

Market Data — Biblioteca Python (CD3 Connector)

Cedro Technologies · Documentação para o cliente · Versão 2026.1

1. Sobre este documento

Este documento descreve a biblioteca oficial `cd3_connector` para **Python**, que conecta a sua aplicação à API **Socket** do Cedro Market Data (protocolo **Cedro Crystal**, servido pelo Crystal Difusor Cedro).

Ele cobre instalação, uso e boas práticas da biblioteca. **Os comandos do protocolo** (`SQT`, `BQT`, `GQT`, `GCH` etc.), o formato das mensagens e os limites do seu pacote estão na **documentação da API Socket** do seu plano, enviada junto com este documento.

2. O que é a biblioteca

O Cedro Crystal é um servidor de streaming sobre TCP: você abre uma conexão, autentica, assina os dados que deseja e o servidor **empurra** as atualizações continuamente.

A `cd3_connector` encapsula toda a parte de transporte dessa conexão:

- **Handshake** — a sequência de autenticação logo após abrir a conexão.
- **Leitura do socket** — recebe as mensagens e as entrega à sua aplicação por *callback*.
- **Envio de comandos** — você envia os comandos do protocolo como texto (`"SQT PETR4"`).

Você passa usuário, senha e três funções de *callback* — e trata apenas as **mensagens** do protocolo. **Esta é a forma recomendada de integrar com a API Socket.**



3. Instalação

Item	Valor
Pacote	cd3_connector (PyPI)
Versões do Python suportadas	3.8, 3.9, 3.10, 3.11, 3.12 e 3.13

```
pip install cd3_connector
# para atualizar:
pip install --upgrade cd3_connector
```

4. Como usar

4.1 Inicialize o conector

Crie uma instância da classe `CD3Connector`, passando usuário, senha e as funções de *callback* para os eventos de desconexão, mensagem recebida e conexão — **nessa ordem**:

```
import os
import cd3_connector

class Receiver:
    def __init__(self, user: str, password: str):
        self._conn = cd3_connector.CD3Connector(
            user, password, self._on_disconnect, self._on_message, self._on_connect)
        self._conn.start() # inicia a conexão (faz o handshake)

    def _on_connect(self):
        print("Conectado!")

    def _on_disconnect(self):
        print("Desconectado!")
```

2



Av. Anselmo Alves dos Santos, 1111
Uberlândia-MG



www.cedrotech.com



contato@cedrotech.com



(34) 3239-0003

```
def _on_message(self, msg: str):  
    print("Mensagem recebida:", msg)  
  
receiver = Receiver(os.environ["CEDRO_USER"], os.environ["CEDRO_PASSWORD"])
```

⚠ **Não escreva usuário e senha no código.** Leia as credenciais de variáveis de ambiente ou de um cofre de segredos, como no exemplo acima.

4.2 Métodos principais

Método	O que faz
<code>start()</code>	Inicia a conexão com o servidor (faz o handshake)
<code>stop()</code>	Finaliza a conexão com o servidor
<code>send_command(message: str)</code>	Envia um comando do protocolo ao servidor

4.3 Enviar comandos

Com a conexão estabelecida, envie os comandos com `send_command`. Cada comando é uma **string**, exatamente como descrito na documentação da API Socket:

```
self._conn.send_command("SQT PETR4") # assina a cotação de PETR4  
self._conn.send_command("BQT PETR4") # assina o livro de ofertas de PETR4
```

i O **livro de ofertas** (BQT) está disponível a partir do **Pacote PRO**. No Pacote Basic, use apenas os comandos listados na documentação da API Socket do seu plano.

💡 O lugar certo para enviar as assinaturas iniciais é o *callback* `on_connect`: assim elas são reenviadas automaticamente sempre que a conexão for restabelecida.



