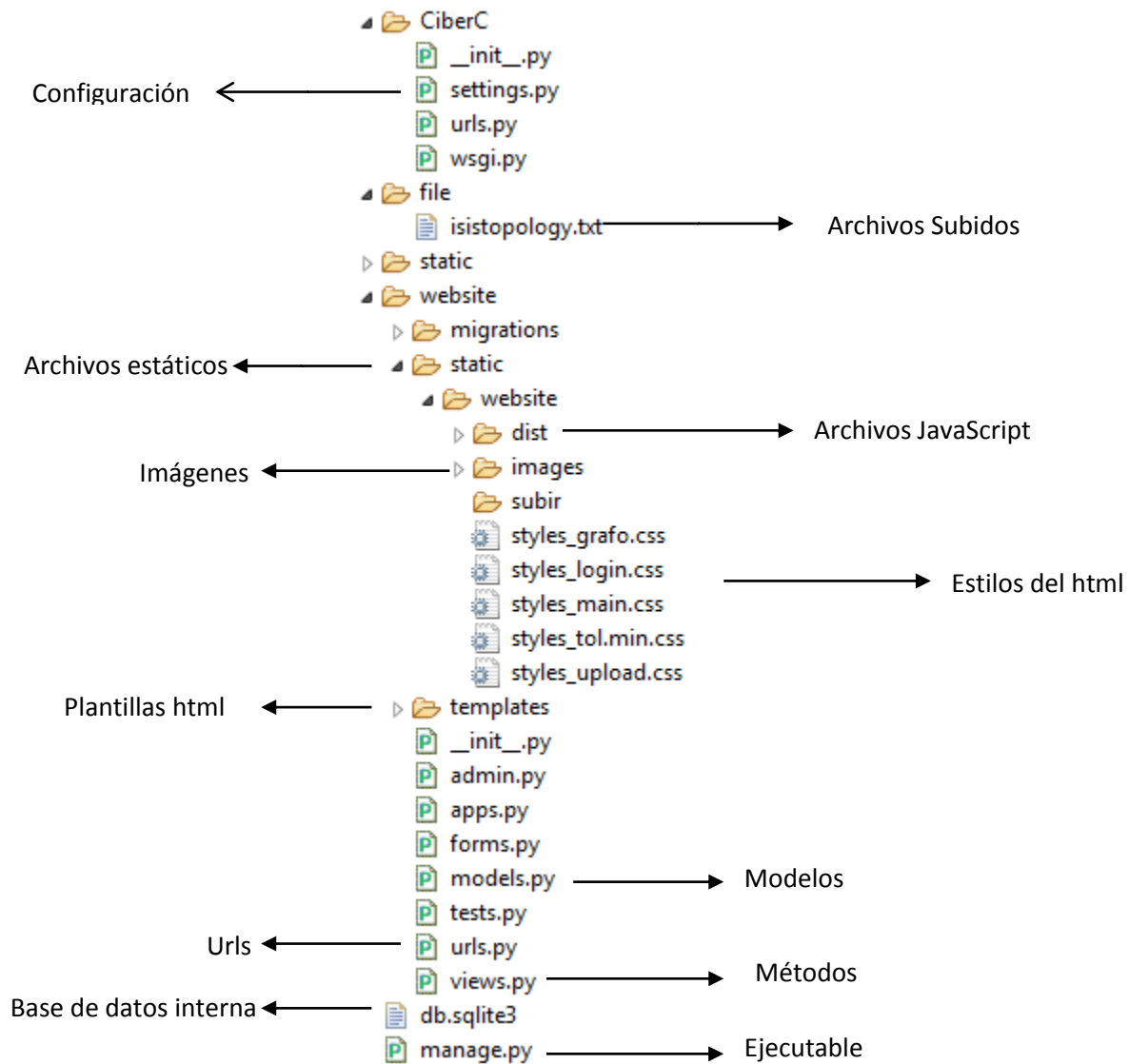


MANUAL TÉCNICO

El siguiente proyecto está estructurado de la siguiente manera:

1. Microservicio
 - 1.1. Django
 - 1.1.1. Urls
 - 1.1.2. Views
 - 1.1.3. Template
 - 1.2. Archivos Estáticos
 - 1.2.1. JavaScript
 - 1.2.2. Imágenes
 - 1.2.3. Estilos
2. Servidor
 - 2.1. Amazon Server
 - 2.1.1. Instalación
 - 2.1.2. Configuración
3. Base de Datos
 - 3.1. Estructura

1. MICROSERVICIO



1.1.DJANGO

Django es un framework Open Source para aplicaciones web gratuito, trabaja con Python. Este framework trabaja de manera ordenada como se observa en la **Figura.1** lo que permite realizar páginas web de manera rápida, segura y sostenible.

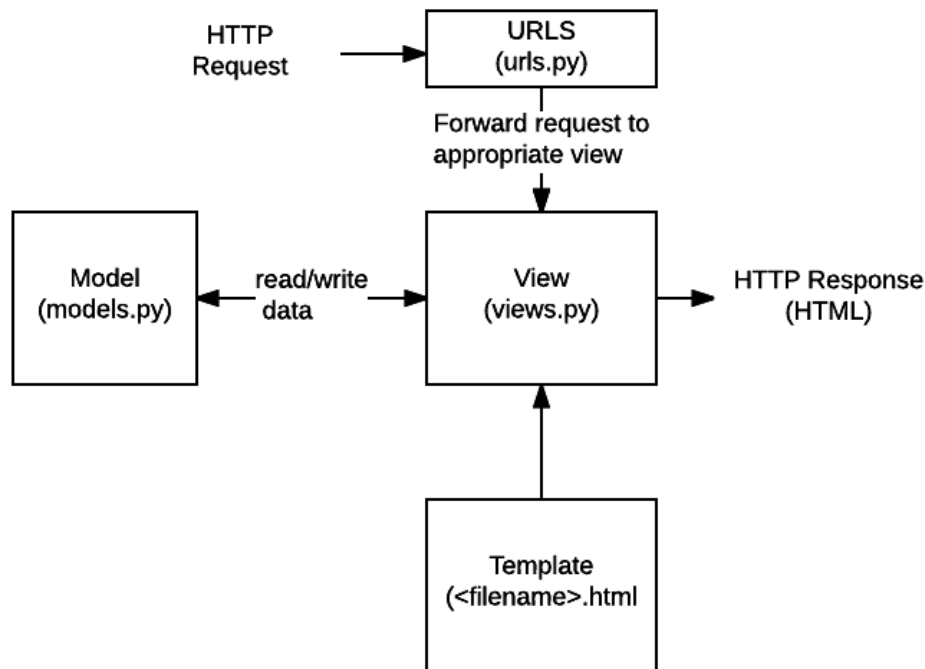


Figura 1: Estructura de Django

1.1.1. URLS

Django utiliza un mapeado de URL, agrupando todas las URL y redirigiendo todas las peticiones de la página web a una vista (view) apropiada para cada URL de manera independiente, además se puede utilizar para enviar parámetros ingresados en el portal web.

Estructura

```
url(r'^$', views.auth_login, name = 'auth_login'),
```

↓ ↓ ↓

Url Método (view) nombre

Código

```
from django.conf.urls import url
from . import views
```

```
urlpatterns = [
```

```
    # /CiberC/
    url(r'^$', views.auth_login, name = 'auth_login'),
```

—————> Login

```
    #CiberC/registration/
    url(r'^registration/$', views.registration, name = 'registration'),
```

```
    #/CiberC/register/
    url(r'^register/$', views.register, name = 'register'),
```

—————> Registrase

```
    # /CiberC/main/
    url(r'^main/$', views.main, name = 'main'),
```

—————> Página Principal

```
    #/CiberC/login_validation/
    url(r'^login_validation/$', views.login_validation, name = 'login_validation'),
```

—————> Validar Usuario

```
    #/CiberC/uploadFile/
    url(r'^uploadFile/$', views.upload, name = 'uploadFile'),
```

```
    #/CiberC/uploadingFile/
    url(r'^uploadingFile/$', views.upload_file, name = 'uploadingFile'),
```

—————> Subir Archivo

```
    #/CiberC/logout/
    url(r'^logout/$', views.auth_logout, name = 'auth_logout'),
```

—————> Salir

```
    #/CiberC/downloadFile/
    url(r'^downloadFile/$', views.download, name = 'download'),
```

—————> Generar Reporte

```
    # /CiberC/downloadingFile/
    url(r'^downloadingFile/$', views.download_file, name='downloadingFile'),
```

```
    #/CiberC/view/
    url(r'^view/$', views.view, name='view'),
```

—————> Generar Grafo

```
    #/CiberC/viewGraph/
    url(r'^viewGraph/$', views.view_graph, name='view_graph'),
```

```
    #/CiberC/modules/
    url(r'^modules/$', views.modules, name='modules'),
```

—————> Módulos

```
    #/CiberC/convertModules/
    url(r'^convertModules/$', views.convertModules, name='convertModules')
```

```
]
```

1.1.2. VIEW

Una vista es un método (función) para el control de peticiones que se realiza en la página web. Cuando se realiza una petición, la vista accede a los datos por medio de modelos y define un formato o un template como respuesta HTTP al portal web.

