Condoor Documentation

Release 1.0.17

Klaudiusz Staniek

Jun 27, 2018

Contents

Condoor is a pure Python module providing the telnet and ssh connectivity to the Cisco devices. It supports multiple jumphosts to reach the target device.

CHAPTER 1

Contents

1.1 Overview

The name is taken from conglomerate of two words: connection and door. Condoor provides an easy way to connect to Cisco devices over SSH and Telnet. It provides the connection door to the Cisco devices using standard Telnet and/or SSH protocol.

Condoor supports various software platforms including:

- Cisco IOS,
- Cisco IOS XE,
- Cisco IOS XR,
- Cisco IOS XR 64 bits,
- Cisco IOS XRv,
- Cisco NX-OS.

Condoor automatically adapts to different configuration modes and shells, i.e. XR Classic Admin mode, XR 64 bits Calvados Admin mode or Windriver Linux when connecting to the Line Cards.

Here is the command line which installs together with Condoor module:

```
$ condoor --help
Usage: condoor [OPTIONS]

Options:
  --url URL                       The connection url to the host (i.e.
                                  telnet://user:pass@hostname). The --url
                                  option can be repeated to define multiple
                                  jumphost urls. If no --url option provided
                                  the CONDOOR_URLS environment variable is
                                  used.  [required]
  --log-path PATH                 The logging path. If no path specified
```

(continues on next page)

(continued from previous page)

```
                                condoor logs are sent to stdout and session
                                logs are sent to stderr.
  --log-level [NONE|DEBUG|INFO|ERROR]
                                Logging level.  [default: ERROR]
  --log-session                 Log terminal session.
  --force-discovery             Force full device discovery.
  --cmd TEXT                    The command to be send to the device. The
                                --cmd option can be repeated multiple times.
  --print-info                  Print the discovered information.
  --help                        Show this message and exit.
```

1.2 Installation

Condoor is on PyPI, and can be installed with standard tools:

```
pip install condoor
```

Or:

```
easy_install condoor
```

1.2.1 Requirements

This version of Condoor requires Python 2.7.

1.3 API documentation

1.3.1 Core condoor components

Init file for condoor.

Connection class

class Connection(*name*, *urls=[]*, *log_dir=None*, *log_level=10*, *log_session=True*)

Connection class providing the condoor API.

Main API class providing the basic API to the physical devices. It implements the following methods:

- connect
- reconnect
- disconnect
- send
- reload
- enable
- run_fsm

__init__(*name*, *urls=[]*, *log_dir=None*, *log_level=10*, *log_session=True*)
Initialize the *condoor.Connection* object.

Args: name (str): The connection name.

urls (str or list): This argument may be a string or list of strings or list of list of strings. When **urls** type is string it must be valid URL in the following format:

```
urls = "<protocol>://<user>:<password>@<host>:<port>/<enable_password>
```

Example:

```
urls = "telnet://cisco:cisco@192.168.1.1"
```

The *<port>* can be omitted and default port for protocol will be used. When **urls** type is list of strings it can provide multiple intermediate hosts (jumphosts) with their credentials before making the final connection the target device. Example:

```
urls = ["ssh://admin:secretpass@jumphost", "telnet://cisco:cisco@192.
↪168.1.1"]

urls = ["ssh://admin:pass@jumphost1", "ssh://admin:pass@jumphost2",
        "telnet://cisco:cisco@192.168.1.1"]
```

The **urls** can be list of list of strings. In this case the multiple connection chains can be provided to the target device. This is used when device has two processor cards with console connected to both of them. Example:

```
urls = [["ssh://admin:pass@jumphost1", "telnet://
↪cisco:cisco@termserv:2001"],
        ["ssh://admin:pass@jumphost1", "telnet://
↪cisco:cisco@termserv:2002"]]
```

log_dir (str): The path to the directory when *session.log* and *condoor.log* is stored. If *None* the condoor log and session log is redirected to *stdout*

log_level (int): The condoor logging level.

log_session (Bool): If **True** the terminal session is logged.

connect(*logfile=None*, *force_discovery=False*, *tracefile=None*)
Connect to the device.

