Setup and Installation

OVERVIEW:
This project sets up a webserver that contains a map interface. The webserver sends requests to Tilecache when it needs to load new chunks of the map. If Tilecache does not have the tiles already cached it sends a request to OGCServer to render a new tile. OGCServer uses mapnik to talk to it's local PostGRES database that has GIS data stored in its tables and renders the tile.

PROJECT SOURCE:

1.1) Download git hub repository from https://github.com/junosw/juno-public

1.2) Or setup a git repository with the following:

- Install git with `sudo apt-get install git`
- Initialize git with `git init`
- enter `git clone git@github.com:junosw/juno-public.git`
- *NOTE: this repository subject to change*

DATABASE SETUP:

1.1) First install the regular postgres program:

```
sudo apt-get install postgresql
```

1.2) Then install the PostGIS addon:

```
sudo apt-get install python-software-properties
sudo apt-add-repository ppa:ubuntugis/ppa
sudo apt-get update
sudo apt-get install postgresql-9.1-postgis
```

1.3) Then you must set up a super user for the database. There are two methods for this.

1.3a) Setup the generic user:

The easier to set up is to use the user that PostGRES creates when it installs. The standard name it uses is postgres. at this point all you have to do is set a password:

```
sudo -u postgres psql
\password postgres
```

Then enter in your password, and then type it in again to confirm. And lastly do Ctrl-D to exit.

Note: for our example setup we use the postgres user and use the password shotfirst. If you use a different password you must edit the map.xml file as detailed in the OGCServer setup section.

1.3b) Setup personal user:

To create a user under the name of your computer do the following. You may prefer to do this setup so you don't have to declare your user as postgres in

other steps.

```
sudo -u createuser <username>
```

it will ask you if you want to make the user a super user. For simplicity it is recommended that you do, but if this computer is used by more than one person this is something you should look into that more on postgresl.org for security reasons.
Note: for our example setup we use the postgres user and use the password shotfirst. Whether or not you use the same password as us, you have to at least specify your username in the map.xml file as specified in the OGCServer setup section.

Then you have to set up your password.

```
sudo -u postgres psql
```

```
\password postgres
```

Then enter in your password, and then type it in again to confirm. And lastly do Ctrl-D to exit.

1.4) Setup database.

Create database: `sudo -u postgres createdb <dbname>`

If you did the second setup for a user just use: `createdb <dbname>`

Note: for our example setup we use the dbname falcon. If you use a different dbname you must edit the map.xml file as detailed in the OGCServer setup section.

And then you need to set your new database to be able to use postgis

data.

```
sudo -u postgres psql -d <dbname> -f
/usr/share/postgresql/9.1/contrib/postgis-2.0/postgis.sql

sudo -u postgres psql -d <dbname> -f
/usr/share/postgresql/9.1/contrib/postgis-2.0/spatial_ref_sys.sql

sudo -u postgres psql -d <dbname> -f
/usr/share/postgresql/9.1/contrib/postgis_comments.sql

sudo -u postgres psql -d <dbname> -f
/usr/share/postgresql/9.1/contrib/postgis-2.0/rtpostgis.sql
```

Or (for personal user)

```
psql -d <dbname> -f /usr/share/postgresql/9.1/contrib/postgis-2.0/postgis.sql

psql -d <dbname> -f
/usr/share/postgresql/9.1/contrib/postgis-2.0/spatial_ref_sys.sql

psql -d <dbname> -f /usr/share/postgresql/9.1/contrib/postgis_comments.sql

psql -d <dbname> -f /usr/share/postgresql/9.1/contrib/postgis-2.0/rtpostgis.sql
```

1.5) Create a table of data.

First you have to navigate to the folder that you have the shapefile in a terminal. We provide you with the world-borders-dataset from the git repository, or if you don't have any shape files you can go to wiki.openstreetmap.org/wiki/Shapefiles. Once you are in the folder with your shapefile you use the following commands.

```
shp2pgsql -c -I -W "LATIN1" -s 4326 <shapefile> <schema>.<table> > world.sql

sudo -u postgres psql -d <dbname> -f world.sql
```

Or (for personal user)

```
psql -d <dbname> -f world.sql
```

Note: for our example setup we use the table name world. If you use a different table name you must edit the map.xml file as detailed in the OGCServer setup section.

