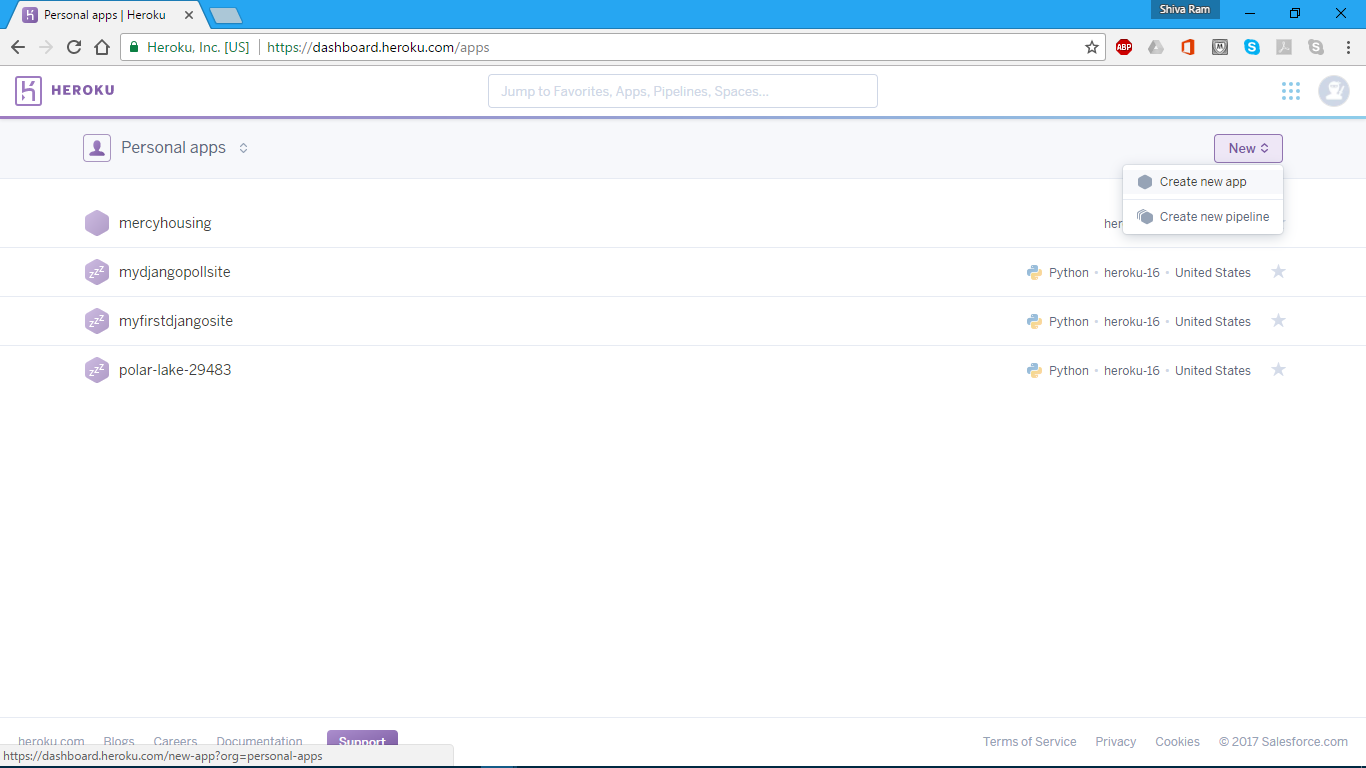
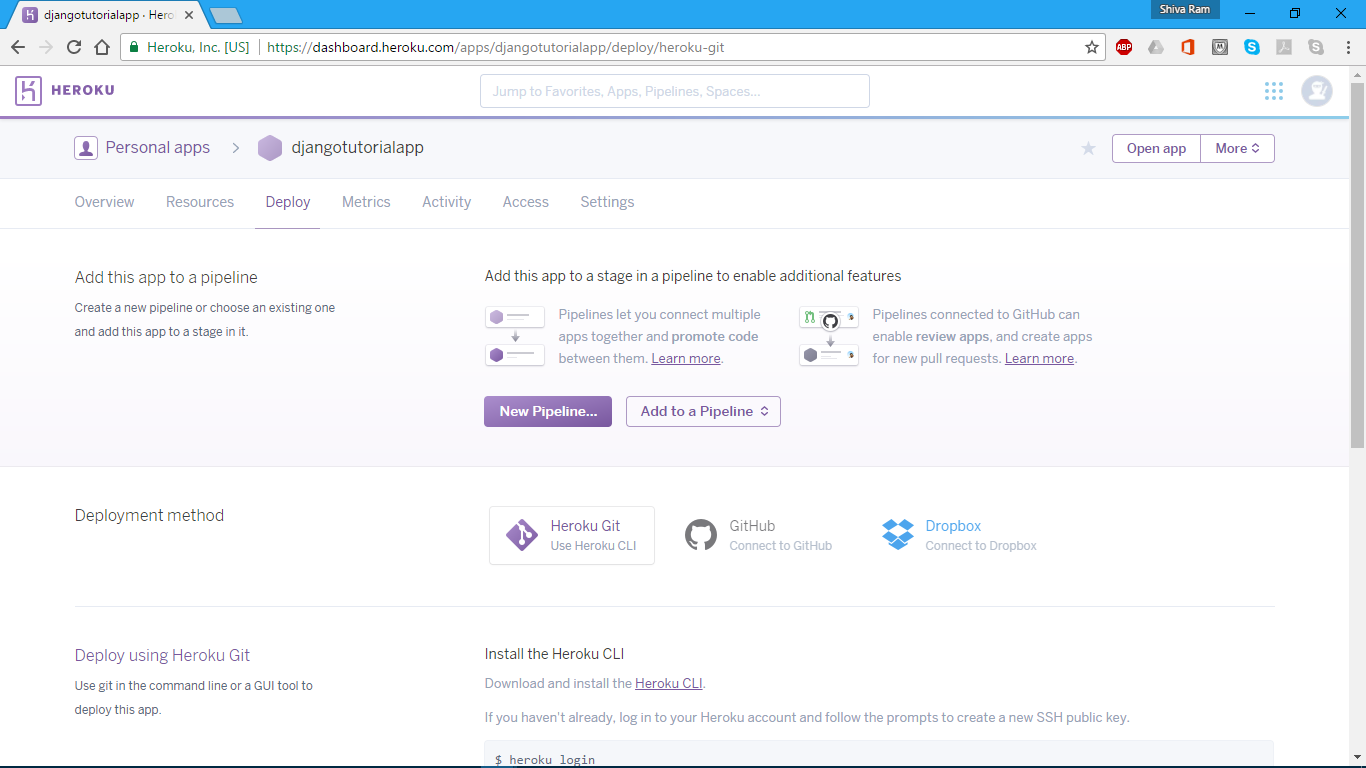
