Ya creada la instancia en Amazon instalar euca2tools

```
sudo yum install euca2tools
```

Crear el siguiente programa:

```
import sys
print('Hello, World!')
print('The sum of 2 and 3 is 5.')
sum = int(sys.argv[1]) + int(sys.argv[2])
print('The sum of {0} and {1} is {2}.'.format(sys.argv[1], sys.argv[2], sum))
```

Verificando que este instalado euca2tools

```
% python
Python 2.7.1 (r271:86882M, Nov 30 2010, 10:35:34)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto
>>> ec2 = boto.connect_ec2()
>>> ec2.get_all_zones()
[Zone:us-east-1a, Zone:us-east-1b, Zone:us-east-1c, Zone:us-east-1d]
>>>
```

Ejemplo de conexión a un almacenamiento en  Google Cloud Storage:

```
% python
Python 2.7.1 (r271:86882M, Nov 30 2010, 10:35:34)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto
>>> gs = boto.connect_gs()
```

Mostrar Regiones:

```
$ python
Python 2.7.1 (r271:86882M, Nov 30 2010, 10:35:34)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto.ec2
>>> boto.ec2.regions()
[RegionInfo:eu-west-1, RegionInfo:us-east-1, RegionInfo:ap-northeast-1,
RegionInfo:us-west-1, RegionInfo:ap-southeast-1]
>>> eu_conn = _[0].connect()
```

Conectarse a una región:

```
$ python
Python 2.7.1 (r271:86882M, Nov 30 2010, 10:35:34)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto.ec2
>>> eu_conn = boto.ec2.connect_to_region('eu-west-1')
>>>
```

Trabajar con una Instancia:

```
import os
import time
import boto
import boto.manage.cmdshell
def launch_instance(ami='ami-7341831a',
                instance_type='t1.micro',
                key_name='paws',
                key_extension='.pem',
                key_dir='~/.ssh',
                group_name='paws',
                ssh_port=22,
                cidr='0.0.0.0/0',
                tag='paws',
                user_data=None,
                cmd_shell=True,
                login_user='ec2-user',
                ssh_passwd=None):
    """
    Launch an instance and wait for it to start running.
    Returns a tuple consisting of the Instance object and the CmdShell
    object, if request, or None.
    ami        The ID of the Amazon Machine Image that this instance will
            be based on.  Default is a 64-bit Amazon Linux EBS image.
    instance_type The type of the instance.
    key_name   The name of the SSH Key used for logging into the instance.
            It will be created if it does not exist.
    key_extension The file extension for SSH private key files.
```

```
key_dir    The path to the directory containing SSH private keys.
          This is usually ~/.ssh.
group_name The name of the security group used to control access
          to the instance.  It will be created if it does not exist.
ssh_port   The port number you want to use for SSH access (default 22).
cidr       The CIDR block used to limit access to your instance.
tag        A name that will be used to tag the instance so we can
          easily find it later.
user_data  Data that will be passed to the newly started
          instance at launch and will be accessible via
          the metadata service running at http://169.254.169.254.
cmd_shell  If true, a boto CmdShell object will be created and returned.
          This allows programmatic SSH access to the new instance.
login_user The user name used when SSH'ing into new instance.  The


          default is 'ec2-user'
ssh_passwd The password for your SSH key if it is encrypted with a
          passphrase.
"""
cmd = None

# Create a connection to EC2 service.
# You can pass credentials in to the connect_ec2 method explicitly
# or you can use the default credentials in your ~/.boto config file
# as we are doing here.
ec2 = boto.connect_ec2()
# Check to see if specified keypair already exists.
# If we get an InvalidKeyPair.NotFound error back from EC2,
# it means that it doesn't exist and we need to create it.
try:
  key = ec2.get_all_key_pairs(keynames=[key_name])[0]
except ec2.ResponseError, e:
  if e.code == 'InvalidKeyPair.NotFound':
    print 'Creating keypair: %s' % key_name
    # Create an SSH key to use when logging into instances.
    key = ec2.create_key_pair(key_name)

    # AWS will store the public key but the private key is
    # generated and returned and needs to be stored locally.
    # The save method will also chmod the file to protect
    # your private key.
    key.save(key_dir)
  else:
    raise
# Check to see if specified security group already exists.
# If we get an InvalidGroup.NotFound error back from EC2,
# it means that it doesn't exist and we need to create it.
try:
  group = ec2.get_all_security_groups(groupnames=[group_name])[0]
except ec2.ResponseError, e:
  if e.code == 'InvalidGroup.NotFound':
    print 'Creating Security Group: %s' % group_name
    # Create a security group to control access to instance via SSH.
    group = ec2.create_security_group(group_name,
                                      'A group that allows SSH access')
  else:
        raise
# Add a rule to the security group to authorize SSH traffic
# on the specified port.
try:
    group.authorize('tcp', ssh_port, ssh_port, cidr)
except ec2.ResponseError, e:
    if e.code == 'InvalidPermission.Duplicate':
        print 'Security Group: %s already authorized' % group_name
    else:
        raise

# Now start up the instance.  The run_instances method
# has many, many parameters but these are all we need
# for now.
reservation = ec2.run_instances(ami,
                                key_name=key_name,
                                security_groups=[group_name],
                                instance_type=instance_type,
                                user_data=user_data)
# Find the actual Instance object inside the Reservation object
# returned by EC2.
instance = reservation.instances[0]
# The instance has been launched but it's not yet up and
# running.  Let's wait for its state to change to 'running'.
print 'waiting for instance'
while instance.state != 'running':
    print '.'
    time.sleep(5)
    instance.update()
print 'done'
# Let's tag the instance with the specified label so we can
# identify it later.
instance.add_tag(tag)
# The instance is now running, let's try to programmatically
# SSH to the instance using Paramiko via boto CmdShell.

if cmd_shell:
    key_path = os.path.join(os.path.expanduser(key_dir),
                            key_name+key_extension)
    cmd = boto.manage.cmdshell.sshclient_from_instance(instance,
                                                       key_path,
                                                       user_name=login_user)

return (instance, cmd)
```

