flask in AWS Cloud9

Overview:
This document provides the configuration for AWS Cloud9 and some sample code that takes advantage of the Flask module for Web application development using Python.

Background:
According to the Flask web site, "Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions" (Flask, 2019). Frameworks are typically a bundle of libraries providing significant functionality. A microframework contains a smaller subset of functionality when compared to a full stack. Django is considered a full stack framework whereas Flask is a microframework. For this course, the use of Flask will suffice. Be aware more functionality is available in other full stacks.

Your reading for this week provided an overview of HTML with specific HTML code for creating lists and links. Since we will be using Flask and AWS Cloud 9, there are some configurations needed to be successful. In addition, we won't be using specific flask tag generations and modules such flask-table. We will be doing more HTML coding from scratch as opposed to generating. This involved wrapping around the HTML tags in Python calls and functions. You can use the generators after this course is complete and becoming comfortable with the basic HTML tags.

Install Flask in AWS Cloud 9:
For all applications you should be using Python 3. Python 3 is the latest version and the most secure. To install the Flask Python module you need to login into the AWS Classroom and start your Cloud9 environment as shown in Figure 1. Click Open IDE to continue.


SDEV300JAR

Type
EC2

Permissions
Owner

Description
My Cloud9 IDE

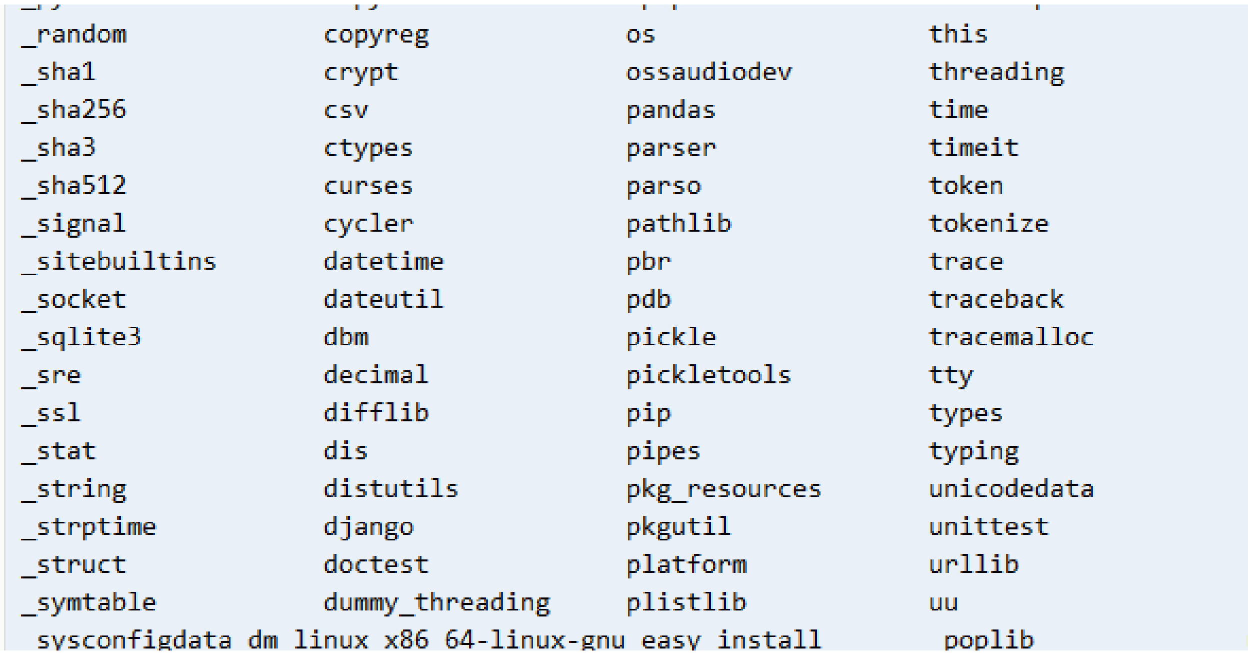
Figure 1 Open your Cloud9 Environment

After the IDE launches, you can go to the work area and click on the bash shell. The bash shell is just an Operating System shell. You can run Linux commands at this prompt. As before, we will use PIP to install the flask module. Specifically, to install flask type the following command at the bash shell:

```
sudo python3 -m pip install flask
```

You will get a success install message upon success. If you receive a message about updating pip, you should ignore this. The PIP version AWS has installed is sufficient.

