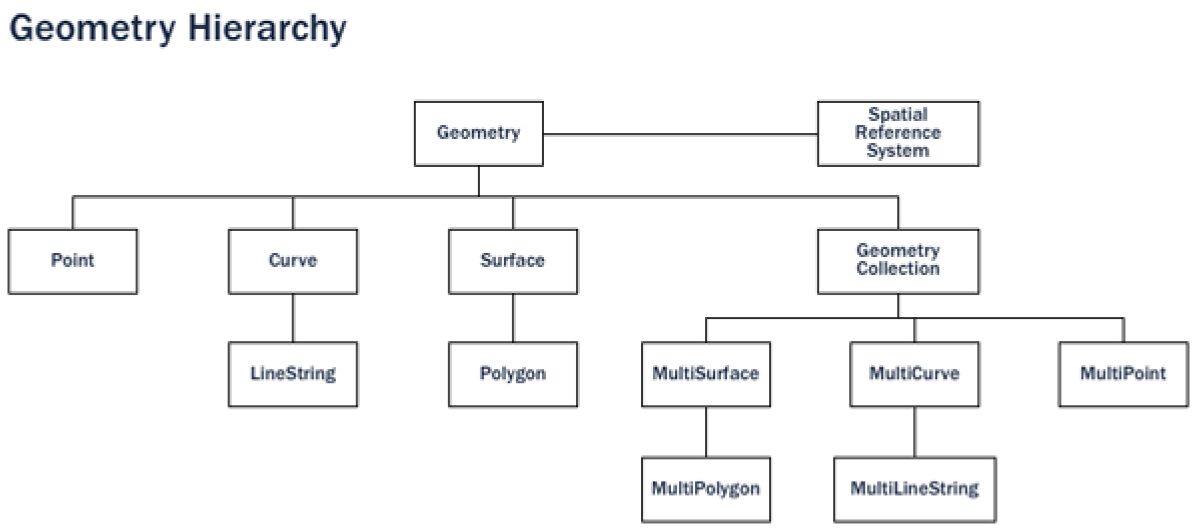
A summary of why PostgreSQL + PostGIS is an important factor in a GIS stack.

Why any GIS Application would need a database

A GIS Application has to deal with huge data in different forms, layer – both raster and vector. Ordinary data handling applications like Excel, MS Access cannot handle them and so we need Relational Database Systems, which provide the flexibility to store the data in different tables, each interlinked with others and provide the flexibility to edit, fetch and augment it on the fly.

Why special database (we call them spatial database) are needed for GIS Stack

An ordinary database has strings, numbers, and dates. A spatial database adds additional (spatial) types for representing **geographic features**. These spatial data types abstract and encapsulate spatial structures such as boundary and dimension. In many respects, spatial data types can be understood simply as shapes.



Source: https://postgis.net/workshops/postgis-intro/introduction.html

What is PostgreSQL

Once we are able to understand, why special databases are needed for GIS application, we look forward to what suits the requirement. There are many ORDBMS available in the market like ORACLE, Microsoft SQL etc., but PostgreSQL is often a preferred choice for many developers, specially open source community. So, let's explore, what PostgreSQL is all about.

PostgreSQL is a powerful, object-relational database management system (ORDBMS). It is released under a BSD-style license and is thus free and open source software. As with many other open source programs, PostgreSQL is not controlled by any single company, but has a global community of developers and companies to develop it.

PostgreSQL was designed from the very start with type extension in mind – the ability to add new data types, functions and access methods at run-time.

Why choose PostgreSQL?

A common question from people familiar with open source databases is, "Why wasn't PostGIS built on MySQL?".

PostgreSQL has:

- Proven reliability and transactional integrity by default (ACID)
- Careful support for SQL standards (full SQL92)
- Pluggable type extension and function extension
- Community-oriented development model
- No limit on column sizes ("TOAST"able tuples) to support big GIS objects
- Generic index structure (GiST) to allow R-Tree index
- Easy to add custom functions

Combined, PostgreSQL provides a very easy development path to add new spatial types.

Why PostgreSQL alone is not sufficient

Evan Carol on stackexchange.com aptly summarizes this. PostgreSQL does not support Spatial data types. It supports geometric types. These are perfectly fine for some things, but they're totally separate from real world coordinate systems. Native types

- have no Spatial Reference System ID/SRSID.

- can never be re-projected.
- even on a spheroid, they provide very limited functionality.
- They're not standardized.
- They don't provide GIST indexes.

Hence we need a special application, which works on the top of PostgreSQL database and extends its capability to handle spatial data types. This is done by Postgis.

What is Postgis

PostGIS adds extra types (geometry, geography, raster and others) to the PostgreSQL database. It also adds functions, operators, and index enhancements that apply to these spatial types. These additonal functions, operators, index bindings and types augment the power of the core PostgreSQL DBMS, making it a fast, feature-plenty, and robust spatial database management system.

The latest release version of PostGIS now comes packaged with PostgreSQL.

Essentially, the above write-up summarizes, why a combination of Postgis + PostgreSQL has the ability to fully meet the requirement of a GIS stack for data management.

A summary of what Docker is and why Docker should be used to containerize PostgreSQL (with PostGIS)