Args:

logfile (file): Optional file descriptor for session logging. The file must be open for write. The session is logged only if `log_session=True` was passed to the constructor. If *None* then the default *session.log* file is created in *log_dir*.

force_discovery (Bool): Optional. If True the device discover process will start after getting connected.

Raises:

ConnectionError: If the discovery method was not called first or there was a problem with getting the connection.

ConnectionAuthenticationError: If the authentication failed.

ConnectionTimeoutError: If the connection timeout happened.

reconnect(*logfile=None*, *max_timeout=360*, *force_discovery=False*, *tracefile=None*, *retry=True*)
Reconnect to the device.

It can be called when after device reloads or the session was disconnected either by device or jumphost. If multiple jumphosts are used then *reconnect* starts from the last valid connection.

Args:

logfile (file): Optional file descriptor for session logging. The file must be open for write. The session is logged only if `log_session=True` was passed to the constructor. It the parameter is None the default *session.log* file is created in *log_dir*.

max_timeout (int): This is the maximum amount of time during the session tries to reconnect. It may take longer depending on the TELNET or SSH default timeout.

force_discovery (Bool): Optional. If True the device discover process will start after getting connected.

tracefile (file): Optional file descriptor for condoor logging. The file must be open for write. It the parameter is None the default *condoor.log* file is created in *log_dir*.

retry (bool): Optional parameter causing the connnection to retry until timeout

Raises:

ConnectionError: If the discovery method was not called first or there was a problem with getting the connection.

ConnectionAuthenticationError: If the authentication failed.

ConnectionTimeoutError: If the connection timeout happened.

disconnect()
Disconnect the session from the device and all the jumphosts in the path.

reload(*reload_timeout=300*, *save_config=True*, *no_reload_cmd=False*)
Reload the device and wait for device to boot up.

Returns False if reload was not successful.

send(*cmd=''*, *timeout=300*, *wait_for_string=None*, *password=False*)
Send the command to the device and return the output.

Args: cmd (str): Command string for execution. Defaults to empty string. timeout (int): Timeout in seconds. Defaults to 300 sec (5 min) wait_for_string (str): This is optional string that driver waits for after command execution. If none the detected prompt will be used. password (bool): If true cmd representing password is not logged

and condoor waits for noecho.

Returns: A string containing the command output.

Raises: ConnectionError: General connection error during command execution CommandSyntaxError: Command syntax error or unknown command. CommandTimeoutError: Timeout during command execution

enable(*enable_password=None*)
Change the device mode to privileged.

If device does not support privileged mode the the informational message to the log will be posted.

Args:

enable_password (str): The privileged mode password. This is optional parameter. If password is not provided but required the password from url will be used. Refer to *condoor.Connection*

run_fsm(*name*, *command*, *events*, *transitions*, *timeout*, *max_transitions=20*)

Instantiate and run the Finite State Machine for the current device connection.

Here is the example of usage:

```
test_dir = "rw_test"
dir = "disk0:" + test_dir
REMOVE_DIR = re.compile(re.escape("Remove directory filename [{}]?".
↪format(test_dir)))
DELETE_CONFIRM = re.compile(re.escape("Delete {}/{}[confirm]".
↪format(filesystem, test_dir)))
REMOVE_ERROR = re.compile(re.escape("%Error Removing dir {} (Directory␣
↪doesnot exist)".format(test_dir)))

command = "rmdir {}".format(dir)
events = [device.prompt, REMOVE_DIR, DELETE_CONFIRM, REMOVE_ERROR, pexpect.
↪TIMEOUT]
transitions = [
    (REMOVE_DIR, [0], 1, send_newline, 5),
    (DELETE_CONFIRM, [1], 2, send_newline, 5),
    # if dir does not exist initially it's ok
    (REMOVE_ERROR, [0], 2, None, 0),
    (device.prompt, [2], -1, None, 0),
    (pexpect.TIMEOUT, [0, 1, 2], -1, error, 0)

]
if not conn.run_fsm("DELETE_DIR", command, events, transitions, timeout=5):
    return False
```