Estructura

Método: Realiza proceso de la View

View: Redirige la Url

Código

```
import os
import pymysql, collections
from django.http import HttpResponse
from django.template import loader
from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login, logout
from docx import Document
from docx.enum.text import WD_ALIGN_PARAGRAPH
from .models import Nodes
from .forms import UserForm
# -*- coding: utf-8 -*-

#1 view for url
#render to the url
def auth_login(request): → View Login
def registration(request): → View Registrarse
def main(request): → View Página principal
def auth_logout(request): → View Salir
def upload(request): → View Cargar Archivo
def download(request): → View Descargar Archivo
def modules(request): → View Módulos
def register(request): → Método Registrarse
def login_validation(request): → Método Validar Login
```

<code>def upload_file(request):</code>	→	Método Cargar Archivo
<code>def download_file(request):</code>	→	Método Descargar Archivo
<code>def view(request):</code>	→	View Graficar Grafo
<code>def view_graph(request):</code>	→	Método Graficar Grafo
<code>def handle_uploaded_file(file, filename):</code>	→	Método Crear Copia del archivo Subido
<code>def load_data(documento):</code>	→	Método Guardar Datos en Mysql
<code>def convert(nodo):</code>	→	Método Convierte IP del nodo a Nombre del nodo
<code>def connected_components(nodes, diccionario):</code>	→	Método Componentes conexas del Grafo
<code>def conexas(nodo):</code>	→	Método Genera El diccionario de los nodos actuales del Grafo
<code>def convertModules(request):</code>	→	Método Convertir Comandos
<code>def xconnects(commands_ios):</code>	→	Método Comandos Xconnects
<code>def vfis(commands_ios):</code>	→	Método Comandos Vfis
<code>def prefix(commands_ios):</code>	→	Método Comandos Prefix-list

Método para cargar el Archivo

Pseudocódigo

- Abro el archivo ingresado
- Recorro las líneas del archivo
 - ✓ Separo las líneas por palabras
 - ✓ Si la línea contiene la palabra "Connection"
 - ❖ Guardo todos los datos en las listas
 - ❖ Vacío las listas
 - ❖ Si el Vecino no tiene Ip o Hostame lo guardo como "0"
 - ✓ Si la línea contiene la palabra "Hostname"
 - ❖ Guardo el nombre del nodo en una lista de hostname
 - ❖ Guardo todas las listas anteriores en su respectiva lista
 - ❖ Vacío las listas
 - ❖ Si el Vecino no tiene Ip o Hostame lo guardo como "0"
 - ✓ Si la línea contiene la palabra "Ip address"
 - ❖ Guardo la ip del nodo en una lista de Ip
 - ✓ Si la línea contiene la palabra "Neighbor"
 - ❖ Guardo el nombre del nodo vecino en una lista de Vecinos
 - ✓ Si la línea contiene la palabra "Metrics"
 - ❖ Guardo la métrica del nodo en una lista de Métricas

- Cierro el archivo
- Si no existe ningún Nodo en el archivo
 - ✓ Retorno que el archivo ingresado es incorrecto
- Si existe Nodos en el archivo
 - ✓ Elimino todos los datos de la base de Datos actual
 - ✓ Guardo toda la lista de Nodos en la tabla Isis
 - ✓ Recorro toda la lista de Neighbor
 - ❖ Valido que tipo de nodo es y lo guardo en su tabla correspondiente con el siguiente formato
 - Hostname
 - Neighbor
 - Ip
 - Métrica

Código

```
def load_data(documento):
    # connect with the database
    conn = pymysql.connect(
        host="localhost", port=3306, user="root",
        passwd="hotmail003", db="red"
    )
    #open the file "isitopology"
    fileShow = open(documento, "r")
    #read the file
    lineas = fileShow.readlines()
    acum3 = 0;
    #list
    hostname = []
    ip = []
    ipGlobal = []
    metric = []
    extended = []
    vecinosGlobal = []
    metricGlobal = []
    extendedGlobal = []
    vecinos = []
    #for the file to create the lists of neighboring nodes
    for i in lineas:
        #separate each word from the line
        palabras = i.split(" ")
        #valid if the first word is "Connection"
        if palabras[0] == "Connection":
            vecinosGlobal.append(vecinos)
            metricGlobal.append(metric)
            extendedGlobal.append(extended)
            #valid if the node have ip
            if (ip):
                ipGlobal.append(ip)
            else:
                ip.append("0")
```

Conexión con la Base de datos

Leo el documento ingresado

Recorro el documento

Separo las palabras y las hago arreglo

Si la línea es igual a Connection closed guardo todas las listas actuales

```

        ipGlobal.append(ip)
    ip = []
    vecinos = []
    metric = []
    extended = []
if len(palabras) >= 3:
    #valid if the word is "Hostname:"
    if palabras[2] == "Hostname:":
        # valid if the node have neighbor
        if acum3 == 1:
            vecinos.append("0")
            acum3 = 0
            palabras1 = palabras[3].split("\n")
            hostname.append(palabras1[0])
            #valid if the list hostname have one node
            if len(hostname) == 1:
                vecinos = []
                metric = []
                extended = []
                ip = []
            else:
                vecinosGlobal.append(vecinos)
                metricGlobal.append(metric)
                extendedGlobal.append(extended)
            # valid if the node have ip
            if (ip):
                ipGlobal.append(ip)
            else:
                ip.append("0")
                ipGlobal.append(ip)
            ip = []
            vecinos = []
            metric = []
            extended = []
if len(palabras) >= 5:
    # valid if the line have the word "IP Address"
    if palabras[2] == "IP" and palabras[3] == "Address:":
        palabras1 = palabras[-1].split("\n")
        ip.append(palabras1[0])
    # valid if the line have the word "Neighbor"
    if palabras[4] == "Neighbor":
        palabras1 = palabras[-1].split("\n")
        vecinos.append(palabras1[0])
        acum3 = 0
    # valid if the node have neighbor
    if acum3 == 1:
        vecinos.append("0")
        acum3 = 0
    # valid if the line have the word "Metric"
    if palabras[2] == "Metric:":
        acum3 = 1
        metric.append(palabras[3])
        palabras1 = palabras[-1].split("\n")
        extended.append(palabras1[0])

```

Si la línea es igual a hostname inicializa las listas

EL nombre o ip del nodo vecino no está en el archivo

Si la línea es igual a Ip address guarda la ip en una lista

Si la línea contiene la palabra Neighbor agrega el nombre del nodo a una lista

Si la línea contiene la palabra métrica agrega la métrica del nodo a una lista


```

acum = 0;
#close the file
fileShow.close() → Cierro el archivo
#valid if the file is correct
if len(hostname)<1:
    return 0
#if the file is correct then we delete all the information from the database
else:
    try:
        sql = "DELETE FROM `border` "
        sql1 = "DELETE FROM `provider` "
        sql2 = "DELETE FROM `provideredge` "
        sql3 = "DELETE FROM `isis`"
        with conn.cursor() as cursor:
            cursor.execute(sql)
            conn.commit()
            cursor.execute(sql1)
            conn.commit()
            cursor.execute(sql2)
            conn.commit()
        with conn.cursor() as cursor:
            cursor.execute(sql3)
            conn.commit()

        for i in hostname:
            # insert the isis nodes in the database
            sql = "INSERT INTO `isis`(`Hostname`, `Ip`) VALUES ('" +
hostname[acum] + "', '" + ipGlobal[acum][0] + "')"
            for m in ipGlobal[acum]:
                if not m==ipGlobal[acum][0]:
                    sql1 = "INSERT INTO `Ip`(`Hostname`, `Ip`) VALUES ('" +
hostname[acum] + "', '" + m + "')"
                    with conn.cursor() as cursor:
                        cursor.execute(sql1)
                        conn.commit()

            print(acum)
            with conn.cursor() as cursor:
                cursor.execute(sql)
                conn.commit()
            acum1 = 0;
            # insert the neighbors nodes in the database
            with conn.cursor() as cursor:
                for j in extendedGlobal[acum]:
                    palabras1 = extendedGlobal[acum][acum1].split(".")
                    cadena = list(palabras1[0])

                    if not cadena[1] == "-":
                        palabras2 = palabras1[0].split("-")
                        cadena1 = list(palabras2[0])
                    else:
                        palabras2 = palabras1[0] → Tipo de Nodo lo guardo en su tabla correspondiente
                        cadena1 = list(palabras2)

                    if cadena1[-3] == "P":

```

```

        sql = "INSERT INTO `provider`(`Hostname`,
`Neighbor`, `Ip`, `Metrica`) VALUES ('" + hostname[
        acum] + "','"+ extendedGlobal[acum][acum1] +
        "','"+ vecinosGlobal[acum][acum1] + "','"+ \
        metricGlobal[acum][acum1] + "') "
        if cadena1[-3] == "E":
            sql = "INSERT INTO `provideredge`(`Hostname`,
`Neighbor`, `Ip`, `Metrica`) VALUES ('" + \
            hostname[acum] + "','"+ +
            extendedGlobal[acum][acum1] + "','"+ + vecinosGlobal[acum][
            acum1] + "','"+ + metricGlobal[acum][acum1]
+ "') "
            if cadena1[-3] == "B":
                sql = "INSERT INTO `border`(`Hostname`, `Neighbor`,
`Ip`, `Metrica`) VALUES ('" + hostname[
                acum] + "','"+ + extendedGlobal[acum][acum1] +
                "','"+ + vecinosGlobal[acum][acum1] + "','"+ + \
                metricGlobal[acum][acum1] + "') "
                if cadena1[-3] == "R":
                    sql = "INSERT INTO `border`(`Hostname`, `Neighbor`,
`Ip`, `Metrica`) VALUES ('" + hostname[
                    acum] + "','"+ + extendedGlobal[acum][acum1] +
                    "','"+ + vecinosGlobal[acum][acum1] + "','"+ + \
                    metricGlobal[acum][acum1] + "')"
                    cursor.execute(sql)
                    acum1 = acum1 + 1
                    conn.commit()
            acum = acum + 1
finally:
    # Close connection.
    conn.close()

```

Método para generar el reporte de Nodos Afectados

CiberC-Nodos

The selected node is: **MCSCNTE01**

The following groups of affected nodes were created:

Group #1:

'MCSSUCUE01'

Group #2:

'MCSSDMZE01', 'MCSCNTE02'

Group #3:

'MCSHUA01'

Group #4:

'MCSRBLE01'

