Parallel-SSH Documentation

Release 0.80.0

P Kittenis

December 24, 2016

Contents

CHAPTER 1

ParallelSSHClient API Documentation

Package containing ParallelSSHClient class.

class pssh.pssh_client.**ParallelSSHClient**(*hosts*, *user=None*, *password=None*, *port=None*, *pkey=None*, *forward_ssh_agent=True*, *num_retries=3*, *timeout=120*, *pool_size=10*, *proxy_host=None*, *proxy_port=22*)

Uses *pssh.ssh_client.SSHClient*, performs tasks over SSH on multiple hosts in parallel.

Connections to hosts are established in parallel when `run_command` is called, therefor any connection and/or authentication exceptions will happen on the call to `run_command` and need to be caught.

Parameters

- **hosts** (*list(str)*) – Hosts to connect to
- **user** (*str*) – (Optional) User to login as. Defaults to logged in user or user from ~/.ssh/config or /etc/ssh/ssh_config if set
- **password** (*str*) – (Optional) Password to use for login. Defaults to no password
- **port** (*int*) – (Optional) Port number to use for SSH connection. Defaults to None which uses SSH default
- **pkey** (paramiko.PKey) – (Optional) Client's private key to be used to connect with
- **num_retries** (*int*) – (Optional) Number of retries for connection attempts before the client gives up. Defaults to 3.
- **timeout** (*int*) – (Optional) Number of seconds to timeout connection attempts before the client gives up. Defaults to 10.
- **forward_ssh_agent** (*bool*) – (Optional) Turn on SSH agent forwarding - equivalent to *ssh -A* from the *ssh* command line utility. Defaults to True if not set.
- **pool_size** (*int*) – (Optional) Greenlet pool size. Controls on how many hosts to execute tasks in parallel. Defaults to number of hosts or 10, whichever is lower. Pool size will be *equal to* number of hosts if number of hosts is lower than the pool size specified as that would only increase overhead with no benefits.
- **proxy_host** (*str*) – (Optional) SSH host to tunnel connection through so that SSH clients connect to self.host via client -> proxy_host -> host
- **proxy_port** (*int*) – (Optional) SSH port to use to login to proxy host if set. Defaults to 22.

Example Usage

```
>>> from pssh.pssh_client import ParallelSSHClient
>>> from pssh.exceptions import AuthenticationException, UnknownHostException, ConnectionErrorEx
```

```
>>> client = ParallelSSHClient(['myhost1', 'myhost2'])
>>> try:
>>> ... output = client.run_command('ls -ltrh /tmp/aasdfasdf', sudo=True)
>>> except (AuthenticationException, UnknownHostException, ConnectionErrorException):
>>> ... pass
```

```
>>> # Commands have started executing at this point
>>> # Exit code will probably not be available immediately
>>> print output
```

```
{'myhost1': {'exit_code': None,
             'stdout' : <generator>,
             'stderr' : <generator>,
             'cmd' : <greenlet>,
             'exception' : None,
             },
 'myhost2': {'exit_code': None,
             'stdout' : <generator>,
             'stderr' : <generator>,
             'cmd' : <greenlet>,
             'exception' : None,
}}
```

Enabling host logger

There is a host logger in parallel-ssh that can be enabled to show stdout *in parallel* from remote commands on hosts as it comes in.

This allows for stdout to be automatically displayed without having to print it serially per host.

```
>>> import pssh.utils
>>> pssh.utils.enable_host_logger()
>>> output = client.run_command('ls -ltrh')
[myhost1]     drwxrwxr-x 6 user group 4.0K Jan 1 HH:MM x
[myhost2]     drwxrwxr-x 6 user group 4.0K Jan 1 HH:MM x
```

Retrieve exit codes after commands have finished as below. This is only necessary for long running commands that do not exit immediately.

`exit_code` in `output` will be `None` if command has not finished.

`get_exit_codes` is not a blocking function and will not wait for commands to finish. Use `client.pool.join()` to block until all commands have finished.

`output` parameter is modified in-place.

```
>>> client.get_exit_codes(output)
>>> for host in output:
>>> ... print output[host]['exit_code']
0
0
```

Print stdout serially per host as it becomes available.

```
>>> for host in output: for line in output[host]['stdout']: print line
[myhost1]     ls: cannot access /tmp/aasdfasdf: No such file or directory
[myhost2]     ls: cannot access /tmp/aasdfasdf: No such file or directory
```

Example with specified private key

```
>>> import paramiko
>>> client_key = paramiko.RSAKey.from_private_key_file('user.key')
>>> client = ParallelSSHClient(['myhost1', 'myhost2'], pkey=client_key)
```

Note: Connection persistence

Connections to hosts will remain established for the duration of the object's life. To close them, just *del* or reuse the object reference.

```
>>> client = ParallelSSHClient(['localhost'])
>>> output = client.run_command('ls -ltrh /tmp/aasdfasdf')
>>> client.pool.join()
```

netstat `tcp 0 0 127.0.0.1:53054 127.0.0.1:22 ESTABLISHED`

Connection remains active after commands have finished executing. Any additional commands will use the same connection.

```
>>> del client
```

Connection is terminated.

