IB Python API

Jay Parmar

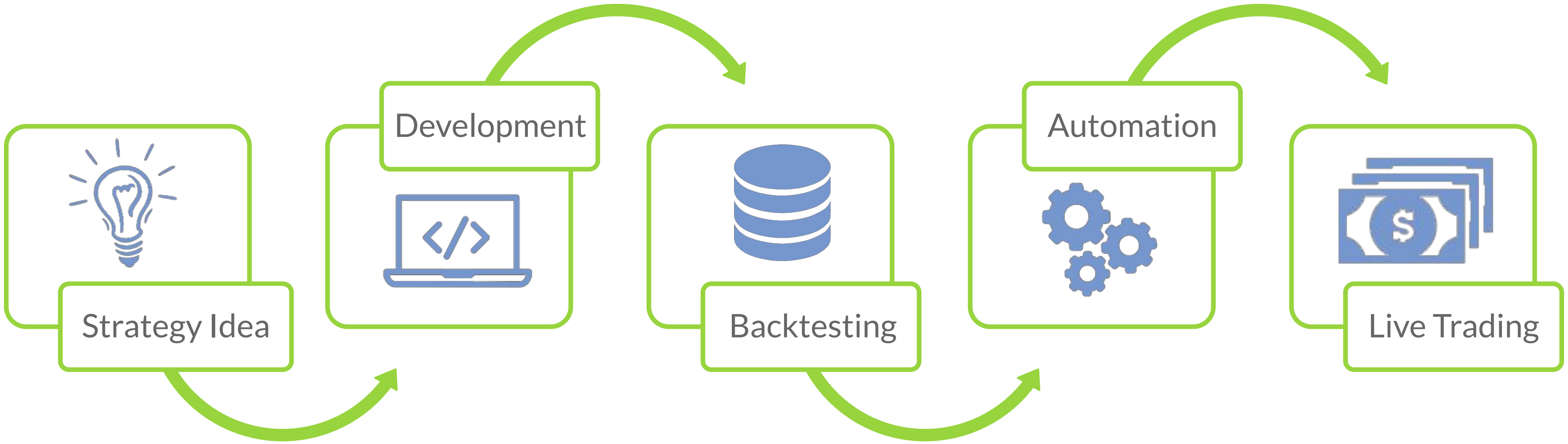
Goal

- Able to trade algorithmically using IB TWS API using Python on a cloud instance.

Agenda

- 1) Strategy Life Cycle
- 2) Manual Trading using IB Trader Workstation (TWS)
- 3) Understanding IB TWS API architecture
- 4) IB API Demo
    - A) Connecting API to TWS
    - B) Fetching contract details and options chain
    - C) Working with historical and live market data
    - D) Placing orders
    - E) Fetching account information
    - F) Limitations and troubleshooting
- 5) Overview of Cloud Computing - Demo

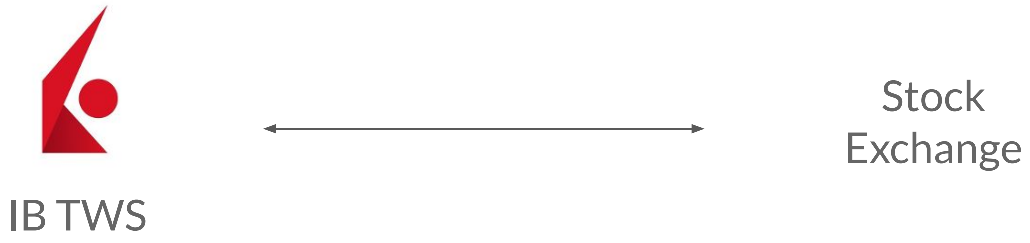
1) Strategy Life Cycle



2) Manual Trading using IB TWS

- The official trading terminal by IB
- Cross platform application
- Demo
    - Watchlist
    - Ticker details
    - Order placing
    - Account Information
- Download: *Link*