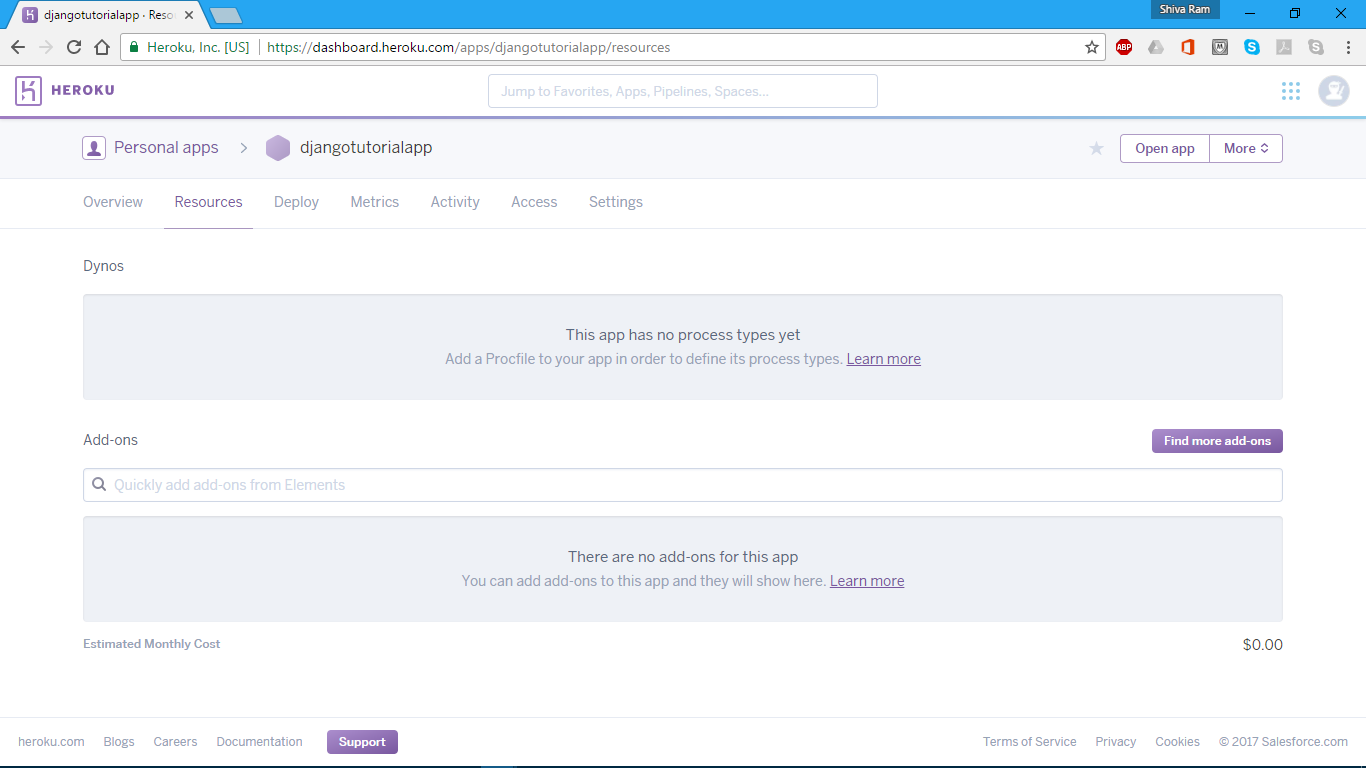
