PlanetLab Node Manager API Documentation

PlanetLab Node Manager API Documentation

Table of Contents

Chapter 1. Introduction

The PlanetLab Node Manager API (NMAPI) is the interface through which the slices access the Node API.

Authentication

Authentication for NM operations is based on the identity of the connecting slice. For slices whose roles are defined as 'nm-controller', the target slice must be listed delegated and as controlled by the calling slice.

Delegation

None

Connection

The NM XMLRPC server listens locally on every PlanetLab node at http://localhost:812.

The NM XMLRPC server can be accessed remotely using an SSH connection through the nm-controller account. Rather than a standard shell, a special command is run that forwards all standard input to the local XMLRPC server, essentially XML-RPC over ssh.

An Example using the PLC and NM API

The nm-controller slice is given a stub account such that it can be accessed over ssh. So rather than logging into NM server listens locally on every PlanetLab node at http://localhost:812.

```
controller_slice_fields = {'name'       : 'princeton_mycontroller',
                           'instantiation' : 'nm-controller',
                           'url'           : 'http://www.yourhost.com',
                           'description'   : 'a brief description of this slice.', }
controller_slice_id = api.AddSlice(plauth, controller_slice_fields)
```

After this, the controller owner, should both add users and nodes to this slice. As well, the controller slice is created using the standard PlanetLab and NM mechanism. So, wait at least 15 minutes before attempting to access the controller slice on any node.

Subsequently, slices that will be delegated to this controller will be registered at PLC. An example follows.

```
delegated_slice_fields = {'name'        : 'anothersite_mydelegated',
                          'instantiation' : 'delegated',
                          'url'           : 'http://www.yourhost.com',
                          'description'   : 'a brief description of this slice.', }
delegated_slice_id = api.AddSlice(plauth, delegated_slice_fields)

# Get ticket for this slice.
ticket = api.GetSliceTicket(plauth, "princetondsl_solteszdelegated")
```

After the slice is registered with PLC, and your application has the Ticket, the last step is to redeem the ticket by presenting it to the NM through the nm-controller account. The following code formats the message correctly.

```
# generate an XMLRPC request.
print xmlrpclib.dumps((ticket,), 'Ticket')
```

Finally, this message must be sent to the NM using the controller account. It should be possible to create a program that creates the ssh connection or to use a library that does this automatically such as: pyXMLRPCssh[1]

Or, you could use something much simpler. Assuming the output from `dumps()` above, is saved to a file called `ticket.txt`, you could run a command like:

```
cat ticket.txt | ssh princeton_mycontroller@mynode.someuniversity.edu
```

Alternately,

```
p = subprocess.Popen(['/usr/bin/ssh', 'princeton_mycontroller@mynode.someuniversity.
     stdin=subprocess.PIPE, stdout=subprocess.PIPE)
print >>p.stdin, xmlrpclib.dumps((ticket,), 'Ticket')
p.stdin.close()
print xmlrpclib.loads(p.stdout.read())
p.wait()
```

The following is a stub to use as you would use the current xmlrpclib.Server() object, but redirects the connection of SSH.

```
"""XML-RPC over SSH.

 To use, create an XmlRpcOverSsh object like so:
  >>> api = XmlRpcOverSsh('princeton_deisenst@planetlab-1.cs.princeton.edu')
 and call methods as with the normal xmlrpclib.ServerProxy interface.
"""

from subprocess import PIPE, Popen
from xmlrpclib import Fault, dumps, loads

__all__ = ['XmlRpcOverSsh']


class XmlRpcOverSsh:
    def __init__(self, userAtHost):
        self.userAtHost = userAtHost

    def __getattr__(self, method):
        return _Method(self.userAtHost, method)


class _Method:
    def __init__(self, userAtHost, method):
        self.userAtHost = userAtHost
        self.method = method

    def __call__(self, *args):
        p = Popen(['ssh', self.userAtHost], stdin=PIPE, stdout=PIPE)
        stdout, stderr = p.communicate(dumps(args, self.method))
        if stderr:
            raise Fault(1, stderr)
        else:
            return loads(stdout)
```

Notes

1. http://cheeseshop.python.org/pypi/pyXMLRPCssh/1.0-0

Chapter 2. PlanetLab API Methods