SSH & SCP IN PYTHON WITH

PARAMIKO

- "Paramiko" is a combination of the Esperanto words for "paranoid" and "friend".
- It's a module for Python 2.7/3.4+ that implements the SSH2 protocol for secure (encrypted and authenticated) connections to remote machines.
- Emphasis is on using SSH2 as an alternative to SSL for making secure connections between python scripts.

❑ All major ciphers and hash methods are supported.

❑ SFTP (Secure File Transfer Protocol) client and server mode are both supported too.

API documentation

- ❑The high-level client API starts with creation of an **SSHClient** object.

- ❑For more direct control, pass a socket (or socket-like object) to a **Transport**, and use **start_server** or **start_client** to negotiate with the remote host as either a server or client.

- ❑As a client, you are responsible for authenticating using a password or private key, and checking the server's host key. (Key signature and verification is done by paramiko, but you will need to provide private keys and check that the content of a public key matches what you expected to see.)

- ❑As a server, you are responsible for deciding which users, passwords, and keys to allow, and what kind of channels to allow.

- Once you have finished, either side may request flow-controlled **channels** to the other side, which are Python objects that act like sockets, but send and receive data over the encrypted session.

Installation

For most users, the recommended method to install is via pip:

```
pip install paramiko
```

Paramiko has only a few **direct dependencies**:

- The big one, with its own sub-dependencies, is Cryptography; see its specific note below for more details;
- bcrypt, for Ed25519 key support;
- pynacl, also for Ed25519 key support.

There are also a number of **optional dependencies** you may install using setuptools ‘extras’:

- If you want all optional dependencies at once, use paramiko[all].
- For Match exec config support, use paramiko[invoke] (which installs Invoke).
- For GSS-API / SSPI support, use paramiko[gssapi], though also see the below subsection on it for details.

- paramiko[ed25519] references the dependencies for Ed25519 key support.
  - As of Paramiko 2.x this doesn’t technically do anything, as those dependencies are core installation requirements.
  - However, you should use this for forwards compatibility; 3.0 will drop those dependencies from core, leaving them purely optional.

Demo

```
import base64
import paramiko



key = paramiko.RSAKey(data=base64.b64decode(b'AAA...'))
client = paramiko.SSHClient()
client.get_host_keys().add('ssh.example.com', 'ssh-rsa', key)
client.connect('ssh.example.com', username='strongbad',
password='thecheat')
stdin, stdout, stderr = client.exec_command('ls')

for line in stdout:
    print('... ' + line.strip('\n'))
client.close()
```

This prints out the results of executing ls on a remote server. The host key b'AAA...' should of course be replaced by the actual base64 encoding of the host key. If you skip host key verification, the connection is not secure!

Demo of scp

```
import sys, paramiko

if len(sys.argv) < 5:
    print "args missing"
    sys.exit(1)

hostname = sys.argv[1]
password = sys.argv[2]
source = sys.argv[3]
dest = sys.argv[4]
```

```
username = "root"
port = 22

try:
t = paramiko.Transport((hostname, port))

t.connect(username=username, password=password)
sftp = paramiko.SFTPClient.from_transport(t)

sftp.get(source, dest)

finally:
t.close()
```

How to SSH and transfer files with python

- SSH is the method typically used to access a remote machine and run commands, retrieve files or upload files.

- You can transfer files from the remote machine to the local or vice versa using SFTP (Secure File Transfer Protocol) and SCP(Secure Copy Protocol).

- According to paramiko.org, The python paramiko model gives an abstraction of the SSHv2 protocol with both the client side and server side functionality.

- As a client, you can authenticate yourself using a password or key and as a server you can decide which users are allowed access and the channels you allow

Let’s get on with it

The primary client of Paramiko as documented in the API, is Paramiko.SSHClient. An instance of the Paramiko.SSHClient can be used to make connections to the remote server and transfer files

MAKING A CONNECTION

```
import paramiko

ssh_client=paramiko.SSHClient()

ssh_client.connect(hostname=’hostname’,username=’mokgadi’,password=’
mypassword’)

#Raises BadHostKeyException,AuthenticationException,SSHException,socket
error
```

when you try this, you get the following error:

```
missing_host_key raise SSHException('Server %r not found in known_hosts' % hostname)

paramiko.ssh_exception.SSHException: Server 'hostname' not found in known_hosts
```

Understanding Known Hosts

You see this error because you have not informed your machine that the remote server you “trust” the server you are trying to access. If you go onto you command line or terminal and try to connect to a server for the first time, You will get a message similar to this:

- *The authenticity of host ‘hostname’ can’t be established.RSA key fingerprint is ‘key’. Are you sure you want to continue connecting (yes/no)?*

When you select yes here, you let your machine know that it can trust the machine and you can now access it without the prompt until the key for that machine changes.

- Paramiko similarly requires that you validate your trust with the machine.

- This validation is handled by calling set_missing_host_key_policy() on the SSHClient an passing the policy you want implemented when accessing a new remote machine.

- By default, the paramiko.SSHclient sets the policy to the RejectPolicy. The policy rejects connection without validating as we saw above.

- Paramiko does however give you a way to sort of “Trust all” key policy, the AutoAddPolicy. Parsing an instance of the AutoAddPolicy to set_missing_host_key_policy() changes it to allow any host.

```
import paramiko

ssh_client =paramiko.SSHClient()

ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

ssh_client.connect(hostname='hostname',username='mokgadi',passwo
rd='mypassword')
```

- You should now be in the green

RUNNING COMMANDS ON THE REMOTE MACHINE

To run a command exec_command is called on the SSHClient with the command passed.

The response is returned as a tuple (stdin,stdout,stderr)

For example to list all the files in a directory:

- stdin,stdout,stderr=ssh_client.exec_command(“ls”)
- Getting the type for each of the returned,
  type(stdin) and type(stdout) is ‘paramiko.channel.ChannelFile’
  type(stderr) is class ‘paramiko.channel.ChannelStderrFile’
- According to paramiko.org they are all python file like objects.

The stdin is a write-only file which can be used for commands requiring input,

The stdout file give the output of the command,

The stderr gives the errors returned on executing the command. Will be empty if there is no error

for the command above

```
>>>print(stdout.readlines()) → [u’anaconda-ks.cfg\n’, u’database_backup\n’, u’Desktop\n’,
u’Documents\n’, u’Downloads\n’, …. u’Public\n’, u’Templates\n’, u’Videos\n’]
```

```
>>>print(stderr.readlines) → []
```

COMMANDS REQUIRING INPUT

- Sometimes you need to provide a password or extra input to run a command. This is what stdin is used for. Let’s run the same command above with sudo.

```
stdin, stdout, stderr = ssh.exec_command(“sudo ls”)
stdin.write(‘mypassword\n’)
print stdout.readlines()
```

- Should return list of files and folders as above.

FILE TRANSFERS

File transfers are handled by the paramiko.SFTPClient which you get from calling open_sftp() on an instance of Paramiko.SSHClient.

Downloading a file from remote machine

```
ftp_client=ssh_client.open_sftp()
ftp_client.get(‘remotefileth’,’localfilepath’)
ftp_client.close()
```

Uploading file from local to remote machine

```
ftp_client=ssh.open_sftp()
ftp_client.put(‘localfilepath’,remotefilepath’)
ftp_client.close()
```