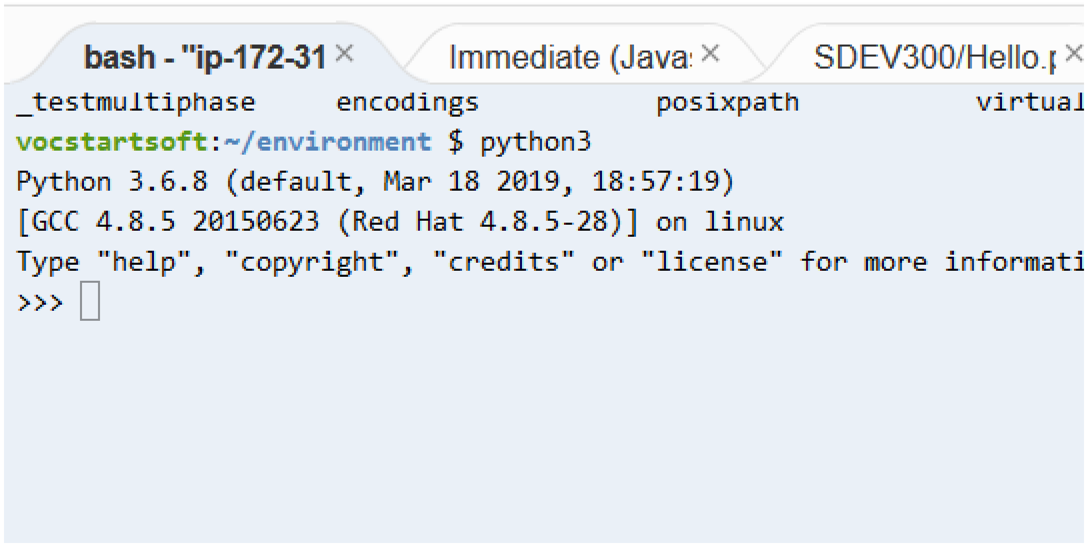
You can list the modules currently installed by running the command line version of Python3 and typing help('modules'). See figure 2.



```
_random                  copyreg               os                  this
_sha1                    crypt                 ossaudiodev         threading
_sha256                  csv                   pandas              time
_sha3                    ctypes                parser              timeit
_sha512                  curses                parso               token
_signal                  cycler                pathlib             tokenize
_sitebuiltins            datetime              pbr                 trace
_socket                  dateutil              pdb                 traceback
_sqlite3                 dbm                   pickle              tracemalloc
_sre                     decimal               pickletools         tty
_ssl                     difflib               pip                 types
_stat                    dis                   pipes               typing
_string                  distutils             pkg_resources       unicodedata
_strptime                django                pkgutil             unittest
_struct                  doctest               platform            urllib
_symtable                dummy_threading       plistlib            uu
 sysconfigdata dm linux x86 64-linux-gnu easy install               poplib
```

Figure 2 Running help('modules') from Python3

Note: to use the help(‘modules’) command you must have launched the Python3 by typing Python3 in the shell. See figure 3.



```
_testmultiphase     encodings          posixpath          virtual
vocstartsoft:~/environment $ python3
Python 3.6.8 (default, Mar 18 2019, 18:57:19)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux
Type "help", "copyright", "credits" or "license" for more informati
>>> 
```

Figure 3 Launching Python3

You can also list help on specific modules using help('module_name'). For example, help('flask')

Once you have successfully installed in the module, you can use it by properly importing it and using the available functions.

AWS Cloud 9 Security Settings:
When you run the Cloud9 IDE from your browser, a specific port was already configured to allow access from your desktop machine. To allow our Web pages to be accessible, we must open up an inbound port. Since the Cloud 9 IDE runs on an Amazon EC2 instance, we just have to add another security group in bound rule. To do this, login to your AWS Educate account, navigate to the AWS console and go the EC2 service. See figure 3.