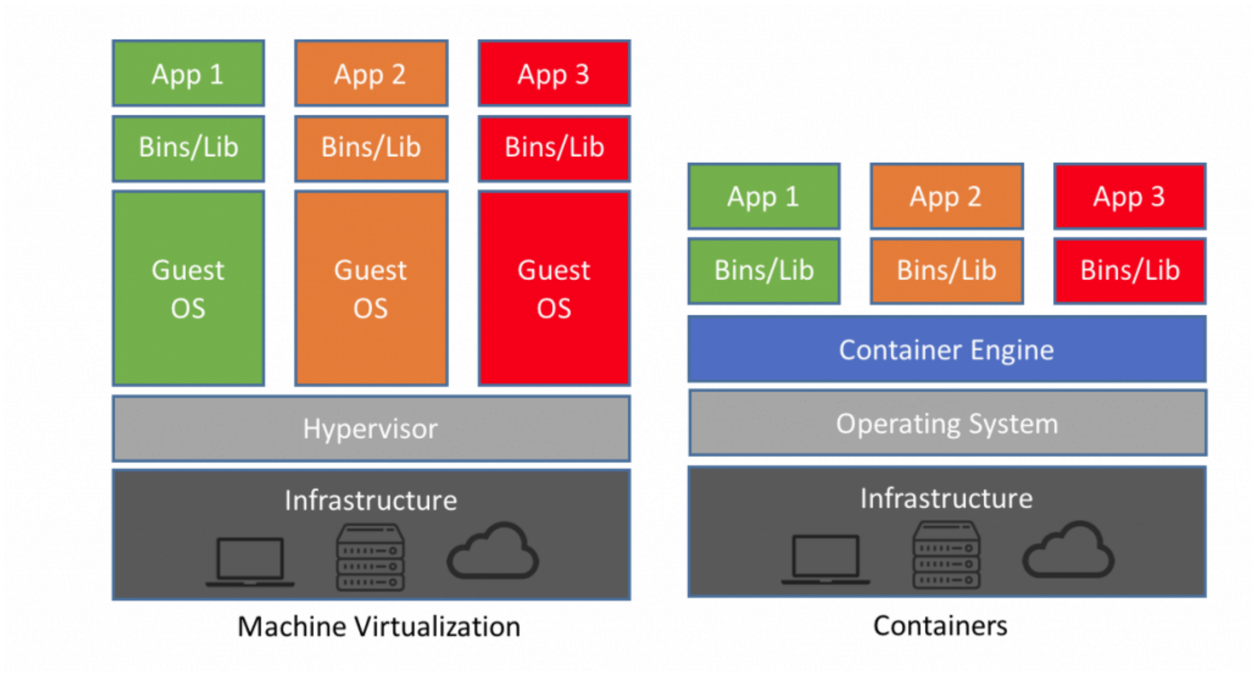
I visualize Docker as an extension of the concept of Virtual Machine in the sense that both provides a user the ability to use the host environment to different other use cases. For example you can run a linux machine within a Windows environment.

However, there are a lot of difference between them.

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code. [Source: https://opensource.com/resources/what-docker]

In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application.

The difference between Docker and a Virtual Machine can be easily understood from the following pictorial diagram.



Source: https://www.guru99.com/docker-tutorial.html

What is Docker Container

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

(Source: https://www.docker.com/resources/what-container)

So essentially, to run Postgis+PostgreSQL on a Windows machine, you just have to load Docker on the machine, pull a suitable Postgis image hub from Docker hub and run it within a container. (As I mentioned in the previous section of this writeup, PostgreSQL already come bundled with Postgis). It hardly takes a set of few commands to establish the setup.

Why to run containerizes Postgis+PostgreSQL

There are many advantages of running the spatial database combination of Postgis+PostgreSQl in a containerized environment like Docker.

For instance, each application can have its own dedicated database. The data base just becomes an on-demand utility. You no longer require the services of large database servers and the huge overheads they require for their upkeep and maintenance.

Containerized databases separate storage from compute, meaning storage performance and capacity can be scaled independently of compute resources. This provides more flexibility in upfront database capacity planning and provisioning, since changes are much easier to make later.

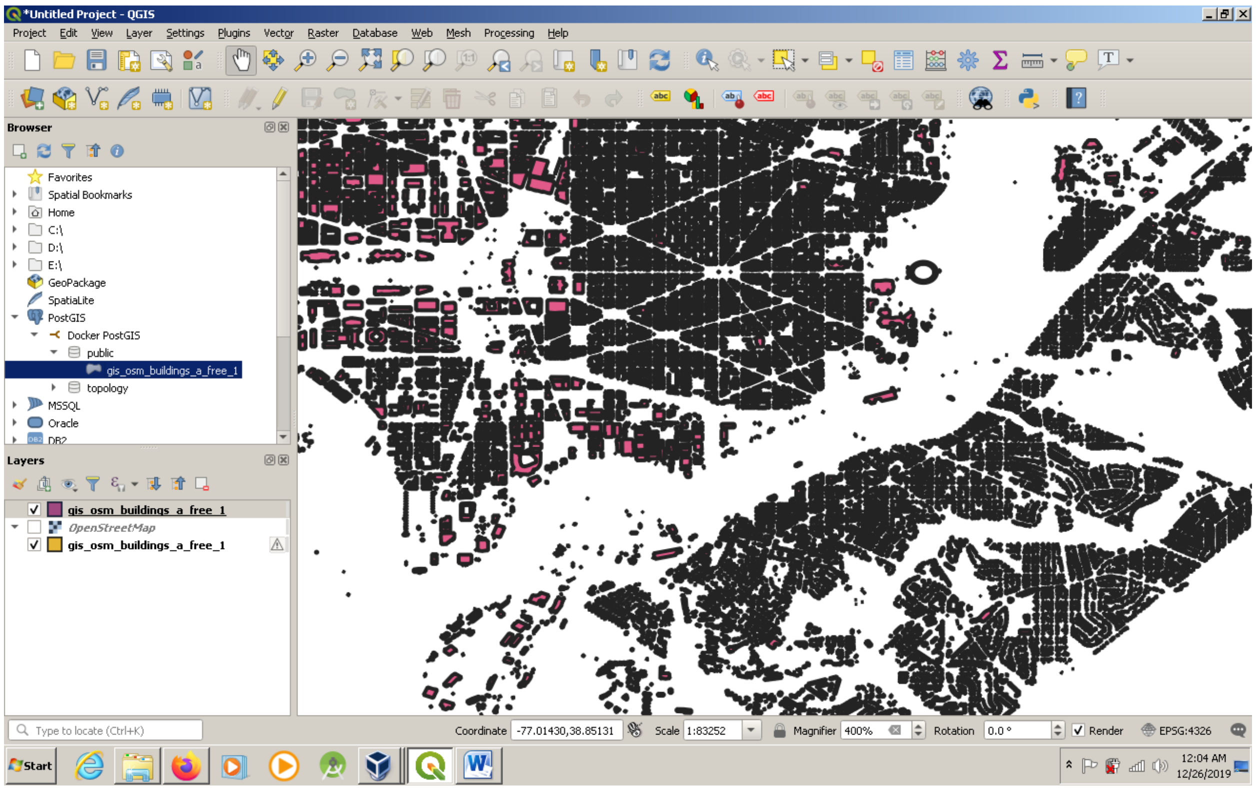
Containerized database provide flexibility and speed of operations in terms of management and upkeep of database.