4.4 Callbacks

Callback	Quando é invocado
<code>on_connect</code>	Quando a conexão é estabelecida com sucesso
<code>on_disconnect</code>	Quando a conexão é perdida
<code>on_message</code>	Sempre que uma nova mensagem é recebida

Exemplo de *callback* de mensagens tratando falha de login:

```
def _on_message(self, msg: str):
    if msg.lower() == "invalid login.":
        print("Login inválido. Verifique suas credenciais.")
    else:
        print(f"Mensagem recebida: {msg}")
```

5. Parâmetros opcionais do construtor

Parâmetro	Padrão	Efeito
<code>active_realtime</code>	False	<code>True</code> recebe do datafeed em tempo real ; o padrão vem em delay
<code>log_level</code>	INFO	DEBUG / INFO / WARNING / ERROR
<code>log_path</code>	vazio	Pasta onde salvar os logs (padrão: pasta <code>cd3_connector</code> no diretório atual)

i Receber em tempo real depende também da **permissão da sua conta** por mercado — consulte com o comando `GUP PM` (documentação da API Socket, seção Usuário e sistema).



6. Exemplo completo

Exemplo com uma **fila** e uma **thread de consumo** separada, reconexão automática e parada da reconexão quando o erro é de credencial ou permissão:

```
import os
import threading
from queue import Queue

import cd3_connector

# Mensagens que indicam erro de credencial/permissão: reconectar não resolve.
ERROS_DEFINITIVOS = {
    "invalid login.",
    "software key not found.",
    "you don't have any permission for this software.",
}

class Receiver:
    def __init__(self, user: str, password: str):
        self._conn = cd3_connector.CD3Connector(
            user, password, self._on_disconnect, self._on_message, self._on_connect)
        self._queue = Queue()
        self._consumer = threading.Thread(target=self._process_messages)
        self._signal = threading.Event()
        self._restart_conn = True
        self._consumer.start()

    def start(self):
        self._conn.start()

    def stop(self):
        self._restart_conn = False
        self._signal.set()
        self._queue.put("") # desbloqueia o consumidor
        self._conn.stop()
```

5



Av. Anselmo Alves dos Santos, 1111
Uberlândia-MG



www.cedrotech.com



contato@cedrotech.com



(34) 3239-0003

```
def join(self):
    self._consumer.join() # bloqueia até o consumidor encerrar

def _on_connect(self):
    self._conn.send_command("sql petr4")
    self._conn.send_command("bqt petr4")

def _on_disconnect(self):
    if self._restart_conn:
        self._conn.start()

def _on_message(self, msg: str):
    self._queue.put(msg) # só enfileira – nunca processa aqui

def _process_messages(self):
    while not self._signal.is_set():
        msg = self._queue.get()
        print(msg)
        self._queue.task_done()
        if msg.lower() in ERROS_DEFINITIVOS:
            self._restart_conn = False
            return

def main():
    receiver = Receiver(os.environ["CEDRO_USER"], os.environ["CEDRO_PASSWORD"])
    receiver.start()
    receiver.join()

if __name__ == "__main__":
    main()
```



7. Boas práticas

7.1 Nunca processe dentro do `on_message`

O servidor **desconecta o consumidor lento**: se a sua aplicação não drenar as mensagens rápido o suficiente e a fila do socket acumular, a conexão é derrubada. Por isso o `on_message` deve apenas **enfileirar** a mensagem — o processamento acontece em outra thread, como no exemplo da seção 6.

7.2 Reconexão com critério

Reconectar no `on_disconnect` é o padrão recomendado, **exceto** quando a mensagem recebida indica erro de credencial ou permissão — nesse caso, reconectar em laço não resolve e pode bloquear a conta:

Mensagem	Significado
<code>invalid login.</code>	Usuário ou senha inválidos
<code>software key not found.</code>	Chave de software não encontrada
<code>you don't have any permission for this software.</code>	Usuário sem permissão para este software

7.3 Uma conexão por usuário

As contas aceitam **pouquíssimas conexões simultâneas (na prática, ~1)**. Uma segunda conexão com o **mesmo usuário** derruba a primeira (erro `E:6`). Garanta que só exista **uma instância** do conector por usuário na sua aplicação — inclusive entre processos e servidores diferentes.

7.4 Encerre a conexão ao terminar

Chame `stop()` ao final da coleta. Nunca deixe uma conexão pendurada.

8. Próximos passos

Com a conexão funcionando, consulte a **documentação da API Socket** do seu plano para:



- A referência completa de comandos (SQT , BQT , GQT , GCH , ...) e o formato de cada mensagem.
- Os limites do seu pacote (ativos assinados simultaneamente, retenção de candles).
- A tabela de erros E: e os guias práticos (watchlist, fita de negócios, gráfico de candles).

