Scripting para SysAdmins:

Conexiones SSH y SFTP desde Python con paramiko

Crear conexiones SSH con Python, es tan simple como escribir unas pocas líneas de código gracias al módulo paramiko[1] y con él, comenzamos la nueva serie de artículos *«Scripting para SysAadmins»*, especialmente diseñada para Administradores de Sistemas que deseen incursionar en la programación.

Escrito por: **Eugenia Bahit** (GLAMP Hacker & eXtreme Programmer)

Eugenia es **Arquitecta de Software**, docente e instructora de tecnologías **GLAMP** (GNU/Linux, Apache, MySQL, Python y PHP) y **Agile coach** (UTN) especializada en Scrum y **eXtreme Programming**. Miembro de la **Free Software Foundation**, **The Linux Foundation** y **Debian Hackers**. Creadora de python-printr, Europio Engine y colaboradora de Vim.

Webs:
Cursos de programación: www.cursosdeprogramacionadistancia.com
Web personal: www.eugeniabahit.com

Redes sociales:
Twitter / Identi.ca: **@eugeniabahit**

Combinar el poder de SSH con el de Python es un sueño muy fácil de cumplir: solo basta con recurrir al módulo `paramiko` para que con unas pocas líneas de código, podamos crear *scripts* que nos permitan crear conexiones SSH y ejecutar comandos de forma remota.

```
import paramiko
```

`paramiko` permite que el usuario se valide tanto por contraseña como por par de llaves,

1 https://pypi.python.org/pypi/paramiko

por lo cual es ideal para autenticar usuarios más allá de las políticas del servidor.

Conexión con autenticación por contraseña

El siguiente, es un ejemplo de conexión estándar autenticada por contraseña:

```
import paramiko


# Inicia un cliente SSH
ssh_client = paramiko.SSHClient()

# Establecer política por defecto para localizar la llave del host localmente
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

# Conectarse
ssh_client.connect('123.456.78.90', 22, 'user', 'secretpassword')

# Ejecutar un comando de forma remota capturando entrada, salida y error estándar
entrada, salida, error = ssh_client.exec_command('ls -la')

# Mostrar la salida estándar en pantalla
print salida.read()

# Cerrar la conexión
ssh_client.close()
```

Como se puede observar, utilizar `paramiko` es sumamente simple y no tiene demasiada ciencia. El único "truco" a tener en cuenta, es establecer la política por defecto para la localización de la llave del host en el ordenador del cliente. De lo contrario, si no se encontrara la llave del host (*host key*, usualmente localizada en el archivo `~/.ssh/know_hosts`), Python nos arrojaría la siguiente excepción de `paramiko`:

```
    raise SSHException('Unknown server %s' % hostname)
paramiko.SSHException: Unknown server 123.456.78.90
```

Un consejo: para evitar dejar la contraseña almacenada en texto plano (lo cual es de sumo riesgo y sin sentido), éste se le puede solicitar al usuario mediante `getpass`.

```
from getpass import getpass
import paramiko


clave = getpass('Clave: ')

HOST = '123.456.78.90'
PUERTO = 372
USUARIO = 'juanperez'
```

```
datos = dict(hostname=HOST, port=PUERTO, username=USUARIO, password=CLAVE)

ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(**datos)
entrada, salida, error = ssh_client.exec_command('ls -la')
print salida.read()
ssh_client.close()
```

Conexión autenticada por par de llaves

Para conectarse evitando la utilización de una contraseña, se pudo haber creado previamente, una llave pública que el usuario debió enviar al servidor, para que el servidor la pueda contrarrestar contra la llave privada del usuario. En este caso, solo podrá bastar con no enviar la contraseña para autenticar al usuario:

```
import paramiko


HOST = '123.456.78.90'
PUERTO = 372
USUARIO = 'juanperez'
datos = dict(hostname=HOST, port=PUERTO, username=USUARIO)

ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(**datos)
entrada, salida, error = ssh_client.exec_command('ls -la')
print salida.read()
ssh_client.close()
```