(source: https://containerjournal.com/topics/container-management/11-things-to-know-about-databases-and-postgres-in-containers/)

My Experience with Docker

My newer machine running Windows 10, has recently got a crash and is not yet up and running. For the past few days, I have been working on my old machine, which is a Windows 7 machine, with a meagre 4 GB of RAM. After a lot of hesitation, I have loaded docker as well as postgis on this Windows 7 machine and have been able to connect the QGIS and complete this task.

It gave me huge surprise that the docker and able to run the container having postgis while the Windows 7 machine was also running QGIS – it talks a lot about how efficiently these applications utlize the resources.

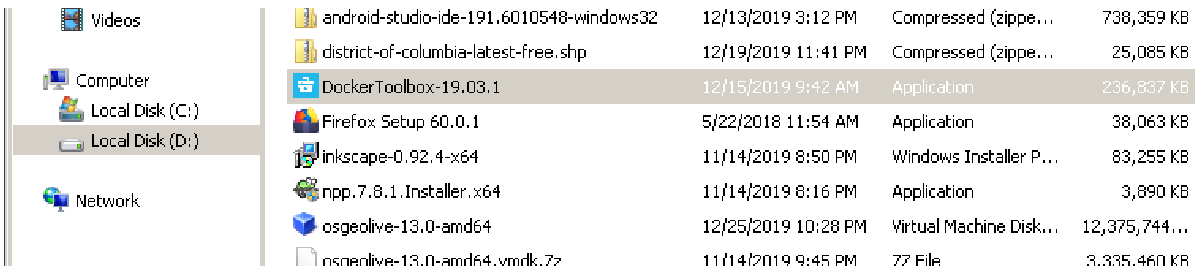


Coordinate -77.01430,38.85131 Scale 1:83252 Magnifier 400% Rotation 0.0 ° ✓ Render EPSG:4326

Environment Setup

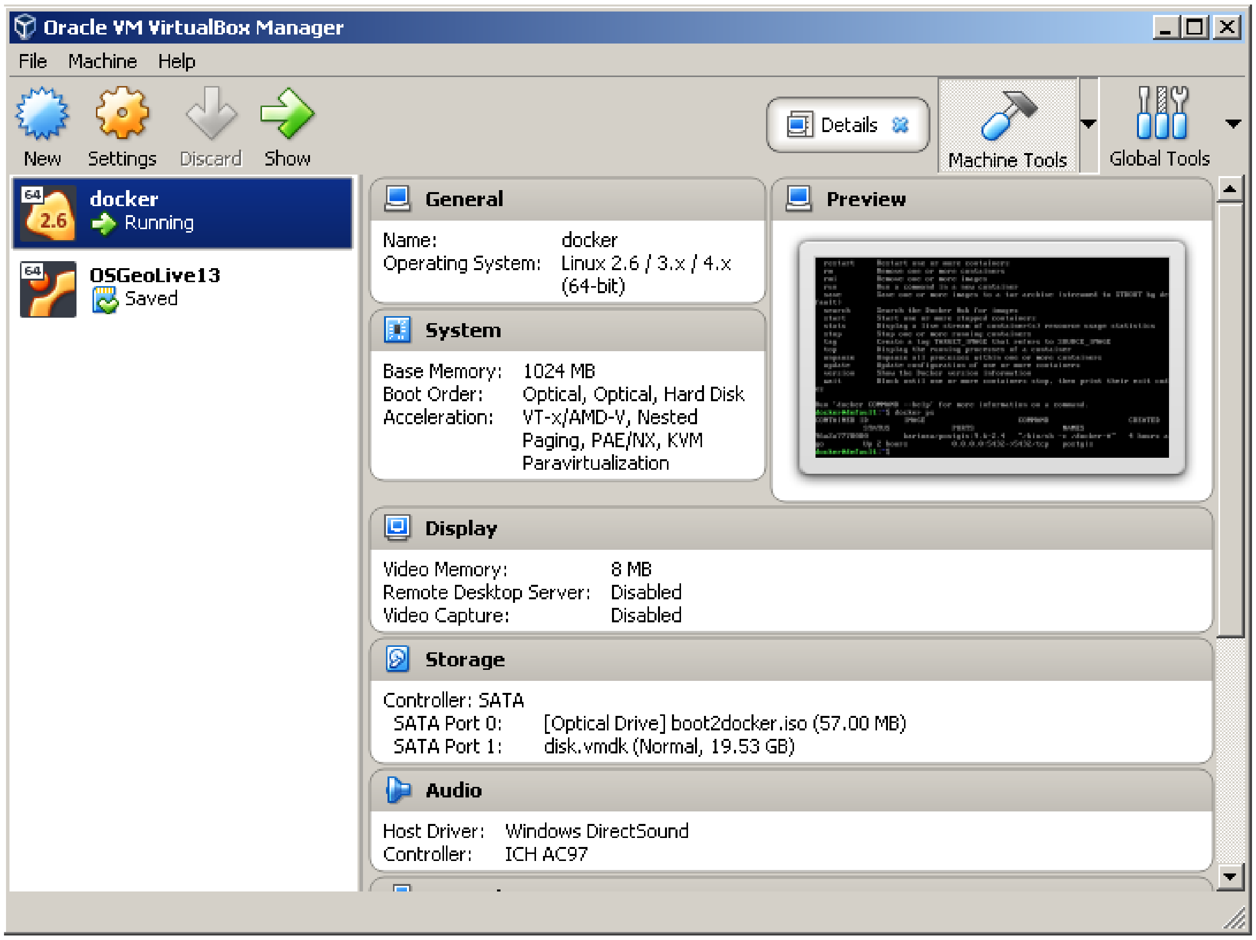
Installation of Docker on local machine

I am working on an older machine, which does not support the installation of Docker for Windows. So, I had to install Docker Toolbox. I downloaded Docker Toolbox in D:// drive.



android-studio-ide-191.6010548-windows32	12/13/2019 3:12 PM	Compressed (zippe...	738,359 KB
district-of-columbia-latest-free.shp	12/19/2019 11:41 PM	Compressed (zippe...	25,085 KB
DockerToolbox-19.03.1	12/15/2019 9:42 AM	Application	236,837 KB
Firefox Setup 60.0.1	5/22/2018 11:54 AM	Application	38,063 KB
inkscape-0.92.4-x64	11/14/2019 8:50 PM	Windows Installer P...	83,255 KB
npp.7.8.1.Installer.x64	11/14/2019 8:16 PM	Application	3,890 KB
osgeolive-13.0-amd64	12/25/2019 10:28 PM	Virtual Machine Disk...	12,375,744...
osgeolive-13.0-amd64.vmdk.7z	11/14/2019 9:45 PM	7Z File	3,335,460 KB

And ran the installer to install the Docker in my already existing Oracle VM VirtualBox.