Código

```
def download_file(request):
    if (request.user.is_authenticated()): → Valido que realizo el Inicio de Sesión
        if (request.method == "POST"):
            nodo = request.POST['node'] → Obtengo el nodo Ingresado
            #get the group of affected nodes
            texto=conexas(nodo) → Obtengo una Lista de las componentes Conexas del grafo
                                   eliminando el nodo ingresado
            #create the docx
            document = Document() → Creo el documento
            number=0
            #Title of the document
            d=document.add_heading('CiberC-Nodos', 0)
            #center the title
            d.alignment = WD_ALIGN_PARAGRAPH.CENTER
            #Create one paragraph
            p=document.add_paragraph('The selected node is: ')
            #change the name of the node to bold and center the text
            p.add_run(nodo).bold = True
            p.alignment = WD_ALIGN_PARAGRAPH.CENTER
            #condition to validate if there are groups of affected nodes
            if len(texto)>1: → Valido que exista varios grupos afectados

                p = document.add_paragraph('The following groups of affected nodes were created: ')
                p.alignment = WD_ALIGN_PARAGRAPH.JUSTIFY
            else:
                p = document.add_paragraph('No affected nodes')
                p.alignment = WD_ALIGN_PARAGRAPH.JUSTIFY
            #for the groups of affected nodes and saved them in the document
            for components in texto: → Recorro la lista de Componentes Conexas
                if number>0:
                    s=document.add_heading("Group #i:" % (number), level=1)
                    s.alignment = WD_ALIGN_PARAGRAPH.CENTER
                    p = document.add_paragraph("%s" %
                        (list(components).__str__().replace('[','').replace(']', '')))
                    p.alignment = WD_ALIGN_PARAGRAPH.JUSTIFY
                    number += 1
                response = HttpResponse(content_type='application/vnd.openxmlformats-
officedocument.wordprocessingml.document') → Nombre y tipo del archivo
                response['Content-Disposition'] = 'attachment; filename=download.docx'
                document.save(response)
                return response → Retorno el archivo creado

    return render(request, 'website/download.html', {'success_message': 'Please insert the name
or ip addres of the node:'})
else:
    return render(request, 'website/login.html')
```

Método para obtener las componentes conexas del grafo

PseudoCódigo

- Creo una copia de la lista de los nodos del grafo
- Recorro la lista de nodos
 - ✓ Obtengo un nodo aleatorio y lo elimino de la copia de la lista
 - ✓ Creo un grupo conjunto que contendrá el siguiente grupo de nodos conectados entre sí.
 - ✓ Construyó una cola con el nodo obtenido
 - ✓ Recorro la cola
 - ❖ Obtengo el primer nodo de la cola
 - ❖ Obtengo los nodos vecinos del nodo obtenido
 - ❖ Eliminamos los nodos vecinos que ya visitamos del grupo
 - ❖ Eliminamos los nodos restantes de la copia de la lista
 - ❖ Agregamos de nuevo los nodos vecinos al grupo para próximas iteraciones
- Retorno los grupos creados

Código

```
def connected_components(nodos, diccionario):  
  
    result = []  
  
    # Make a copy of the set, so we can modify it.  
    nodes = set(nodos)  —————> Copia de toda la lista de nodos  
  
    # Iterate while we still have nodes to process.  
    while nodes:  
  
        # Get a random node and remove it from the global set.  
        n = nodes.pop()  —————> Obtenemos Nodo aleatorio  
  
        # This set will contain the next group of nodes connected to each other.  
        group = {n}  —————> Creamos un grupo con todos los nodos sin el  
  
        # Build a queue with this node in it.  
        queue = [n]  —————> Creamos una cola con cada nodo recorrido  
  
        # Iterate the queue.  
        # When it's empty, we finished visiting a group of connected nodes.  
        while queue:  
            # Consume the next item from the queue.  
            n = queue.pop(0)  
            # Fetch the neighbors.  
  
            neighbors = diccionario.get(n)  —————> Obtengo los nodos vecinos del  
Nodo obtenido aleatoriamente  
  
            if neighbors==None:  
                break  
            if len(neighbors)>0:
```

```

# Remove the neighbors we already visited.
neighbors.difference_update(group)  → Remuevo todos los nodos vecinos de la lista grupo

# Remove the remaining nodes from the global set.
nodes.difference_update(neighbors) → Remuevo todos los nodos que no sean nodos vecinos del nodo obtenido

# Add them to the group of connected nodes.
group.update(neighbors)  → Actualizo la lista grupo

# Add them to the queue, so we visit them in the next
iterations.
queue.extend(neighbors)

# Add the group to the list of groups.
result.append(group)  → Retorno los grupos creados
# Return the list of groups.

return result

```

Método para graficar el grafo de la red

PseudoCódigo

- Valido la autenticación
- Obtengo las opciones ingresadas en la página Network Grapher
- Valido si el nodo ingresado es ip
 - ✓ Si es Ip utilizo la función Convert que convierte la Ip en el Nombre del nodo
- Obtengo la lista de los nodos de la base de datos
- Uno las tablas de nodos vecinos y hago una sola lista
- Recorro la lista de nodos vecinos
 - ✓ Si la opción ingresada es toda la red
 - ❖ Guardo una lista1 de nodos vecinos del nodo recorrido
 - ❖ Valido que el nodo actual no esté en la lista de nodos visitados
 - Si no está lo agrego
 - ❖ Si la opción ingresada es 1 enlace
 - Creo un texto de enlaces que no se repitan
 - ❖ Si la opción ingresada es todos los enlaces
 - Creo un texto de enlaces
 - ✓ Si la opción ingresada no es toda la red
 - ❖ Si el nodo ingresado es igual al nodo actual recorrido
 - Guardo una lista1 de nodos vecinos del nodo ingresado
 - Valido que el nodo actual no esté en la lista de nodos visitados
 - Si no está lo agrego
 - Si la opción ingresada es 1 enlace
 - Creo un texto de enlaces que no se repitan
 - Si la opción ingresada es todos los enlaces
 - Creo un texto de enlaces
 - Recorro la lista de nodos vecinos del nodo ingresado

- Creo un texto de enlaces
- Vuelvo a recorrer la lista de nodos vecinos
 - ✓ Si el nodo actual está en lista1
 - ❖ Si la opción ingresada es 3 hops
 - Guardo una lista2 de nodos vecinos del nodo recorrido
 - Valido que el nodo actual no esté en la lista de nodos visitados
 - SI no está lo agrego
 - ❖ Si la opción ingresada es toda la red
 - Valido que el nodo actual no esté en la lista de nodos visitados
 - SI no está lo agrego
 - Si la opción ingresada es 1 enlace
 - Creo un texto de enlaces que no se repitan
 - Si la opción ingresada es todos los enlaces
 - Creo un texto de enlaces
 - ❖ Si la opción ingresada 2 o 3 hops
 - Valido que el nodo actual no esté en la lista de nodos visitados
 - SI no está lo agrego
 - Si la opción ingresada es 1 enlace
 - Creo un texto de enlaces que no se repitan
 - Si la opción ingresada es todos los enlaces
 - Creo un texto de enlaces
- Vuelvo a recorrer la lista de nodos vecinos
 - ✓ Si el nodo actual está en lista2
 - ❖ Valido que el nodo actual no esté en la lista de nodos visitados
 - SI no está lo agrego
 - ❖ Si la opción ingresada es 1 enlace
 - Creo un texto de enlaces que no se repitan
 - ❖ Si la opción ingresada es todos los enlaces
 - Creo un texto de enlaces
- Recorro la lista de Nodos Visitados
 - ✓ Valido que tipo de nodo es
 - ✓ Asigno estructura de nodos en el grafo
 - ✓ Si la opción ingresada es Grafo dinámico
 - ❖ Retorno la url Grafo.html
 - ✓ Si la opción ingresada es Grafo estático
 - ❖ Retorno la url GrafoR.html

Código

```
# create the nodes and edges of the graph
def view_graph(request):
    if (request.user.is_authenticated()): → Valido que se realizó el Inicio de Sesión
        if (request.method == "POST"):
            # obtain the chosen options to graph the graph
            nodo = request.POST['node'] → Nodo Ingresado
            level = request.POST['level'] → Level del grafo (1,2,3 Hops- toda la red)
            red = request.POST['red'] → Tipo de Red (Estática o dinámica)
            link = request.POST['Link'] → Con o sin métrica
            #connect with the database
            conn = pymysql.connect(
                host="localhost", port=3306, user="root",
                passwd="hotmail003", db="red" → Conexión con la base de Datos
            )
            texto = ""
            textono = ""
            #Get the hostname of the node entered
            nodo=convert(nodo) → Obtengo el nombre del nodo ingresado
            try:
                list = []
                listIp = []
                #keep all the isis nodes in a list
                with conn.cursor() as cursor:
                    sql0 = "SELECT * FROM `isis`"
                    cursor.execute(sql0)
                    for row in cursor:
                        nodo1 = row[0].rstrip("\r")
                        ip = row[1].rstrip("\r") → Obtengo una lista de todos los
                        listIp.append(ip) → nodos de la red
                        list.append(nodo1)
                # keep all the data of the neighbors nodes in a list
                with conn.cursor() as cursor:
                    sql1 = "SELECT * FROM `isis` INNER JOIN provider
USING(Hostname)"
                    sql2 = "SELECT * FROM `isis` INNER JOIN provideredge
USING(Hostname)"
                    sql3 = "SELECT * FROM `isis` INNER JOIN border
USING(Hostname)"
                    sql = sql1 + " UNION " + sql2 + " UNION " + sql3 + " ORDER
BY Hostname ASC"
                    cursor.execute(sql)
                    result=cursor.fetchall() → Uno todas las tablas de Vecinos de
                    la base de datos
            finally:
                # Close connection.
                conn.close()

            listaT = []
            listaT.append(nodo)
            lista = []
            lista1 = []
            lista2 = []
            edge= []
            #for the list of the neighbors nodes
```

```

# row[0]= Hostname of the root node
# row[1]= ip of the root node
# row[2]= Hostname of the neighbors nodo
# row[3]= ip of the neighbors nodo
# row[4]= metric
for row in result:
    #eliminate the line break of each row
    nodo1=row[0].rstrip("\r")
    lista.append(row)
    #the name of the neighbors nodes
    palabras = row[2].split(".")
    #all levels
    if level == "4":
        #problem with one node and validate the name
        palabras1 = palabras[0].split("-")
        if (len(palabras1) >= 2):
            if (palabras1[1] == "COR"):
                palabras[0] = palabras[0] + "E"
        #valid that the neighbor node is not in the list of nodes
        if not palabras[0] in listaT:
            lista1.append(palabras[0])
            listaT.append(palabras[0])
            #1 link
            if link == "1":
                #structure to create the edge
                edgeT = "{\\"from\\": " + str(list.index(nodo1)) + ",
                \\"to\\": " + str(list.index(palabras[0])) + ", \\"arrows\\": \\"to\\",\n"
                edgeT1 = "{\\"from\\": " + str(list.index(palabras[0])) +
                ", \\"to\\": " + str(list.index(nodo1)) + ", \\"arrows\\": \\"to\\",\n"
                #valid if the link between the two nodes was already
                if not edgeT in edge:
                    if not edgeT1 in edge:
                        edge.append(edgeT)
                        textono = textono + edgeT
                    else:
                        textono = textono + "{\\"from\\": " +
                        str(list.index(nodo1)) + ", \\"to\\": " + str(
                        list.index(palabras[0])) + ", \\"Label\\": \"\" +
                        str(row[4]) + "\", \\"arrows\\": \\"to\\",\n"
                    else:
                        #valid if the node entered is equal to current node
                        if nodo1==nodo:
                            # problem with one node and validate the name
                            palabras1 = palabras[0].split("-")
                            if (len(palabras1) >= 2):
                                if (palabras1[1] == "COR"):
                                    palabras[0] = palabras[0] + "E"
                            #valid that the neighbor node is not in the list of
                            if not palabras[0] in listaT:

