

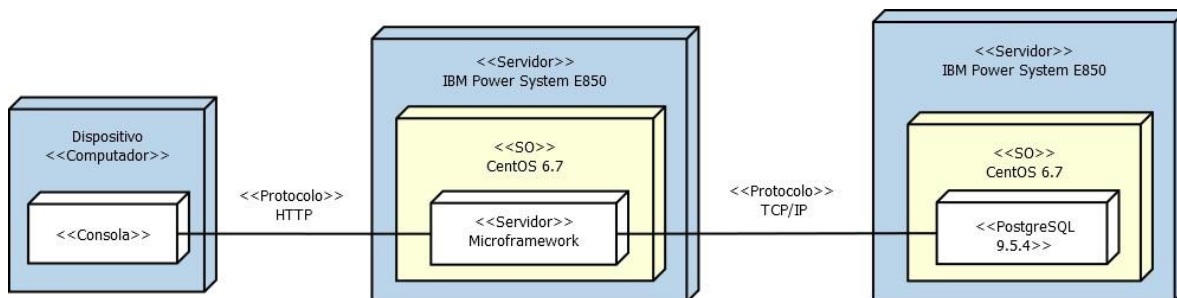
Sistemas Distribuidos

Informe del Parcial 1

https://github.com/diegoedelgado/distributed_system/tree/master/Vagrant/Flask_Postgres

Introducción

A continuación, se expone la solución planteada para un esquema de aprovisionamiento como el presentado en la siguiente imagen:



Se plantea la siguiente distribución para el desarrollo de este proyecto, se hará uso de la herramienta **Vagrant** y una serie de Cookbooks de chef. Inicialmente se mostrará la configuración del archivo Vagrantfile, posteriormente los cookbooks necesarios para el aprovisionamiento de un micro-Framework disponible para usar en el lenguaje Python llamado **Flask**, el cual nos servirá como puente entre el nodo de consulta y la base de datos, para esta última usaremos **PostgreSQL** como motor. Finalmente se mostrarán las recetas necesarias para el despliegue de la base de datos.

1. Despliegue Vagrant

Con base en la arquitectura propuesta anteriormente, se decide aprovisionar dos máquinas virtuales, una encargada de contener el micro framework Flask y la segunda soportara todo lo correspondiente a la base de datos PostgreSQL.

Partimos con un script, provisioner, ubicado en el archivo VagrantFile, el cual se encarga de instalar todas las dependencias necesarias para la primera máquina con el micro framework.

```

4 $provisioner = <<SCRIPT
5 echo "#!/bin/bash
6 function InstallPip {
7   if [ '$(which pip)' ]; then
8     echo '-- Already installed.'
9     return
10  fi
11  sudo yum install python -y
12  sudo yum install nano -y
13  sudo yum install python-psycopg2.i686 -y
14  curl -O https://bootstrap.pypa.io/get-pip.py
15  sudo python get-pip.py
16  rm get-pip.py
17 }
18 echo 'Installing Pip...'; InstallPip
19 echo 'Installing Flask...'; pip install flask
20 exit 0" | /bin/bash
21 SCRIPT
22

```

Ilustración 1 Script provisioner

Luego de este script iniciamos con la configuración para aprovisionar las dos máquinas, maquina uno **centos_flask** y **centos_databases**.

```

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.define :centos_flask do |node|
    node.vm.box = "centos66"
    node.vm.network :private_network, ip: "192.168.56.51"
    node.vm.network :forwarded_port, guest: 80, host: 8080
    node.vm.network "public_network", :bridge => "eth4", ip:"192.168.131.51", :auto_config => "false", :netmask => "255.255.255.0"
    node.vm.provider :virtualbox do |vb|
      vb.customize ["modifyvm", :id, "--memory", "1024", "--cpus", "2", "--name", "centos_flask" ]
    end
    config.vm.provision :chef_solo do |chef|
      chef.cookbooks_path = "cookbooks"
      |chef.add_recipe "mirror"
      chef.add_recipe "flask"
      chef.json = {"aptmirror" => {"server" => "192.168.131.254", "webip1" => "192.168.56.52", "webip2" => "192.168.56.53"}}
    end
    node.vm.provision :shell, :inline => $provisioner # runs as root
  end
end

