Python XMPP Client (Twisted)

Introducción

En este post vamos a realizar un cliente de xmpp en python. Es de obligada mención Twisted Matrix, una increíble framework para python el cual facilita muchíssimo la programación enfocada a redes. En cuanto a protocolos podemos trabajar con HTTP, XMPP, NNTP, IMAP, SSH, IRC, FTP y muchos más, también podemos trabajar con varias arquitecturas (TCP, UDP, SSL/TLS, IP Multicast, Unix domain sockets).

Características

- Separación de los protocolos y transportes

El diseño de Twisted se basa en la separación completa entre los protocolos lógicos (que por lo general dependen de la conexión semántica basada en streams —flujos—, como el HTTP o POP3) y el transporte en capas físicas soportado como la semántica basada en streams (como archivos, bibliotecas sockets o SSL). La conexión entre un protocolo lógico y una capa de transporte que ocurre en el último momento posible, justo antes de la información se pase a la instancia de protocolo lógico. El protocolo lógico es informado de la instancia de capa de transporte, y puede utilizarlo para enviar mensajes de un lado para comprobar la identidad del otro extremo. Tenga en cuenta que todavía es posible, en el código de protocolo, para consultar profundamente la capa de transporte en cuestiones de transporte (como la comprobación de un certificado SSL del lado del cliente). Naturalmente, el código de dicho protocolo, se producirá un error (lanzar una excepción) si la capa de transporte no es compatible con tales semánticas.

- Deferreds

El modelo central de aplicación para Twisted es el concepto de

un deferred (predefinir algo que se usara como valor futuro). Un deferred es un valor que no se ha calculado todavía, por ejemplo, porque las necesidades de datos desde un equipo remoto. Los deferreds se pueden transferir, al igual que los objetos normales, pero no se puede pedir por su valor. Cada deferred es compatible con una cadena de devolución de llamada. Cuando el deferred toma el valor, es transferido a través de la cadena de devolución de llamada, con el resultado de cada de callback (devolución) siendo la entrada (input) para la siguiente. Esto permite que operen en los valores de un deferred sin saber lo que son. Por ejemplo, si un deferred devuelve una cadena desde un equipo remoto con una dirección IP en formato quad, un callback se puede adjuntar para traducirla a un número de 32 bits. Cualquier usuario del deferred puede ahora tratarlo como deferred de retorno de un número de 32 bits. Esto, y la capacidad de relación para definir «errbacks» (callbacks que son llamados como controladores de errores), permite que el código que se ve como si fuera de serie, mientras que todavía mantiene la abstracción por eventos.

- Soporte de Thread (hilos o subprocesos)

Twisted soporta una abstracción sobre threads en crudo usando un thread como una fuente deferred. Por lo tanto, un deferred que es retornado inmediatamente, recibirá un valor cuando finalice el thread. Los callbacks se pueden adjuntar cuando corran en el thread principal, a fin de aliviar la necesidad de soluciones complejas de bloqueo. Un buen ejemplo de tal uso, que viene de las bibliotecas de soporte de Twisted, es usar este modelo para llamadas en bases de datos. La llamada de la base de datos misma pasa de un thread exterior, pero el análisis del resultado que sucede en el thread principal.

- Soporte de bucle de externos

Twisted se puede integrar con bucles de eventos externos, tales como los de GTK+, Qt y Cocoa (a través de PyObjC). Esto

le permite el uso de Twisted como la capa de soporte de red en aplicaciones GUI, usando todas sus colecciones sin tener que añadir una sobrecarga de thread-por-socket, como lo haria cualquier biblioteca nativa de Python. Se pude integrar en proceso un completo web server con una aplicación interfaz gráfica utilizando este modelo, por ejemplo.

Cabe decir que no es objetivo de este post profundizar en twisted matrix, pues se necesitarían sería muy extenso (de echo harían falta muchos :p). También hace falta añadir, que hay una muy buena documentación en la página oficial.

Instalación

Podemos descargar el framework des de la página oficial:
Twisted Matrix Downloads
O bien odemos instalarlo fácilmente en Debian y derivados, puesto que esta en los repositorios (sino en el link de la página oficial antes citada te facilitan la PPA):

```
$>sudo apt-get install python-twisted python-twisted-words
```

El modulo twisted-words contiene las librerías de jabber.