```

Recorro toda la tabla de Nodos Vecinos

traveled

Lista de nodos Visitados

Valido que los enlaces no se repitan

Valido que el nodo actual es igual al nodo ingresado

nodes traveled


```

        lista1.append(palabras[0])
        listaT.append(palabras[0]) → Lista de nodos Visitados
    if link == "1":
        # structure to create the edge
        edgeT = "{\\"from\\": " + str(list.index(nodo1)) + ",
        list.index(palabras[0])) + ", \\"arrows\\": \\"to\\"
    },\n"

    edgeT1 = "{\\"from\\": " +
    str(list.index(palabras[0])) + ", \\"to\\": " + str(
    list.index(nodo1)) + ", \\"arrows\\": \\"to\\" },\n"
    # valid if the link between the two nodes was
    already created

    if not edgeT in edge:
        if not edgeT1 in edge: → Valido que los enlaces no
            edge.append(edgeT)          se repitan
            textono = textono + edgeT
        else:
            textono = textono + "{\\"from\\": " +
            str(list.index(nodo1)) + ", \\"to\\": " + str(
            list.index(palabras[0])) + ", \\"Label\\":
            \\""+str(row[4])+"\\", \\"arrows\\": \\"to\\" },\n"
            # 'for' the list of the neighbors nodes
    for k in lista: → Recorro nuevamente la
        # eliminate the line break of each row          lista de nodos
        nodo1 = k[0].rstrip("\r")
        # valid if the current node to be on the visited list
        if nodo1 in lista1: → Valido que el nodo actual este en
            # the name of the neighbors nodes          lista 1
            palabras = k[2].split(".")
            palabras1 = palabras[0].split("-")
            # problem with one node and validate the name
            if (len(palabras1) >= 2):
                if (palabras1[1] == "COR"):
                    palabras[0] = palabras[0] + "E"
            #4 levels
            if level == "3":
                # valid that the neighbor node is not in the list of
                nodes traveled

                if not palabras[0] in listaT: → Lista de nodos Visitados
                    lista2.append(palabras[0])
                #all levels
                if level=="4":
                    # valid that the neighbor node is not in the list of
                    nodes traveled

                    if not palabras[0] in listaT: → Lista de nodos Visitados
                        listaT.append(palabras[0])
                    #1 link
                    if link == "1":
                        # structure to create the edge
                        edgeT = "{\\"from\\": " + str(list.index(nodo1)) + ",
                        list.index(palabras[0])) + ", \\"arrows\\": \\"to\\"
                        },\n"

```

```

edgeT1 = "{\\"from\\": " +
str(list.index(palabras[0])) + ", \\"to\\": " + str(
list.index(nodo1)) + ", \\"arrows\\": \\"to\\" },\n"
# valid if the link between the two nodes was

```

already created

```

if not edgeT in edge:
    if not edgeT1 in edge:
        edge.append(edgeT)
        textono = textono + edgeT

```

Valido que los enlaces no se repitan

```

else:
    textono = textono + "{\\"from\\": " +
str(list.index(nodo1)) + ", \\"to\\": " + str(
list.index(palabras[0])) + ", \\"label\\": \\"" +
str(k[4]) + "\\", \\"arrows\\": \\"to\\" },\n"

```

#3 and 4 levels

```

if level == "3" or level == "2" :
    # valid that the neighbor node is not in the list of

```

nodes traveled

```

if not palabras[0] in listaT:
    listaT.append(palabras[0])
# 1 link
if link == "1":

```

Lista de nodos Visitados

```

    # structure to create the edge
    edgeT = "{\\"from\\": " + str(list.index(nodo1)) + ",
\\\"to\\": " + str(
list.index(palabras[0])) + ", \\"arrows\\": \\"to\\"
},\n"

```

```

    edgeT1 = "{\\"from\\": " +
str(list.index(palabras[0])) + ", \\"to\\": " + str(
list.index(nodo1)) + ", \\"arrows\\": \\"to\\" },\n"
# valid if the link between the two nodes was

```

already created

```

if not edgeT in edge:
    if not edgeT1 in edge:
        edge.append(edgeT)
        textono = textono + edgeT

```

Valido que los enlaces no se repitan

```

else:
    textono = textono + "{\\"from\\": " +
str(list.index(nodo1)) + ", \\"to\\": " + str(
list.index(palabras[0])) + ", \\"label\\": \\"" +
str(k[4]) + "\\", \\"arrows\\": \\"to\\" },\n"

```

4 levels

```

if level == "3":
    # 'for' the list of the neighbors nodes
    for j in lista:

```

Recorro nuevamente la lista de nodos

eliminate the line break of each row

nodo1 = j[0].rstrip("\\r")

valid if the current node to be on the visited list

if nodo1 in lista2:

palabras = j[2].split(".")

palabras1 = palabras[0].split("-")

problem with one node and validate the name

if (len(palabras1) >= 2):

if (palabras1[1] == "COR"):

palabras[0] = palabras[0] + "E"

```

        if level == "3":
            if not palabras[0] in listaT:
                listaT.append(palabras[0])
                Lista de nodos Visitados

            if link == "1":
                # structure to create the edge
                edgeT = "{\\"from\\": " + str(list.index(nodo1)) + ",
                list.index(palabras[0])) + ", \\"arrows\\": \\"to\\"
                \\"to\\": " + str(
                },\n"
                list.index(palabras[0])) + ", \\"arrows\\": \\"to\\" },\n"
                edgeT1 = "{\\"from\\": " +
                str(list.index(palabras[0])) + ", \\"to\\": " + str(
                list.index(nodo1)) + ", \\"arrows\\": \\"to\\" },\n"
                # valid if the link between the two nodes was
                already created

            if not edgeT in edge:
                if not edgeT1 in edge:
                    edge.append(edgeT)
                    textono = textono + edgeT
                    Valido que los enlaces no se repitan
                else:
                    textono = textono + "{\\"from\\": " +
                    str(list.index(nodo1)) + ", \\"to\\": " + str(
                    list.index(palabras[0])) + ", \\"label\\": \\"" +
                    str(j[4]) + "\", \\"arrows\\": \\"to\\" },\n"

        # 'for'the list of graph nodes
        for t in listaT:
            Lista de Nodos Visitados
            #Provider node
            if t[-3] == "P":
                Tipo = "P"
            #ProviderEdge node
            if t[-3] == "E":
                Tipo = "E"
            Tipo de Nodo
            #backup node
            if t[-3] == "R":
                Tipo = "R"
            #border node
            if t[-3] == "B":
                Tipo = "B"
            #select node
            if t == nodo:
                Tipo = "C"
            #structure to create the nodes
            if t in list:
                texto = texto + "{id:" + str(list.index(t)) + ", \\"label\\":
                \\"" + t + "\", \\"title\\": \\"" + str(listIp[list.index(t)]) + "\", \\"group\\": \\""
                + Tipo + "\",\n"
                Base de datos de los Nodos del Grafo

            if red=="1":
                return render(request, 'website/GrafoR.html', {'nodos': texto,
                'edge': textono})
                Grafo Estático
            else :
                return render(request, 'website/Grafo.html', {'nodos':
                texto, 'edge': textono})
                Grafo Dinámico

```

Método para convertir la ip ingresada a nombre del nodo

PseudoCódigo

- Conexión con la base de datos
- Valido si el Nodo es Ip
 - ✓ Recorro la tabla Isis
 - ❖ Si la ip del nodo ingresado es igual a la ip del nodo actual
 - Retorno el nombre del nodo
 - ✓ Recorro la tabla Ip
 - ❖ Si la ip del nodo ingresado es igual a la ip del nodo actual
 - Retorno el nombre del nodo
 - ✓ Uno las tablas de Neighbors y la guardo en una lista
 - ❖ Recorro la lista de Neighbors
 - Si la ip del nodo ingresado es igual a la ip del nodo actual
 - Retorno el nombre del nodo

Código

```
def convert(nodo):
    #connect with the database
    conn = pymysql.connect(
        host="localhost", port=3306, user="root",
        passwd="hotmail003", db="red"
    )
    # separate each word from the line if the line have dots
    nod = nodo.split(".")
    bandera=0
    #valid if the node entered is an ip
    if (nod >= 2):
        bandera=1
    try:
        #if the node entered is in the isis table return the node
        with conn.cursor() as cursor:
            sql0 = "SELECT * FROM `isis`"
            cursor.execute(sql0)
            for row in cursor:
                nodo1 = row[0].rstrip("\r")
                nodo2 = row[1].rstrip("\r")
                if nodo2 == nodo:
                    bandera = 0
                    nodo = nodo1
            #if the entered node is an ip and it is not in isis table, look in
            the ip table
            if bandera==1:
                with conn.cursor() as cursor:
                    sql0 = "SELECT * FROM `Ip`"
                    cursor.execute(sql0)
                    for row in cursor:
                        nodo2 = row[1].rstrip("\r")
```