```

Ilustración 2 Configuración Centos_Flask

De la anterior imagen se puede observar que se usara una imagen de centos6.6, se determinan dos direcciones Ips, una interna y otra externa con su respectiva mascara de red, además de esto se determinan parámetros como memoria, cpus, nombre de la máquina entre otros. Posterior a esto se determinan las recetas de Chef a usarse, para este caso tendremos una receta, <<mirror>> la cual nos ayudara a que los repositorios de la maquina apunten a otra máquina ubicada de forma local. Posterior a eso tenemos <<flask>> receta que se encarga de instalar todas las dependencias necesarias para que el micro framework funcione.

Por otra parte, contamos con la configuración de la maquina **centos_databases**:

```

end
config.vm.define :centos_databases do |db|
  db.vm.box = "centos66"
  db.vm.network :private_network, ip: "192.168.56.54"
  db.vm.network :forwarded_port, guest: 80, host: 8080
  #db.vm.network :public_network, :bridge => "eth4", ip:"192.168.131.54", :auto_config => "false", :netmask => "255.255.255.0"
  db.vm.provider :virtualbox do |vb|
    vb.customize ["modifyvm", :id, "--memory", "1024", "--cpus", "1", "--name", "centos_databases" ]
  end
  config.vm.provision :chef_solo do |chef|
    chef.cookbooks_path = "cookbooks"
    #chef.add_recipe "mirror"
    chef.add_recipe "postgres"
    chef.json = {"aptmirror" => {"server" => "192.168.131.254", "webip1" => "192.168.56.52", "webip2" => "192.168.56.53"}}
  end
end
end

```

Ilustración 3 Centos_databases

De forma inicial el aprovisionamiento de la maquina **centos_databases** es muy similar a la primera a excepción porque esta cuenta con un récipe llamado <<postgres>> con el cual se realizan todas las configuraciones necesarias para que esta máquina pueda operar. El funcionamiento de este servicio será explicado con más detalle más adelante.

2. Cookbooks Micro Framework Flask

En esta sección se establecen las configuraciones necesarias para el aprovisionamiento de la máquina que tendrá el Flask como micro framework.

Se dispone de la siguiente distribución de archivos:

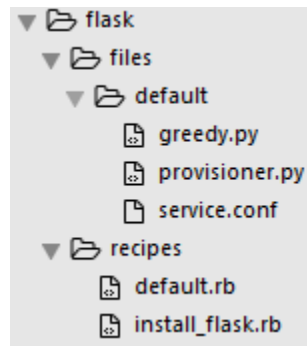


Ilustración 4 Distribución de Archivos recipe Flask

Como se observa en la anterior distribución de archivos, tenemos una sección de files y otra se récipes. En la primera encontramos todos los archivos que incluiremos dentro de nuestra máquina, el primero archivo **greedy.py** se encarga de realizar la consulta con la máquina **centos_databases** en la que se encuentra la base de datos PostgreSQL. El archivo **service.conf** se encarga de convertir las instrucciones escritas en el archivo greedy.py en un servicio nativo del sistema facilitando el proceso de prueba y ejecución.

```
start on started sshd
stop on runlevel [!2345]

exec /usr/bin/python2.6 /home/vagrant/greedy.py
respawn
```

Ilustración 5 Archivo service.conf

```
import psycopg2
from flask import Flask
app = Flask(__name__)

def greedy():
    host = "host="+192.168.56.54+" "
    port = "port="+str(5432)+" "
    db = "dbname="+swn+" "
    user = "user="+pi+" "
    passwd = "password="+security+++" "

    conn_string = host+port+db+user+passwd
    conn = psycopg2.connect(conn_string)
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM devices")
    records = cursor.fetchall()
    return "value: "+str(records)

@app.route("/hi")
def hi():
    return "Hi!, I am a greedy algorithm"

@app.route("/greedy")
def greedy():
    host = "host="+192.168.56.54+" "
    port = "port="+str(5432)+" "
    db = "dbname="+swn+" "
    user = "user="+pi+" "
    passwd = "password="+security+++" "

    conn_string = host+port+db+user+passwd
    conn = psycopg2.connect(conn_string)
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM devices")
    records = cursor.fetchall()
    return "value: "+str(records)

if __name__ == "__main__":
    app.run('192.168.56.51', port=80)
```