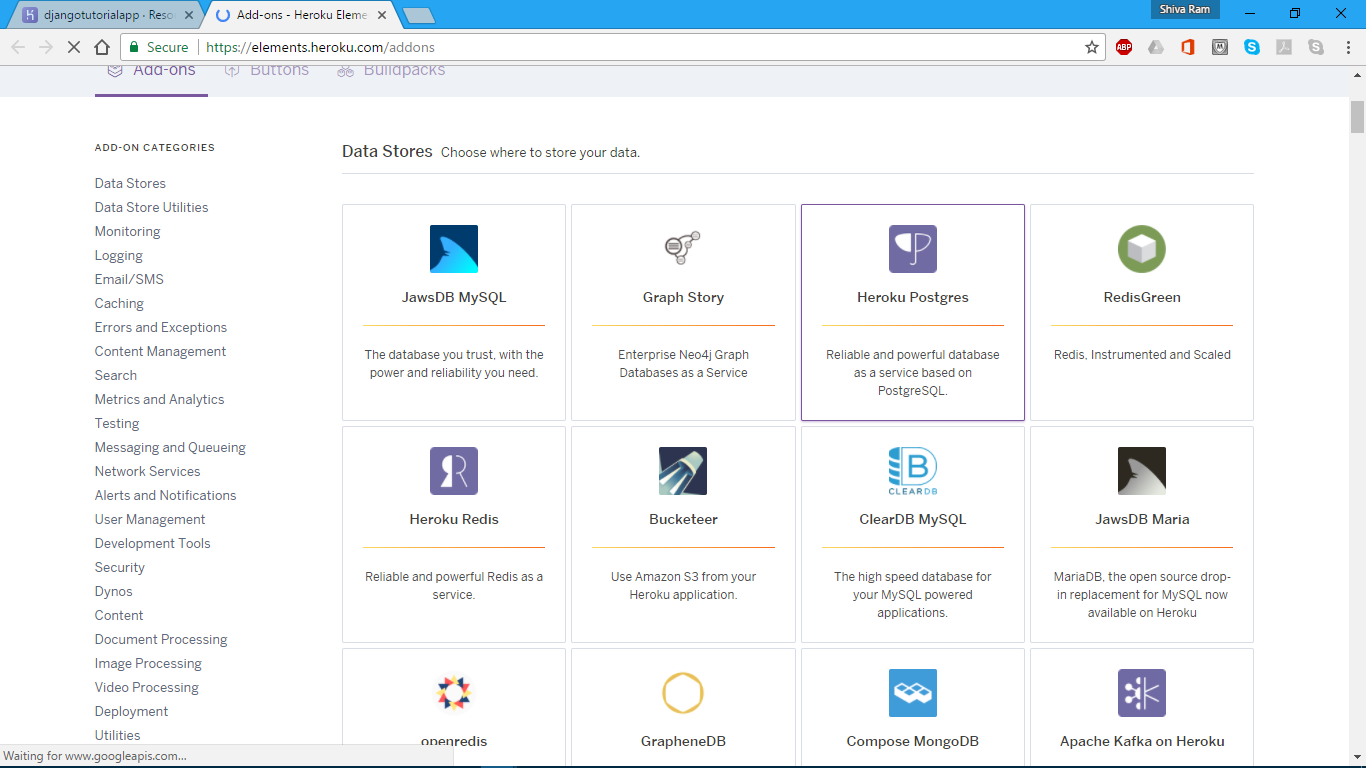
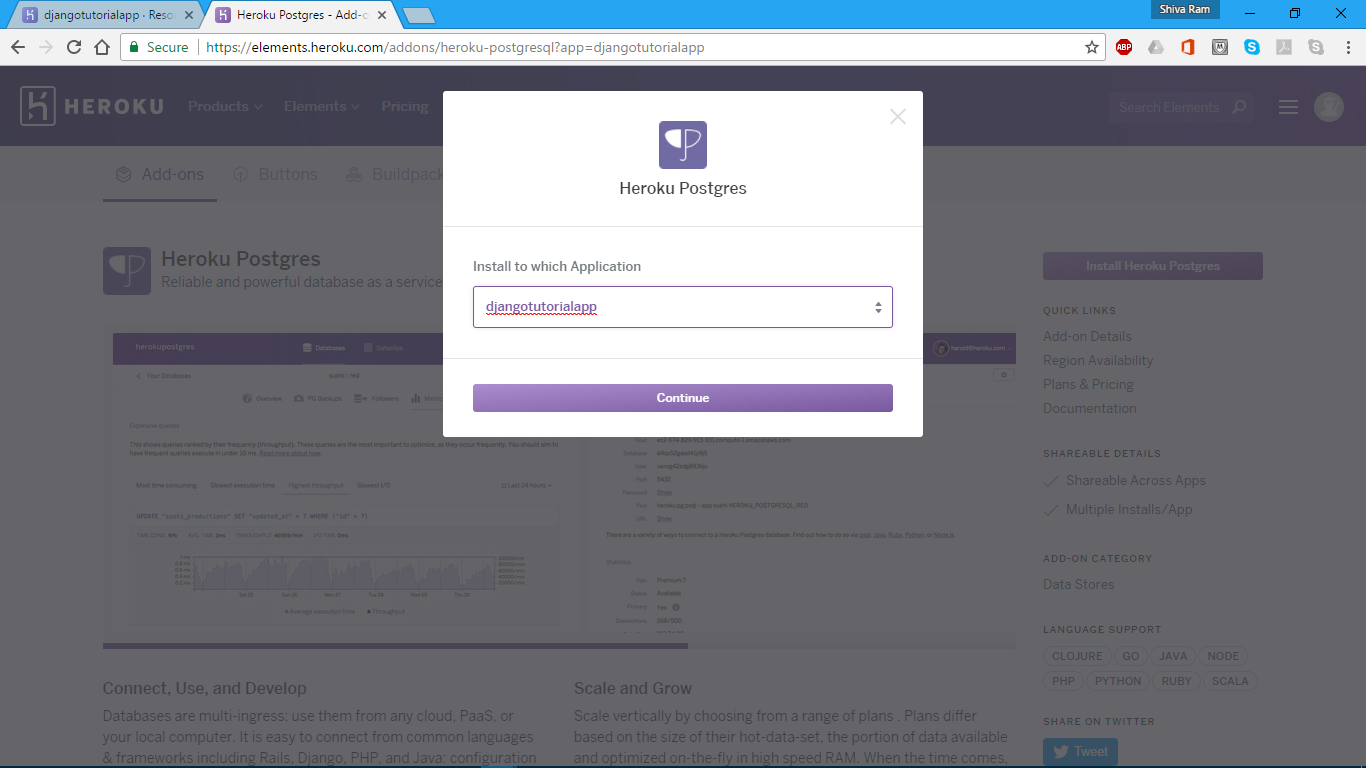