Conexión con la base de datos

Recorro la tabla isis

Recorro la tabla ip

```

        if nodo2 == nodo:
            nodo1 = row[0].rstrip("\r")
            bandera = 0
            nodo = nodo1

    # if the entered node is an ip and it is not in ip table, look in
    the neighbors table
    if bandera==1:
        with conn.cursor() as cursor:
            sql1 = "SELECT * FROM `isis` INNER JOIN provider
USING(Hostname)"
            sql2 = "SELECT * FROM `isis` INNER JOIN provideredge
USING(Hostname)"
            sql3 = "SELECT * FROM `isis` INNER JOIN border
USING(Hostname)"
            sql = sql1 + " UNION " + sql2 + " UNION " + sql3 + " ORDER
BY Hostname ASC"
            cursor.execute(sql)
            for row in cursor:
                nodo2 = row[3].rstrip("\r")
                if nodo2 == nodo:
                    nodo1 = row[2].rstrip("\r")
                    nod2 = nodo1.split(".")
                    nodo = nod2[0]
                    Recorro la lista Neighbors

            finally:
                # Close connection.
                conn.close()
    return nodo

```

1.1.3. PLANTILLAS

Un template es un fichero de texto, el cual se utiliza para el diseño del portal web. Define la estructura, el formato y el diseño de la página html.

website	
base.html	→ Estructura Estática
download.html	→ Pagina para Generar Reporte
Grafo.html	→ Grafo Dinámico
GrafoR.html	→ Grafo Estático
login.html	→ Página de Ingreso
main.html	→ Página Principal
modules.html	→ Página de los Módulos de Conversión
registration.html	→ Página para Registrarse
upload.html	→ Página para Cargar el Archivo
view.html	→ Página para Generar el Grafo

Grafo.html

Formato para nodos

```
{id:482, "label": "UIODTIE02", "title": "10.8.0.97", "group": "E"},
```

↓ ↓ ↓ ↓

Id del nodo nombre del nodo ip del nodo grupo del nodo

Formato para Edge

```
{"from": 91, "to": 499, "arrows": "to" },
```

↓ ↓

Id del nodo Id del nodo
pivote vecino

Código JavaScript

```
var nodes = null;  
var edges = null;  
var network = null;  

```

```
// create a network
```

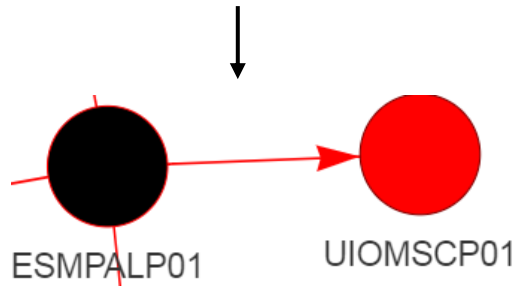
```
var container = document.getElementById('mynetwork');
```

Creación de la red

```
// create options
```

```
var options = {  
  edges: {arrows:{to:{enabled:true}}},
```

Habilitamos las flechas de
dirección de los nodos vecinos



```
// Color of the nodes
```

```
groups:{
```

```
  "P":{
```

```
    color:{background:'red',border:'maroon'},
```

```
  },
```

```
  "E":{color:{background:'coral',border:'black'},
```

```
  },
```

```
  "R":{color:{background:'green',border:'darkGreen'},
```

```
  },
```






```
  "B":{color:{background:'gold',border:'brown'},
```

```
  },
```

```
  "C":{color:{background:'black',border:'red'},
```

```
  }
```

Asignamos Color a los nodos
Dependiendo el tipo de nodo

Color	Node
	Chosen Node
	Provider Node
	Provider Edge Node
	Border Node
	Backup Node

```
interaction: {
  tooltipDelay: 200,
  hideEdgesOnDrag: true,
  navigationButtons: true,
  keyboard: true
},
```

Habilitamos los botones de Navegación



```
manipulation: {
```

Habilitamos La manipulación de los nodos

```
  addNode: function (data, callback) {
    // filling in the popup DOM elements
    document.getElementById('node-operation').innerHTML = "Add Node";
    editNode(data, clearNodePopUp, callback);
  },
  editNode: function (data, callback) {
    // filling in the popup DOM elements
    document.getElementById('node-operation').innerHTML = "Edit Node";
    editNode(data, cancelNodeEdit, callback);
  },
  addEdge: function (data, callback) {
    if (data.from == data.to) {
      var r = confirm("Do you want to connect the node to itself?");
      if (r != true) {
        callback(null);
        return;
      }
    }
    document.getElementById('edge-operation').innerHTML = "Add Edge";
    editEdgeWithoutDrag(data, callback);
  },
  editEdge: {
    editWithoutDrag: function (data, callback) {
      document.getElementById('edge-operation').innerHTML = "Edit Edge";
      editEdgeWithoutDrag(data, callback);
    }
  }
},
```




```

physics: {
  stabilization: {
    enabled: true,
    iterations: 100,
    updateInterval: 25
  }
},
// create the options to edit the graph
configure: {
  filter: function (option, path) {
    if (path.indexOf('physics') !== -1 || option === 'width' ||
option === 'shape') {
      return true;
    }
  }
}

```

Habilitamos La manipulación del grafo

physics

enabled: ☒

barnesHut:

gravitationalConstant: -2000

centralGravity: 0.3

springLength: 95

springConstant: 0.04

damping: 0.09

avoidOverlap: 0

maxVelocity: 50

minVelocity: 0.75

solver:

timestep: 0.5

```

if (path.indexOf('font') !== -1 || option === 'font') {
  return true;
}

```

nodes

font:

color:

size: 14

face:

shape:

edges

font:

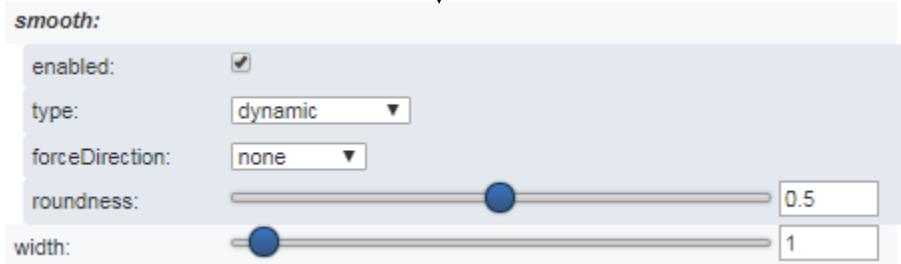
color:

size: 14

face:

align:

```
if (path.indexOf('smooth') !== -1 || option === 'smooth') {
  return true;
}
```



```
return false;
},
container: document.getElementById('config')
}
```

```
};
network = new vis.Network(container, data, options);
```

```
// add event listeners
network.on("click", function (params) {
```

→ Muestra la Ip del nodo

```
// Check if you clicked on a node; if so, display the title (if any) in a
popup
network.interactionHandler._checkShowPopup(params.pointer.DOM);
});
```



```
network.on('select', function(params) {
  document.getElementById('selection').innerHTML = 'Selection: ' +
params.nodes;
});
```

```
network.on("stabilizationProgress", function(params) {
  var maxWidth = 496;
  var minWidth = 20;
  var widthFactor = params.iterations/params.total;
  var width = Math.max(minWidth,maxWidth * widthFactor);
```

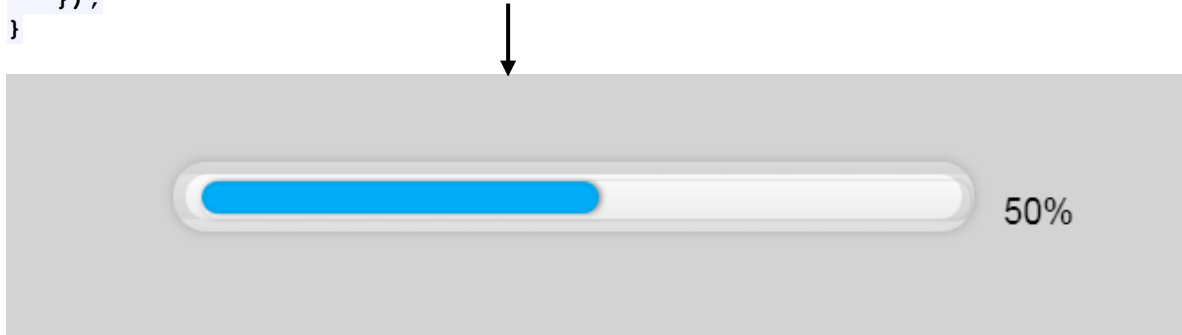
→ Estabilización del Grafo

```
document.getElementById('bar').style.width = width + 'px';
document.getElementById('text').innerHTML =
Math.round(widthFactor*100) + '%';
});
network.once("stabilizationIterationsDone", function() {
```

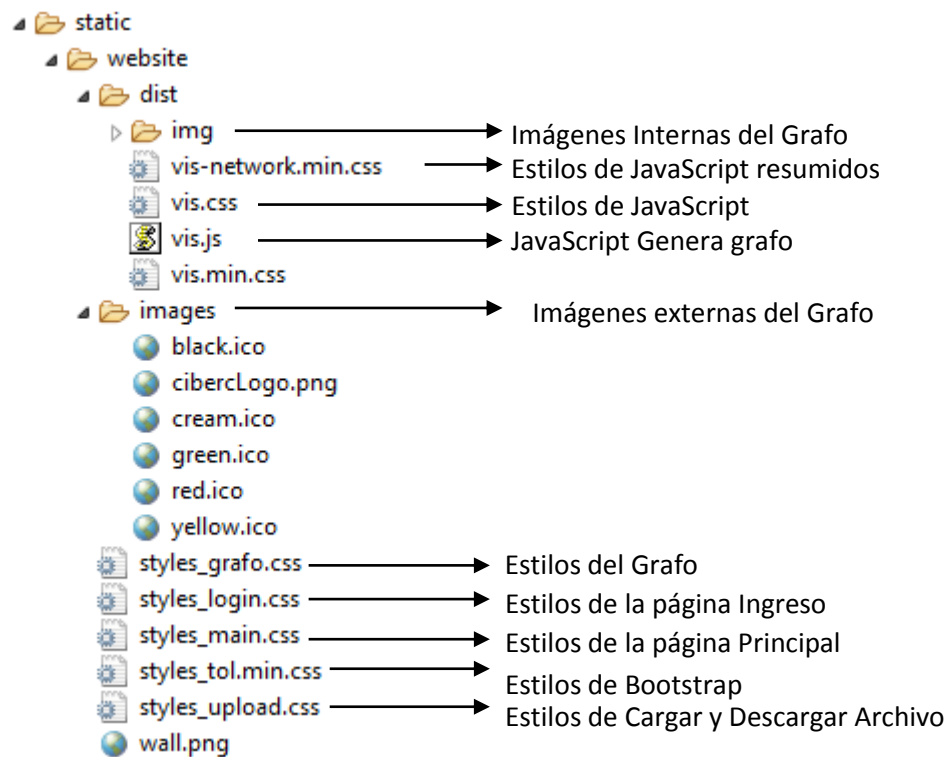
```

document.getElementById('text').innerHTML = '100%';
document.getElementById('bar').style.width = '496px';
document.getElementById('loadingBar').style.opacity = 0;
// really clean the dom element
setTimeout(function ()
{document.getElementById('loadingBar').style.display = 'none';}, 500);
});
}

```



1.2.ARCHIVOS ESTÁTICOS



2. SERVIDOR

2.1.Instalación

Pasos para crear un servidor en Aws

1. Ingresamos a la Página de Amazon Server (<https://aws.amazon.com>), como se observa en la **Figura 2**



Figura 2: Página Principal de Amazon Server

2. Creamos una cuenta de forma gratuita
3. Una vez creada la cuenta, Iniciamos sesión en la consola de administración de AWS como se puede observar en la Figura 3.