Ilustración 6 Archivo greedy.py

En cuanto a la sección de récipes, contamos con un archivo llamado **install_flask.rb** en el cual se definen las dependencias necesarias para que Flask funcione.

```

execute 'pg' do
  command 'yum localinstall http://yum.postgresql.org/9.4/redhat/rhel-6-x86_64/pgdg-centos94-9.4-3.noarch.rpm -y'
end

#yum_package 'postgresql94' do
#  action :install
#end

bash "open port" do
  user "root"
  code <<-EOH
iptables -I INPUT 5 -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT
service iptables save
EOH
end
cookbook_file "/home/vagrant/provisioner.py" do
  source "provisioner.py"
  mode 0644
  owner "vagrant"
  group "wheel"
end
cookbook_file "/home/vagrant/greedy.py" do
  source "greedy.py"
  mode 0644
  owner "vagrant"
  group "wheel"
end
cookbook_file "/etc/init/service.conf" do
  source "service.conf"
  mode 0644
  owner "root"
  group "wheel"
end

#execute 'provisioner' do
#  command sudo python provisioner.py
#end

```

Ilustración 7 Archivo install_flask.rb

En este script podemos observar las distintas dependencias que instalaremos para el funcionamiento de Flask, inicialmente movemos los archivos provisioner.py, greedy.py y service.conf a las rutas determinadas en el formato. Además, se especifica en las líneas de bash el puerto por el cual se estará escuchando el servicio web, para este caso el puerto 80.

3. Cookbooks Aprovisionamiento PostgreSQL

Para la sección de aprovisionamiento de la máquina de base de datos haremos uso de los siguientes archivos:

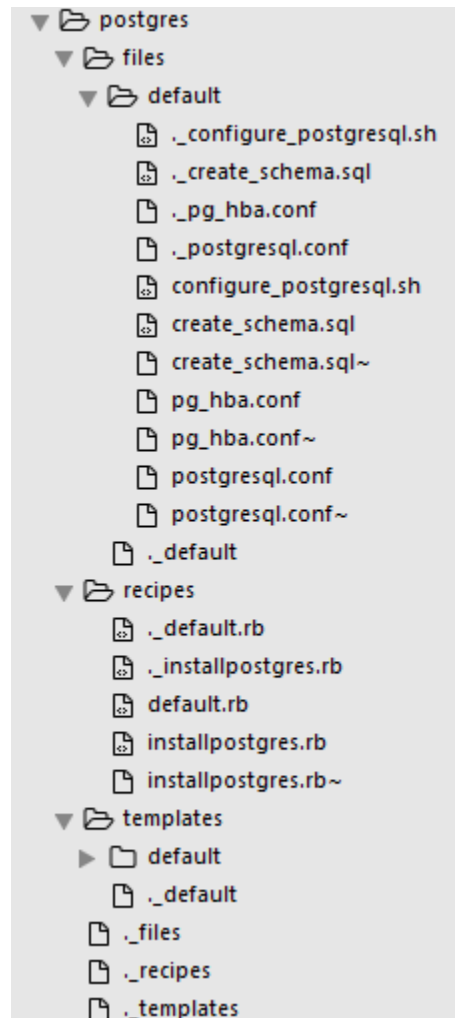


Ilustración 8 Directorio de Archivos Cookbook PostgreSQL

Para la configuración de esta máquina se hace uso de un cookbook en el que se modificaron los siguientes archivos.

Inicialmente el archivo **pg_hba.conf** establece las políticas de conexiones seguras a la base de datos, es decir en este archivo de configuración se encuentran las direcciones ips desde las cuales se pueden realizar consultas a la base de datos.