Codigo del Cliente

```
#!/usr/bin/python

# Twisted Imports
from twisted.words.protocols.jabber import client, jid , xmlstream
from twisted.words.xish import domish
from twisted.internet import reactor

name = None
server = None
resource = None
password = None
me = None

thexmlstream = None
```

```
tryandregister = 1

def initOnline(xmlstream):
     # creamos los observadores hacia las respuestas xml
message y una general (podemos incluir presence, iq...)
    global factory
    print 'Initializing...'
    xmlstream.addObserver('/message', gotMessage)
    xmlstream.addObserver('/*', gotSomething)

def authd(xmlstream):
    # Autentificacion
    global thexmlstream
    thexmlstream = xmlstream
    print "we've authd!"
    print repr(xmlstream)

    # se envia la presencia a los demas clientes
    presence = domish.Element(('jabber:client', 'presence'))
    presence.addElement('status').addContent('Online')
    xmlstream.send(presence)

    initOnline(xmlstream)

def send(author, to, msg):
    # esta funcion envia los mensajes
    global thexmlstream
    message = domish.Element(('jabber:client','message'))
    message["to"] = jid.JID(to).full()
    message["from"] = jid.JID(author).full()
    message["type"] = "chat"
     message.addElement("body", "jabber:client", msg+ "- ya lo
sabia adicotalainformatica1")

    thexmlstream.send(message)

def gotMessage(el):
    # esta funcion parsea los mensajes recibidos
    global me
    # print 'Got message: %s' % str(el.attributes)
    from_id = el["from"]
```

```
    body = "empty"
    for e in el.elements():
        if e.name == "body":
            body = unicode(e.__str__())
        break

    send(me, from_id, body)

def gotSomething(el):
    # Observador general
      print 'Got something: %s -> %s' % (el.name,
str(el.attributes))

def authfailedEvent(xmlstream):
    global reactor
    print 'Auth failed!'
    reactor.stop()

def invaliduserEvent(self,xmlstream):
    print "Invalid User"

def registerfailedEvent(self,xmlstream):
    print 'Register failed!'

if __name__ == '__main__':
    #Parametrizamos la conexion
    PASSWORD = '123456'
    myJid = jid.JID('adictoalainformatica2@antitot-linux')
    me = 'adictoalainformatica2@antitot-linux'
    factory = client.XMPPClientFactory(myJid, PASSWORD)

    # Registramos las callbacks de autentificacion
    print 'register callbacks'
    factory.addBootstrap(xmlstream.STREAM_AUTHD_EVENT, authd)
factory.addBootstrap(client.BasicAuthenticator.INVALID_USER_EV
ENT, invaliduserEvent)
factory.addBootstrap(client.BasicAuthenticator.AUTH_FAILED_EVE
NT, authfailedEvent)
factory.addBootstrap(client.BasicAuthenticator.REGISTER_FAILED
_EVENT, registerfailedEvent)
```

```
    # Paarametrizamos y arrancamos el reactor (encargado de
mantener y gestionar las callbacks
    # producidas por las acciones de la conexion)
    reactor.connectTCP('localhost', 5222, factory)
```