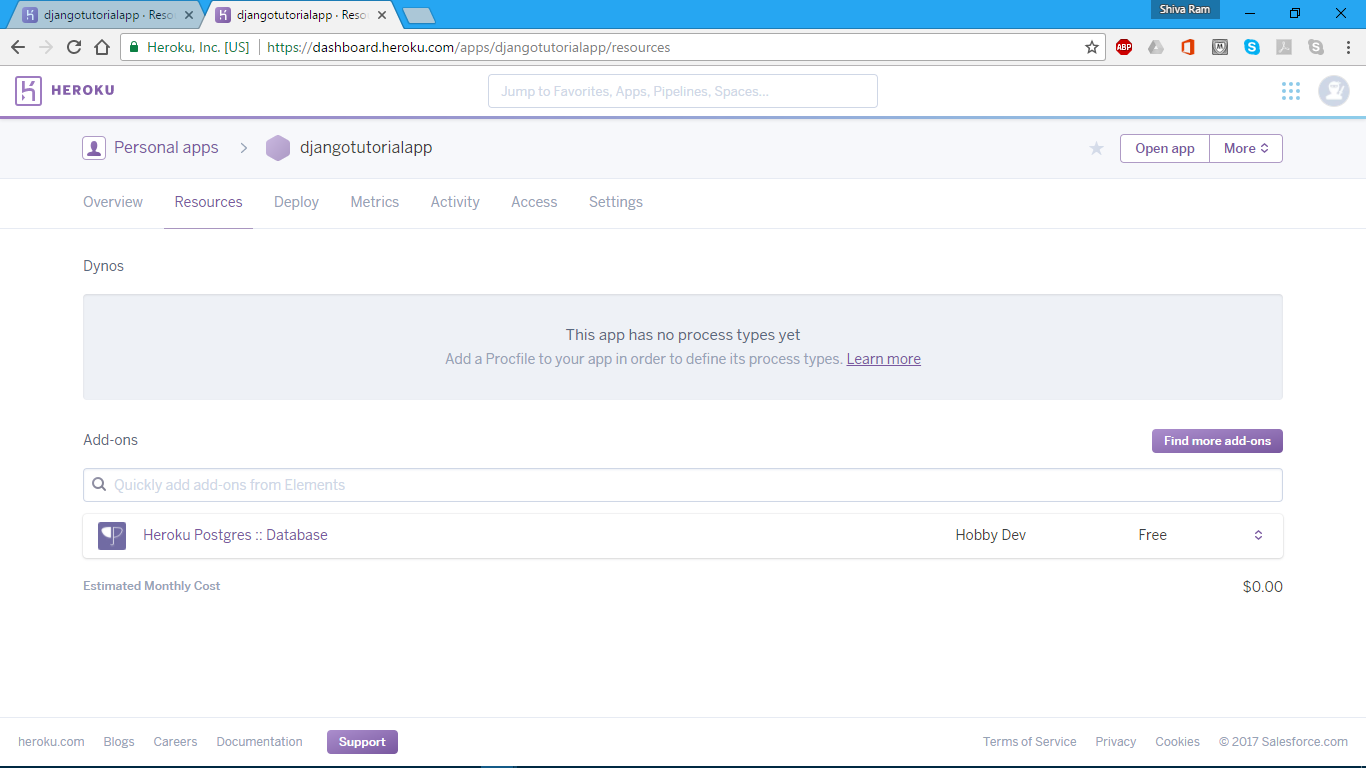
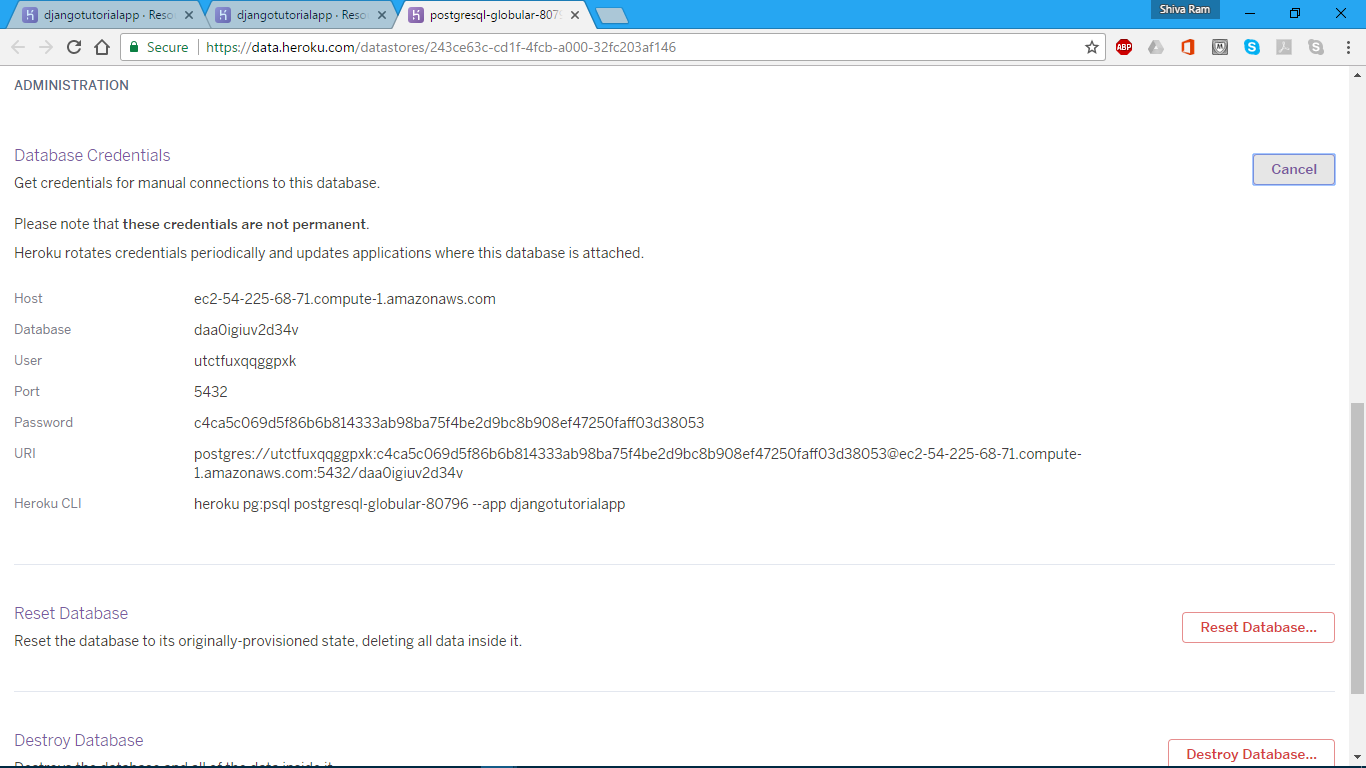
# Deploying your first application using Python & Django