Usando una función para una instancia:

```
>>> from ec2_launch_instance import launch_instance
>>> launch_instance()
Creating keypair: paws
Security Group: paws already authorized
waiting for instance
.
.
done
SSH Connection refused, will retry in 5 seconds
```

```
(Instance:i-98847ef8, ≤boto.manage.cmdshell.SSHClient object at 0x10141fb90>)
>>> instance, cmdshell = _
>>> cmdshell.shell()
```

```
       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|
See /usr/share/doc/system-release/ for latest release notes.
No packages needed for security; 1 packages available
[ec2-user@domU-12-31-39-00-E4-53 ~]$ ls
[ec2-user@domU-12-31-39-00-E4-53 ~]$ pwd
/home/ec2-user
[ec2-user@domU-12-31-39-00-E4-53 ~]$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/xvda1             8256952    918228   7254864  12% /
tmpfs                   305624         0    305624   0% /dev/shm
[ec2-user@domU-12-31-39-00-E4-53 ~]$
[ec2-user@domU-12-31-39-00-E4-53 ~]$ logout
*** EOF
```

Seguimiento de instancias con etiquetas:

```
import boto
ec2 = boto.connect_ec2()
# Get a list of all current instances.  We will assume that the only
# instance running is the one we started in the previous recipe.
reservations = ec2.get_all_instances()
# Despite the name, get_all_instances does not return Instance
# objects directly.  What it returns are Reservation objects
# as returned by run_instances.  This is a confusing aspect of
# the EC2 API that we have decided to be consistent with in boto.
# The following incantation will return the actual Instance
# object embedded within the Reservation.  We are assuming we
# have a single Reservation which launched a single Instance.
instance = reservations[0].instances[0]
# We could call create_tags directly here but boto provides
# some nice convenience methods to make it even easier.
# We are going to store a single tag on this instance.
instance.add_tag('paws')
# We can now ask for all instances that have the tag name "paws"
# and get our instance back again.
reservations = ec2.get_all_instances(filters={'paws' : None})
new_instance = reservations[0].instances[0]
assert new_instance.id == instance.id
```

Para Cargar el par de claves SSH:

```
import boto.ec2
def sync_keypairs(keypair_name, public_key_file):
    """
    Synchronize SSH keypairs across all EC2 regions.
    keypair_name    The name of the keypair.
    public_key_file The path to the file containing the
                    public key portion of the keypair.
    """
    fp = open(public_key_file)
    material = fp.read()
    fp.close()

    for region in boto.ec2.regions():
        ec2 = region.connect()
        # Try to list the keypair.  If it doesn't exist
        # in this region, then import it.
        try:
            key = ec2.get_all_key_pairs(keynames=[keypair_name])[0]
            print 'Keypair(%s) already exists in %s' % (keypair_name,
                                                         region.name)
        except ec2.ResponseError, e:
            if e.code == 'InvalidKeyPair.NotFound':
                print 'Importing keypair(%s) to %s' % (keypair_name,
                                                        region.name)
                ec2.import_key_pair(keypair_name, material)
```

Asociar la dirección IP elástica con una instancia:

```
import boto
ec2 = boto.connect_ec2()
# Let's assume the instance we are interested in has already been started
# in the previous examples and is tagged with "paws".  This little
# incantation will retrieve it for us.
instance = ec2.get_all_instances(filters={'paws' : None})[0].instances[0]
# Allocate an Elastic IP Address.  This will be associated with your
# account until you explicitly release it.
address = ec2.allocate_address()
# Associate our new Elastic IP Address with our instance.
ec2.associate_address(instance.id, address.public_ip)
# Alternatively, you could do this.
instance.use_ip(address)
```