General

```
Name:              docker
Operating System:  Linux 2.6 / 3.x / 4.x
                   (64-bit)
```

System

```
Base Memory:   1024 MB
Boot Order:    Optical, Optical, Hard Disk
Acceleration:  VT-x/AMD-V, Nested
               Paging, PAE/NX, KVM
               Paravirtualization
```

Display

```
Video Memory:            8 MB
Remote Desktop Server:   Disabled
Video Capture:           Disabled
```

Storage

```
Controller: SATA
 SATA Port 0:     [Optical Drive] boot2docker.iso (57.00 MB)
 SATA Port 1:     disk.vmdk (Normal, 19.53 GB)
```

Audio

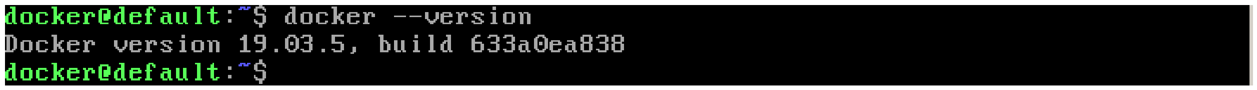
```
Host Driver:   Windows DirectSound
Controller:    ICH AC97
```

Get familiar with some basic Docker commands (docker run, docker pull, docker stop, docker ps, etc)

Docker is really a beast. The different combinations of command pieces can easily multiply to give hundreds of thousands of commands and I was just overwhelmed with the enormity and versatility they provide to a user.

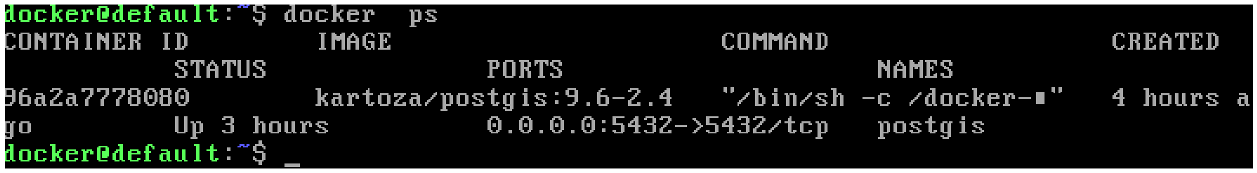
Starting with the basics, I quickly learnt following commands:

1. **docker –version:** This command is used to get the currently installed version of docker.



```
docker@default:~$ docker --version
Docker version 19.03.5, build 633a0ea838
docker@default:~$
```

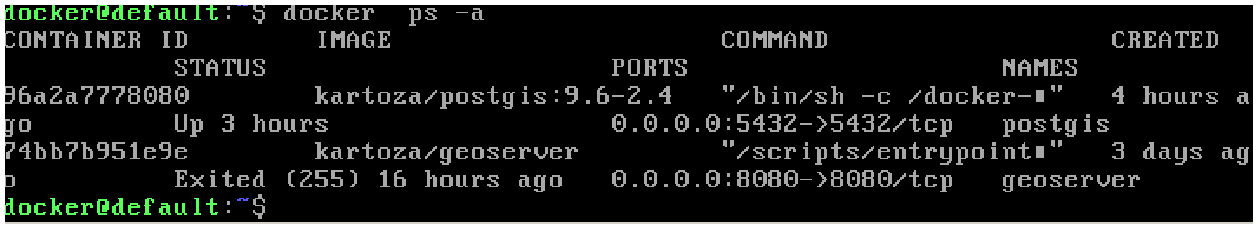
2. docker ps: This command is used to list the running containers.



```
docker@default:~$ docker  ps
CONTAINER ID        IMAGE                        COMMAND                  CREATED
           STATUS              PORTS                    NAMES
96a2a7778080        kartoza/postgis:9.6-2.4    "/bin/sh -c /docker-■"   4 hours a
go         Up 3 hours          0.0.0.0:5432->5432/tcp   postgis
docker@default:~$ _
```

As can be seen, right now my docker instance is running kartoza/postgis:9.6-2.4 container, whose name I have give as "postgis". The docker gives every container a unique Container Id. My docker has given the container id as 96a2a7778080.

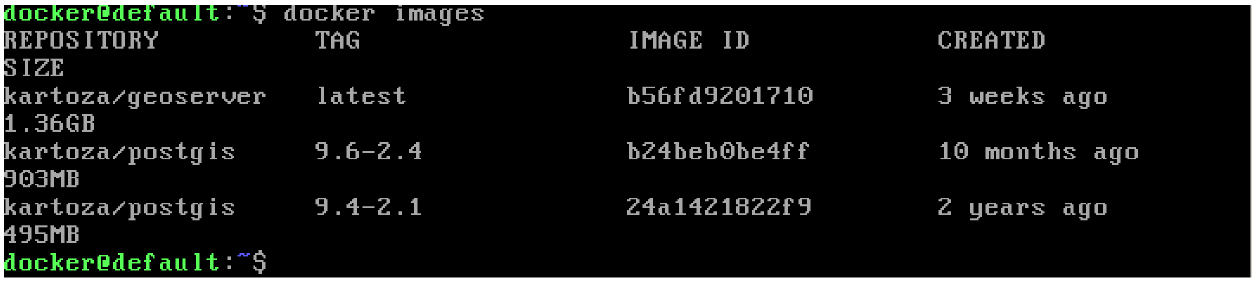
3. docker ps -a: If we give docker ps command with flag –a, then it gives us a list of all the containers, whether they are in running state or not.



```
docker@default:~$ docker  ps -a
CONTAINER ID        IMAGE                        COMMAND                  CREATED
           STATUS                      PORTS                    NAMES
96a2a7778080        kartoza/postgis:9.6-2.4    "/bin/sh -c /docker-■"   4 hours a
go         Up 3 hours                  0.0.0.0:5432->5432/tcp   postgis
74bb7b951e9e        kartoza/geoserver          "/scripts/entrypoint■"   3 days ag
o          Exited (255) 16 hours ago   0.0.0.0:8080->8080/tcp   geoserver
docker@default:~$
```

As can be seen from the above output, my docker is having two container – one is postgis and the other one is geoserver, which I have installed for another task of GCI 2019. The status of postgis container is "Up for 3 hours", while the status of the other container "geoserver" is "Exited", because it is currently not in a running condition.