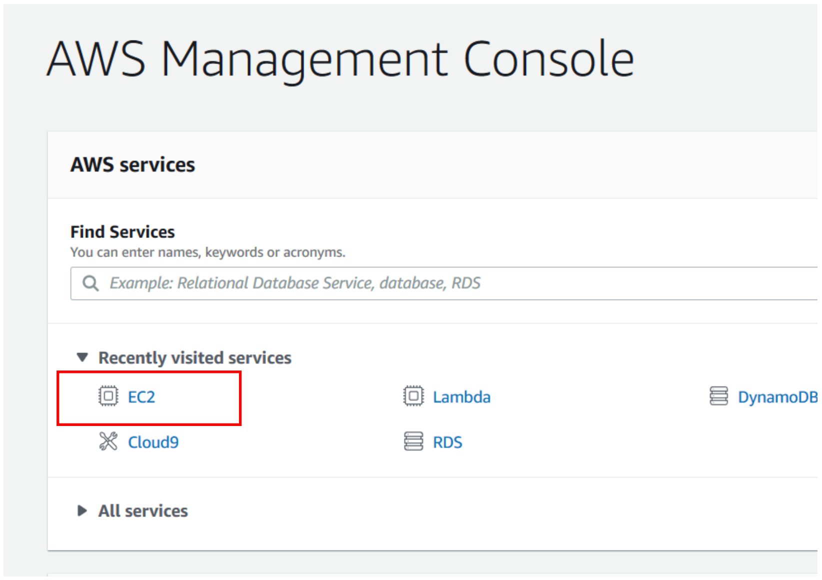
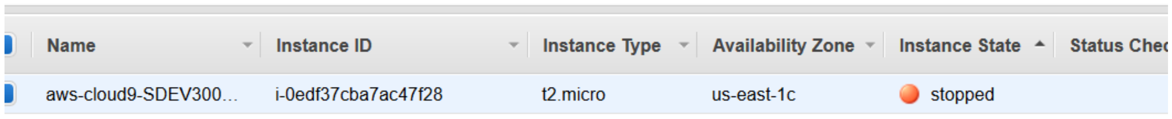

AWS Management Console
AWS services
Find Services
You can enter names, keywords or acronyms.
Example: Relational Database Service, database, RDS
Recently visited services
EC2
Lambda
DynamoDB
Cloud9
RDS
All services

Figure 4 Navigate to the EC2 Service

You can also reach the EC2 service by clicking on the Services menu in the upper right menu bar and then click on EC2 under the Compute category.

Click on the EC2 service to continue.

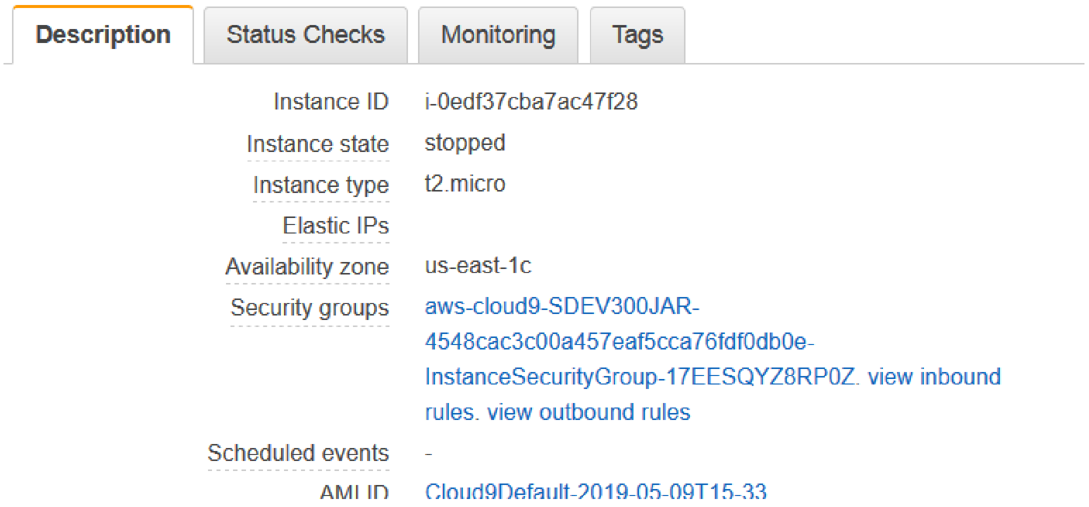
Selecting the Running instances link will expose the AWS Cloud9 EC instance where your IDE is running. See figure 5. Note your instance state may be stopped or running. Either state is appropriate for applying the settings needed for access.



