Parallel-SSH Documentation

Release 0+unknown

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*, *proxy_user=None*, *proxy_password=None*, *proxy_pkey=None*, *agent=None*, *allow_agent=True*, *host_config=None*, *channel_timeout=None*)

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 10. Values over 500 are not likely to increase performance due to overhead in the single thread running our event loop.
- **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.

- **proxy_user** (*str*) – (Optional) User to login to `proxy_host` as. Defaults to logged in user.
- **proxy_password** (*str*) – (Optional) Password to login to `proxy_host` with. Defaults to no password
- **proxy_pkey** (`paramiko.PKey`) – (Optional) Private key to be used for authentication with `proxy_host`. Defaults to available keys from SSHAgent and user's home directory keys
- **agent** (`pssh.agent.SSHAgent`) – (Optional) SSH agent object to programmatically supply an agent to override system SSH agent with
- **host_config** (*dict*) – (Optional) Per-host configuration for cases where not all hosts use the same configuration values.
- **channel_timeout** (*int*) – (Optional) Time in seconds before an SSH operation times out.
- **allow_agent** (*bool*) – (Optional) set to False to disable connecting to the SSH agent

Example Usage

```
from pssh.pssh_client import ParallelSSHClient
from pssh.exceptions import AuthenticationException, \
  UnknownHostException, ConnectionErrorException

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(s) will 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 from remote commands on hosts as it comes in.

This allows for stdout to be automatically logged 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.

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

parallel-ssh starts commands asynchronously to enable starting multiple commands in parallel without blocking.

Because of this, exit codes will not be immediately available even for commands that exit immediately.

Waiting for command completion

At least one of

- Iterating over stdout/stderr
- Calling `client.join(output)`

is necessary to cause *parallel-ssh* to wait for commands to finish and be able to gather exit codes.

Note: Joining on client pool

`client.pool.join()` only blocks *until greenlets have started executing* which will be immediately as long as pool is not full.

Checking command completion

To check if commands have finished *without blocking* use

```
client.finished(output)
False
```

which returns `True` if and only if all commands in output have finished.

For individual commands the status of channel can be checked

```
output[host]['channel'].closed
False
```

which returns `True` if command has finished.

Either iterating over stdout/stderr or *client.join(output)* will cause exit codes to become available in output without explicitly calling *get_exit_codes*.

Use `client.join(output)` to block until all commands have finished and gather exit codes at same time.

Exit code retrieval

`get_exit_codes` is not a blocking function and will not wait for commands to finish.

`output` parameter is modified in-place.

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

Stdout from each 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)
```

Multiple commands

```
for cmd in ['uname', 'whoami']:
    client.run_command(cmd)
```

Per-Host configuration

Per host configuration can be provided for any or all of user, password port and private key. Private key value is a `paramiko.PKey` object as returned by `pssh.utils.load_private_key`.

`pssh.utils.load_private_key` accepts both file names and file-like objects and will attempt to load all available key types, returning *None* if they all fail.

```
from pssh.utils import load_private_key
host_config = { 'host1' : {'user': 'user1', 'password': 'pass',
                           'port': 2222,
                           'private_key': load_private_key('my_key.pem')},
                'host2' : {'user': 'user2', 'password': 'pass',
                           'port': 2223,
                           'private_key': load_private_key(open('my_other_key.pem'))},
                }
hosts = host_config.keys()
client = ParallelSSHClient(hosts, host_config=host_config)
client.run_command('uname')
```

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.join(output)
```

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*, *recurse=False*)

Copy local file to remote file in parallel

This function returns a list of greenlets which can be 'join'ed on to wait for completion.

Use *.get* on each greenlet to raise any exceptions from them.

Exceptions listed here are raised when *.get* is called on each greenlet, not this function itself.

Parameters

- **local_file** (*str*) – Local filepath to copy to remote host
- **remote_file** (*str*) – Remote filepath on remote host to copy file to
- **recurse** (*bool*) – Whether or not to descend into directories recursively.

Raises `ValueError` when a directory is supplied to local_file and recurse is not set

Raises `IOError` on I/O errors writing files

Raises `OSError` on OS errors like permission denied

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

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`

finished(*output*)
Check if commands have finished without blocking

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

Return type bool

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>,}}`

join(*output*)

Block until all remote commands in output have finished and retrieve exit codes

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 been *sent* to remote servers and then return immediately.

Any connection and/or authentication exceptions will be raised here and need catching *unless run_command* is called with *stop_on_errors=False*.

Parameters

- **args** (*tuple*) – Positional arguments for command
- **sudo** (*bool*) – (Optional) Run with sudo. Defaults to False
- **user** (*str*) – (Optional) User to run command as. Requires sudo access for that user from the logged in user account.

- **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.
- **shell** (*str*) – (Optional) Override shell to use to run command with. Defaults to logged in user's defined shell. Use the shell's command syntax, eg *shell='bash -c'* or *shell='zsh -c'*.
- **use_shell** (*bool*) – (Optional) Run command with or without shell. Defaults to True - use shell defined in user login to run command string
- **host_args** (*tuple or list*) – (Optional) Format command string with per-host arguments in `host_args`. `host_args` length must equal length of host list - *pssh.exceptions.HostArgumentException* is raised otherwise

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

Raises *pssh.exceptions.HostArgumentException* on number of host arguments not equal to number of hosts

Raises *TypeError* on not enough host arguments for cmd string format

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, update output with exit codes