The image shows the AWS login page. It features the AWS logo at the top. Below it are three input fields labeled 'Cuenta:', 'Nombre de usuario:', and 'Contraseña:'. At the bottom of the form is a blue button labeled 'Iniciar sesión'. Below the button is a link that says 'Iniciar sesión utilizando credenciales de cuenta raíz'.

Figura 3: Inicio de Sesión de AWS

4. Al ingresar a la consola, hacemos click en “EC2”, como se puede observar en la Figura 4.



Figura 4: Consola de Administración de AWS

5. Al ingresar a EC2, hacemos click en “launch Instance” para crear una instancia del servidor como se puede observar en la Figura 5.

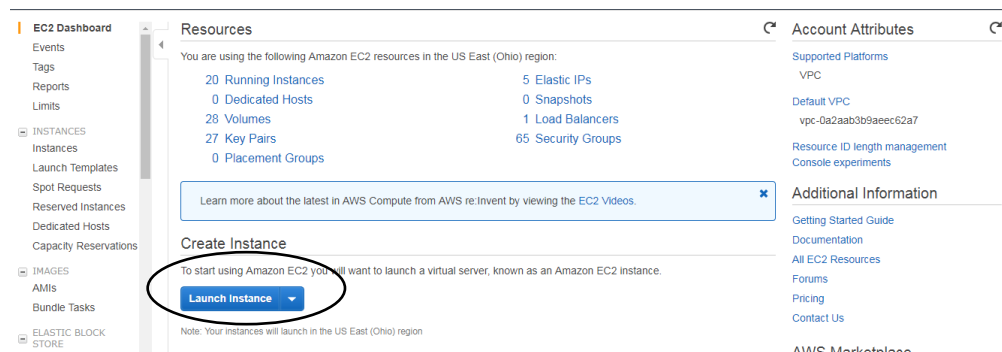


Figura 5: Crear Instancia

6. Al ingresar a “launch Instance”, seleccionamos el sistema operativo del servidor a utilizar, en nuestro caso usamos Amazon Linux 2, por la facilidad de instalación de paquetes como se puede observar en la Figura 6.

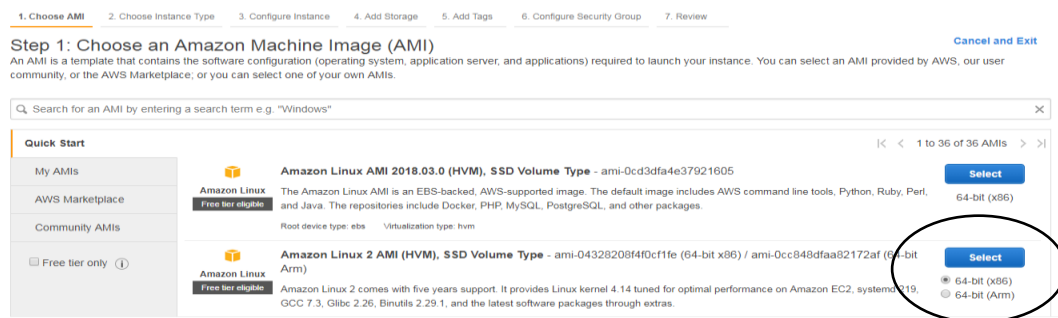


Figura 6: Sistema Operativo

7. Después escogemos el tipo de instancia que vamos a crear, en nuestro caso escogemos una gratuita como se puede observar en la Figura 7.

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: All instance types Current generation Show/Hide Columns

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GiB)	EBS-Optimized Available	Network Performance	IPv6 Support
<input type="checkbox"/>	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	General purpose	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes

Cancel Previous **Review and Launch** Next: Configure Instance Details

Figura 7: Tipo de Instancia

8. Después configuramos la instancia, en nuestro caso usamos la configuración por defecto como se puede observar en la Figura 8.

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances 1 Launch into Auto Scaling Group

Purchasing option ☐ Request Spot instances

Network vpc-0a2aab3b9aee62a7 (default) Create new VPC

Subnet No preference (default subnet in any Availability Zone) Create new subnet

Auto-assign Public IP Use subnet setting (Enable)

Placement group ☐ Add instance to placement group

Capacity Reservation Open Create new Capacity Reservation

IAM role None Create new IAM role

Cancel Previous **Review and Launch** Next: Add Storage

Figura 8: Configuración de la Instancia

9. Una vez configurado la instancia seleccionamos la capacidad de disco de nuestro servidor como se puede observar en la Figura 9.

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more](#) about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/xvda	snap-0ba107d12d889f010	8	General Purpose SSD (gp2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	Not Encrypted

Add New Volume

Figura 9: Capacidad de la Instancia

10. Finalmente configuramos la seguridad de nuestro servidor, en este caso habilitamos los puertos http (80 y 8000), ssh(23) y mysql(3306) como se puede observar en la Figura 10.

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 6: Configure Security Group

Assign a security group: ☒ Create a new security group ☐ Select an existing security group

Security group name:

Description:

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
HTTP	TCP	80	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
HTTPS	TCP	443	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
Custom TCP f	TCP	8000	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
MYSQL/Aurora	TCP	3306	Custom CIDR, IP or Security Group	e.g. SSH for Admin Desktop

Add Rule

Cancel Previous Review and Launch

Figura 10: Selección de Puertos

11. Al finalizar la configuración presionamos launch y creamos la llave de ingreso a nuestro servidor, una vez creada la descargamos como se puede observar en la Figura 11.

Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Create a new key pair

Key pair name

Llave

Download Key Pair

You have to download the **private key file** (*.pem file) before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created.

Cancel Launch Instances

Figura 11: Creación de Llave privada

12. Una vez descargada nuestra llave de ingreso privada, debemos abrir PuttyGen para generar una llave pública como se puede observar en la Figura 12.

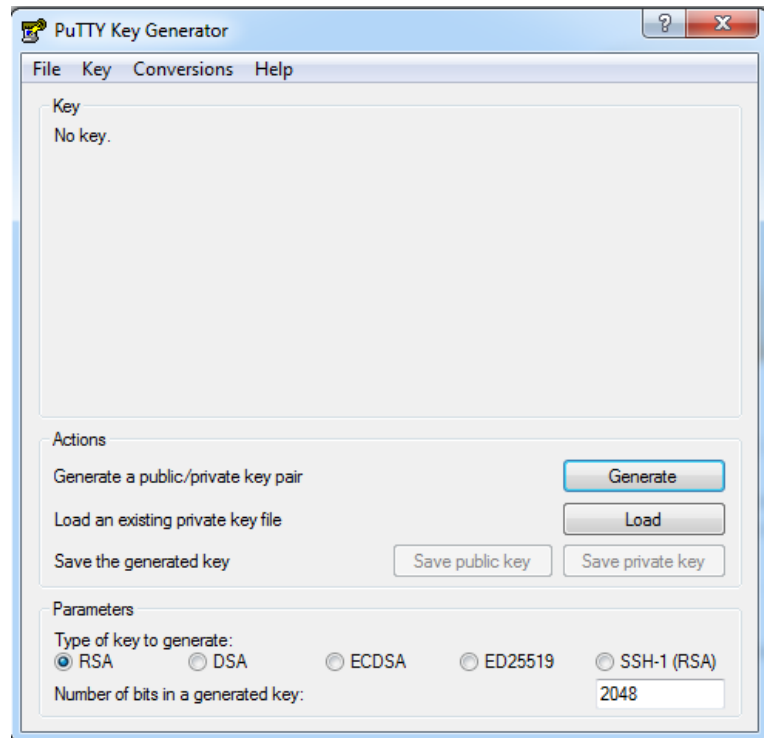


Figura 12: PuttyGen

13. En PuttyGen hacemos click en Load y seleccionamos la llave privada como se puede observar en la Figura 13.

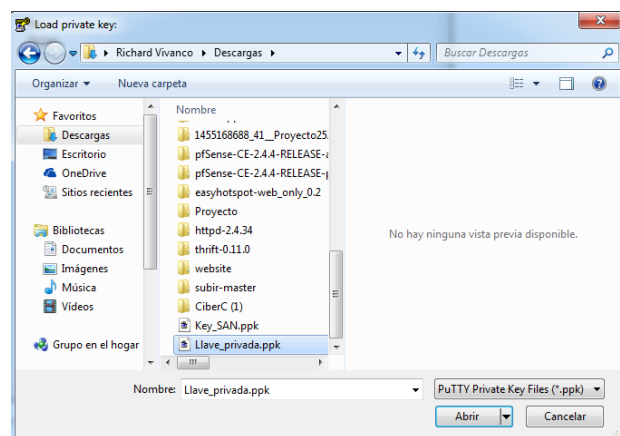


Figura 13: Selección de llave Privada

14. Finalmente presionamos Generate y guardamos nuestra llave publica como se puede observar en la Figura 14.

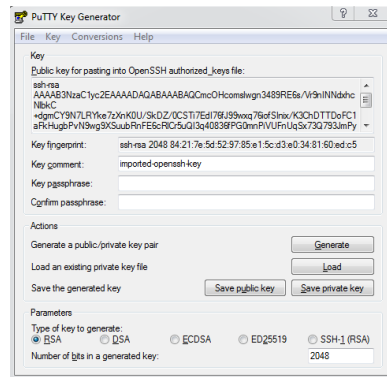


Figura 14: Generar llave publica

15. Al finalizar todo el proceso de configuración de nuestro servidor, Amazon Server nos redirige a la siguiente pestaña donde podemos observar todas las instancias creadas en AWS como se puede observar en la Figura 15.