Ejecución remota de comandos como súper usuario

Si el usuario `root` no tiene permitidas las conexiones remotas ¿cómo ejecutar comandos con privilegios de súper usuario si nos estamos conectando con `paramiko`? Ésta, suele ser una de las dudas más frecuente y sin embargo, la solución no está en `paramiko` sino en los comandos ejecutados en el propio *shell*. Véamoslo con un ejemplo en el que listamos las reglas establecidas en `iptables` con `sudo -S`:

```
from getpass import getpass
import paramiko


HOST = '123.456.78.90'
PUERTO = 372
USUARIO = 'juanperez'
datos = dict(hostname=HOST, port=PUERTO, username=USUARIO)

password = getpass('Clave: ')
ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(**datos)
```

```
comando = 'echo %s | sudo -S iptables -nL' % password
entrada, salida, error = ssh_client.exec_command(comando)

print salida.read()
ssh_client.close()
```

Guardando los *logs* de paramiko

Con `paramiko` también es posible registrar toda la actividad que `paramiko` realice desde el *script*:

```
from getpass import getpass
import paramiko


HOST = '123.456.78.90'
PUERTO = 372
USUARIO = 'juanperez'
datos = dict(hostname=HOST, port=PUERTO, username=USUARIO)
password = getpass('Clave: ')

paramiko.util.log_to_file('paramiko.log')

ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(**datos)
comando = 'echo %s | sudo -S iptables -nL' % password
entrada, salida, error = ssh_client.exec_command(comando)

print salida.read()
ssh_client.close()
```

Manipulando archivos remotos mediante SFTP

Con paramiko es posible crear conexiones SFTP directas (sin recurrir al cliente SSH) por medio del objeto `SFTPClient()`. Aunque también, podremos obtener una instancia de `SFTPClient` desde el propio cliente SSH (`SSHClient`) recurriendo al método `open_sftp()`.

```
import paramiko

paramiko.util.log_to_file('paramiko.log')

HOST = '123.456.78.90'
PUERTO = 372
USUARIO = 'juanperez'
datos = dict(hostname=HOST, port=PUERTO, username=USUARIO)

ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(**datos)
entrada, salida, error = ssh_client.exec_command('pwd')
ruta = salida.read().replace('\n', '')

sftp = ssh_client.open_sftp()  # Crea un objeto SFTPClient()
```

```
# Descargando archivos
archivos = sftp.listdir()

for archivo in archivos:
    archivo_remoto = "%(ruta)s/%(nombre)s" % dict(ruta=ruta, nombre=archivo)

    print "Descargando: %s" % archivo_remoto

    try:
        sftp.get(archivo_remoto, "paramii/%s" % archivo)
        print "copiado archivo %s." % archivo
    except:
        print "Fallo al intentar copiar %s. Tal vez es un directorio." % archivo

sftp.close()
ssh_client.close()
```

El cliente SFTP de `paramiko`, provee de los mismos métodos que un cliente FTP normal. Todos los métodos pueden ser consultados desde el manual oficial ingresando en http://www.lag.net/paramiko/docs/paramiko.SFTPClient-class.html

Entre los métodos provistos por el cliente SFTP, podremos encontrar:

Método	**Descripción**
`get(remoto, local)`	Trae un archivo remoto a un directorio local
`put(local, remoto)`	Envía un archivo local al servidor
`chdir(ruta)`	Cambia el directorio de trabajo actual
`chmod(ruta, modo)`	Cambia permisos en un archivo
`mkdir(ruta, modo=511)`	Crea un directorio
`rename(anterior, nuevo)`	Renombra un archivo o directorio
`remove(archivo)`	Elimina un archivo
`rmdir(ruta)`	Elimina un directorio
`symlink(fuente, destino)`	Crea un enlace simbólico