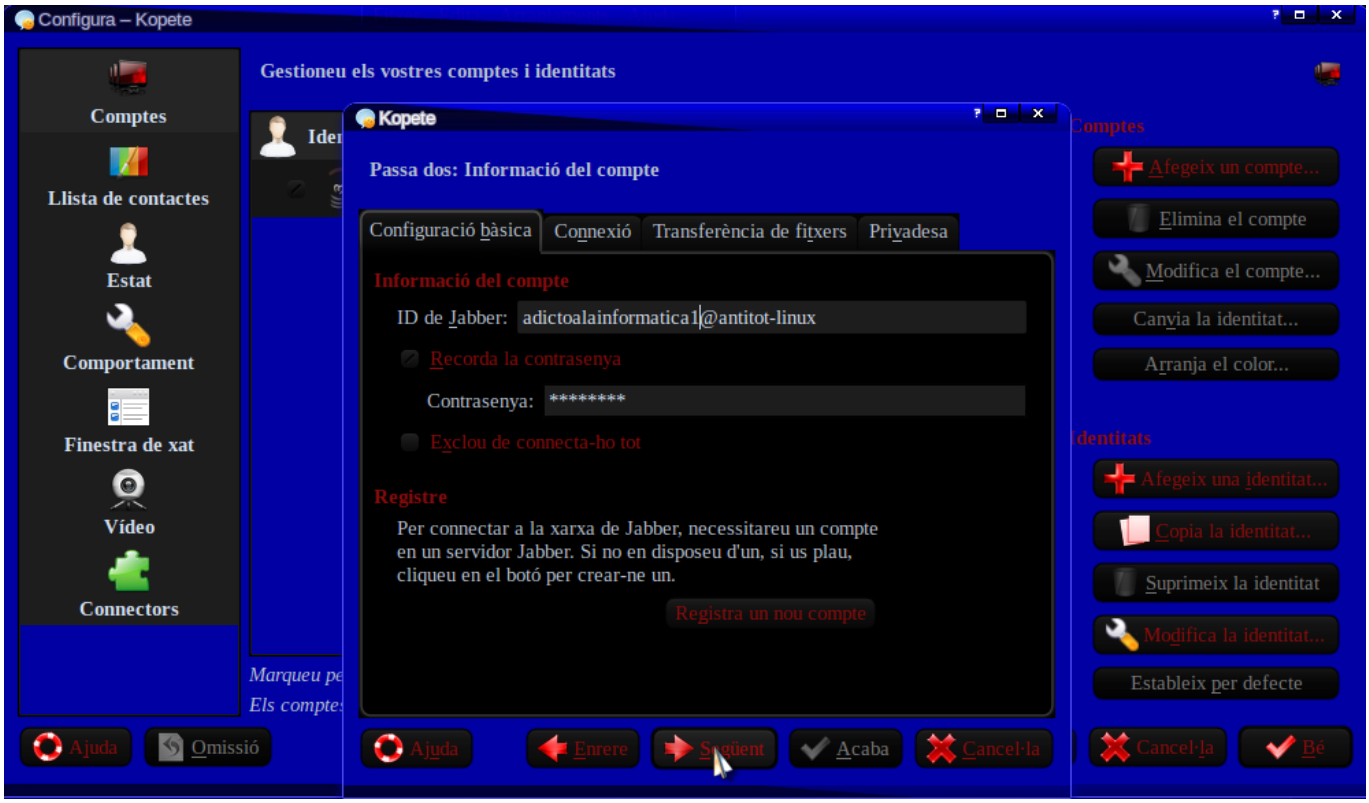
Test

Para realizar el test haremos referencia al post anteriror sobre xmpp donde configuramos y creamos dos usuarios (adictosalainformatica1, adictosalainformatica2) con el servidor openfire. Aprovecharemos los usuarios
para configurar el script e indicaremos al reactor donde esta el servidor xmpp:

```
#Parametrizamos la conexion
PASSWORD = '123456'
myJid = jid.JID('adictoalainformatica2@antitot-linux')
me = 'adictoalainformatica2@antitot-linux'
factory = client.XMPPClientFactory(myJid, PASSWORD)
reactor.connectTCP('antitot-linux', 5222, factory)
```

y cualquier cliente como por ejemplo Kopete. Abriremos Kopete e hiremos a configuraciones -> configura. Se abrirá una nueva ventana en esta pincharemos en añadir nuevo elemento.
Seguidamente deberemos escoger el protocolo:

Configuramos la cuenta:



Y ya esta, cuando des del usuario adictosalainformatica enviemos un mensaje a nuestro script éste le responderá el mismo mensaje con la coletilla » – ya lo sabia adicotalainformatica2″.

Código del cliente Xmpp

Puedes descargar el código de este post desde la cuenta de GitHub

Observaciones

Y con este post cierro el tema de redes xmpp dando una vista superficial pero intuitiva de lo que puede ofrecernos este nuevo protocolo tanto a nivel de mensajería como a nivel de distribuidor de comandos. espero que haya sido de ayuda.

Fuentes

- Twisted Matrix
- Wikipedia – Twisted Matrix

Ruben