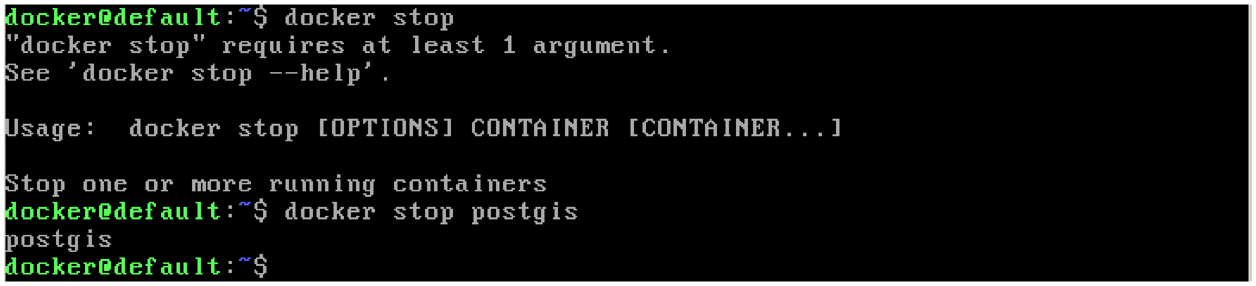
4. docker images: if you want to know what images are already present on your docker setup locally, you can give this command.



```
docker@default:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
kartoza/geoserver   latest              b56fd9201710        3 weeks ago
1.36GB
kartoza/postgis     9.6-2.4             b24beb0be4ff        10 months ago
903MB
kartoza/postgis     9.4-2.1             24a1421822f9        2 years ago
495MB
docker@default:~$
```

This tells us the images, which we have downloaded in the docker.

5. docker stop: This command is used to stop the container from running.



```
docker@default:~$ docker stop
"docker stop" requires at least 1 argument.
See 'docker stop --help'.

Usage:  docker stop [OPTIONS] CONTAINER [CONTAINER...]

Stop one or more running containers
docker@default:~$ docker stop postgis
postgis
docker@default:~$
```

Here I have stopped the container postgis

6. docker pull: This command is used to download image file from the docker hub repository. For example, to pull postgis image from docker hub repository, the following commands can be used.

docker pull kartoza/postgis:9.6-2.4

Creating the container

Now, for the installation of the postgis, following commands were used:

Creation of a volume to persist the data

Following command was used for creation of the volume:

docker volume create pg_data

Create a database container and start the container

Following command was used for the creation of the database container and running it simultaneously.

```
docker run --name=postgis -d -e POSTGRES_USER=navya -e POSTGRES_PASS=password -e POSTGRES_DBNAME=gis -e ALLOW_IP_RANGE=0.0.0.0/0 -p 5432:5432 -v pg_data:/var/lib/postgresql --restart=always kartoza/postgis:9.6-2.4
```

Following were the important parameters in this command, which were later used for establishing the connection with QGIS.