This FSM tries to remove directory from disk0:

Args: name (str): Name of the state machine used for logging purposes. Can't be *None* command (str): The command sent to the device before FSM starts events (list): List of expected strings or pexpect.TIMEOUT exception expected from the device. transitions (list): List of tuples in defining the state machine transitions. timeout (int): Default timeout between states in seconds. max_transitions (int): Default maximum number of transitions allowed for FSM.

The transition tuple format is as follows:

```
(event, [list_of_states], next_state, action, timeout)
```

Where:

- **event** (str): string from the *events* list which is expected to be received from device.
- **list_of_states** (list): List of FSM states that triggers the action in case of event occurrence.
- **next_state** (int): Next state for FSM transition.
- **action** (func): function to be executed if the current FSM state belongs to *list_of_states* and the *event* occurred. The action can be also *None* then FSM transits to the next state without any action. Action can be also the exception, which is raised and FSM stops.

The example action:

```
def send_newline(ctx):
    ctx.ctrl.sendline()
    return True

def error(ctx):
```

(continues on next page)

(continued from previous page)

```
    ctx.message = "Filesystem error"
    return False

def readonly(ctx):
    ctx.message = "Filesystem is readonly"
    return False
```

The ctx object description refer to `condoor.fsm.FSM`.

If the action returns True then the FSM continues processing. If the action returns False then FSM stops and the error message passed back to the ctx object is posted to the log.

The FSM state is the integer number. The FSM starts with initial `state=0` and finishes if the `next_state` is set to -1.

If action returns False then FSM returns False. FSM returns True if reaches the -1 state.

discovery(*logfile=None*, *tracefile=None*)
Discover the device details.

This method discover several device attributes.

Args:

logfile (file): Optional file descriptor for session logging. The file must be open for write. The session is logged only if `log_session=True` was passed to the constructor. It the parameter is not passed then the default *session.log* file is created in *log_dir*.

family
Return the string representing hardware platform family.

For example: ASR9K, ASR900, NCS6K, etc.

platform
Return the string representing hardware platform model.

For example: ASR-9010, ASR922, NCS-4006, etc.

os_type
Return the string representing the target device OS type.

For example: IOS, XR, eXR. If not detected returns *None*

os_version
Return the string representing the target device OS version.

For example 5.3.1. If not detected returns *None*

hostname
Return target device hostname.

prompt
Return target device prompt.

is_connected
Return if target device is connected.

is_discovered
Return if target device is discovered.

is_console
Return if target device is connected via console.

mode
Return the sting representing the current device mode.

For example: Calvados, Windriver, Rommon.

name
Return the chassis name.

description
Return the chassis description.

pid
Return the chassis PID.

vid
Return the chassis VID.

sn
Return the chassis SN.

udi
Return the dict representing the udi hardware record.

Example:

```
{
'description': 'ASR-9904 AC Chassis',
'name': 'Rack 0',
'pid': 'ASR-9904-AC',
'sn': 'FOX1830GT5W ',
'vid': 'V01'
}
```

device_info
Return the dict representing the target device info record.

Example:

```
{
'family': 'ASR9K',
'os_type': 'eXR',
'os_version': '6.1.0.06I',
'platform': 'ASR-9904'
}
```

description_record
Return dict describing *condoor.Connection* object.