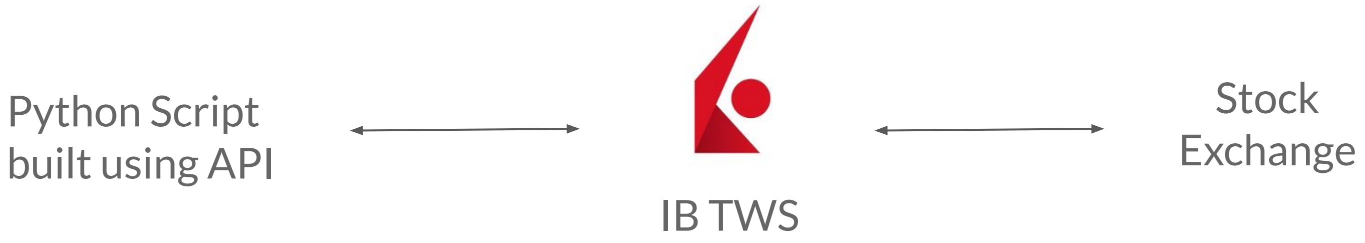
2) Manual Trading Workflow



2) IB TWS Demo

IB TWS Demo

3) Automated Trading Workflow



3) IB TWS API Overview

- Official Application Programming Interface by IB
- Different from IB TWS
- Allows operations including but not limited to
    - Order placement
    - Receiving market data and portfolio data
    - Receiving account values
    - Querying financial instrument details
- Supports multiple account types and programming languages
- Open source: *Link*
- Excessive Documentation: *Link*

3) Installation and Configuration

- Installation
  - Download IB API from GitHub: *https://interactivebrokers.github.io/*
  - Install it
  - Go to installation folder and run the `setup.py` file

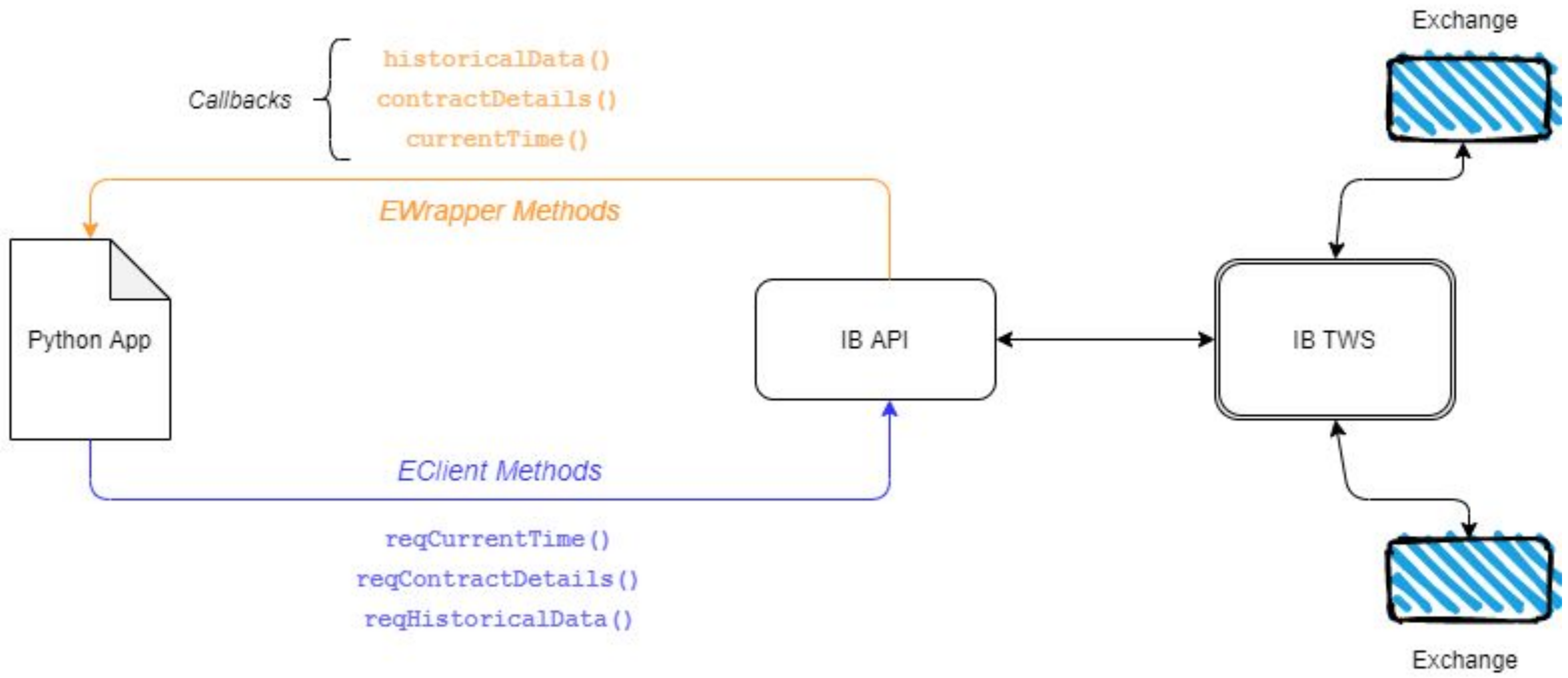
- Configuring IB TWS
  - Go to File menu and click on Global Configuration option
  - Click on the API and select Settings sub-option
    - Check “Enable ActiveX and Socket Clients” option
    - Uncheck “Read-Only API” option
    - Change the socket port to 7497 (for paper trading) or 7496 (for live trading)

