Virc

Personal project

Software goals : Automate crypto trading on Coinbase Pro's platform.

Educational goals : Learn and improve my devops and full stack skills.

Problems solved : Automatically open/close positions on market events, provides range trading bots. Written extensible, it is open for other uses and other platforms.

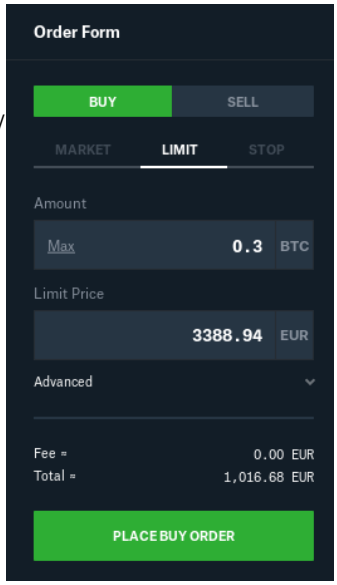
Table of Contents

Overview

Coinbase Pro is an online digital currency exchange, where you can buy and sell the major cryptocurrencies as bitcoin and ether. It provides some basic operations through its interface: sell/buy at market price, set stop losses and sell/buy when the price has reached a certain value.

It is impossible to define elaborate actions. For example, when the market goes down, we could want to cancel our higher sell orders and set stop loss on this assets.

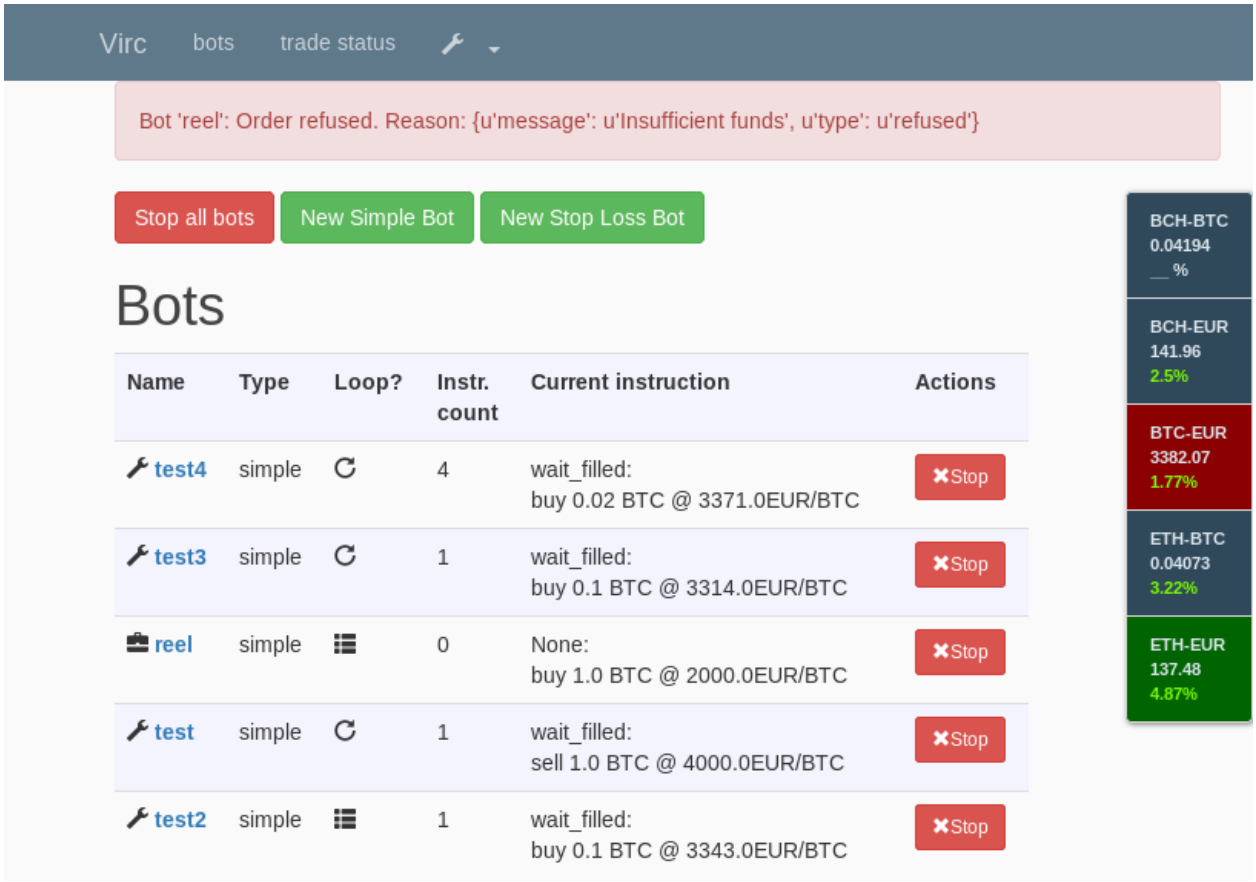
But Coinbase Pro provides an API...