The image shows the AWS Management Console 'Instances' page. A table lists several EC2 instances. The instance named 'proyecto' is highlighted in blue. The table columns include Name, Instance ID, Instance Type, Availability Zone, Instance State, Status Checks, Alarm Status, Public DNS (IPv4), and IPv4.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4
Web Server ...	i-0080d9b662f1fea0	t2.medium	us-east-2a	running	2/2 checks ...	None	ec2-18-188-68-224.us-...	16.2...
distrib_PAP_...	i-00b23ece5e2e74378	t2.micro	us-east-2c	running	2/2 checks ...	None	ec2-18-222-148-179.us...	16.2...
proyecto	i-027f8df6b7e56a6ef	t2.micro	us-east-2b	running	2/2 checks ...	None	ec2-18-220-110-43.us-...	16.2...
FRMSlave_DB	i-02efebac90ca53bc8	t2.small	us-east-2c	running	2/2 checks ...	None	ec2-18-188-180-45.us-...	16.1...
ruddy_instan...	i-0330f237f188af1be	t2.micro	us-east-2c	running	2/2 checks ...	None	ec2-13-58-52-105.us-e...	16.1...

Figura 15: Instancias Creadas en AWS

16. Nos ubicamos en nuestra instancia creada y observamos la dirección publica DNS y la IPv4 publica de nuestro servidor como se puede observar en la Figura 16.

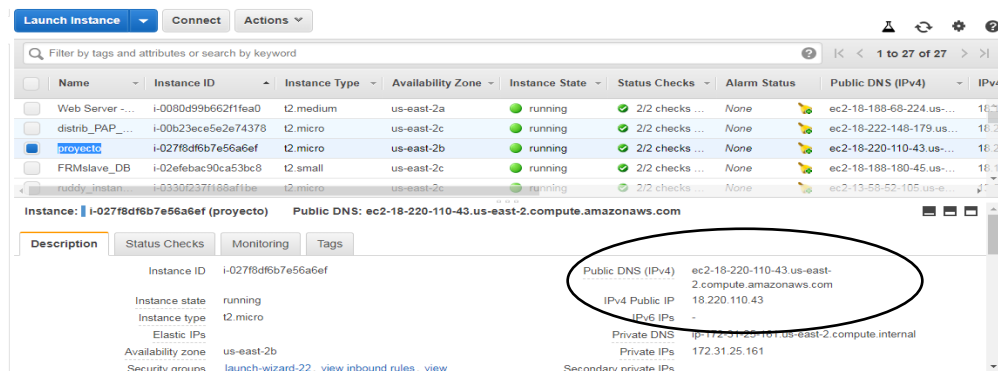


Figura 16: Selección de llave Privada

17. Finalmente, para ingresar a nuestra consola del servidor, abrimos Putty como se puede observar en la Figura 17.

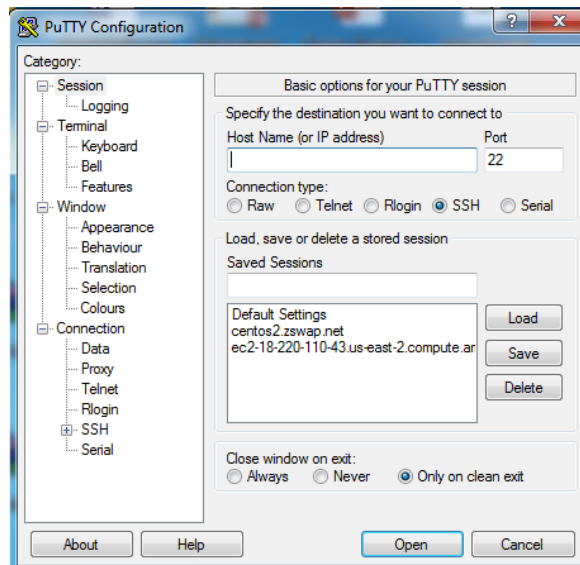


Figura 17: Putty

18. Después editamos las opciones de SSH, para hacerlo nos dirigimos a SSH -Auth y presionamos el botón Browse para seleccionar la llave Publica convertida en el paso 14 como se puede observar en la Figura 18.

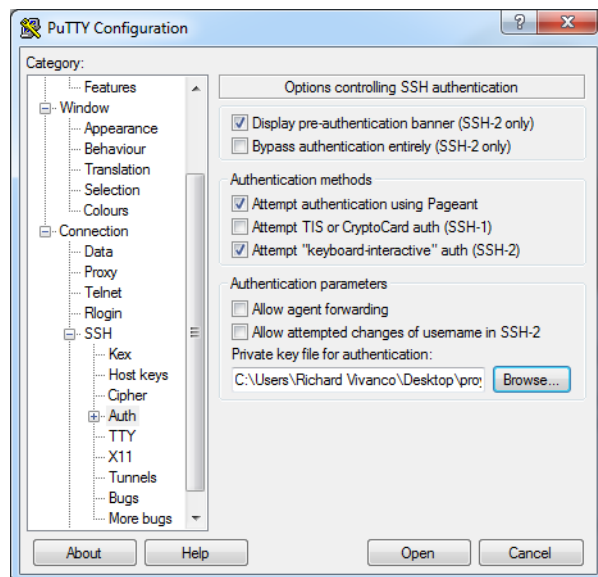


Figura 18: Selección de llave publica

19. Después nos redirigimos a Session e ingresamos la dirección DNS de nuestro servidor, está la podemos observar en el paso 16 y presionamos Open como se puede observar en la Figura 19.

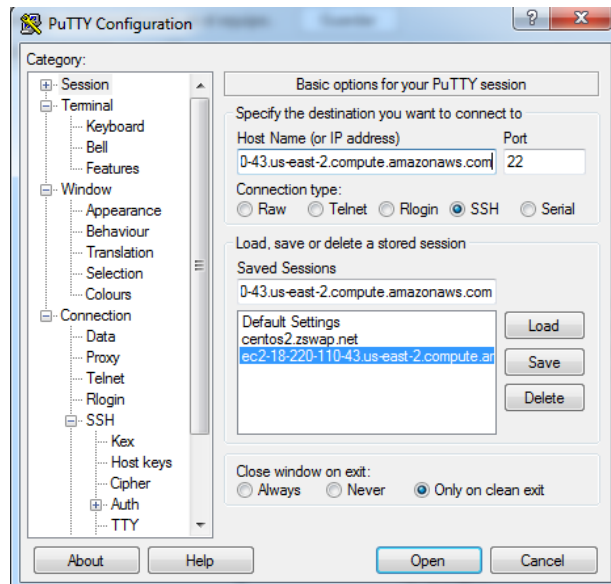


Figura 19: Ingreso al Servidor

20. Finalmente ingresamos a la consola del servidor, el usuario de nuestro servidor escogido en nuestro caso es ec2-user como se puede observar en la Figura 20.

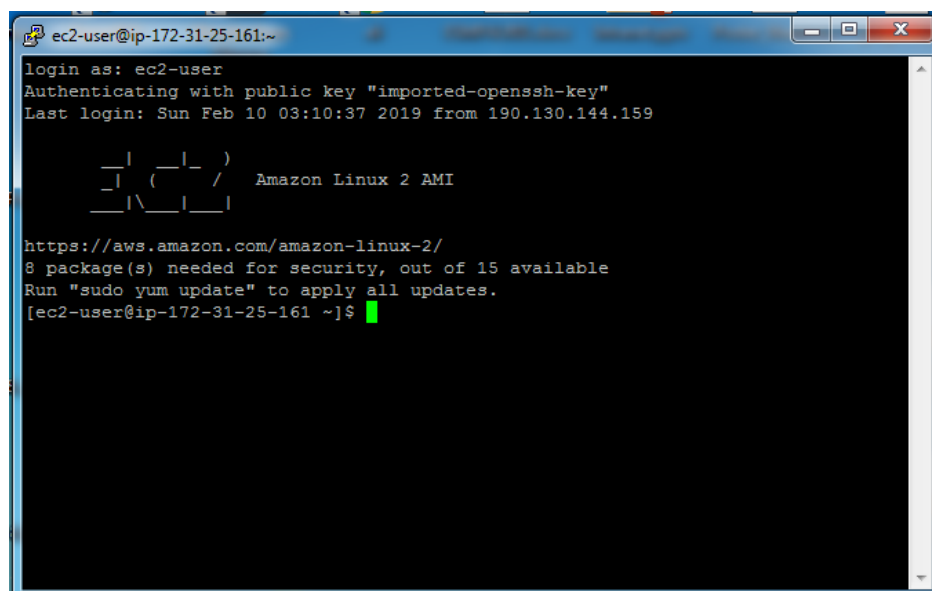


Figura 20: Consola del Servidor

2.2. Configuración

- **Actualización del servidor**
`sudo yum update -y`
- **Instalación de las librerías que nos brinda Amazon**
`sudo amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2`
- **Instalación de apache y MariaDb**
`sudo yum install -y httpd mariadb-server`
- **Configuración de Mysql**
`sudo mysql_secure_installation`
- **Instalación de Php-Fpm**
`sudo yum install php-mbstring -y`
- **Instalación de GitHub**
`sudo yum install git -y`
- **Instalación de pip**
`curl -O https://bootstrap.pypa.io/get-pip.py
python get-pip.py --user
export PATH=~/.local/bin:$PATH
source ~/.bash_profile`
- **Instalación de django**
`pip install django`
- **Instalación de la librería pymysql**
`pip install pymysql`
- **Instalación de la librería python-docx**
`pip install python-docx`
- **Instalación de PhpMyAdmin**
`cd /var/www/html
wget https://www.phpmyadmin.net/downloads/phpMyAdmin-latest-all-languages.tar.gz
mkdir phpMyAdmin && tar -xvzf phpMyAdmin-latest-all-languages.tar.gz -C
phpMyAdmin --strip-components 1
rm -rf phpMyAdmin-latest-all-languages.tar.gz`