#	TYPE	DATABASE	USER	ADDRESS	METHOD
# "local" is for Unix domain socket connections only					
local	all		all		trust
# IPv4 local connections:					
host	all		all	127.0.0.1/32	ident
host	all		all	192.168.56.52/24	trust
host	all		all	192.168.56.53/24	trust
host	all		all	192.168.56.51/24	trust
host	all		all	192.168.131.53/24	trust
host	all		all	192.168.131.52/24	trust
host	all		all	192.168.131.51/24	trust
# IPv6 local connections:					
host	all		all	:::1/128	ident
# Allow replication connections from localhost, by a user with the					
# replication privilege.					
#local	replication		postgres		peer
#host	replication		postgres	127.0.0.1/32	ident
#host	replication		postgres	:::1/128	ident

Archivo de configuración pg_hba.conf

En el siguiente archivo **postgresql.conf** determina los distintos parámetros de configuración para la base de datos como las unidades métricas de memoria, kB, MB, GB, TB, los directorios para los archivos y dentro de los parámetros de nuestro interés los de conexión a la base de datos y los de autenticación.

```

# - Connection Settings -

listen_addresses = '*'      # what IP address(es) to listen on;
                             # comma-separated list of addresses;
                             # defaults to 'localhost'; use '*' for all
                             # (change requires restart)
port = 5432                 # (change requires restart)
max_connections = 100       # (change requires restart)
# Note: Increasing max_connections costs ~400 bytes of shared memory per
# connection slot, plus lock space (see max_locks_per_transaction).
#superuser_reserved_connections = 3 # (change requires restart)
unix_socket_directories = '/tmp' # comma-separated list of directories
                                # (change requires restart)
unix_socket_group = ''      # (change requires restart)
unix_socket_permissions = 0777 # begin with 0 to use octal notation
                                # (change requires restart)
#bonjour = off              # advertise server via Bonjour
                                # (change requires restart)
#bonjour_name = ''          # defaults to the computer name
                                # (change requires restart)

# - Security and Authentication -

#authentication_timeout = 1min # 1s-600s
#ssl = off                   # (change requires restart)
#ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL' # allowed SSL ciphers
                                # (change requires restart)
#ssl_prefer_server_ciphers = on # (change requires restart)
#ssl_ecdh_curve = 'prime256v1' # (change requires restart)
#ssl_renegotiation_limit = 0   # amount of data between renegotiations
#ssl_cert_file = 'server.crt' # (change requires restart)
#ssl_key_file = 'server.key'  # (change requires restart)
#ssl_ca_file = ''             # (change requires restart)
#ssl_crl_file = ''            # (change requires restart)
#password_encryption = on
#db_user_namespace = off

```

Archivo de configuración postgresql.conf

En este caso el parámetro más significativo es el puerto de conexión establecido en el **5432**, el cual será el puerto por el que la base de datos estará escuchando.

En la carpeta `recipes` se encuentra el archivo **installpostgres.rb** en el cual se encuentran los scripts necesarios para la instalación de todas las dependencias usadas para que el servicio de la base de datos funcione de forma correcta, dentro de este archivo se adicionaron las siguientes líneas las cuales permiten reiniciar el servicio de la base de datos una vez terminadas todas las configuraciones, esto con el fin de resolver un error que impedía que se aplicarían de forma correcta las configuraciones.


```
#Ejecuta el archivo create_schema.sql a traves del motor de postgres ejecuta
bash "create schema" do
  user "postgres"
  cwd "/tmp"
  code <<-EOH
  psql < create_schema.sql
  EOH
end
bash "postgres restart" do
  user "root"
  code <<-EOH
  service postgresql-9.4 restart
  EOH
end
```

Líneas de configuración archivo, **installpostgres.rb**

Con esta línea se termina la configuración respectiva para el funcionamiento de la base de datos, los demás archivos se dejaron en su normalidad y pueden ser encontrados en este directorio.

