Setting up AWS Cloud and Running Apache Server on Custom Domain

1. Login to AWS console - https://aws.amazon.com/console/
2. Choose build a solution -> Launch a virtual machine
3. Choose Ubuntu Server 20.04 LTS -> Select
4. Configure instance details
5. Review and Launch
6. Launch
7. Create a new key pair
8. Download the key pair. This will give you a .pem file
9. Launch Instances
10. Click the instance id and see that it is up and running

Open Puttygen

1. Load the .pem downloaded from AWS and produce a .ppk file

Open Putty

1. Copy the public DNS of your AWS instance
2. SSH-Auth->Browse
3. Add .ppk that you produced with PuttyGen
4. Go to the session and save it with a name like sd3bAWSInstance
5. Login to the instance as Ubuntu

On the AWS instance

1. sudo apt-get update
2. sudo apt-get install apache2
3. vim /etc/apache2/apache2.conf
4. Add ServerName Public DNS for AWS instance at the bottom of this file
5. sudo apachectl configtest
6. sudo systemctl restart apache2

Add inbound security rule

1. Cannot connect until we add inbound security rule for the instance
2. Look at the instance and notice the security group column on the right
3. On left in Network and Security -> Security Group
4. Select Launch-wizard-1
5. Inbound rules
6. Edit inbound rules
7. Add rule http -> 0.0.0.0/0
8. Now in a browser type in the public DNS of the instance and you should see the default Ubuntu page

Create your own custom domain using freenom.com

1. Register a new domain
2. Check availability
3. Search for the domain you want
4. Get it now->selected->checkout->continue

AWS Route 53 and freenom

1. AWS -> Services -> Route 53
2. Create hosted zone
3. Copy domain name
4. Create hosted zone
5. Copy the name servers from Route 53
6. In freenom -> Manage domains -> Management tools
7. Use custom nameservers
8. Remove dots from end of copied name servers
9. Click change name servers
10. Go to EC2 instances and copy public ipv4 address
11. In Route 53 create A record
12. Copy public ip to the value field
13. Click create record
14. Create CNAME record
15. Add www as an alias for your domain
16. You have successfully routed your freenom url to your AWS instance. Type in the url and you should see the default Apache page

Deploying IoT Flask App on Apache

wsgi stands for web server gateway interface. It allows a Flask app to talk to an Apache server.

1. sudo apt-get install libapache2-mod-wsgi-py3
2. sudo apt update
3. sudo apt install python3-pip
4. pip install flask
5. cd /var/www
6. mkdir FlaskApp
7. cd FlaskApp
8. mkdir FlaskApp
9. cd FlaskApp

Copy files using Winscp

1. Enter public ip address of Aws instance
2. Advanced ssh add .ppk file
3. Looks like it does not allow remote login of root users
4. sudo vim /etc/ssh/sshd_config

5. Look for PermitRootLogin prohibit-password change to PermitRootLogin yes
6. sudo passwd root
7. sudo cp /home/Ubuntu/.ssh/authorized_keys /root/.ssh
8. sudo reboot
9. Now should be able to winscp as root to AWS instance
10. Go to /var/www/FlaskApp/FlaskApp on AWS
11. Copy app.py, static and templates
12. Rename app.py as __init__.py (underscore underscore init underscore underscore)
13. Close Winscp
14. Putty to Aws instance

Configure Virtual Host

1. sudo vim /etc/apache2/site-available/FlaskApp.conf

```
<VirtualHost *:80>
    ServerName sd3biot22.ml
    ServerAdmin john.loane@dkit.ie
    ServerAlias www.sd3biot22.ml
    WSGIScriptAlias / /var/www/FlaskApp/flaskapp.wsgi
    <Directory /var/www/FlaskApp/FlaskApp/>
        Order allow,deny
        Allow from all
    </Directory>
    Alias /static /var/www/FlaskApp/FlaskApp/static
    <Directory /var/www/FlaskApp/FlaskApp/static/>
        Order allow,deny
        Allow from all
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    LogLevel warn
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

2. Apache server uses wsgi file to serve the flask app
3. sudo vim flaskapp.wsgi

```
#!/usr/bin/python
import sys
import logging
import os
logging.basicConfig(stream=sys.stderr)
sys.path.insert(0,"/var/www/FlaskApp/")
os.environ['TEST'] = 'test'
os.environ['FACEBOOK_APP'] = 'Your facebook app id'
os.environ['FACEBOOK_SECRET']='Your facebook app secret'
```

```
def application(environ, start_response):
    for key in ['TEST', 'FACEBOOK_APP', 'FACEBOOK_SECRET']:
        os.environ[key] = environ.get(key, '')
    from FlaskApp import app as _application
    _application.secret_key='secret'
    return _application(environ, start_response)
```

4. Make sure that your flask app in __init__.py is running with no arguments – app.run(). These will be handled by the Apache server
5. sudo service apache2 restart
6. sudo service apache2 reload
7. To debug look at /var/log/apache2/error.log
8. sudo a2dissite 000-default.conf
9. sudo adensite FlaskApp
10. service apache2 restart

Securing Custom Domain

Get SSL cert from letsencrypt.org

1. sudo snap install certbot –classic
2. sudo apt-get update
3. sudo apt-get install python3-certbot-apache
4. sudo certbot –apache –d testdomain.tk –d www.testdomain.tk
5. Agree
6. N to sharing email
7. 2 to redirect http traffic to https
8. ls /etc/letsencrypt/live
9. Go to ssllabs.com/ssltest. Enter url testdomain.tk and check
10. No inbound security rule on AWS for https
11. Go to EC2 instance and add inbound security rule for https

Configure Facebook Login

1. Login to developers.facebook.com
2. Select MyApps
3. Create a new app
4. Consumer
5. Display name -> testApp or whatever you want
6. Create App
7. Enter password
8. Select Facebook login
9. Set up - > www
10. Add site url, app domains, privacy policy url
11. On left hand panel select Facebook login
12. Settings

13. Valid oauth redirect URIs add https://testdomain.tk, /main, /facebook_login, /facebook_authorized
14. Permissions and features get Advanced access to public profile
15. Settings -> Basic copy app id and app secret
16. Go to Github for the code for the server needed to add facebook login to flask server

Database Integration

1. Mysql comes installed on AWS Ubuntu 20.04
2. You just need to add a password for the root user –

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY
'your_password';
FLUSH PRIVILEGES;
```

3. Note that the password you enter above will need a capital letter, number and special character as the password security is set to strong
4. pip install Flask-SQLAlchemy
5. apt-get install python-mysqldb
6. mysql –u root –p
7. Enter the password you created above in 2
8. show databases;
9. create database sd3iot;
10. use sd3iot;
11. show tables;
12. create table users(id int not null auto_increment, name varchar(20) not null, user_id bigint not null, auth_key varchar(255), login int, read_access int, write_access int, primary key(id));
13. select * from users;
14. See Github code for how to write to this database from the server

How to set up environment variables in Apache and wsgi

1. The os.getenv() we used previously will not work when running our app on Apache through wsgi
2. We need to change the Apache.conf-le-ssl file and the .wsgi file to be able to use environment variables

```
 <IfModule mod_ssl.c>
<VirtualHost *:443>
        ServerName sd3biot22.ml
        ServerAdmin john.loane@dkit.ie
        ServerAlias www.sd3biot22.ml
        WSGIScriptAlias / /var/www/FlaskApp/flaskapp.wsgi
        <Directory /var/www/FlaskApp/FlaskApp/>
                Order allow,deny
                Allow from all
        </Directory>
        SetEnv TEST test
```

```
        SetEnv FACEBOOK_APP 'Your Facebook App id'
        SetEnv FACEBOOK_SECRET 'Your Facebook App secret'
        Alias /static /var/www/FlaskApp/FlaskApp/static
        <Directory /var/www/FlaskApp/FlaskApp/static/>
            Order allow,deny
            Allow from all
        </Directory>
        ErrorLog ${APACHE_LOG_DIR}/error.log
        LogLevel warn
        CustomLog ${APACHE_LOG_DIR}/access.log combined

    Include /etc/letsencrypt/options-ssl-apache.conf
    SSLCertificateFile /etc/letsencrypt/live/sd3biot22.ml/fullchain.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/sd3biot22.ml/privkey.pem
    </VirtualHost>
    </IfModule>
```

3. You also need to update your wsgi file to something like the following depending on the names of the environment variables

```
#!/usr/bin/python
import sys
import logging
import os
logging.basicConfig(stream=sys.stderr)
sys.path.insert(0,"/var/www/FlaskApp/")
#from FlaskApp import app
#app.secret_key='secret'
os.environ['TEST'] = 'test'
os.environ['FACEBOOK_APP'] = 'Your facebook app id'
os.environ['FACEBOOK_SECRET']='Your facebook app secret'

def application(environ, start_response):
    for key in ['TEST', 'FACEBOOK_APP', 'FACEBOOK_SECRET']:
        os.environ[key] = environ.get(key, '')
    from FlaskApp import app as _application
    _application.secret_key='secret'
    return _application(environ, start_response)
```

How to add Facebook login button to the login page

1. Go to https://codepen.io/davidelrizzo/pen/vEYvyv copy html for facebook login button and add to login.html
2. In the static folder create another folder called styles and in this folder add a file called login.css – path is /static/styles/login.css. Copy the css from the site above to this page

In login.html add the link to the css - <link rel= "stylesheet" type= "text/css" href= "{{ url_for('static',filename='styles/login.css') }}">

3. In the button add onClick = facebookLogin()
4. In main.js add

```
function facebookLogin()
{
location.replace("/facebook_login");
}
```

Generate ssh key pair on AWS instance so that you can commit straight to Github

1. ssh-keygen -t rsa -b 4096 -C "your_email@dkit.ie"
2. ls -l ~/.ssh/id_*.pub is the file you will need to add to github
3. vim ~/.ssh/id_*.pub and copy the public key
4. Go to Github ->Settings->SSH and GPGKeys
5. Select add new key and paste your public key