jumpssh Documentation

Release 1.6.5

Thibaud Castaing

Apr 20, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 JumpSSH

JumpSSH Python module to run commands on remote servers

Copyright Copyright (c) 2017 Amadeus sas

License MIT

Documentation https://jumpssh.readthedocs.io

Development https://github.com/AmadeusITGroup/JumpSSH

1.1.1 What

JumpSSH is a module for Python 2.7+/3.5+ that can be used to run commands on remote servers through a gateway.

It is based on paramiko library. It provides the ability to execute commands on hosts that are not directly accessible but only through one or more servers. Script does not need to be uploaded on a remote server and can be run locally.

Several authentication methods are supported (password, ssh key).

Commands can be run through several jump servers before reaching the remote server. No need to establish a session for each command, a single ssh session can run as many command as you want, including parallel queries, and you will get result for each command independently.

So, why another python library to setup remote server through ssh ? Here is a quick comparison with the most known existing p

- Paramiko: provide very good implementation of SSHv2 protocol in python but with a low level api a bit complex
- Ansible: require more configuration and understanding to start. Moreover, support of bastion host is done with modification of local ssh config to use ProxyCommand, and this is needed for each bastion host.
- Fabric: use of jump server is much easier than Ansible thanks to 'env.gateway' parameter, but does not allow jump through several servers.

1.1.2 Installation

To install JumpSSH, simply:

```
$ pip install jumpssh
```

1.1.3 Examples

establish ssh session with a remote host through a gateway:

```
>>> from jumpssh import SSHSession

# establish ssh connection between your local machine and the jump server
>>> gateway_session = SSHSession('gateway.example.com',
...                              'my_user', password='my_password').open()

# from jump server, establish connection with a remote server
>>> remote_session = gateway_session.get_remote_session('remote.example.com',
...                                                     password='my_password2')
```

run commands on remote host:

```
# command will be executed remotely and output will be returned locally and printed
>>> print(remote_session.get_cmd_output('ls -lta'))
total 28
drwxr-xr-x. 412 root    root    12288 Mar 21 14:25 ..
drwx------.   2 my_user my_user    28 Mar  6 19:25 .ssh
drwx------.   3 my_user my_user    70 Mar  6 19:25 .
-rw-r--r--.   1 my_user my_user    18 Jul 12  2016 .bash_logout
-rw-r--r--.   1 my_user my_user   193 Jul 12  2016 .bash_profile
-rw-r--r--.   1 my_user my_user   231 Jul 12  2016 .bashrc

# get exit code of the remotely executed command (here to check if a package is␣
↪installed)
>>> remote_session.get_exit_code('yum list installed package_name')
0
```

remote rest api usage:

```
# calling rest api on remote host that is only accessible from the gateway
>>> from jumpssh import RestSshClient
>>> rest_client = RestSshClient(gateway_session)

# syntax is similar to requests library (http://docs.python-requests.org)
>>> http_response = rest_client.get('http://remote.example.com/helloworld')
>>> http_response.status_code
200
>>> http_response.text
u'Hello, World!'
```

remote files operations:

```
# check if remote path exists
>>> remote_session.exists('/path/to/a/file')
True
```

(continues on next page)

(continued from previous page)

```
# copy file from local machine to remote host through gateway
>>> remote_session.put('/local/path/to/a/file', '/remote/path/to/the/file')

# create file on remote host from local content
>>> remote_session.file('/remote/path/to/the/file',
...                     content='remote file content', permissions='600')

# download remote file on local machine from remote host through gateway
>>> remote_session.get('/remote/path/to/the/file', '/local/path/')
```

1.1.4 Tests

jumpssh tests require docker, check docker documentation for how to install it depending on your OS. it also requires few python packages. To install them, run:

```
$ pip install -r requirements_dev.txt
```

To run the test suite, clone the repository and run:

```
$ pytest -sv tests/
```

or simply:

```
$ tox
```

1.1.5 Contributing

Bug Reports

Bug reports are hugely important! Before you raise one, though, please check through the GitHub issues, both open and closed, to confirm that the bug hasn't been reported before.

Feature Requests

If you think a feature is missing and could be useful in this module, feel free to raise a feature request through the GitHub issues

Code Contributions

When contributing code, please follow this project-agnostic contribution guide.