PLAY FRAMEWORK

- Download play framework from http://www.playframework.com/download (2.0.2)
    - You will need JDK to run play. It can be found here: http://www.oracle.com/technetwork/java/javase/downloads/index.html (7u21 -- extracted it is 1.7.0_21)
    - Play installation instructions: http://www.playframework.com/documentation/2.1.1/Installin
    - Set path variables for play and java:
        - Type `gedit ~/.profile`
        - Scroll to the bottom of the file and add

```
PATH=$PATH:/<path to play folder>
export PATH
PATH=$PATH:/<path to java folder>
export PATH
```

        - Restart the computer
    - type ‘play’ into the terminal to ensure the PATH variables are set correctly

INTELLIJ

- Download intellij from http://devnet.jetbrains.com/docs/DOC-1228 (Version 12.0.4 Community Edition, Build ic-123.169)

- Set the java path variables for intellij at the bottom of .profile again:

```
gedit ~/.profile
export IDEA_JDK=/<path to java folder>
export JAVA_HOME=/<path to java folder>
```

- Download and install intellij plugins:

    - scala plugin (version 0.7.134): http://plugins.jetbrains.com/plugin/1347?pr=

    - play plugin (version 0.2.16): http://plugins.jetbrains.com/plugin/index?pr=&pluginId=7080

    - Run intellij (idea.sh in the bin folder)
        - Select “Configure → Plugins” from the quick start menu *or* “File → Settings → Plugins” from the main menu
        - Select “Install plugin from disk” → Scala
        - Select “Install plugin from disk” → Play 2.0

- Type the following in juno-shared (and anytime you make a change to it hereafter):

```
play
clean
compile
publish-local
```

- Type `play idea` inside the slippy-map, tile-cache, and map-server folders from terminal (once in each)

- NOTE -- Open Intellij projects with:
    - File → Open
    - Select <play project name>.iml from slippy-map, tile-cache, map-server, or juno-public

OPEN LAYERS

- The Open Layers (version 2.12) library comes packaged with our git repository

- Or, download Open Layers from http://openlayers.org/

MAPNIK:

- For installation, follow the instructions at https://github.com/mapnik/mapnik/wiki/UbuntuInstallation (version 2.2)

TILECACHE:

- Requires python paste:
  ```
  sudo apt-get install python-paste
  ```

- Requires python setuptools
  ```
  sudo apt-get install python-setuptools
  ```

- In the git repository, go to tile-cache/tilecache and type
  ```
  sudo python setup.py install
  python tilecache_seed.py
  ```

- Or download Tilecache at http://tilecache.org/

- In juno-public/tile-cache/tilecache/tilecache.cfg at the very bottom are three sections for our different layers that we use with openlayers. “[shape]” is for our main layer which is called “world” in the slippy-map project. “[custom]” is commented out but can be used if desired. You will also need to uncomment the code for custom in the slippy-map project and in the map.xml file for ogcserver. You will also have to setup a table in the database called “custom” or change the name of the table in map.xml to match what you call it in the database.

OGC SERVER:

1.1) Install dependencies for OGC Server:

- PIL -- python imaging library
  - Get python.h with `sudo apt-get install python-dev`
  - http://www.pythonware.com/products/pil/ (version 1.1.7 or greater)
  - Run `sudo python setup.py install`

- lxml -- xml parsing in python
  - `sudo apt-get install python-lxml`
  - Install libxml2 and libsxlt with `sudo apt-get install libxml2-dev`
  - Install libsxlt from https://launchpad.net/ubuntu/+source/libxslt/1.1.28-1ubuntu1 (version 1.1.28 or greater)
    - `./configure`
    - `make`
    - `sudo make install`
  - lxml may be already installed with python. If you aren’t sure, do all the other dependencies first and see if OGCserver installation will work.
- Pastescript
  - `sudo apt-get install python-pastescript`
- WebOB

- `sudo apt-get install python-webob`

1.2) OGCserver

- Download from https://github.com/mapnik/OGCServer (version 0.1.0 or greater)
- Type `sudo python setup.py install`

1.3) Setup Map.xml:

In the OGCServer-master folder and inside the demo subfolder You'll want to edit the map.xml file to fit your setup.

```
<Parameter name="host">localhost</Parameter>
<Parameter name="user">postgres</Parameter>
<Parameter name="password">shotfirst</Parameter>
<Parameter name="dbname">falcon</Parameter>
<Parameter name="table">world</Parameter>
<Parameter name="type">postgis</Parameter>
```

- host should be left as localhost unless you are running the OGCServer on a different machine than the PostgreSQL database is located.
- user will either be postgres or the user that you created in the database setup.
- password will be the password you used in the database setup.
- dbname will be the name that you used in the database setup.
- table will be the name of the specific table you want to pull the data from. In our example setup we only have the "world" table, you may have multiple in your setup.
- type should always be postgis for the way we have it set up. If you are going to use a different method of storing data of your shape files then you'll have to look into to other types that are supported.