Example:

```
{'connections': [{'chain': [{'driver_name': 'eXR',
                 'family': 'ASR9K',
                 'hostname': 'vkg3',
                 'is_console': True,
                 'is_target': True,
                 'mode': 'global',
                 'os_type': 'eXR',
                 'os_version': '6.1.2.06I',
                 'platform': 'ASR-9904',
                 'prompt': 'RP/0/RSP0/CPU0:vkg3#',
                 'udi': {'description': 'ASR-9904 AC Chassis',
```

(continues on next page)

(continued from previous page)

```
                          'name': 'Rack 0',
                          'pid': 'ASR-9904-AC',
                          'sn': 'FOX2024GKDE ',
                          'vid': 'V01'}}]},
     {'chain': [{'driver_name': 'generic',
                 'family': None,
                 'hostname': '172.27.41.52:2045',
                 'is_console': None,
                 'is_target': True,
                 'mode': None,
                 'os_type': None,
                 'os_version': None,
                 'platform': None,
                 'prompt': None,
                 'udi': None}]}],
'last_chain': 0}
```

1.3.2 Exceptions

This chapter describes all the exceptions used by condoor module.

Init file for condoor.

exception GeneralError(*message=None*, *host=None*)

Bases: exceptions.Exception

General error.

This is a base class for all exceptions raised by condoor module.

__init__(*message=None*, *host=None*)

Initialize the GeneralError object.

Args:

message (str): Custom message to be passed to the exceptions. Defaults to *None*. If *None* then the general class *__doc__* is used.

host (str): Custom string which can be used to enhance the exception message by adding the "*host*: " prefix to the message string. Defaults to *None*. If *host* is *None* then message stays unchanged.

Connection exceptions

The exceptions below are related to connection handling events. There are covered three cases:

- general connection errors caused by device disconnect or jumphosts disconnects,
- authentication errors caused by using wrong credentials to access the device,
- timeout errors caused by lack of response within defined amount of time.

exception ConnectionError(*message=None*, *host=None*)

General connection error.

Bases: *condoor.GeneralError*

exception ConnectionAuthenticationError(*message=None*, *host=None*)

Connection authentication error.

Bases: *condoor.ConnectionError*

exception ConnectionTimeoutError(*message=None*, *host=None*)
Connection timeout error.

Bases: *condoor.ConnectionError*

Command exceptions

The exceptions below are related to command execution. There are covered three cases:

- generic command execution error,
- command syntax error,
- command execution timeout.

exception CommandError(*message=None*, *host=None*, *command=None*)
Command execution error.

This is base class for command related exceptions which extends the standard message with a 'command' string for better user experience and error reporting.

Bases: *condoor.GeneralError*

__init__(*message=None*, *host=None*, *command=None*)
Initialize CommandError object.

Args:

message (str): Custom message to be passed to the exceptions. Defaults to *None*. If *None* then the general class *__doc__* is used.

host (str): Custom string which can be used to enhance the exception message by adding the "*host*: " prefix to the message string. Defaults to *None*. If *host* is *None* then message stays unchanged.

command (str): Custom string which can be used enhance the exception message by adding the "*command*" suffix to the message string. Defaults to *None*. If *command* is *None* then the message stays unchanged.

exception CommandSyntaxError(*message=None*, *host=None*, *command=None*)
Command syntax error.

Bases: *condoor.CommandError*

exception CommandTimeoutError(*message=None*, *host=None*, *command=None*)
Command timeout error.

Bases: *condoor.CommandError*

URL exceptions

This exception is raised when invalid URL to the *condoor.Connection* class is passed.

exception InvalidHopInfoError(*message=None*, *host=None*)
Invalid device connection parameters.

Bases: *condoor.GeneralError*

Pexpect exceptions

Those are exceptions derived from pexpect module. This exception is used in FSM and *condoor.Connection.run_fsm()*

exception TIMEOUT(*value*)

Bases: pexpect.exceptions.ExceptionPexpect

Raised when a read time exceeds the timeout.