CHAPTER

TWO

API REFERENCE

2.1 jumpssh.exception

exception jumpssh.exception.**ConnectionError**(*msg*, *original_exception=None*)

Bases: *jumpssh.exception.SSHException*

Exception raised when unable to establish SSHSession with remote host

exception jumpssh.exception.**RestClientError**(*msg*, *original_exception=None*)

Bases: *jumpssh.exception.SSHException*

Exception raised when error occurs during rest ssh calls

exception jumpssh.exception.**RunCmdError**(*exit_code*, *success_exit_code*, *command*, *error*, *runs_nb=1*)

Bases: *jumpssh.exception.SSHException*

Exception raised when remote command return a non success exit code

Variables

- ***exit_code*** (*int*) – The exit code from the run command.
- **list (int)** – List of expected success exit codes for run command.
- **command** (*str*) – The command that is generating this exception.
- **error** (*str*) – The error captured from the command output.

exception jumpssh.exception.**SSHException**(*msg*, *original_exception=None*)

Bases: Exception

Generic exception for jumpssh

Allow to chain exceptions keeping track of origin exception

exception jumpssh.exception.**TimeoutError**(*msg*, *original_exception=None*)

Bases: *jumpssh.exception.SSHException*

Exception raised when remote command execution reached specified timeout

2.2 jumpssh.restclient

class jumpssh.restclient.**HTTPResponse**(*http_response_str*)

Bases: object

check_for_success()

is_valid_json_body()

json(***kwargs*)

class jumpssh.restclient.**RestSshClient**(*ssh_session=None*, ***kwargs*)

Bases: object

delete(*uri*, ***kwargs*)

Sends a DELETE request.

Parameters

- **uri** – URL of the http request.
- ****kwargs** – Optional arguments that *request()* takes.

Returns *HTTPResponse* object

Return type *restclient.HTTPResponse*

get(*uri*, ***kwargs*)

Sends a GET request.

Parameters

- **uri** – URL of the http request.
- ****kwargs** – Optional arguments that *request()* takes.

Returns *HTTPResponse* object

Return type *restclient.HTTPResponse*

head(*uri*, ***kwargs*)

Sends a HEAD request.

Parameters

- **uri** – URL of the http request.
- ****kwargs** – Optional arguments that *request()* takes.

Returns *HTTPResponse* object

Return type *restclient.HTTPResponse*

options(*uri*, ***kwargs*)

Sends a OPTIONS request.

Parameters

- **uri** – URL of the http request.
- ****kwargs** – Optional arguments that *request()* takes.

Returns *HTTPResponse* object

Return type *restclient.HTTPResponse*

patch(*uri*, ***kwargs*)

Sends a PATCH request.

Parameters

- **uri** – URL of the http request.
- ****kwargs** – Optional arguments that *request()* takes.

Returns *HTTPResponse* object

Return type *restclient.HTTPResponse*

post (*uri*, ***kwargs*)
Sends a POST request.

Parameters

- **uri** – URL of the http request.
- ****kwargs** – Optional arguments that *request()* takes.

Returns *HTTPResponse* object

Return type *restclient.HTTPResponse*

put (*uri*, ***kwargs*)
Sends a PUT request.

Parameters

- **uri** – URL of the http request.
- ****kwargs** – Optional arguments that *request()* takes.

Returns *HTTPResponse* object

Return type *restclient.HTTPResponse*

request (*method*, *uri*, ***kwargs*)
Perform http request and send back http response.

Parameters

- **method** – http method.
- **uri** – remote URL to target.
- **params** – (optional) Dictionary to be sent in the query string.
- **data** – (optional) Content to send in the body of the http request.
- **headers** – (optional) Dictionary of HTTP Headers to send with the http request.
- **remote_file** – (optional) File on the remote host with content to send in the body of the http request.
- **local_file** – (optional) Local file with content to send in the body of the http request.
- **document_info_only** – (optional) if True, only HTTP Headers are returned in http response (default=False).
- **auth** – (optional) Auth tuple to enable Basic/Digest/Custom HTTP Auth.
- **verify** – (optional) whether the SSL cert will be verified.
- **silent** – if True, does not log the command run (useful if sensitive information are used in command)