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Chec
aws-cloud9-SDEV300...	i-0edf37cba7ac47f28	t2.micro	us-east-1c	● stopped	

Figure 5 Viewing the EC2 Instance

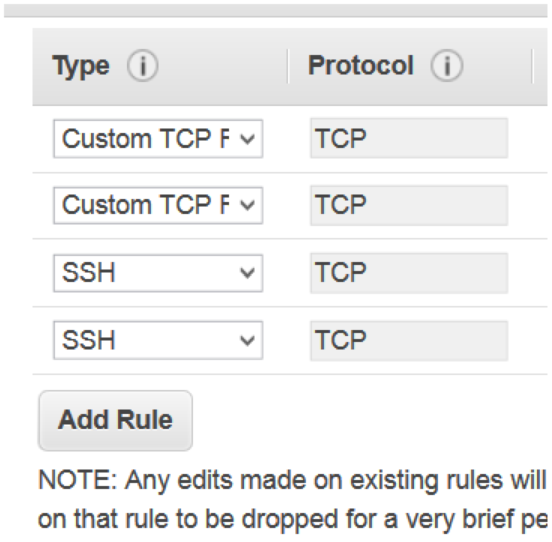
Scroll down and see the security group settings under the description tab as shown In figure 6.



Instance ID	i-0edf37cba7ac47f28
Instance state	stopped
Instance type	t2.micro
Elastic IPs	
Availability zone	us-east-1c
Security groups	aws-cloud9-SDEV300JAR-4548cac3c00a457eaf5cca76fdf0db0e-InstanceSecurityGroup-17EESQYZ8RP0Z. view inbound rules. view outbound rules
Scheduled events	-
AMI ID	Cloud9Default-2019-05-09T15-33

Figure 6 Scroll down to the Security Settings

Click on the security group link and select the inbound rules tab to add a rule. Select Add rule as shown in figure 7.

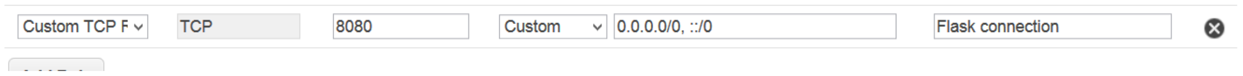


Type	Protocol
Custom TCP F	TCP
Custom TCP F	TCP
SSH	TCP
SSH	TCP

NOTE: Any edits made on existing rules will
on that rule to be dropped for a very brief pe

Figure 7 Adding an Inbound rule

Click on the Add Rule button and then add a Custom TCP Protocol for port 8080. See figure 8.

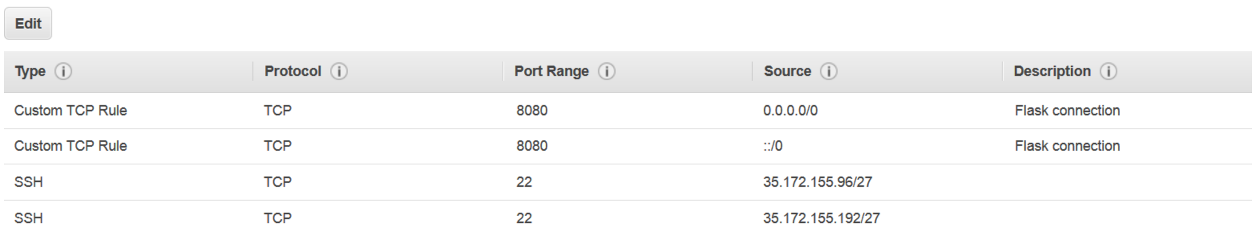


Custom TCP F	TCP	8080	Custom	0.0.0.0/0, ::/0	Flask connection

Figure 8 Adding port 8080

The port must match the port you list in the Flask code. For example: `app.run(host='0.0.0.0', port= 8080)`
The 0.0.0.0/0 option allows access from any inbound desktop. You can use the drop down source menu to restrict to a specific IP. For example, your IP. For web applications, they are typically open to the world, so for now this is Okay but we may need to revisit this and further restrict.