**Prerequisites**

1. You have developed your application and it is successfully running, error free in the python development environment. This means that you can bring up the application and run it on 127.0.0.1:8000. The admin sites comes up and works in your browser at 127.0.0.1:8000/admin.

2. You have a solid and reliable WiFi connection. Be sure you are NOT connected to UNO’s guest WiFi. Deploying to a hosting site like Heroku will fail if you are using the guest WiFi at UNO!!

**Deploying your Application to Heroku**

* + 1. Firstly, signup for an Heroku account and login to your account.
    2. Create a new Heroku instance, by clicking “New” and selecting “Create a New App” option on top right of your dashboard. 
    3. Give your application a unique name and hit “Create App”. Now you should see this screen, 
    4. While you are here, the next step is to setup a database for the application that you are going to deploy on Heroku. We will choose Postgresql as our backend database. For this, click on “Resources” tab and you should see this screen, 
    5. Now click on “Find more add-ons” button, you will be now redirected to a page where you find numerous options/applications that Heroku supports, but our interest is “Postgresql” – find it and select that option. 
    6. Once you click the option, it redirects to a different page, towards right you see “Login to install” – click the option and click on “Install Heroku Postgres” – now, on the pop-up window select the appropriate application for which you are attaching Postgres to and hit continue. 
    7. It takes a moment to install the database to your application, once done you should see a pop-up asking for plan details – select the default “Hobby Dev-Free” option and hit “Provision”.
    8. You should now see the database has been installed to your application, 
    9. While you are here, click on the “Heroku Postgres : Database” and you will be directed to a new window where you see certain configurations, scroll down and click on “View Credentials” to view “Database Credentials” we will need these to link our local application to Heroku during deployment. 
    10. Now switch back to our application on local machine, we will come back to Heroku in a bit.
    11. First, open up code editor and within your project directory create an empty file and name it “Procfile” this tells Heroku about application’s process types and entry points. Once you create the file, add this line and save it.

web: gunicorn yourprojectname.wsgi --log-file -

* + 1. Open up “GIT BASH” and navigate to the project directory. Here, we would need to install few “requirements” to make our application compatible with Heroku production environment.
    2. For this we would need to make a requirements list within our project directory, to do this follow these commands,
* First we install gunicorn – it is a production web server which Heroku recommends for Django applications.

pip install gunicorn

then,

pip freeze > requirements.txt

* Second, database configuration where we store config variables,

pip install dj\_database\_url

then,

pip freeze > requirements.txt

* Once you install dj-database-url, go to settings.py and add these lines,

import dj\_database\_url -- place this code right under import os line at the top.

Then add the below code snippet at the bottom.

# Update database configuration with $DATABASE\_URL.

db\_from\_env = dj\_database\_url.config(conn\_max\_age=500)

DATABASES['default'].update(db\_from\_env)

* While you are at settings.py, it is also time to add certain settings for our STATIC assets, find the below section in settings.py and replace the content with the below code,

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/1.9/howto/static-files/

PROJECT\_ROOT = os.path.dirname(os.path.abspath(\_\_file\_\_))

STATIC\_ROOT = os.path.join(PROJECT\_ROOT, 'staticfiles')

STATIC\_URL = '/static/'

# Extra places for collectstatic to find static files.

STATICFILES\_DIRS = (

os.path.join(PROJECT\_ROOT, 'static'),

)

* Now switch back to BASH and install something called “whitenoise” the application that supports static files in production environment.

pip install whitenoise

then,

pip freeze > requirements.txt

* Go back to settings.py and add this line,

# Simplified static file serving.

# https://warehouse.python.org/project/whitenoise/

STATICFILES\_STORAGE = 'whitenoise.django.GzipManifestStaticFilesStorage'

And in your wsgi.py file add these lines,

from django.core.wsgi import get\_wsgi\_application

from whitenoise.django import DjangoWhiteNoise

application = get\_wsgi\_application() – this line of code is already present

application = DjangoWhiteNoise(application)

* Open your requirements.txt file and add this line,

psycopg2==2.7.1

* Just to make sure all your requirements were installed correctly, run this command.

pip install -r requirements.txt

* While you are in code editor, create an empty file called runtime.txt and add this line and save it,

python-3.6.1

Please Note: you might be using a different version of python on your local machine, hence it is advised to know which version you are using assigning that number here. This is important because your entire application would become compatible with this version and the same will be installed in production as well. Incorrect information will lead to compatibility issues during deployment.

* + 1. We would now need to separate our “settings” to “local” & “global” for this, create a new Python file “local\_settings.py” alongside your “settings.py” and this piece of code and save it,

**import** os  
  
BASE\_DIR = os.path.dirname(os.path.dirname(\_\_file\_\_))  
  
DATABASES = {  
 **'default'**: {  
 **'ENGINE'**: **'django.db.backends.sqlite3'**,  
 **'NAME'**: os.path.join(BASE\_DIR, **'db.sqlite3'**),  
 }  
}  
  
DEBUG = **True**

* + 1. Next up, add/edit this piece of code to your “settings.py”

SECURE\_PROXY\_SSL\_HEADER = ('HTTP\_X\_FORWARDED\_PROTO', 'https')

ALLOWED\_HOSTS = ['\*']

DEBUG = False

try:

from .local\_settings import \*

except ImportError:

pass

Please note, your “settings.py” already has **ALLOWED\_HOSTS** and **DEBUG** set to default values, you just need to modify these settings as above.

* + 1. Next we will create a “.gitignore” file that tells GIT **NOT** to export these files to production environment, from your code editor, right click on your project directory “Create New File” name it .gitignore and hit ok and associate it with Python (if asked).
    2. Open the “.gitignore” clear everything and add these lines to it and save it,

\*.pyc

db.sqlite3

myvenv

\_\_pycache\_\_

local\_settings.py

* + 1. Now go back to Heroku database credentials tab that I asked you leave open, we will now copy those credentials into our “settings.py”, search for “DATABASES” section and edit it this way,

DATABASES = {  
 **'default'**: {  
 **'ENGINE'**: **'django.db.backends.postgresql'**,  
 **'NAME'**: **'database name from Heroku'**,  
 **'USER'**: **'User name'**,  
 **'PASSWORD'**: **'Paste the password here'**,  
 **'HOST'**: **'add host here'**,  
 **'PORT'**: **'5432'**,  
 }  
}

* + 1. It is now time to deploy the application, before this just make sure it is running fine on your local machine, this confirms that all the settings are now properly configured.