Returns *HTTPResponse* object

Return type *restclient.HTTPResponse*

Usage:

```
>>> from jumpssh import RestSshClient
>>> with RestSshClient(host='gateway.example.com', username='my_user') as␣
↪rest_client:
>>> ... http_response = rest_client.request('GET', 'http://remote.example.com
↪')
>>> ... http_response.status_code
200
```

2.3 jumpssh.session

class `jumpssh.session.`**`RunCmdResult`**(*exit_code*, *output*, *result_list*, *command*, *success_exit_code*, *runs_nb*)

Bases: *jumpssh.session.RunSSHCmdResult*

Result of a command run with SSHSession

Parameters

- **exit_code** – exit code of the run command (last run exit_code in case of retries)
- **output** – output of the command run (last run output only in case of retries)
- **command** – the command run
- **result_list** – list of RunSSHCmdResult, 1 item for each retry
- **success_exit_code** – list of integer considered as a success exit code for command run
- **runs_nb** – number of times the command has been run

Usage:

```
>>> result = ssh_session.run_cmd('hostname')

# access to both exit_code and command output using tuple
>>> (exit_code, output) = result

# access directly to single attributes
>>> result.exit_code
0

>>> result.output
u'gateway.example.com'

>>> result.command
'hostname'
```

class `jumpssh.session.`**`RunSSHCmdResult`**(*exit_code*, *output*)

Bases: `tuple`

property exit_code

Alias for field number 0

property output

Alias for field number 1

class jumpssh.session.**SSHSession**(*host, username, proxy_transport=None, private_key_file=None, port=22, password=None, missing_host_key_policy=None, compress=False, timeout=None, **kwargs*)

Bases: `object`

Establish SSH session with a remote host

Parameters

- **host** – name or ip of the remote host
- **username** – user to be used for remote ssh session
- **proxy_transport** – `paramiko.transport.Transport` object for an SSH connection used to establish ssh session between 2 remotes hosts
- **private_key_file** – local path to a private key file to use if key needed for authentication and not present in standard path (~/.ssh/)
- **port** – port to connect to the remote host (default 22)
- **password** – password to be used for authentication with remote host
- **missing_host_key_policy** – set policy to use when connecting to servers without a known host key. This parameter is a class **instance** of type `paramiko.client.MissingHostKeyPolicy`, not a **class** itself
- **compress** – set to True to turn on compression for this session
- **timeout** – optional timeout opening SSH session, default 3600s (1h)
- ****kwargs** – any parameter taken by `paramiko.client.SSHClient.connect` and not already explicitly covered by *SSHSession*

Usage:

```
>>> from jumpssh import SSHSession
>>> gateway_session = SSHSession('gateway.example.com', 'my_user', password='my_
↪password')
```

close()

Close connection with remote host

Usage::

```
>>> from jumpssh import SSHSession
>>> ssh_session = SSHSession('gateway.example.com', 'my_user', password=
↪'my_password').open()
>>> ssh_session.is_active()
True
>>> ssh_session.close()
>>> ssh_session.is_active()
False
```

exists(*path, use_sudo=False*)

Check if path exists on the remote host

Parameters

- **path** – remote path to check for existence
- **use_sudo** – if True, allow to check path current user doesn't have access by default

Returns True, if specified *path* exists on the remote host else False

Return type bool

Usage::

```
>>> with SSHSession('gateway.example.com', 'my_user', password='my_
↪password') as ssh_session:
>>> ... ssh_session.exists('/path/to/remote/file')
False
>>> ... ssh_session.exists('/home/other_user/.ssh', use_sudo=True)
True
```

file (*remote_path*, *content*, *use_sudo=False*, *owner=None*, *permissions=None*, *username=None*, *silent=False*)
Method to create a remote file with the specified *content*

Parameters

- **remote_path** – destination folder in which to copy the local file
- **content** – content of the file
- **use_sudo** – allow to copy file in location with restricted permissions
- **owner** – user that will own the file on the remote host
- **permissions** – permissions to apply on the remote file (chmod format)
- **username** – sudo user
- **silent** – disable logging