1.4 Examples

There is a execute_command.py example file in code repository:

```
#!/usr/bin/env python
# ==============================================================================
# Copyright (c)  2017, Cisco Systems
# All rights reserved.
#
# # Author: Klaudiusz Staniek
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are met:
#
# Redistributions of source code must retain the above copyright notice,
# this list of conditions and the following disclaimer.
# Redistributions in binary form must reproduce the above copyright notice,
# this list of conditions and the following disclaimer in the documentation
# and/or other materials provided with the distribution.
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
# LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
# THE POSSIBILITY OF SUCH DAMAGE.
# ==============================================================================

import logging
import getpass
import optparse
import sys

import condoor


usage = '%prog -H url [-J url] [-d <level>] [-h] command'
usage += '\nCopyright (C) 2017 by Klaudiusz Staniek'
parser = optparse.OptionParser(usage=usage)

parser.add_option(
    '--host_url', '-H', dest='host_url', metavar='URL',
```

(continues on next page)

(continued from previous page)

```
    help='''
target host url e.g.: telnet://user:pass@hostname
'''.strip())

parser.add_option(
    '--jumphost_url', '-J', dest='jumphost_url', default=None, metavar='URL',
    help='''
jump host url e.g.: ssh://user:pass@jumphostname
'''.strip())

parser.add_option(
    '--debug', '-d', dest='debug', type='str', metavar='LEVEL',
    default='CRITICAL', help='''
prints out debug information about the device connection stage.
LEVEL is a string of DEBUG, INFO, WARNING, ERROR, CRITICAL.
Default is CRITICAL.
'''.strip())

logging_map = {
    0: 60, 1: 50, 2: 40, 3: 30, 4: 20, 5: 10
}


def prompt_for_password(prompt):
    print("Password not specified in url.\n"
          "Provided password will be stored in system KeyRing\n")
    return getpass.getpass(prompt)


if __name__ == "__main__":
    options, args = parser.parse_args(sys.argv)

    args.pop(0)

    urls = []

    if options.jumphost_url:
        urls.append(options.jumphost_url)

    host_url = None
    if not options.host_url:
        parser.error('Missing host URL')

    urls.append(options.host_url)


    if len(args) > 0:
        command = " ".join(args)
    else:
        parser.error("Missing command")

    numeric_level = getattr(logging, options.debug.upper(), 50)

    try:
        import keyring
        am = AccountManager(config_file='accounts.cfg', password_cb=prompt_for_
↪password)
```

(continues on next page)

(continued from previous page)

```
    except ImportError:
        print("No keyring library installed. Password must be provided in url.")
        am = None

    try:
        conn = condoor.Connection('host', urls, account_manager=am, log_level=numeric_
↪level)
        conn.discovery()
        conn.connect()
        try:
            output = conn.send(command)
            print output

        except condoor.CommandSyntaxError:
            print "Unknown command error"

    except condoor.ConnectionAuthenticationError as e:
        print "Authentication error: %s" % e
    except condoor.ConnectionTimeoutError as e:
        print "Connection timeout: %s" % e
    except condoor.ConnectionError as e:
        print "Connection error: %s" % e
    except condoor.GeneralError as e:
        print "Error: %s" % e
```

The example help output:

```
./execute_command.py -h
Usage: execute_command.py -H url [-J url] [-d <level>] [-h] command
Copyright (C) 2015 by Klaudiusz Staniek

Options:
  -h, --help            show this help message and exit
  -H URL, --host_url=URL
                        target host url e.g.: telnet://user:pass@hostname
  -J URL, --jumphost_url=URL
                        jump host url e.g.: ssh://user:pass@jumphostname
  -d LEVEL, --debug=LEVEL
                        prints out debug information about the device
                        connection stage. LEVEL is a string of DEBUG, INFO,
                        WARNING, ERROR, CRITICAL. Default is CRITICAL.
```