Make sure that you have HEROKU TOOLBELT (HEROKU CLT) installed on our laptop.

HEROKU files were provided by the instructor. Eg. Proc file, runtime , .gitignore. locat-setting etc.

-------------------------------------------------------------------------------------------------------------------------

* + 1. Open “GIT CMD” navigate to your project directory and enter this commands to login to your Heroku account through GIT (use your Heroku login credentials),

|  |
| --- |
| C:\python\blog\george>heroku login  Enter your Heroku credentials.  Email: [groyce@unomaha.edu](mailto:groyce@unomaha.edu)  Password: \*\*\*\*\*\*\*\*\* |

* + 1. Open GIT BASH, and these commands in series

git status

git init

heroku git:remote -a heroku remote name such as



git add -A .

git commit -m "additional files and changes for Heroku"

git push heroku master

* + 1. Once the push is successful, we will now migrate the db files into Heroku using this command,

heroku run python manage.py migrate

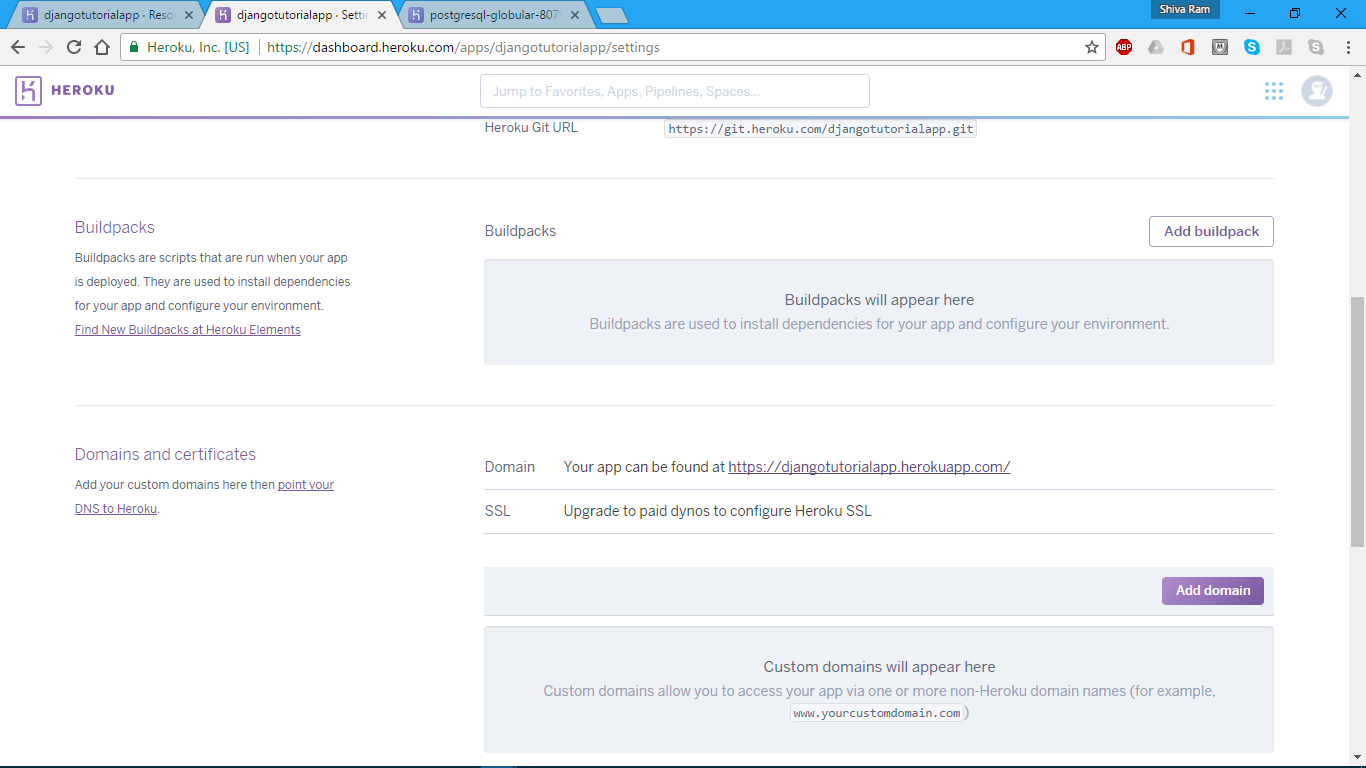
* + 1. After successfully migrating your application, it is time to create a superuser for Heroku as well,

we will do this using following command,

heroku run python manage.py createsuperuser

Enter credentials and finish the process.

* + 1. Now go back to Heroku, click on “settings” tab, scroll down and click on the domain link,



Which will now open a new window with your application, try logging in using the credentials you have used during “createsuperuser” process.