Click Save to continue. You can then view the rules that were just added as shown in figure 9.



Type	Protocol	Port Range	Source	Description
Custom TCP Rule	TCP	8080	0.0.0.0/0	Flask connection
Custom TCP Rule	TCP	8080	::/0	Flask connection
SSH	TCP	22	35.172.155.96/27	
SSH	TCP	22	35.172.155.192/27	

Figure 9 Saving the Inbound Rules

The AWS console duplicates the row for you. For example, notice a second customer TCP rule with the same port and Source of :0 was entered automatically. This might be a display artifact within the AWS console but you don’t need to worry about adding it only the one Custom TCP rule for port 8080.

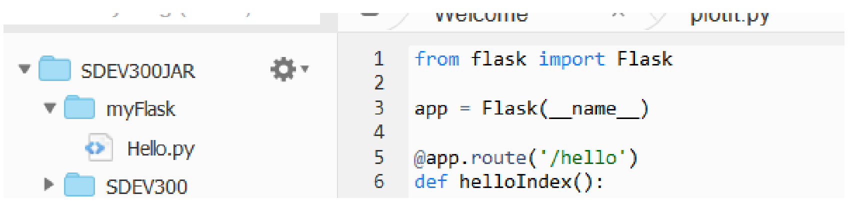
By adding this port to your security group, you are now ready to try a simple Hello, World Flask application.

Hello World Example:

After installing flask, it makes sense to test a simple hello world program. Launch your AWS Cloud 9 IDE and create a new file with the following content:

```
from flask import Flask

app = Flask(__name__)

@app.route('/hello')
def helloIndex():
    myReply = ‘Welcome to SDEV 300- The Web App!’
    return myReply

app.run(host='0.0.0.0', port= 8080)
```

I always recommend creating a new folder and saving the new file in that folder. This allows you to group your application type. As shown in figure 10, I created a folder called myFlask and named the Python file, Hello.py.



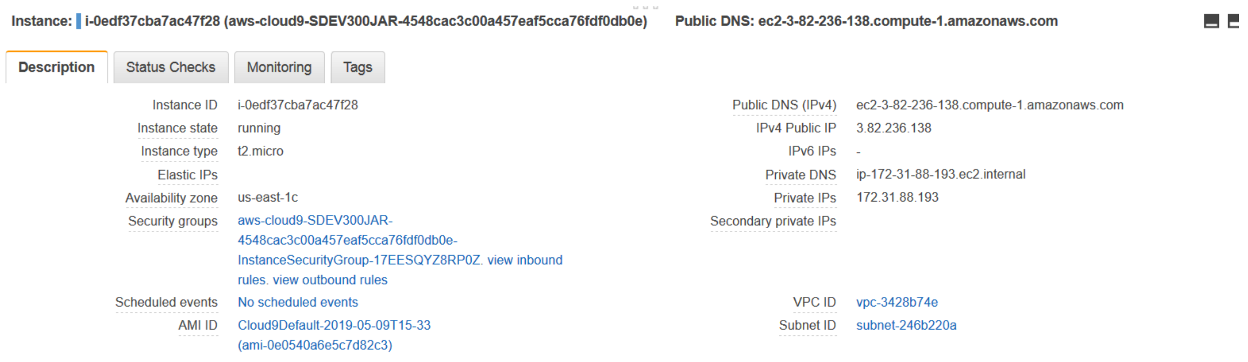
```
from flask import Flask

app = Flask(__name__)

@app.route('/hello')
def helloIndex():
```

Figure 10 Saving Hello.py

Run the Hello.py, just like you would any other Python3 application. Click on the green Run arrow. As shown in figure 11, if successful, the Application will invoke and the http://0.0.0.0:8080 server location is listed.

We can now test the application by launching a browser and connecting to the public IP V4 address using the app name. To find the public IP V4 address, go back to the EC2 service and scroll down to record your IP address.

As shown in figure 12, the IPv4 Public IP is found in the description tab on the right side of the screen.