Coinbase pro's main features

Virc is a way to create autonomous bots able to communicate and trade on Coinbase Pro. As it has be written to be modular, other bot functions and other trade platforms can be added.

It uses several docker containers, managed by docker-compose (see Technical details chapter below).



Virc's main interface

Technical skills used

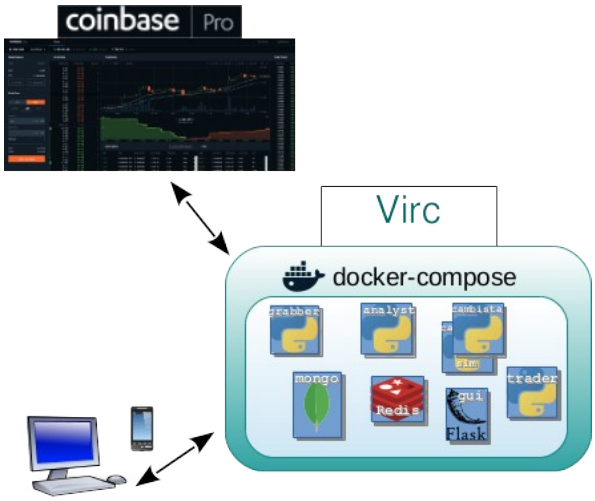
Agile development. Designed to be robust and extensible, the main goal was to make it run as soon as possible. A list of improvements with priorities is updated after each new feature implementation.

Tools

Docker, docker-compose
Git
SSE (Server-sent event), websocket, REST API
jQuery, javascript, jinja2, flask, gunicorn, HTML5, CSS, bootstrap, JSON, Firefox developer tools/debugger
Python, pip
Linux (Ubutnu, Alpine)
MongoDB
Redis as broker, pubsub server, also used for object persistence.
Common unix tools (gvim, vi, bash, debian packaging tools, linux monitoring tools, grep, find, diff...)

Technical details

Overview

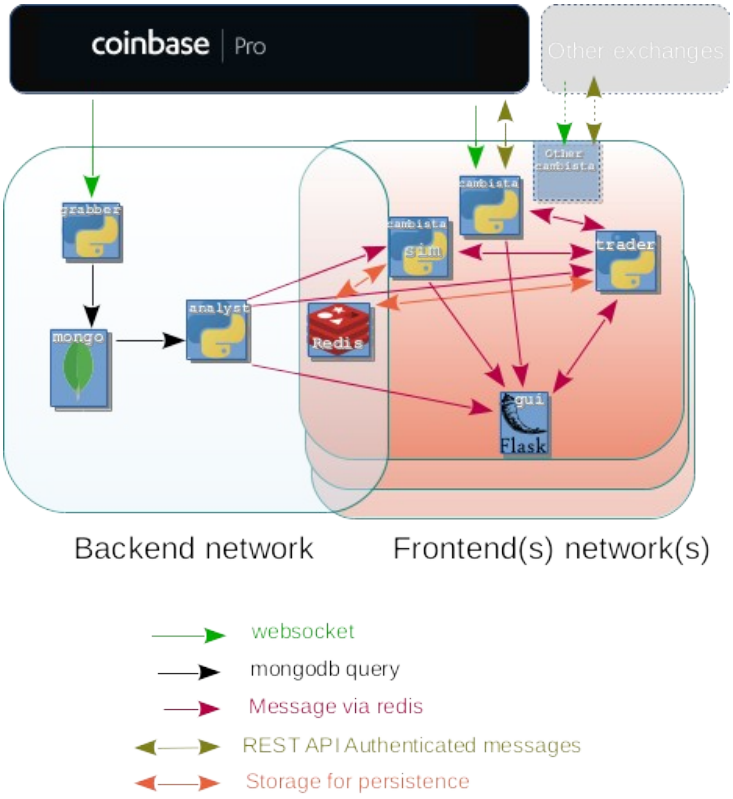


Virc communicates with Coinbase Pro through REST API and receives market update from websocket feeds.