In below example Condoor connects to host 172.28.98.4 using username *cisco*. The debug level is set to INFO and command to be executed is *show version brief*. Condoor asks for password for username *cisco* and then stores it in local KeyRing. Subsequent command execution does not prompt a password again if password is already stored in the KeyRing.:

```
./execute_command.py -H telnet://cisco@172.28.98.4 -d INFO show version brief

2015-11-21 15:22:38,560     INFO: [host]: Connecting to telnet://cisco@172.28.98.4:23
2015-11-21 15:22:42,456     INFO: [host]: [TELNET]: telnet: 172.28.98.4: Acquiring␣
↪password for cisco from system KeyRing
Password not specified in url.
Provided password will be stored in system KeyRing

cisco@172.28.98.4 Password:
```

(continues on next page)

(continued from previous page)

```
2015-11-21 15:22:53,110     INFO: [host]: Connected to telnet://cisco@172.28.98.4:23
2015-11-21 15:22:53,946     INFO: [host]: Command executed successfully: 'terminal␣
↪len 0'
2015-11-21 15:22:54,781     INFO: [host]: Command executed successfully: 'terminal␣
↪width 0'
2015-11-21 15:22:58,741     INFO: [host]: Command executed successfully: 'show version
↪'
2015-11-21 15:22:58,742     INFO: [host]: Disconnecting from telnet://cisco@172.28.98.
↪4:23
2015-11-21 15:22:59,689     INFO: [host]: Disconnected
2015-11-21 15:22:59,691     INFO: Hostname: 'ASR9K-PE2-R1'
2015-11-21 15:22:59,691     INFO: Family: ASR9K
2015-11-21 15:22:59,691     INFO: Platform: ASR-9010
2015-11-21 15:22:59,691     INFO: OS: XR
2015-11-21 15:22:59,691     INFO: Version: 5.3.1[Default]
2015-11-21 15:22:59,691     INFO: Prompt: 'RP/0/RSP0/CPU0:ASR9K-PE2-R1#'
2015-11-21 15:22:59,691     INFO: [ASR9K-PE2-R1]: Connecting to telnet://cisco@172.28.
↪98.4:23
2015-11-21 15:23:11,075     INFO: [ASR9K-PE2-R1]: Connected to telnet://cisco@172.28.
↪98.4:23
2015-11-21 15:23:11,911     INFO: [ASR9K-PE2-R1]: Command executed successfully:
↪'terminal exec prompt no-timestamp'
2015-11-21 15:23:12,747     INFO: [ASR9K-PE2-R1]: Command executed successfully:
↪'terminal len 0'
2015-11-21 15:23:13,582     INFO: [ASR9K-PE2-R1]: Command executed successfully:
↪'terminal width 0'
2015-11-21 15:23:14,441     INFO: [ASR9K-PE2-R1]: Command executed successfully:
↪'show version brief'


Cisco IOS XR Software, Version 5.3.1[Default]
Copyright (c) 2015 by Cisco Systems, Inc.

ROM: System Bootstrap, Version 0.73(c) 1994-2012 by Cisco Systems,  Inc.

ASR9K-PE2-R1 uptime is 2 days, 16 hours, 46 minutes
System image file is "disk0:asr9k-os-mbi-5.3.1/0x100305/mbiasr9k-rsp3.vm"

cisco ASR9K Series (Intel 686 F6M14S4) processor with 12582912K bytes of memory.
Intel 686 F6M14S4 processor at 2134MHz, Revision 2.174
ASR 9010 8 Line Card Slot Chassis with V1 AC PEM

2 Management Ethernet
8 TenGigE
8 DWDM controller(s)
8 WANPHY controller(s)
1 SONET/SDH
503k bytes of non-volatile configuration memory.
6271M bytes of hard disk.
11817968k bytes of disk0: (Sector size 512 bytes).
11817968k bytes of disk1: (Sector size 512 bytes).
```

1.5 Frequently Asked Questions

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

C

D

E

F

G

H

I

M

N

O

P

R

S

T

U

V