Instance: i-0edf37cba7ac47f28 (aws-cloud9-SDEV300JAR-4548cac3c00a457eaf5cca76fdf0db0e) Public DNS: ec2-3-82-236-138.compute-1.amazonaws.com

Instance ID	i-0edf37cba7ac47f28	Public DNS (IPv4)	ec2-3-82-236-138.compute-1.amazonaws.com
Instance state	running	IPv4 Public IP	3.82.236.138
Instance type	t2.micro	IPv6 IPs	-
Elastic IPs		Private DNS	ip-172-31-88-193.ec2.internal
Availability zone	us-east-1c	Private IPs	172.31.88.193
Security groups	aws-cloud9-SDEV300JAR-4548cac3c00a457eaf5cca76fdf0db0e-InstanceSecurityGroup-17EESQYZ8RP0Z. view inbound rules. view outbound rules	Secondary private IPs	
Scheduled events	No scheduled events	VPC ID	vpc-3428b74e
AMI ID	Cloud9Default-2019-05-09T15-33 (ami-0e0540a6e5c7d82c3)	Subnet ID	subnet-246b220a

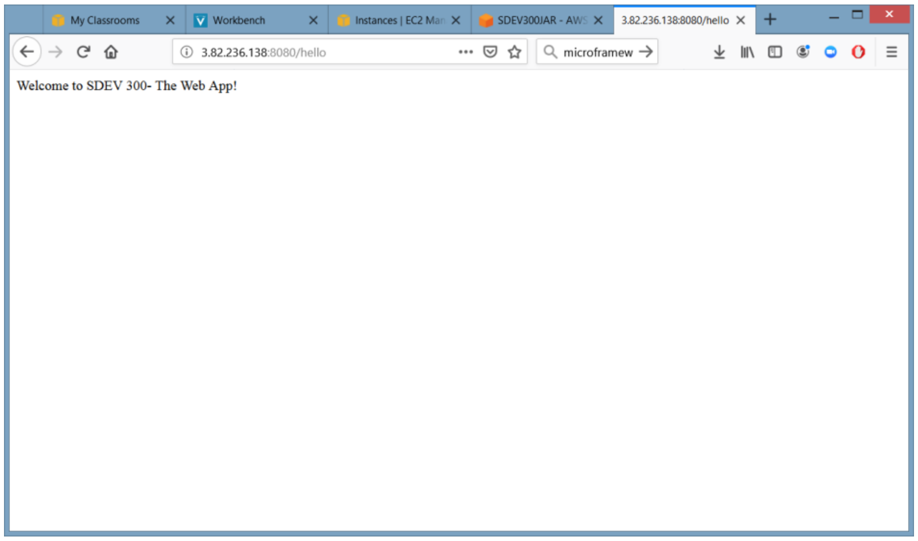
Figure 11 Finding your Public IP address

Be sure you check for your IPv4 Public address each time you run your application. In most cases, this value will change between sessions where the EC2 instance is shut down and restarted.

Web applications run from a browser. To connect, you need access to the port and the location of the URL. The correct URL for this example application is:

http://3.82.236.138:8080/hello

Launching this URL in a desktop browser yields the following expected results:



Welcome to SDEV 300- The Web App!

Figure 12 Running Hello.py

Note: Your URL will be different and based on your Cloud9 EC2 settings as described above.

A couple of important notes about this application code:

1. @app.route('**/hello**') aligns the location of the application on the server. This is not the physical location but a virtual location and how it appears in the URL:

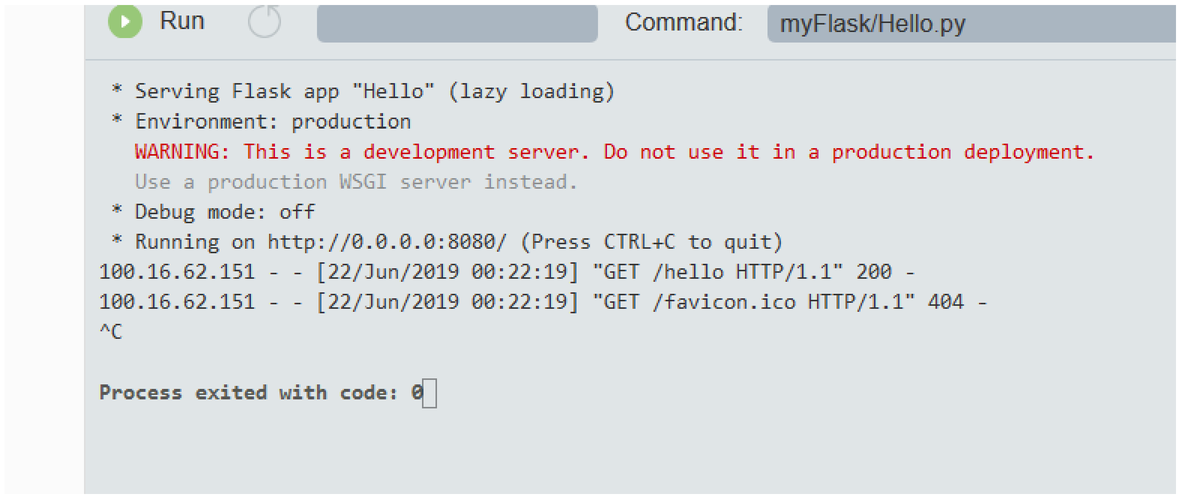
   http://3.82.236.138:8080/**hello**

If we have used **/hello2** in the app.route() method, we would have needed to use:

   http://3.82.236.138:8080/**hello 2**

2. The code: app.run(host='0.0.0.0', port= 8080 specifies the port. Typically AWS Cloud9 does not provide port 80 access. You should use this line of code as is for all code in this class.
3. Don't forget to check your IPv4 public address each time you restart your EC2 instance. This prevents you wondering why your application isn't running

Be sure to clean-up after you test by stopping your web application. To do this type ctrl+c in the AWS Cloud 9 console. You will see the Process exited with code: 0 upon successfully stopping the Web application as shown in figure 13.



```
 * Serving Flask app "Hello" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
100.16.62.151 - - [22/Jun/2019 00:22:19] "GET /hello HTTP/1.1" 200 -
100.16.62.151 - - [22/Jun/2019 00:22:19] "GET /favicon.ico HTTP/1.1" 404 -
^C

Process exited with code: 0
```

Figure 13 Stopping the Web Application

HTML Tags:
The simple hello example, combined with the HTML tag reading you did this week, will allow you to quickly generate simple web pages

The following code, uses the HTML tag functionality along with Python functions to create a simple Web page. Notice, the Python functions build the HTML tags to create the Web page. This same process can be used to build any Web page using Python.

```
from flask import Flask

app = Flask(__name__)

@app.route('/moreHTML')
def helloIndex():
    myReply = popHead();
    myReply += popH1('Welcome to SDEV 300- The Web App!')
    myReply += "<p></p>"
    myReply += popTable()
    myReply += popEnd()
    return myReply

def popH1(myString):
    newString = '<H1>' + myString + '</H1>'
    return newString

def popTable():
    tabledata = "<table> <thead>"
    tabledata += "<tr><th>Date</th><th>Holiday</th></tr></thead>"
    tabledata += "<tbody><tr><td>Jan 1</td> <td> New Years Day
</td></tr>"
    tabledata += "<tr><td>May 26</td> <td> Memorial Day
</td></tr>"
```

```
        tabledata += "<tr><td>Oct 31</td> <td> Halloween </td></tr>"
        tabledata += "</tbody></table>"
        return tabledata

    def popHead():
        headData = "<!DOCTYPE html> "
        headData +="<head> "
        headData +="<title>Flask Shop</title>"
        headData +="<style>" + getStyle() + "</style>"
        headData +="</head>"
        headData +="<body>"
        return headData

    def getStyle():
        myStyle = "table,td {"
        myStyle += "border: 5px solid #117;"
        myStyle += "}"
        return myStyle

    def popEnd():
       endData = "</body>"
       endData += "</html>"
       return endData

    app.run(host='0.0.0.0', port= 8080)
```

Here are some summary notes and suggestions about this process and code:

1. Take your time when you build the HTML strings. Test the tags outside in a Web browser to expedite.
2. Start with the HTML tags and then integrate into Python functions.
3. Use the hello.py example as a starting point
4. Test early and often to make sure the web page is displaying as expected
5. Quotes within quotes need to use single quotes on the inside. For example:

`"<a href='https://umuc.edu'>Go to UMUC</a>"`. This is useful if you have attributes within the HTML tag