The user interface is provided by a webserver using the Flask framework.

Containers

grabber: python script which fetches the market data from Coinbase's websocket API transforms and stores in mongodb



mongo: official mongo container. Stores data in an external volume.

analyst: process data and provides statistics, such as tickers and averages (for now volumes and prices). Planified improvements : provide formatted data to draw candlestick, RSI, …

redis: Standard Alpine-linux container. Configured with an external volume for persistence.

gui: A linux container running an gunicorn web server and the Flask application server. Provides the bot management. All data is provided by redis. Uses SSE for asynchronous messaging with the client (tested with Firefox, should work with any modern web browser).

trader: trader.py is in charge to run, stop and monitor the bots. Bots are forked in this container, gets and set their status in redis for persistence. Each bot type is a script instance, creating a instance of an object which inherits from the base class Bot.

cambista: connects to the private Coinbase's API, communicates with the bots in an unified API. It receives the orders from the bots (buy, sell, cancel, …), transmits orders to coinbase, and sends updates to the bots. Can also be duplicated to connect to Coinbase's sandbox.

cambista sim: Do the same as cambista, but in simulation mode. Nothing is sent to coinbase, so it maintains an order book locally, and emulates the coinbase messages.

Next steps

- Implement new bot types
- Document the messaging system
- Improve web interfaces
- Machine leaning
- ...

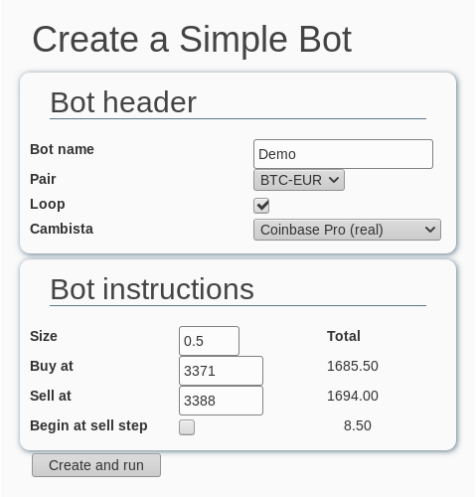
Conclusion

The computer engineer's way to wish a long life to his project could be : "design it simple, modular and robust". That's what I tried to do while working on this project. I spent time to document myself, evaluate the different solutions to solve some a problem, and always tried to choose the lightest way. It is a real pleasure to work on dockerized components to make tests, change some code without halting the whole process.

Virc is not aimed to be a state-of-the-art of what I can do, but how I can design and implement an IT project, and how I learn new fields to go further.

Examples

Screenshots



Simple bot definition

Bot detail, with history

Structure

Source code from cambista, sends a order_filled *message to a bot and sends a message to the user's interface :*

```
rds.lpush(channels["order_filled"] + msg['order_id'], json.dumps(msg))
utils.flash("Order '%s' filled" % msg['order_id'], "info", sync=False)
```

A simple bot structure in JSON, stored in redis :


```
{
    "name": "test4",
    "start_date": "2019-01-05T16:08:15.956086",
    "type": "simple",
    "pair": "BTC-EUR",
    "cambista_link": "virc:cambista:sim",
    "cambista_title": "Coinbase Emulated (sim)",
    "uid": "553c5def-7e1d-44df-aeed-266c399f6b7e",
    "pid": 17,
    "instructions_count": 4,
    "instructions_index": 0,
    "instructions_loop": true,
    "instructions": [
        {
            "pair": "BTC-EUR",
            "price": 3371.0,
            "side": "buy",
            "size": 0.02,
            "start_date": "2019-01-05T17:42:39.622722",
            "status": "wait_filled",
            "type": "order",
            "wait_filled": "2b31f615-b443-4a18-9341-f957d03db2df"
        },
        {
            "pair": "BTC-EUR",
            "price": 3383.0,
            "side": "sell",
            "size": 0.02,
            "start_date": "2019-01-05T16:55:08.028944",
            "status": null,
            "type": "order",
            "wait_filled": null
        }
    ]
}
```


Available on Github !

https://github.com/sznicolas/virc

Still in heavy development, many updates will come soon.

Credits

Cliparts are from openclipart.org :
- Thin client from LindsayBradford
- Smartphone by filtre

Public domain licence CC0 1.0