The below section that is commented out with the layer named custom is there from our custom map data that we had imported. Since it is not included in our basic instructions it is commented out. It is not completely removed as it serves a good example of creating multiple maps to view.

RUNNING THE APPLICATION

Assuming all the programs are installed and linked together correctly, the following will run the system with the default demo data we have in the database.

- Open 4 terminals
    - In one, navigate to the slippy-map folder in juno-public-master and type
        - `play`
        - `run 9000` (starts slippy-map and openlayers)
    - In the second, navigate to the tile-cache folder in juno-public-master and type
        - `play`
        - `run 9001` (starts the tilecache application)
    - In the third, navigate to the tile-cache/tilecache folder and type `./tilecache_http_server.py` (starts the tilecache server)
    - In the fourth, navigate to the OGCServer-master folder and type `./bin/ogcserver-local.py demo/map.xml` (starts the external mapnik server)
- Open a web browser (chrome seems to work best) and go to
    - localhost:9000
- You should be seeing an interactive map of the world (world borders dataset by default -- purple vector-base map)
- Notes:
    - If you choose to use the map-server application and make your

own server, you can run the play project on port 9002
    - All ports are easy to change, but make sure that the corresponding listening programs change as well (for example, the tilecache.cfg file points to localhost:9002 to connect to the map-server in order to get the tiles it needs rendered)

TOOLS, TIPS, & RESOURCES:

- Basic mapnik tutorial to get some familiarization with the program https://github.com/mapnik/mapnik/wiki/GettingStartedInPython

- QGIS Map Creation Tool (version -- 2.6.2 or greater with updated dependencies) http://hub.qgis.org/projects/quantum-gis/wiki/Download

    - Tutorial: http://infogeoblog.wordpress.com/2012/11/07/cartography-in-qgis/
    - To add your new shape files into the database, follow database step 1.5 again with your new data

- Set up your own map server with node-mapnik (beta)

    - Install dependencies
        - Install nodejs from http://nodejs.org/ (click the green 'install' button to download)
        - Install libmapnik with `sudo apt-get install libmapnik-dev`
        - Install nodejs-dev with `sudo apt-get install nodejs-dev`

    - Install node-mapnik from https://github.com/mapnik/node-mapnik
    - Follow README instructions for installation

    - Notes:
        - Currently, OGCserver is an external program servicing tiles

from mapnik to tilecache. It has some customizable properties, but in order to have full control, you will need to create your own map server. We have included a map-server play project in git-hub for you to develop on. If you set up the project with the instructions above, map-server is not used.

- node-mapnik gives you a java API for mapnik, allowing you to develop your own map server within the play map-server project. There does seem to be a number of module bugs with the current versions of nodejs and node-mapnik, so be wary of pending issues when using this approach.

- Using WebGL to animate the map
  - There are lots of things that WebGL can do, but unfortunately there aren't very many ways to use it with OpenLayers. What we were asked to do was to basically use WebGL to display the map as a 3D, rotatable plane, among other things. I did some research and what I found was that for the most part, WebGL can be used to animate html elements, but it uses a "picture" of the element to do the animation, but until the animation is complete, the element would not be usable. But, I found another option...
  - ReadyMap: http://readymap.com/websdk.html. This is an open source sdk that can be used with OpenLayers to display the map as a 3D globe. It utilizes WebGL but makes implementation easy, apparently. The only problem with ReadyMap is there is literally zero documentation that I would find for it. Even the README.txt is practically empty. There are examples that I looked at that might be helpful, but they are far too basic and don't explain a lot. They can be found here: http://demo.pelicanmapping.com/rmweb/webgl/tests/index.html. Basically, I couldn't get it working with what they have and I didn't

much useful information to get it working. There was something about needing to setup a php proxy on the server, but I couldn’t figure out how to do that in time. Hopefully you will have better luck.