---

copy_file(*local_file*, *remote_file*)
Copy local file to remote file in parallel

Parameters

- **local_file** (*str*) – Local filepath to copy to remote host
- **remote_file** (*str*) – Remote filepath on remote host to copy file to

---

Note: Remote directories in *remote_file* that do not exist will be created as long as permissions allow.

Note: Path separation is handled client side so it is possible to copy to/from hosts with differing path separators, like from/to Linux and Windows.

Return type List(`gevent.Greenlet`) of greenlets for remote copy commands

exec_command(**args*, ***kwargs*)
Run command on all hosts in parallel, honoring *self.pool_size*

Deprecated by *pssh.pssh_client.ParallelSSHClient.run_command*

Parameters

- **args** (*tuple*) – Position arguments for command
- **kwargs** (*dict*) – Keyword arguments for command

Return type List of `gevent.Greenlet`

get_exit_code(*host_output*)
Get exit code from host output *if available*.

Parameters **host_output** – Per host output as returned by *pssh.pssh_client.ParallelSSHClient.get_output*

Return type int or None if exit code not ready

get_exit_codes(*output*)

Get exit code for all hosts in output *if available*. Output parameter is modified in-place.

Parameters **output** – As returned by *pssh.pssh_client.ParallelSSHClient.get_output*

Return type None

get_output(*cmd*, *output*)

Get output from command.

Parameters

- **cmd** (gevent.Greenlet) – Command to get output from
- **output** (*dict*) – Dictionary containing output to be updated with output from cmd

Return type None

output parameter is modified in-place and has the following structure

```
{'myhost1': {'exit_code': exit code if ready else None,
             'channel' : SSH channel of command,
             'stdout'  : <iterable>,
             'stderr'  : <iterable>,
             'cmd'     : <greenlet>,
             'exception' : <exception object if applicable>}}
```

Stdout and stderr are also logged via the logger named `host_logger` which can be enabled by calling `enable_host_logger`

Example usage:

```
>>> output = client.get_output()
>>> for host in output: for line in output[host]['stdout']: print line
<stdout>
>>> # Get exit code after command has finished
>>> self.get_exit_code(output[host])
0
```

get_stdout(*greenlet*, *return_buffers=False*)

Get/print stdout from greenlet and return exit code for host

Deprecated - use *pssh.pssh_client.ParallelSSHClient.get_output* instead.

Parameters

- **greenlet** (gevent.Greenlet) – Greenlet object containing an SSH channel reference, hostname, stdout and stderr buffers
- **return_buffers** (*bool*) – Flag to turn on returning stdout and stderr buffers along with exit code. Defaults to off.

Return type Dictionary containing `{host: {'exit_code': exit code}}` entry for example `{'myhost1': {'exit_code': 0}}`

Return type With `return_buffers=True`: `{'myhost1': {'exit_code': 0, 'channel' : None or SSH channel of command if command is still executing, 'stdout' : <iterable>, 'stderr' : <iterable>,}}`

run_command(**args*, ***kwargs*)

Run command on all hosts in parallel, honoring self.pool_size, and return output buffers.

This function will block until all commands have **started** and then return immediately. Any connection and/or authentication exceptions will be raised here and need catching.

Parameters

- **args** (*tuple*) – Positional arguments for command
- **sudo** (*bool*) – (Optional) Run with sudo. Defaults to False
- **stop_on_errors** (*bool*) – (Optional) Raise exception on errors running command. Defaults to True. With stop_on_errors set to False, exceptions are instead added to output of *run_command*. See example usage below.
- **kwargs** (*dict*) – Keyword arguments for command

Return type Dictionary with host as key as per *pssh.pssh_client.ParallelSSHClient.get_output*

Raises *pssh.exceptions.AuthenticationException* on authentication error

Raises *pssh.exceptions.UnknownHostException* on DNS resolution error

Raises *pssh.exceptions.ConnectionErrorException* on error connecting

Raises *pssh.exceptions.SSHException* on other undefined SSH errors

Example Usage

Simple run command

```
>>> output = client.run_command('ls -ltrh')
```

print stdout for each command

```
>>> for host in output:
>>>     for line in output[host]['stdout']: print line
```

Get exit codes after command has finished

```
>>> client.get_exit_codes(output)
>>> for host in output:
>>> ... print output[host]['exit_code']
0
0
```

Wait for completion, no stdout

```
>>> client.pool.join()
```

Run with sudo

```
>>> output = client.run_command('ls -ltrh', sudo=True)
```

Capture stdout - **WARNING** - this will store the entirety of stdout into memory and may exhaust available memory if command output is large enough:

```
>>> for host in output:
>>>     stdout = list(output[host]['stdout'])
>>>     print "Complete stdout for host %s is %s" % (host, stdout,)
```

Example Output

```
{'myhost1': {'exit_code': exit code if ready else None,
             'channel' : SSH channel of command,
             'stdout'  : <iterable>,
             'stderr'  : <iterable>,
             'cmd'     : <greenlet>},
             'exception' : None}
```

Do not stop on errors, return per-host exceptions in output

```
>>> output = client.run_command('ls -ltrh', stop_on_errors=False)
>>> client.pool.join()
>>> print output
```

```
{'myhost1': {'exit_code': None,
             'channel' : None,
             'stdout'  : None,
             'stderr'  : None,
             'cmd'     : None,
             'exception' : ConnectionErrorException(
                 "Error connecting to host '%s:%s' - %s - retry %s/%s",
                 host, port, 'Connection refused', 3, 3)}}
```

CHAPTER 2

SSHClient API Documentation

Package containing SSHClient class.

class pssh.ssh_client.**SSHClient**(*host*, *user=None*, *password=None*, *port=None*, *pkey=None*, *forward_ssh_agent=True*, *num_retries=3*, *_agent=None*, *timeout=10*, *proxy_host=None*, *proxy_port=22*)

Wrapper class over paramiko.SSHClient with sane defaults Honours ~/.ssh/config and /etc/ssh/ssh_config entries for host username overrides

Connect to host honouring any user set configuration in ~/.ssh/config or /etc/ssh/ssh_config

Parameters

- **host** (*str*) – Hostname to connect to
- **user** (*str*) – (Optional) User to login as. Defaults to logged in user or user from ~/.ssh/config if set
- **password** (*str*) – (Optional) Password to use for login. Defaults to no password
- **port** (*int*) – (Optional) Port number to use for SSH connection. Defaults to None which uses SSH default
- **pkey** (paramiko.PKey) – (Optional) Client's private key to be used to connect with
- **num_retries** (*int*) – (Optional) Number of retries for connection attempts before the client gives up. Defaults to 3.
- **timeout** (*int*) – (Optional) Number of seconds to timeout connection attempts before the client gives up. Defaults to 10.
- **forward_ssh_agent** (*bool*) – (Optional) Turn on SSH agent forwarding - equivalent to *ssh -A* from the *ssh* command line utility. Defaults to True if not set.
- **_agent** (paramiko.agent.Agent) – (Optional) Override SSH agent object with the provided. This allows for overriding of the default paramiko behaviour of connecting to local SSH agent to lookup keys with our own SSH agent. Only really useful for testing, hence the internal variable prefix.
- **proxy_host** (*str*) – (Optional) SSH host to tunnel connection through so that SSH clients connects to self.host via client -> proxy_host -> host
- **proxy_port** (*int*) – (Optional) SSH port to use to login to proxy host if set. Defaults to 22.

copy_file(*local_file*, *remote_file*)

Copy local file to host via SFTP/SCP

Copy is done natively using SFTP/SCP version 2 protocol, no scp command is used or required.

Parameters

- **local_file** (*str*) – Local filepath to copy to remote host
- **remote_file** (*str*) – Remote filepath on remote host to copy file to

exec_command(*command*, *sudo=False*, *user=None*, ***kwargs*)
Wrapper to `paramiko.SSHClient.exec_command`

Opens a new SSH session with a new pty and runs command with given *kwargs* if any. Greenlet then yields (sleeps) while waiting for command to finish executing or channel to close indicating the same.

Parameters

- **command** (*str*) – Shell command to execute
- **sudo** (*bool*) – (Optional) Run with sudo. Defaults to False
- **kwargs** (*dict*) – (Optional) Keyword arguments to be passed to remote command

Return type Tuple of *(channel, hostname, stdout, stderr)*. Channel is the remote SSH channel, needed to ensure all of stdout has been got, hostname is remote hostname the copy is to, stdout and stderr are buffers containing command output.

mkdir(*sftp*, *directory*)
Make directory via SFTP channel

Parameters

- **sftp** (`paramiko.SFTPClient`) – SFTP client object
- **directory** (*str*) – Remote directory to create

Catches and logs at error level remote IOErrors on creating directory.

CHAPTER 3

Exceptions

Exceptions raised by parallel-ssh classes.

exception pssh.exceptions.**AuthenticationException**
Raised on authentication error (user/password/ssh key error)

exception pssh.exceptions.**ConnectionErrorException**
Raised on error connecting (connection refused/timed out)

exception pssh.exceptions.**SSHException**
Raised on SSHException error - error authenticating with SSH server

exception pssh.exceptions.**UnknownHostException**
Raised when a host is unknown (dns failure)

Welcome to ParallelSSH's API documentation.

New users should start with *pssh.pssh_client.ParallelSSHClient* and in particular *pssh.pssh_client.ParallelSSHClient.run_command*.

CHAPTER 4

Indices and tables

- genindex

p

A

C

E

G

M

P

R

S

U