Usage:

```
# create file on remote host and with specified content at the specified path
>>> ssh_session.file(remote_path='/path/to/remote/file', content='file content
↪')

# create file on remote host and with specified content at the specified path␣
↪needing sudo permissions
>>> ssh_session.file(remote_path='/path/to/remote/file', content='file content
↪', use_sudo=True)

# create file on remote host and with specified content at the specified path
# with specified owner and permissions
>>> ssh_session.file(remote_path='/path/to/remote/file', content='file content
↪',
...                  owner='other_user', permissions='700')
```

get (*remote_path*, *local_path*, *use_sudo=False*, *username=None*)
Download a file from the remote host

Parameters

- **remote_path** – remote path of the file to download
- **local_path** – local path where to download the file
- **use_sudo** – allow to download a file from a location current user does not have access
- **username** – sudo user

Usage:

```
# download remote file in local directory
>>> ssh_session.get(remote_path='/path/to/remote/file', local_path='/local/
↪folder')

# donload remote file from a path not accessible by current user
>>> ssh_session.get(local_path='/path/to/local/file', remote_path='/path/to/
↪remote/file', use_sudo=True)
```

get_cmd_output(*cmd*, ***kwargs*)

Return output of remotely executed command

Support same parameters than *run_cmd* method

Parameters **cmd** – remote command to execute

Returns output of remotely executed command

Return type str

Usage::

```
>>> from jumpssh import SSHSession
>>> with SSHSession('gateway.example.com', 'my_user', password='my_
↪password') as ssh_session:
>>> ... ssh_session.get_cmd_output('hostname'))
u'gateway.example.com'
```

get_exit_code(*cmd*, ***kwargs*)

Return exit code of remotely executed command

Support same parameters than *run_cmd* method

Parameters **cmd** – remote command to execute

Returns exit code of remotely executed command

Return type int

Usage::

```
>>> from jumpssh import SSHSession
>>> with SSHSession('gateway.example.com', 'my_user', password='my_
↪password') as ssh_session:
>>> ... ssh_session.get_exit_code('ls')
0
>>> ... ssh_session.get_exit_code('dummy_command')
127
```

get_remote_session(*host*, *username=None*, *retry=0*, *private_key_file=None*, *port=22*, *password=None*, *retry_interval=10*, *compress=False*, *timeout=None*, ***kwargs*)

Establish connection with a remote host from current session

Parameters

- **host** – name or ip of the remote host
- **username** – user to be used for remote ssh session
- **retry** – retry number to establish connection with remote host (-1 for infinite retry)

- **private_key_file** – local path to a private key file to use if key needed for authentication
- **port** – port to connect to the remote host (default 22)
- **password** – password to be used for authentication with remote host
- **retry_interval** – number of seconds between each retry
- **compress** – set to True to turn on compression for this session
- **timeout** – optional timeout opening remote session, default 3600s (1h)
- ****kwargs** – any parameter taken by `paramiko.client.SSHClient.connect` and not already explicitly covered by *SSHSession*

Returns session object of the remote host

Return type *SSHSession*

Usage:

```
# open session with remote host
>>> from jumpssh import SSHSession
>>> ssh_session = SSHSession('gateway.example.com', 'my_user', password='my_
↪password').open()

# get remote session using same user than current session and same␣
↪authentication method
>>> remote_session = ssh_session.get_remote_session('remote.example.com')

# get remote session with specific user and password
>>> remote_session = ssh_session.get_remote_session('remote.example.com',
...                                                  username='other_user',
...                                                  password='other_user_
↪password')

# retry indefinitely to connect to remote host until success
>>> remote_session = ssh_session.get_remote_session('remote.example.com',␣
↪retry=-1)
```

get_sftp_client()

See documentation for available methods on paramiko.sftp_client at : http://docs.paramiko.org/en/latest/api/sftp.html

Returns paramiko SFTP client object.