* + 1. That’s it. Your application is now live! Confirm by running following command or go to Heroku and click on domain name.

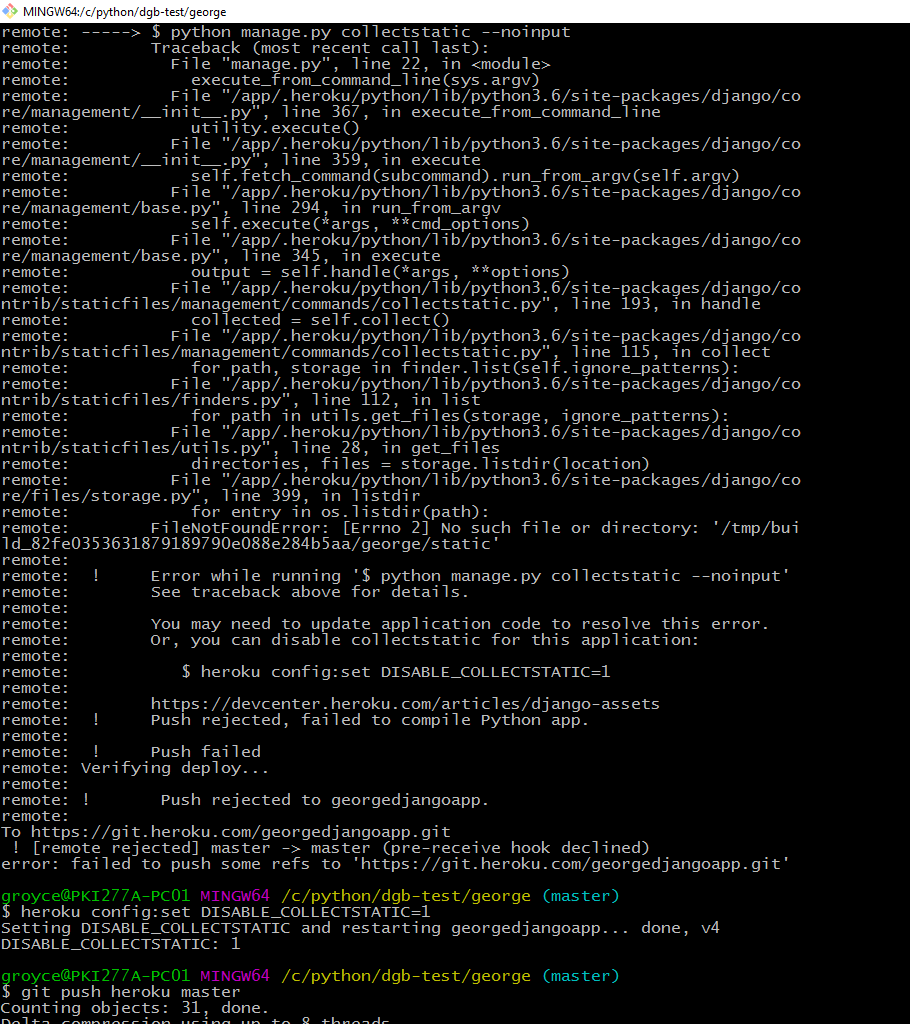
heroku open

**Potential Error Scenarios:**

It is highly likely that you might encounter errors during deployment, it’s actually a good thing, in such cases debug the application, read error codes, understand root cause and fix them as you go.

I have listed some potential error scenarios and how to tackle them just to save some time. If your errors are something out of the box, don’t worry you are not alone, there are numerous articles on the internet, you just need to search. Good sources are [Stackoverflow](https://stackoverflow.com/), [Heroku](https://devcenter.heroku.com/articles/django-app-configuration), [Djangoproject](https://www.djangoproject.com/), [Djangogirls](https://djangogirls.gitbooks.io/django-girls-tutorial-extensions/content/) websites.

1. During deployment, it is possible that git throws and error saying “push failed” the error looks like this,



In this scenario, simply enter the command,

heroku config:set DISABLE\_COLLECTSTATIC=1

or

heroku config:set DISABLE\_COLLECTSTATIC=0

depending on the type of error – it should be displayed on bash console.

then,

git push heroku master

heroku run python manage.py migrate

2) ECONNRESET while running migration to Heroku, solution is either connect through UNOMAHA VPN or connect to UNOSECURE wifi (if on campus).

3) Third type of error might be

*“ImproperlyConfigured: You must either define the environment variable DJANGO\_SETTINGS\_MODULE or call settings.configure() before accessing settings”*

In this case, run these commands in GIT Bash console,

export DJANGO\_SETTINGS\_MODULE=yourappname.settings

heroku config:set DJANGO\_SETTINGS\_MODULE= yourappname.settings

4) “Bad Server 500” – if you face this error go to settings, change DEBUG = True and run following commands,

Git add .

Git commit -m “message”

Git push Heroku master

heroku run python manage.py migrate

If you encounter any other errors, please share and contribute with findings and solutions that worked for you.

**Acknowledgements**

This tutorial was created as a part of independent study by Shiva Ram Chennapragada during Summer 2017.