3) Components of IB API

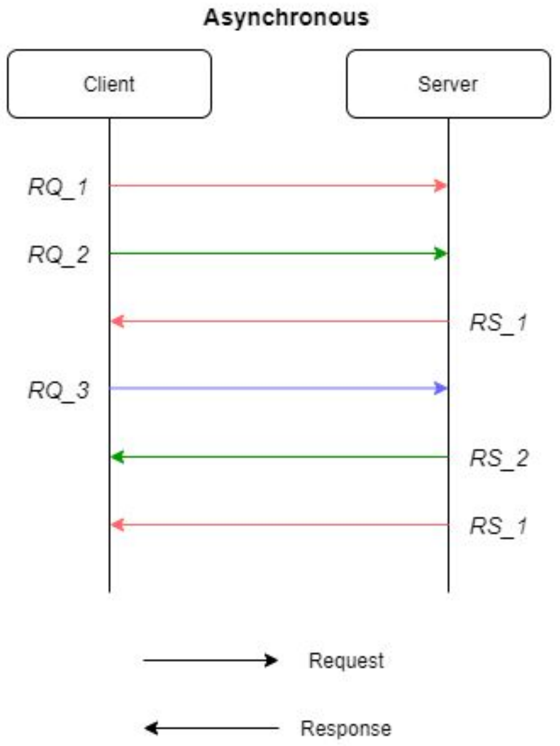
- EClient Class
    - Used to send requests to TWS from Python client
    - Implemented in `client` sub-module
    - Uses: Connect to TWS, request historical data, place orders, etc.

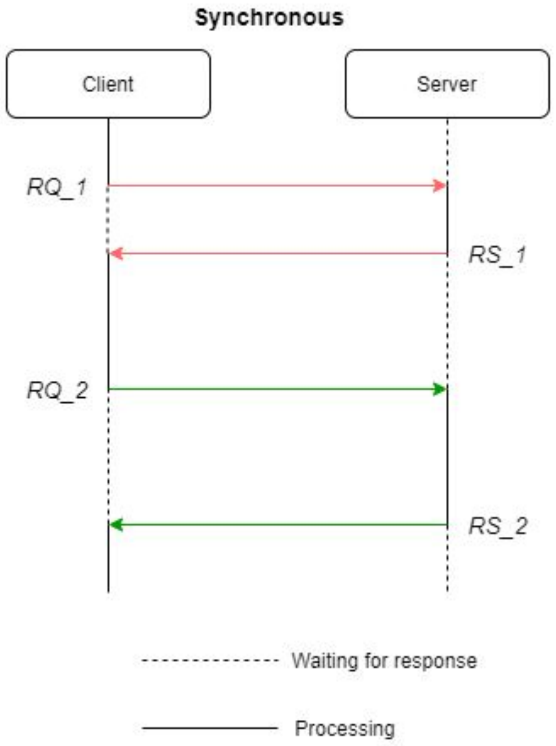
- EWrapper Class
    - Used to receive responses from TWS to Python client
    - Implemented in `wrapper` sub-module
    - Uses: Receive order updates, receive market data, receive positions update, etc.

3) Information Flow



3) Communication Modes

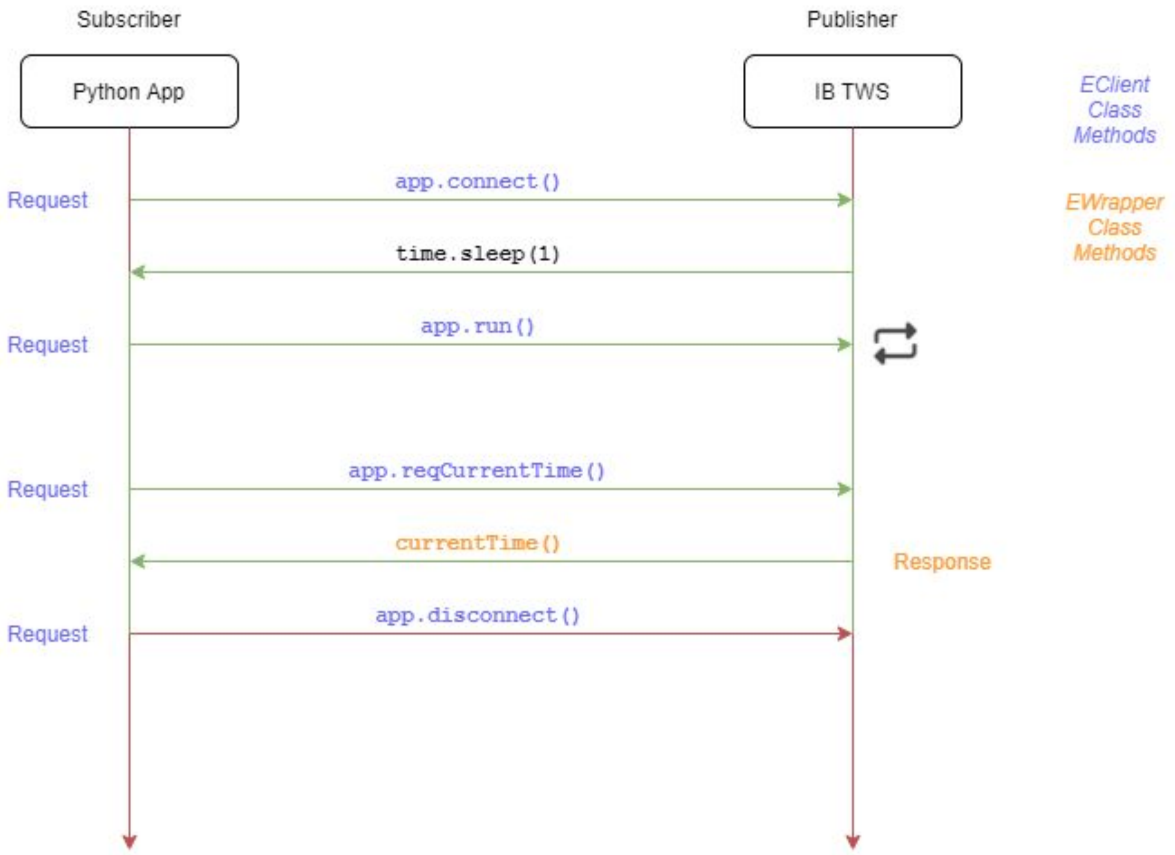




4) Hello World!

- Goal: Connect Python script to TWS
- Derive a strategy class from **`EClient`** and **`EWrapper`** classes
- Create an object of the strategy
- Run the following methods on the object:
    - **`connect()`** to connect with the
    - **`run()`** to send request and receive response from the TWS
- Documentation: *Link*
- Demo filename: **`1_IB_API_Connection.py`**

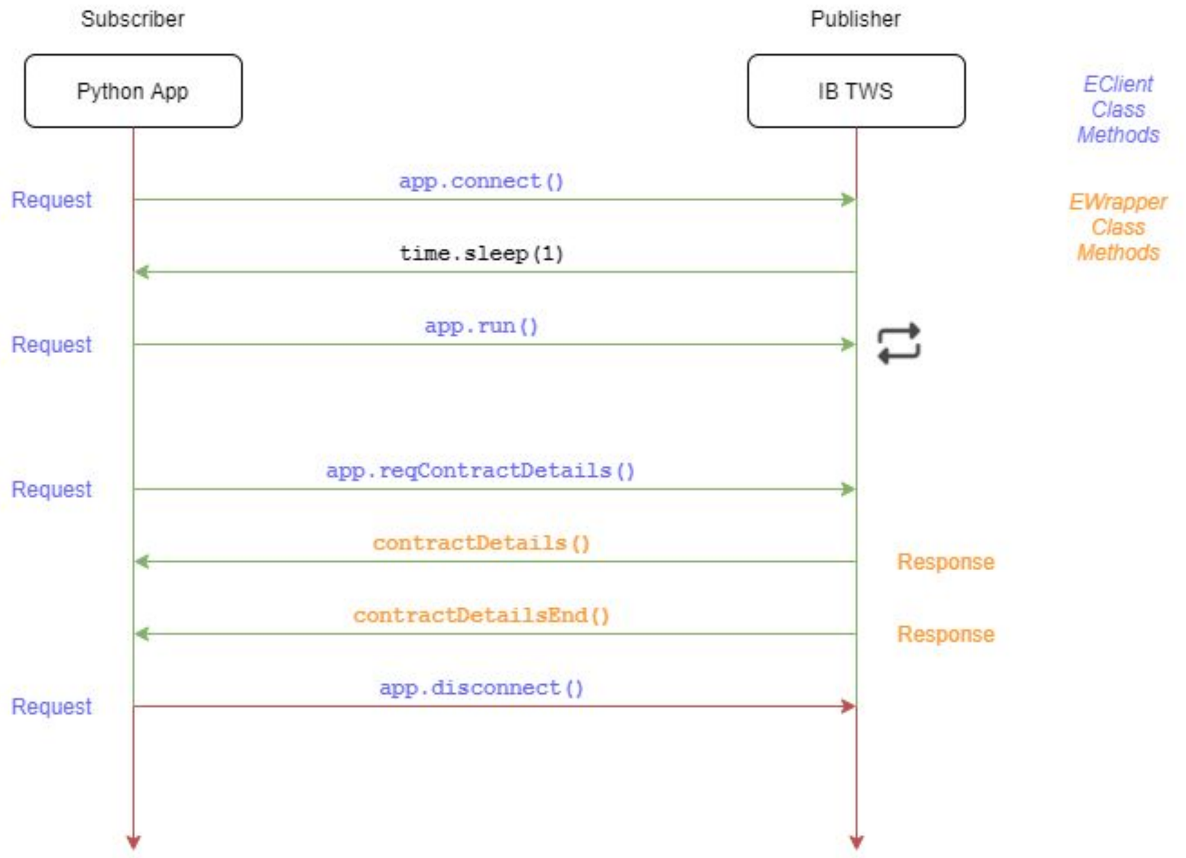
4) Pictorial Representation



4) Fetching Contract Details

- IB API supports trading all assets which can be traded from TWS
- Fetching contract details using API
  - Use `ibapi.contract` module to define contract objects
- Following are the *common* properties of a given contract
  - `symbol`
  - `currency`
  - `secType`
  - `exchange`
  - `primaryExchange`
- Documentation: *Link*
- Demo filename: `2_IB_API_Contract_Details.py`

4) Fetching Contract Details



4) Historical Data

- IB API allows fetching historical data for varied duration
- Use `reqHistoricalData()` method to fetch historical data
    - `contract`
    - `endDateTime`
    - `durationStr`
    - `barSizeSetting`
    - `WhatToShow`
- Receive response from the TWS using:
    - `historicalData()`
    - `historicalDataEnd()`
- Documentation: *Link*
- Limitations: *Link*
- Demo filename: `3_IB_API_Historical_Data.py`

4) Live Streaming Data

- IB API allows fetching real-time market data across exchanges
- Use **`reqMktData()`** method to subscribe to market data
  - **`contract`**
  - **`genericTickList (1:Bid Price, 2:Ask Price)`**
  - **`snapshot`**
  - **`regulatorySnapshot`**
  - **`mktDataOptions`**
- Receive response from the TWS using:
  - **`tickPrice()`**
- Cancel market subscription using
  - **`cancelMktData()`**
- Documentation: *Link*
- Demo filename: **`4_IB_API_Market_Data.py`**

4) Order Management

- Create an order object using the order class from `ibapi.order` module
  - `action`
  - `totalQuantity`
  - `orderType`
  - `lmtPrice`
- Use `placeOrder()` method to place orders
- Receive response from the TWS using:
  - `openOrder()`
  - `orderStatus()`
  - `execDetails()`
- Cancel orders using
  - `cancelOrder()`
- Documentation: *Link*
- Demo filename: `5_IB_API_Place_Order.py`

4) Positions

- IB API allows fetching details of all positions in the market.
- Use **`reqPositions()`** method to fetch details about positions
- Receive response from the TWS using:
  - **`position()`**
  - **`positionEnd()`**
- Documentation: *Link*
- Demo filename: **`6_IB_API_Positions.py`**

4) Fetching Option Chain

- IB API supports trading all assets which can be traded from TWS
- Fetching contract details using API
    - Use **`ibapi.contract`** module to define contract objects
- Following are the *common* properties of a given contract
    - **`strike`**
    - **`right`**
    - **`exchange`**
    - **`lastTradeDateOrContractMonth`**
- Documentation: *Link*
- Demo filename: **`7_IB_API_Option_Chain.py`**

4) IB API - Limitations

- Each TWS instance can listen to 32 client apps simultaneously
- Can send 50 messages per second from the client to the IB applications
  - Order request
  - Historical data request
  - Positions request
  - Account request
- The maximum number of simultaneous open historical data requests from the API is 50.

5) What's Next?

- Deployment of an algorithm
- Setup a trading server in-house
- Issues with traditional in-house setup
- Modern day approach: Cloud Computing
  - What it is?
- Benefits of using cloud computing
  - Scalability
  - Cost-effectiveness
  - Elasticity
  - High availability
- Demo on AWS

5) Components of a Server

- Operating system
- Random Access Memory (RAM)
- Storage media
- Databases
- Network connectivity
- Trading softwares

5) Issues with Traditional Setup

- Pay for electricity, cooling and maintenance
- Adding and replacing hardware takes time
- Scaling is limited
- Might need to hire someone to monitor the infrastructure
- Dealing with disasters becomes difficult (natural calamities, power shutdown, fire, etc.)

Q&A

- Questions, if any!

Resources

- IB TWS Download: *Link*
- IB TWS API Download: *Link*
- IB TWS API Documentation: *Link*
- Spyder IDE Documentation: *Link*