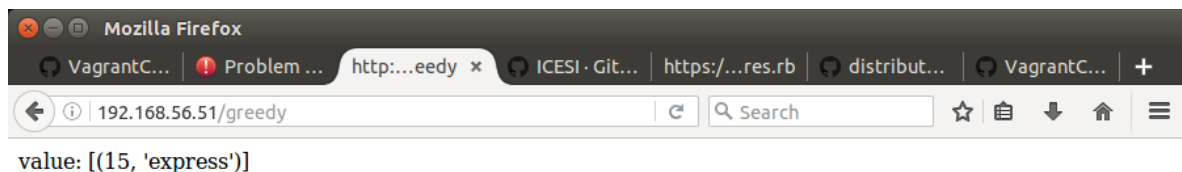
4. Resultados

Una vez realizado todo el proceso anterior obtenemos unos logs similares a los de la siguiente imagen, mostrando que toda la configuración se realizó de forma adecuada.

```
distribuidos@307C: ~/Documentos/SisDistri/Exam1
distribuidos@307C:~/Documentos/SisDistri/Exam1$ vagrant up
Bringing machine 'centos_flask' up with 'virtualbox' provider...
Bringing machine 'centos_databases' up with 'virtualbox' provider...
==> centos_flask: Clearing any previously set forwarded ports...
==> centos_flask: Clearing any previously set network interfaces...
==> centos_flask: Preparing network interfaces based on configuration...
centos_flask: Adapter 1: nat
centos_flask: Adapter 2: hostonly
centos_flask: Adapter 3: bridged
==> centos_flask: Forwarding ports...
centos_flask: 22 => 2222 (adapter 1)
==> centos_flask: Running 'pre-boot' VM customizations...
==> centos_flask: Booting VM...
==> centos_flask: Waiting for machine to boot. This may take a few minutes...
centos_flask: SSH address: 127.0.0.1:2222
centos_flask: SSH username: vagrant
centos_flask: SSH auth method: private key
centos_flask: Warning: Connection timeout. Retrying...
centos_flask: Warning: Connection timeout. Retrying...
centos_flask: Warning: Remote connection disconnect. Retrying...
==> centos_flask: Machine booted and ready!
==> centos_flask: Checking for guest additions in VM...
==> centos_flask: Configuring and enabling network interfaces...
==> centos_flask: Mounting shared folders...
centos_flask: /vagrant => /home/distribuidos/Documentos/SisDistri/Exam1
==> centos_flask: Machine already provisioned. Run 'vagrant provision' or use the '--provision'
==> centos_flask: to force provisioning. Provisioners marked to run always will still run.
==> centos_databases: Clearing any previously set forwarded ports...
==> centos_databases: Fixed port collision for 22 => 2222. Now on port 2200.
==> centos_databases: Clearing any previously set network interfaces...
==> centos_databases: Preparing network interfaces based on configuration...
centos_databases: Adapter 1: nat
centos_databases: Adapter 2: hostonly
centos_databases: Adapter 3: bridged
==> centos_databases: Forwarding ports...
centos_databases: 22 => 2200 (adapter 1)
==> centos_databases: Running 'pre-boot' VM customizations...
==> centos_databases: Booting VM...
==> centos_databases: Waiting for machine to boot. This may take a few minutes...
centos_databases: SSH address: 127.0.0.1:2200
centos_databases: SSH username: vagrant
centos_databases: SSH auth method: private key
centos_databases: Warning: Connection timeout. Retrying...
==> centos_databases: Machine booted and ready!
==> centos_databases: Checking for guest additions in VM...
==> centos_databases: Configuring and enabling network interfaces...
==> centos_databases: Mounting shared folders...
centos_databases: /vagrant => /home/distribuidos/Documentos/SisDistri/Exam1
==> centos_databases: Machine already provisioned. Run 'vagrant provision' or use the '--provision'
==> centos_databases: to force provisioning. Provisioners marked to run always will still run.
distribuidos@307C:~/Documentos/SisDistri/Exam1$
```

Configuración exitosa de las maquinas.

Al realizar la consulta desde la ip determinada para la maquina centos_flask: 192.168.56.51 observamos el funcionamiento de ambas maquinas, una brindándonos un servicio web por el cual se puede consultar a la base de datos.



value: [(15, 'express')]

Se observa la consulta que se realiza a la base de datos, trayendo consigo la información solicitada de la tabla swm.

```
--CREATE EXTENSION postgis;
create user pi with password 'security++';
alter role pi with createdb;
create database swm owner pi;
\connect swm
CREATE TABLE devices(
    id integer NOT NULL PRIMARY KEY,
    device_name varchar(10) NOT NULL UNIQUE
);
grant all privileges on table devices to pi;
CREATE TABLE consumption(
    id integer NOT NULL PRIMARY KEY,
    date date NOT NULL,
    device_id integer references devices(id)
);
INSERT INTO devices
(id, device_name)
VALUES
(15 , 'express');
```

Archivo create_schema.sql