- **Habilitación de Servicios**

```
sudo systemctl start httpd
sudo systemctl enable httpd
sudo systemctl start mariadb
sudo systemctl enable mariadb
sudo systemctl start php-fpm
sudo systemctl enable php-fpm
```

- **Instalación del microservicio**

- **Clonamos el repositorio de gitHub**

git clone <https://github.com/Rimavig/subir.git>

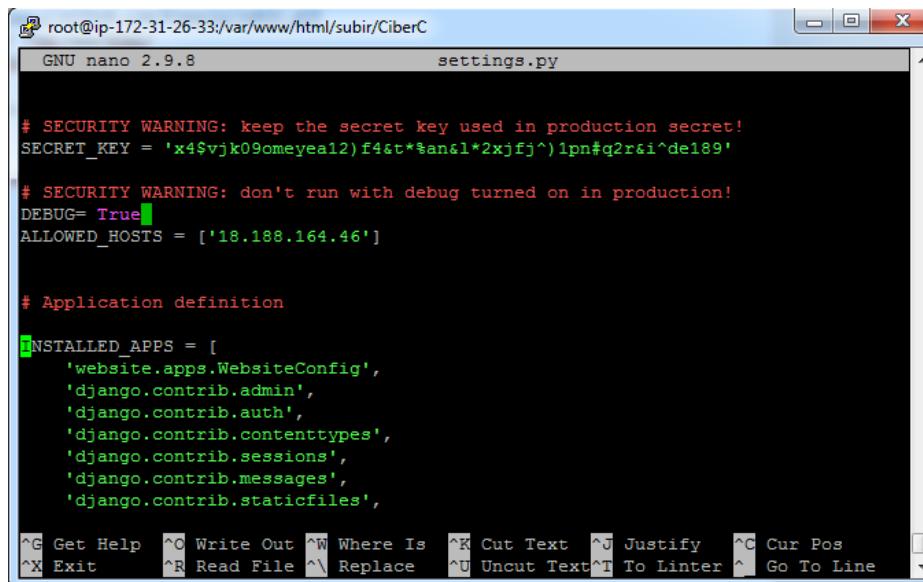
- **Copiamos la carpeta subir al directorio /var/www/html/**

```
cp -rf subir/ /var/www/html/
```

- **Editamos el archivo settings.py**

```
cd /var/www/html/subir
cd CiberC/
nano settings.py
```

Editamos ALLOWED_HOSTS por la dirección pública actual de nuestro servidor como se puede observar en la Figura 21



```
root@ip-172-31-26-33:/var/www/html/subir/CiberC
GNU nano 2.9.8 settings.py

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'x4$vjK09omeyea12)f4&t*%an&l*2xjffj^)1pn#q2r&i^de189'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG= True
ALLOWED_HOSTS = ['18.188.164.46']

# Application definition

INSTALLED_APPS = [
    'website.apps.WebsiteConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^I To Linter ^_ Go To Line
```

Figura 21: Archivo settings.py

- **Ejecutamos el microservicio como se puede observar en la Figura 22.**

```
./manage.py runserver 0.0.0.0:8000 &
```

```
root@ip-172-31-26-33:/var/www/html/subir
[1]+  Exit 127                  ./manage.py runserver 0.0.0.0:8000
[root@ip-172-31-26-33 CiberC]# cd ..
[root@ip-172-31-26-33 subir]# ls
CiberC  db.sqlite3  file  get-pip.py  manage.py  static  website
[root@ip-172-31-26-33 subir]# ./manage.py runserver 0.0.0.0:8000 &
[1] 9849
[root@ip-172-31-26-33 subir]# Performing system checks...


System check identified no issues (0 silenced).
February 11, 2019 - 22:37:38
Django version 1.11.20, using settings 'CiberC.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.
[root@ip-172-31-26-33 subir]#
```

Figura 22: Microservicio Activo

3. BASE DE DATOS

- Actualización de la Base de datos usando PhpMyAdmin
 - Ingresamos a PhpMyAdmin usando la dirección publica de nuestro servidor <http://my.public.dns.amazonaws.com/phpMyAdmin> como se puede observar en la Figura 23

18.188.164.46/phpMyAdmin/



Bienvenido a phpMyAdmin

Idioma - Language
Español - Spanish

Iniciar sesión ⓘ

Usuario:

Contraseña:

Continuar

Figura 23: Ingreso en PhpMyAdmin

➤ Ingresamos nuestro usuario y contraseña de MariaDb

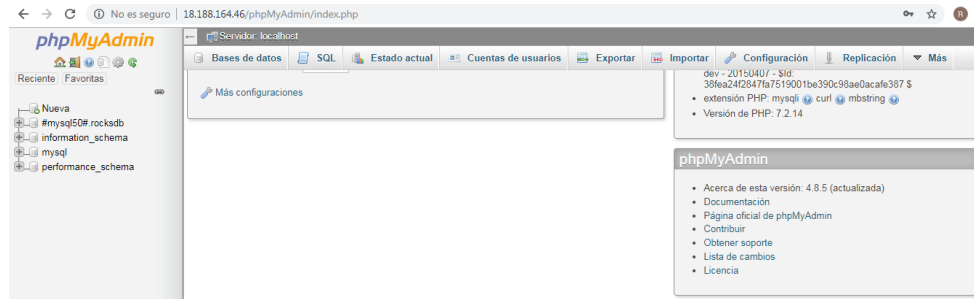


Figura 24: Página Principal PhpMyAdmin

- **Importar Base de datos al servidor**

Para poder importar las tablas de la base de datos es necesario crear la base de datos Con el mismo nombre que exportamos la Base como se observa en la Figura 25



Figura 25: Creación de Base de Datos

Una vez creada la base de datos nos dirigimos a Importar y seleccionamos el archivo .sql exportado del servidor anterior como se observa en la Figura 26

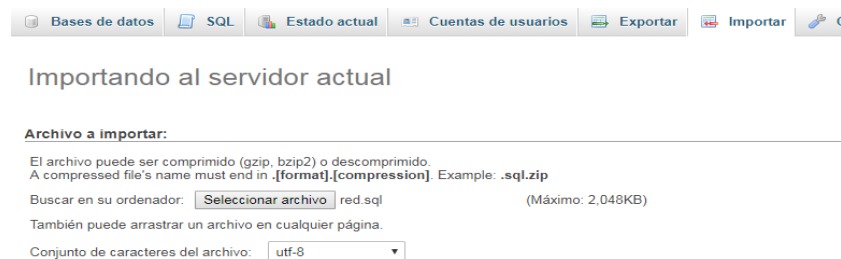


Figura 26: Importar Base de Datos

Si la importación se realizó correctamente podemos observar las tablas creadas como se observa en la Figura 27

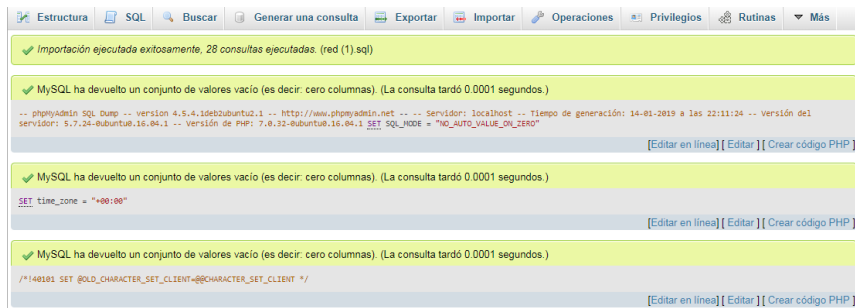


Figura 27: Detalle de Importar Base de datos

- **Exportar Base de datos del servidor**

Seleccionamos la base de datos, después hacemos click en exportar y presionamos continuar como se observa en la Figura 28

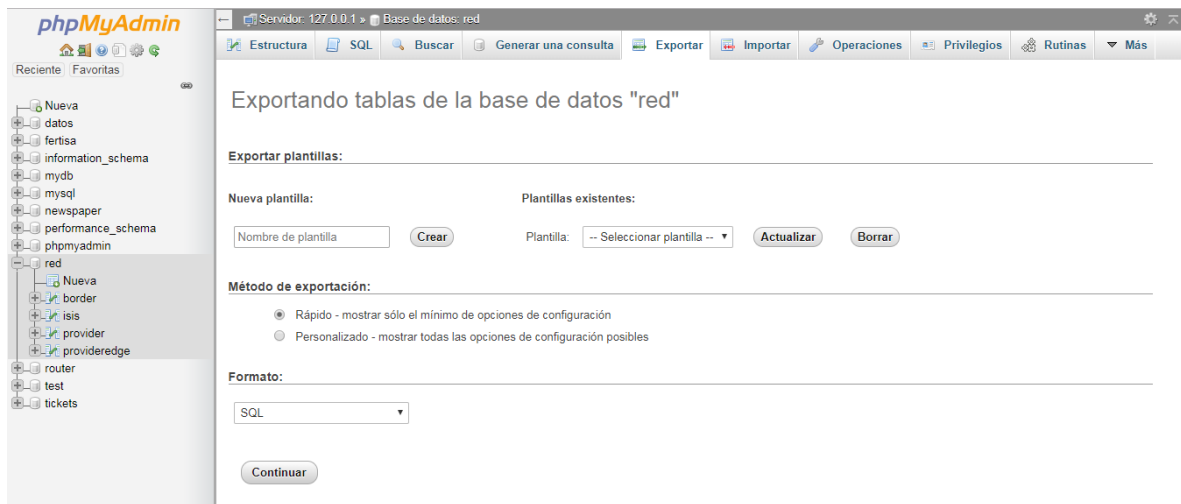


Figura 28: Importar Base de Datos