Return type paramiko.sftp_client.SFTPClient

Usage:: # open session with remote host >>> from jumpssh import SSHSession >>> ssh_session = SSHSession('gateway.example.com', 'my_user', password='my_password').open()

# get sftp client >>> sftp_client = ssh_session.get_sftp_client()

is_active()

Check if connection with remote host is still active

An inactive SSHSession cannot run command on remote host

Returns True if current session is still active, else False

Return type bool

Usage::

```
>>> from jumpssh import SSHSession
>>> with SSHSession('gateway.example.com', 'my_user', password='my_
↪password') as ssh_session:
>>> ... ssh_session.is_active()
True
>>> ssh_session.is_active()
False
```

open (*retry=0*, *retry_interval=10*)
Open session with the remote host

Parameters

- **retry** – number of retry to establish connection with remote host (-1 for infinite retry)
- **retry_interval** – number of seconds between each retry

Returns same SSHSession opened

Usage::

```
>>> from jumpssh import SSHSession
>>> ssh_session = SSHSession('gateway.example.com', 'my_user', password=
↪'my_password').open()
>>> ssh_session.is_active()
True
```

put (*local_path*, *remote_path*, *use_sudo=False*, *owner=None*, *permissions=None*, *username=None*)
Upload a file to the remote host

Parameters

- **local_path** – path of the local file to upload
- **remote_path** – destination folder in which to upload the local file
- **use_sudo** – allow to upload a file in location with restricted permissions
- **owner** – user that will own the copied file on the remote host syntax : *user:group* or simply *user* if same than group
- **permissions** – permissions to apply on the remote file (chmod format)
- **username** – sudo user

Raises IOError – if local file *local_path* does not exist

Usage:

```
# copy local file on remote host
>>> ssh_session.put(local_path='/path/to/local/file', remote_path='/path/to/
↪remote/file')

# copy local file on remote host in a remote path needing sudo permission
>>> ssh_session.put(local_path='/path/to/local/file', remote_path='/path/to/
↪remote/file', use_sudo=True)

# copy local file on remote host with specific owner and permissions
```

(continues on next page)

(continued from previous page)

```
>>> ssh_session.put(local_path='/path/to/local/file', remote_path='/path/to/
↪remote/file',
...                 owner='root', permissions='600')
```

run_cmd(*cmd, username=None, raise_if_error=True, continuous_output=False, silent=False, timeout=None, input_data=None, success_exit_code=0, retry=0, retry_interval=5, keep_retry_history=False*)

Run command on the remote host and return result locally

Parameters

- **cmd** – command to execute on remote host cmd can be a str or a list of str
- **username** – user used to execute the command (sudo privilege needed)
- **raise_if_error** – if True, raise SSHException when exit code of the command is different from 0 else just return exit code and command output
- **continuous_output** – if True, print output all along the command is running
- **silent** – if True, does not log the command run (useful if sensitive information are used in command) if parameter is a list, all strings of the command matching an item of the list will be concealed in logs (regexp supported)
- **timeout** – length in seconds after what a TimeoutError exception is raised
- **input_data** – key/value dictionary used when remote command expects input from user when key is matching command output, value is sent
- **success_exit_code** – integer or list of integer considered as a success exit code for command run
- **retry** – number of retry until exit code is part of successful exit code list (-1 for infinite retry) or RunCmdError exception is raised
- **retry_interval** – number of seconds between each retry
- **keep_retry_history** – if True, all retries results are kept and accessible in return result default is False as we don't want to save by default all output for all retries especially for big output

Raises

- ***TimeoutError*** – if command run longer than the specified timeout
- **TypeError** – if *cmd* parameter is neither a string neither a list of string
- ***SSHException*** – if current SSHSession is already closed
- ***RunCmdError*** – if exit code of the command is different from 0 and raise_if_error is True

Returns a class inheriting from collections.namedtuple containing mainly *exit_code* and *output* of the remotely executed command

Return type *RunCmdResult*

Usage::

```
>>> from jumpssh import SSHSession
>>> with SSHSession('gateway.example.com', 'my_user', password='my_
↪password') as ssh_session:
```

(continues on next page)

(continued from previous page)

```
>>> ...     ssh_session.run_cmd('hostname')
RunSSHCmdResult(exit_code=0, output=u'gateway.example.com')
```

2.4 jumpssh.util

Useful functions used by the rest of jumpssh.

`jumpssh.util.`**`id_generator`**(*size=6*, *chars='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345678*

Generate random string with specified size and set of characters

Parameters

- **`size`** – length of the expected string
- **`chars`** – expected characters in the string

Returns random string

`jumpssh.util.`**`yes_no_query`**(*question*, *default=None*, *interrupt=None*)

Ask a yes/no question via standard input and return a boolean answer.