Username: navya

Password: password

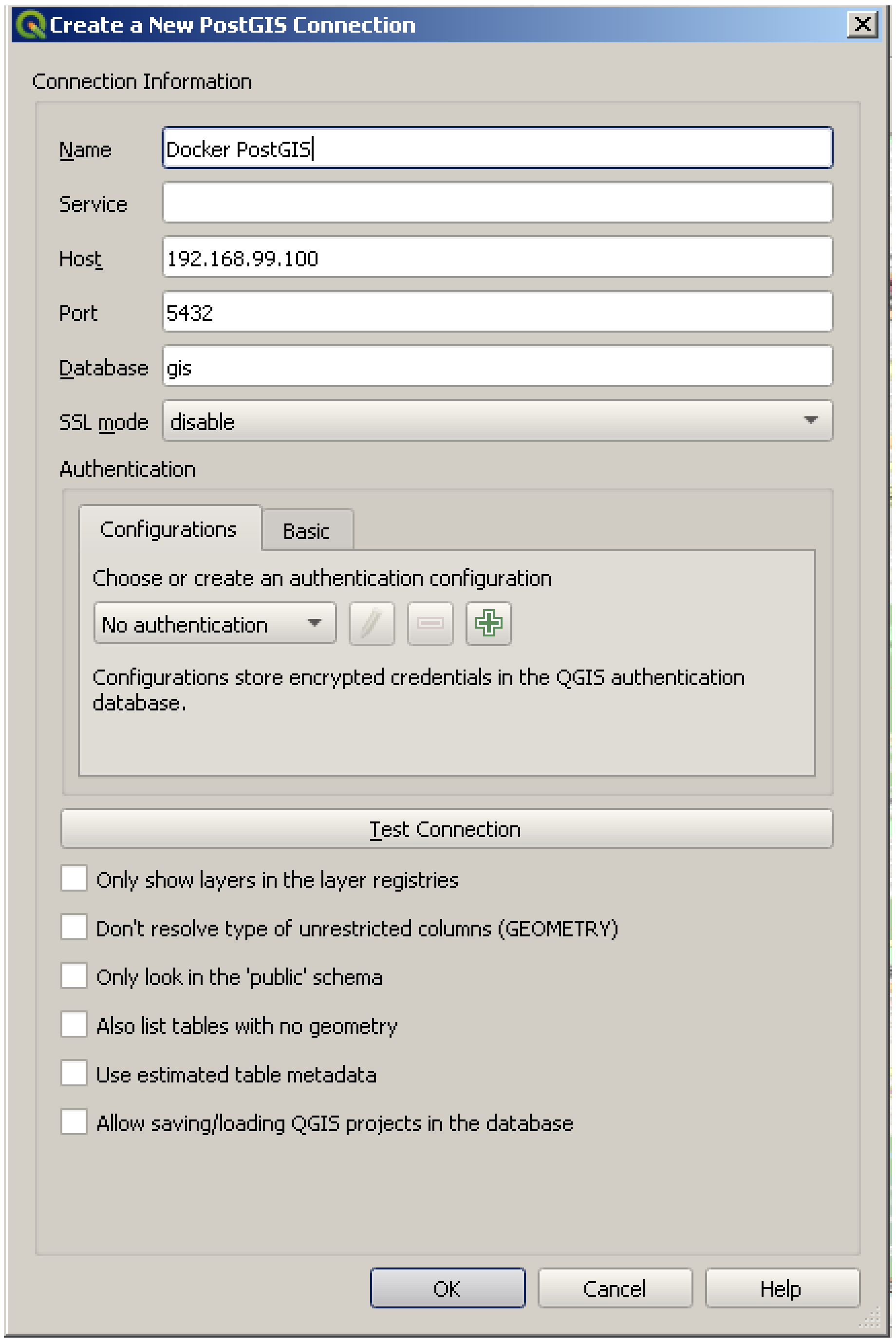
Database: gis

Port: 5432

After the container was created and then started, I used few commands like docker ps to display that the container is properly running.

Connect to the containerized database using QGIS

To connect QGIS with postgis, we start QGIS in windows through Start menu. In the Browser pane on the left hand side, we look for postgis and then right click to open New Connection Window.


Create a New PostGIS Connection

Connection Information

Name: Docker PostGIS

Service:

Host: 192.168.99.100

Port: 5432

Database: gis

SSL mode: disable

Authentication

Choose or create an authentication configuration

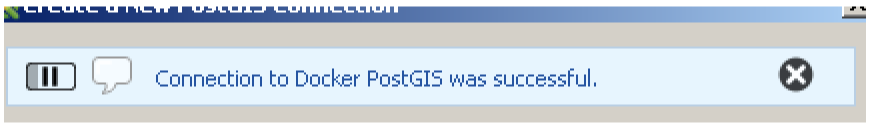
No authentication

Configurations store encrypted credentials in the QGIS authentication database.

- [ ] Only show layers in the layer registries
- [ ] Don't resolve type of unrestricted columns (GEOMETRY)
- [ ] Only look in the 'public' schema
- [ ] Also list tables with no geometry
- [ ] Use estimated table metadata
- [ ] Allow saving/loading QGIS projects in the database

The various parameters to be filled in are shown in the screenshot above. The username and password are to be given in the Authentication area of the new connection window. Once the parameters are given, we need to test the connection.

QGIS responds with the following message.

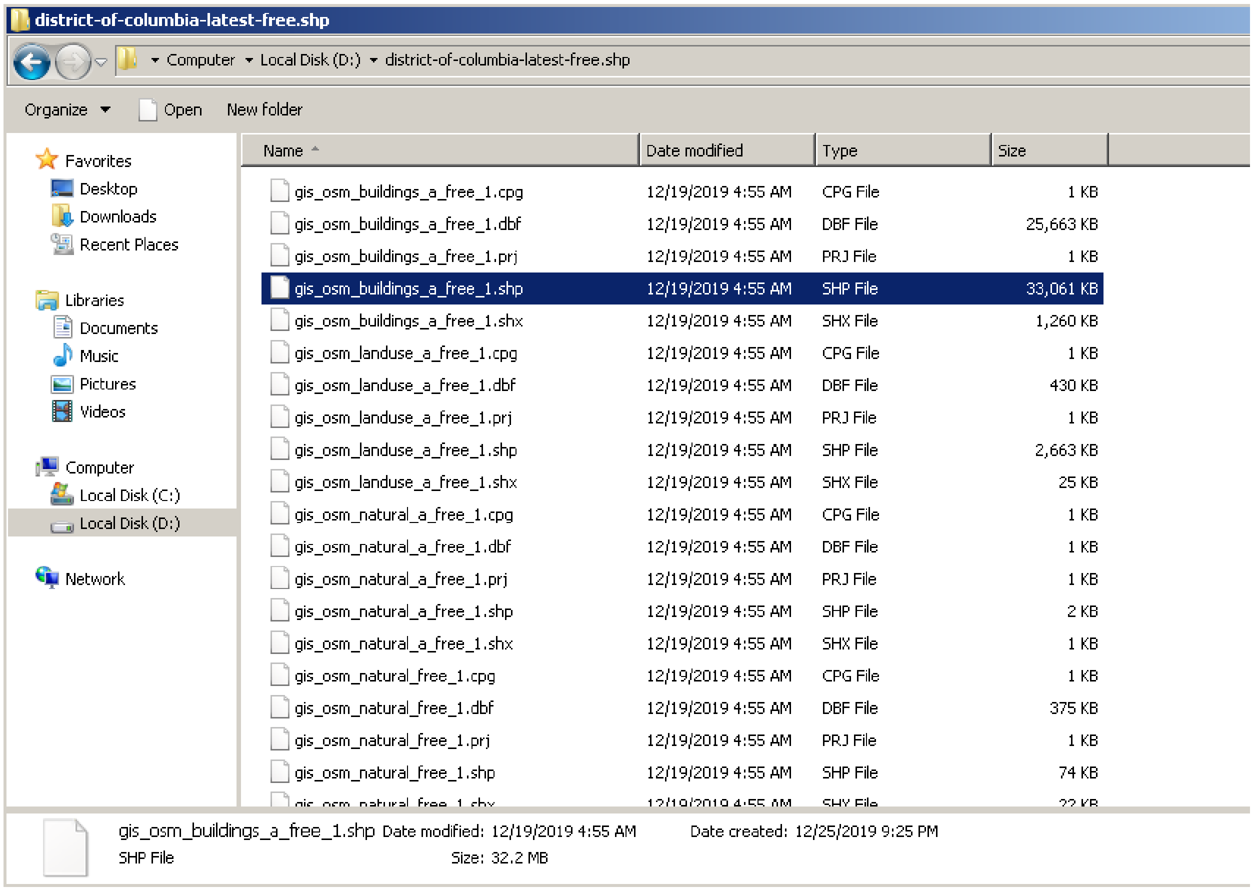

Connection to Docker PostGIS was successful.

Adding some data to the database

To add some data to the postgis database, I had used the shapefiles, which I had used for the task og GeoServer, wherein I was required to upload shapefiles to GeoServer.

I had downloaded the Colombia state data from openstreetmap website.

I chose the buildings data from the district of Columbia database, which was still lying in my hard disk.

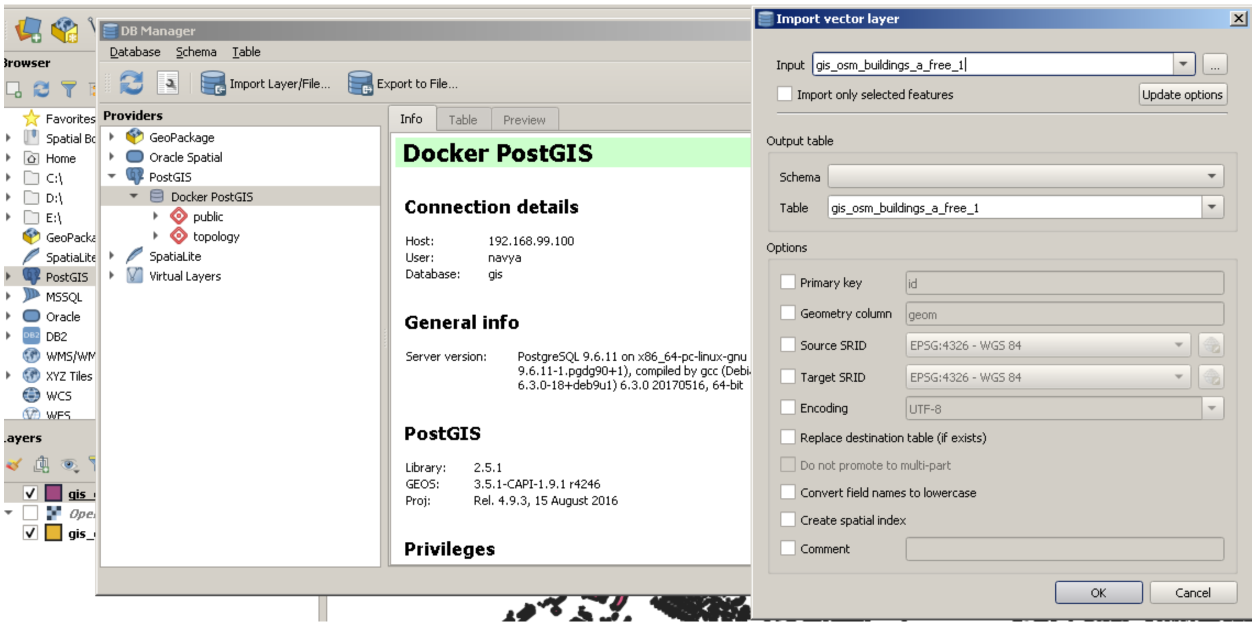


Name	Date modified	Type	Size
gis_osm_buildings_a_free_1.cpg	12/19/2019 4:55 AM	CPG File	1 KB
gis_osm_buildings_a_free_1.dbf	12/19/2019 4:55 AM	DBF File	25,663 KB
gis_osm_buildings_a_free_1.prj	12/19/2019 4:55 AM	PRJ File	1 KB
gis_osm_buildings_a_free_1.shp	12/19/2019 4:55 AM	SHP File	33,061 KB
gis_osm_buildings_a_free_1.shx	12/19/2019 4:55 AM	SHX File	1,260 KB
gis_osm_landuse_a_free_1.cpg	12/19/2019 4:55 AM	CPG File	1 KB
gis_osm_landuse_a_free_1.dbf	12/19/2019 4:55 AM	DBF File	430 KB
gis_osm_landuse_a_free_1.prj	12/19/2019 4:55 AM	PRJ File	1 KB
gis_osm_landuse_a_free_1.shp	12/19/2019 4:55 AM	SHP File	2,663 KB
gis_osm_landuse_a_free_1.shx	12/19/2019 4:55 AM	SHX File	25 KB
gis_osm_natural_a_free_1.cpg	12/19/2019 4:55 AM	CPG File	1 KB
gis_osm_natural_a_free_1.dbf	12/19/2019 4:55 AM	DBF File	1 KB
gis_osm_natural_a_free_1.prj	12/19/2019 4:55 AM	PRJ File	1 KB
gis_osm_natural_a_free_1.shp	12/19/2019 4:55 AM	SHP File	2 KB
gis_osm_natural_a_free_1.shx	12/19/2019 4:55 AM	SHX File	1 KB
gis_osm_natural_free_1.cpg	12/19/2019 4:55 AM	CPG File	1 KB
gis_osm_natural_free_1.dbf	12/19/2019 4:55 AM	DBF File	375 KB
gis_osm_natural_free_1.prj	12/19/2019 4:55 AM	PRJ File	1 KB
gis_osm_natural_free_1.shp	12/19/2019 4:55 AM	SHP File	74 KB
gis_osm_natural_free_1.shx	12/19/2019 4:55 AM	SHX File	22 KB

gis_osm_buildings_a_free_1.shp Date modified: 12/19/2019 4:55 AM Date created: 12/25/2019 9:25 PM
SHP File Size: 32.2 MB

QGIS is presently having a DB Manager plugin, which allows effortless import / export of the geospatial data from postgis database.

After browsing to the desired shape file, I just clicked the update options, which automatically updates the other empty fields on the import data window.



Docker PostGIS

Connection details

```
Host:       192.168.99.100
User:       navya
Database:   gis
```

General info

```
Server version:    PostgreSQL 9.6.11 on x86_64-pc-linux-gnu
                   9.6.11-1.pgdg90+1), compiled by gcc (Debi
                   6.3.0-18+deb9u1) 6.3.0 20170516, 64-bit
```

PostGIS

```
Library:   2.5.1
GEOS:      3.5.1-CAPI-1.9.1 r4246
Proj:      Rel. 4.9.3, 15 August 2016
```

Privileges

Import vector layer

Input: gis_osm_buildings_a_free_1

☐ Import only selected features — Update options

Output table

Schema:

Table: gis_osm_buildings_a_free_1

Options

☐ Primary key: id

☐ Geometry column: geom

☐ Source SRID: EPSG:4326 - WGS 84

☐ Target SRID: EPSG:4326 - WGS 84

☐ Encoding: UTF-8

☐ Replace destination table (if exists)

☐ Do not promote to multi-part

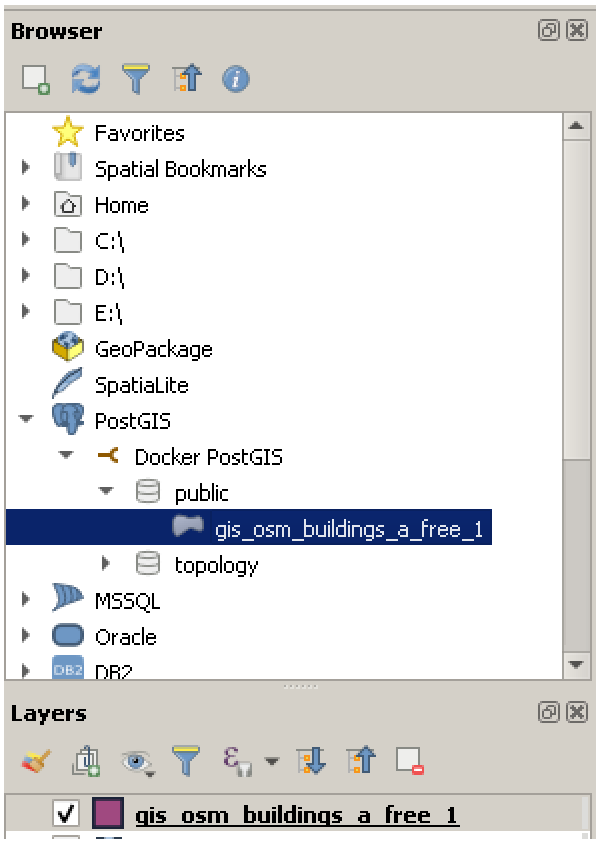
☐ Convert field names to lowercase

☐ Create spatial index

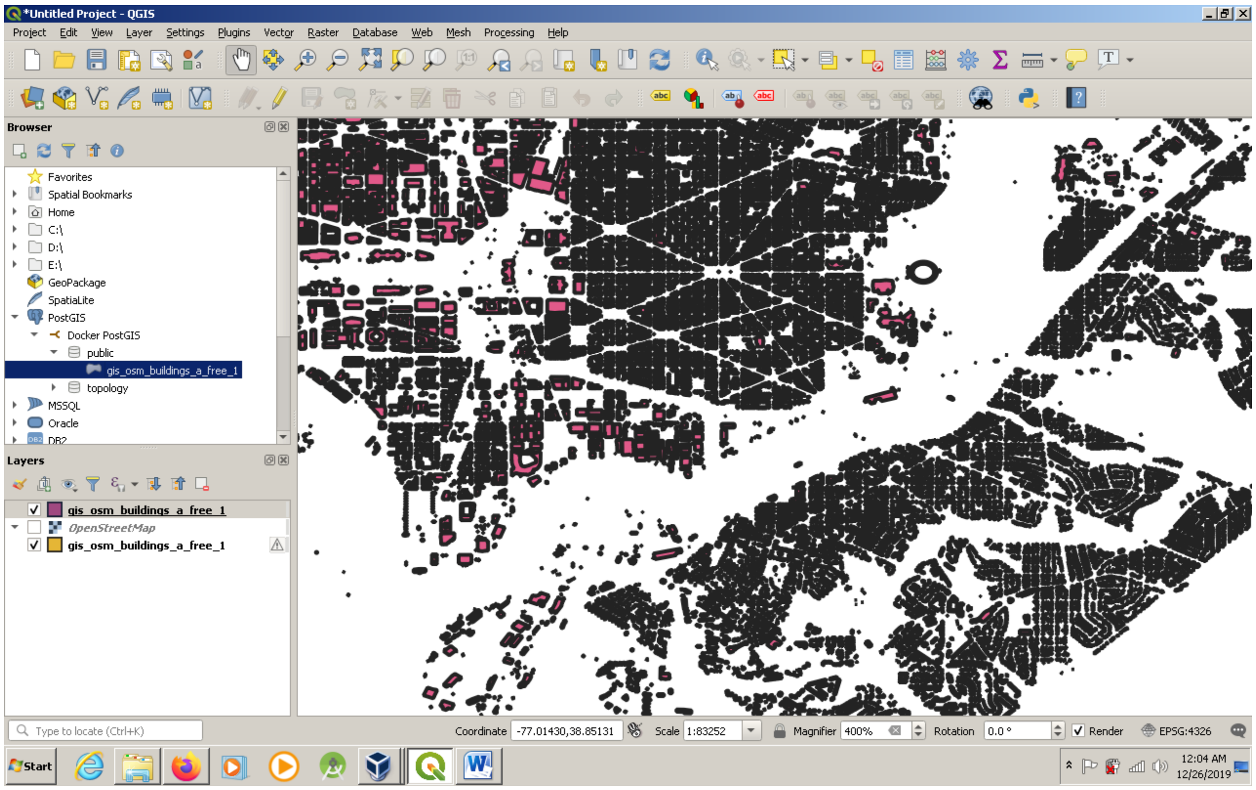
☐ Comment

Clicking on ok, imports the data into the database and you get “Import Successful” message.

After successful import, you will be able to see the data in the postgis database in the browser pane as shown in the screenshot below.



Thereafter, you just drag this layer into the right hand side pane. The geospatial data would be visible in the workspace in the QGIS.



Layers:

- ☑ gis_osm_buildings_a_free_1
- ☐ OpenStreetMap
- ☑ gis_osm_buildings_a_free_1

Coordinate -77.01430,38.85131 Scale 1:83252 Magnifier 400% Rotation 0.0 ° ☑ Render EPSG:4326

Thanks.