```
client.join(output)
print(output[host]['exit_code'])
0
for line in output[host]['stdout']:
    print(line)
```

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.

Iterating over stdout/stderr by definition implies blocking until command has finished. To only see output as it comes in without blocking the host logger can be enabled - see *Enabling Host Logger* above.

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

Command with per-host arguments

`host_args` keyword parameter can be used to provide arguments to use to format the command string.

Number of `host_args` should be at least as many as number of hosts.

Any string format specification characters may be used in command string.

Examples

```
# Tuple
#
# First host in hosts list will use cmd 'host1_cmd',
# second host 'host2_cmd' and so on
output = client.run_command('%s', host_args=('host1_cmd',
                                             'host2_cmd',
                                             'host3_cmd',))

# Multiple arguments
#
output = client.run_command('%s %s',
                            host_args=(('host1_cmd1', 'host1_cmd2'),
                                       ('host2_cmd1', 'host2_cmd2'),
                                       ('host3_cmd1', 'host3_cmd2'),))

# List of dict
#
# Fist host in host list will use cmd 'host-index-0',
# second host 'host-index-1' and so on
output = client.run_command(
  '%(cmd)s', host_args=[{'cmd': 'host-index-%s' % (i,))
                        for i in range(len(client.hosts))]
```

Expression as host list

Any type of iterator may be used as host list, including generator and list comprehension expressions.

```
hosts = ['dc1.myhost1', 'dc2.myhost2']
# List comprehension
client = ParallelSSHClient([h for h in hosts if h.find('dc1')])
# Generator
client = ParallelSSHClient((h for h in hosts if h.find('dc1')))
# Filter
client = ParallelSSHClient(filter(lambda h: h.find('dc1'), hosts))
client.run_command(<..>)
```

Note: Since generators by design only iterate over a sequence once then stop, *client.hosts* should be re-assigned after each call to *run_command* when using iterators as target of *client.hosts*.

Overriding host list

Host list can be modified in place. Call to *run_command* will create new connections as necessary and output will only contain output for hosts command ran on.

```
client.hosts = ['otherhost']
print(client.run_command('exit 0'))
{'otherhost': {'exit_code': None}, <..>}
```

Run multiple commands in parallel

This short example demonstrates running long running commands in parallel, how long it takes for all commands to start, blocking until they complete and how long it takes for all commands to complete.

See examples directory for complete script.

```
output = []
host = 'localhost'

# Run 10 five second sleeps
cmds = ['sleep 5' for _ in xrange(10)]
start = datetime.datetime.now()
for cmd in cmds:
    output.append(client.run_command(cmd, stop_on_errors=False))
end = datetime.datetime.now()
print("Started %s commands in %s" % (len(cmds), end-start,))
start = datetime.datetime.now()
for _output in output:
    for line in _output[host]['stdout']:
        print(line)
end = datetime.datetime.now()
print("All commands finished in %s" % (end-start,))
```

Output

```
Started 10 commands in 0:00:00.428629
All commands finished in 0:00:05.014757
```

Output dictionary

```
{'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.join(output)
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)}}
```

Using stdin

```
output = client.run_command('read')
stdin = output['localhost']['stdin']
stdin.write("writing to stdin\n")
stdin.flush()
for line in output['localhost']['stdout']:
    print(line)

writing to stdin
```

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*, *allow_agent=True*, *timeout=10*, *proxy_host=None*, *proxy_port=22*, *proxy_user=None*, *proxy_password=None*, *proxy_pkey=None*, *channel_timeout=None*, *_openssh_config_file=None*)

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 object.
- **forward_ssh_agent** – (Optional) Turn on SSH agent forwarding - equivalent to *ssh -A* from the *ssh* command line utility. Defaults to True if not set.
- **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.

- **channel_timeout** (*int*) – (Optional) Time in seconds before an SSH operation times out.
- **allow_agent** (*bool*) – (Optional) set to False to disable connecting to the SSH agent

copy_file(*local_file*, *remote_file*, *recurse=False*)

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
- **recurse** (*bool*) – Whether or not to descend into directories recursively.

Raises `ValueError` when a directory is supplied to `local_file` and `recurse` is not set

Raises `IOError` on I/O errors writing files

Raises `OSError` on OS errors like permission denied

exec_command(*command*, *sudo=False*, *user=None*, *shell=None*, *use_shell=True*, *use_pty=True*, ***kwargs*)

Wrapper to `paramiko.SSHClient.exec_command`

Opens a new SSH session with a new pty and runs command before yielding the main gevent loop to allow other greenlets to execute.

Parameters

- **command** (*str*) – Cxommand to execute
- **sudo** (*bool*) – (Optional) Run with sudo. Defaults to False
- **user** (*str*) – (Optional) User to switch to via sudo to run command as. Defaults to user running the python process
- **shell** – (Optional) Shell override to use instead of user login configured shell. For example `shell='bash -c'`
- **use_shell** (*bool*) – (Optional) Force use of shell on/off. Defaults to *True* for on
- **use_pty** (*bool*) – (Optional) Enable/Disable use of pseudo terminal emulation. This is required in vast majority of cases, exception being where a shell is not used and stdout/stderr/stdin buffers are not required. Defaults to `True`
- **kwargs** (*dict*) – (Optional) Keyword arguments to be passed to remote command

Return type Tuple of *(channel, hostname, stdout, stderr, stdin)*. 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.

Parent paths in the directory are created if they do not exist.

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.**HostArgumentException**
Raised on errors with per-host command arguments

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