If default is given, it is used if the user input is empty. If interrupt is given, it is used if the user presses Ctrl-C. An EOF is treated as the default answer. If there is no default, an exception is raised to prevent infinite loops. Valid answers are: y/yes/n/no (match is not case sensitive). If invalid input is given, the user will be asked until they actually give valid input.

Parameters

- **`question`** – A question that is presented to the user.
- **`default`** – The default value when enter is pressed with no value. When None, there is no default value and the query will loop.
- **`interrupt`** – The default value when the user presses Ctrl-C

Returns A bool indicating whether user has entered yes or no.

Return type bool

CHAPTER

THREE

CHANGES

3.1 1.6.5 (11/03/2020)

- [Bug] #152: Remove pkg_info.json file and replace it with python file to avoid access issue at runtime
- [Improvement] add python 3.9 validation

3.2 1.6.4 (08/24/2020)

- [Bug] #109: Fix automated session closure handled by python garbage collection
- [Bug] #120: Fix get_remote_session not respecting 'timeout' parameter
- [Bug] #139: Fix run_cmd raising AuthenticationException if no agent is running
- [Improvement][Tests]: use flaky package to automatically rerun flaky tests

3.3 1.6.3 (03/12/2020)

- [Improvement]: remove pytest-runner from setup_requires as this is deprecated for security reasons, see https://github.com/pytest-dev/pytest-runner
- [Improvement]: use only fixed test dependencies in requirements_dev.txt

3.4 1.6.1 (04/08/2019)

- [Bug] #51: 'get' file was failing if the remote file is binary. Thanks to @pshaobow for the report.
- [Feature]: Ability to use any parameter of *paramiko.client.SSHClient.connect* in *get_remote_session*, was forgotten during implementation of #43.
- [Improvement]: tests migrated to docker-compose to setup docker environment

3.5 1.5.1 (01/14/2019)

- [Feature] #43: Ability to use any parameter of paramiko.client.SSHClient.connect in SSHSession.

3.6 1.4.1 (03/31/2018)

- [Bug] #33: Fix download of file owned by root with *SSHSession.get*
- [Bug] : Automatically open closed session when calling SSHSession.put. Thanks to @fmaupas for the fix.

3.7 1.4.0 (01/29/2018)

- [Feature] #29: Expose compression support from Paramiko (inherited from SSH). Thanks to @fmaupas for the contribution.

3.8 1.3.2 (12/17/2017)

- [Bug] #23: do not print *byte* but *str* in continuous output when running command with python3. Thanks to @nicholasbishop for the report.

3.9 1.3.1 (09/15/2017)

- fix interruption of remote command when transport channel is already closed

3.10 1.3.0 (09/14/2017)

- allow to conceal part of the command run in logs specifying list of pattern in silent parameter (regexp format) For example, if a password is specified in command you may want to conceal it in logs but still want to log the rest of the command run
- ability to customize success exit code when calling run_cmd so that an exit code different from 0 do not raise any exception. Success exit code can be an int or even a list of int if several exit codes are considered a success.
- ability to retry remote command until success or max retry is reached
- ability to forward Ctrl-C to remote host in order to interrupt remote command before stopping local script

3.11 1.2.1 (07/27/2017)

- reduce logging level of some logs
- propagate missing 'silent' parameter in restclient module to run_cmd to control logging

3.12 1.2.0 (07/24/2017)

- automatically open inactive session when running command on it
- automatically open inactive jump session when requesting remote session

3.13 1.1.0 (07/20/2017)

- Each ssh session can be used as a jump server to access multiple remote sessions in parallel. Only 1 remote session per jump server was allowed before.
- ability to customize retry interval when opening a ssh session

3.14 1.0.2 (07/14/2017)

- Fix run of shell builtins commands (source, . . .) when impersonating another user as they cannot be executed without the shell and by default, sudo do not run shell

3.15 1.0.1 (06/11/2017)

- Fix BadHostKeyException raised by paramiko when reusing same ssh session object to connect to a different remote host having same IP than previous host (just TCP port is different)

3.16 1.0.0 (05/24/2017)

- First release

Here are the licenses applicable to the use of the jumpssh library.

CHAPTER

FOUR

LICENSE

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX