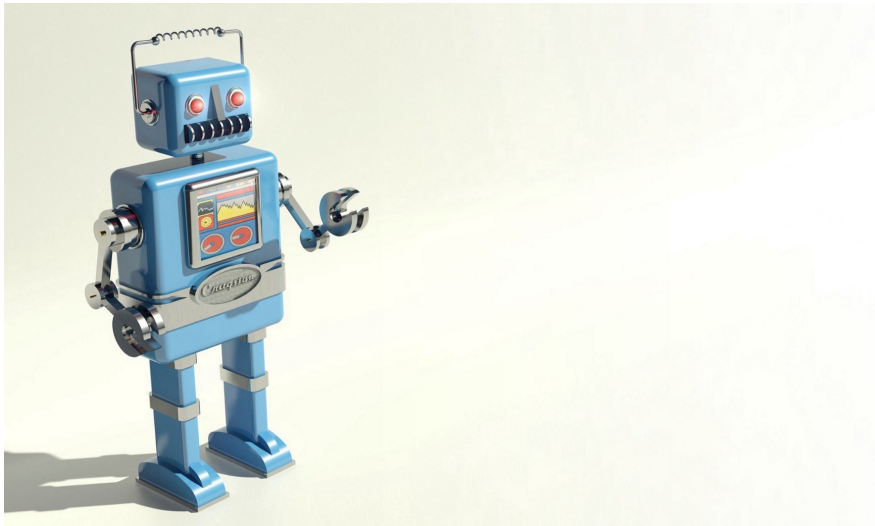
joel degan
I write tutorials that I wish I found, instead of having to write. I like puzzles, games, beer and travel and any combinations of those.
Jun 30, 2017 · 16 min read

Let’s write a cryptocurrency bot. (part 2)



NOTICE: The underlying Github project for this is being rewritten, much of this is to do with the fact that some of what this was originally based on (Bitfinex) has become hostile to persons from the USA and USA persons can’t open an account with them.

PLEASE CHECK THE ROADMAP FOR THE PROJECT. Thanks.

———————————— —

Want to make your own Bitcoin bot? This is part two of a three part series on creating a cryptocurrency trading bot using the boilerplate and framework library called Bowhead for general use in the cryptocurrency space.

— This is a three part article: Part 1, Part 2 and Part 3

In Let's write a cryptocurrency bot. (part 1) we went through the overview of the Bowhead crypto-bot boilerplate and framework and how to set it up, if you have not read it that is where you will want to start.

In part 2 we are going to:

1. Add in ETH data collection so we can trade ETH as well.
2. We are going to write a generalized 'signals' responder which will show us scaled buy/sell signals based on multiple indicators. This will help us narrow down instruments to look more closely at. We have six new indicators to introduce.
3. We need some strategies, I provide enough to keep you busy for quite a while with twenty basic strategies to use on timeframes from one minute to one hour.
4. Run a full test of the strategies, using the signals, on a demo account on an exchange. This way we can see how the strategies

and the signals work together.

5. We are going to write a real-time GDAX straddle-bot using Bowhead with one-minute time-limited orders to try to catch some swings.

6. Bowhead introduction as a crypto-trading REST API so you can use it with any language that supports REST. (This part is unfinished and will be a work in progress going forward)

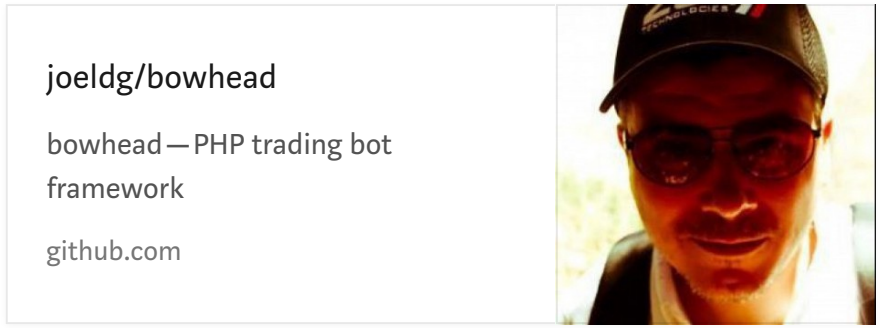
Part 2 is a little heavier on the indicators space, writing a currency bot is 'mostly' using technical indicators and making decisions based on those. These indicators are what you will be using so make sure to understand what technicals you are using and more importantly why they were chosen for use. Where possible I have provided links to further reading.

In the next article in the series, we will code up more advanced Forex actions, including using binary options strategies and we may explore arbitrage opportunities between the brokerages and market makers as well as building a more advanced signals system. Also, we will want to explore setting buys and sells at target instead of trading at market price. Part 3 will conclude the series, though the library will include updates and I may start another series on more advanced strategies.

Lets get going.

> *Note: If you previously went through the tutorial, you will want to recreate the database as the structure has changed to accommodate new periods*

> *for strategies.*



Added Bitfinex ETH support.

In our first iteration we only were looking at Bitcoin, this time around I've added ETH as well with easy instructions for adding other currencies from Bitfinex.

So, Bowhead has a new function to keep up to date ETH data in the database.

```
php artisan bowhead:websocket_bitfinex_eth
```

This cannot be combined with the BTC queries as Bitfinex does not tag the data returning, so to get going quickly, we needed to create a new worker.

Keeping this running in supervisor will make sure we have up to data ETH data.

Forex Signals

A trade signal is typically simply a technical indicator, or it is a series of technical indicators that we use to determine if it is a good time to enter or exit a trade.

We want to create a Forex signals trait that we can add to our bot's and for this we will need a few more technical indicators added to the mix, so I have added six new ones to the indicators class for part 2 of this tutorial. The indicators are fairly important and I have provided links to descriptions of them and the reasons we might want to use them.

- Elder ray—Bulls/Bears Power.
- High-Low Index
- Williams R%
- Ultimate Oscillator
- Price Rate Of Change
- Stochastic RSI

Which along with the following useful indicators we already had previously that we will be using for our signals responder

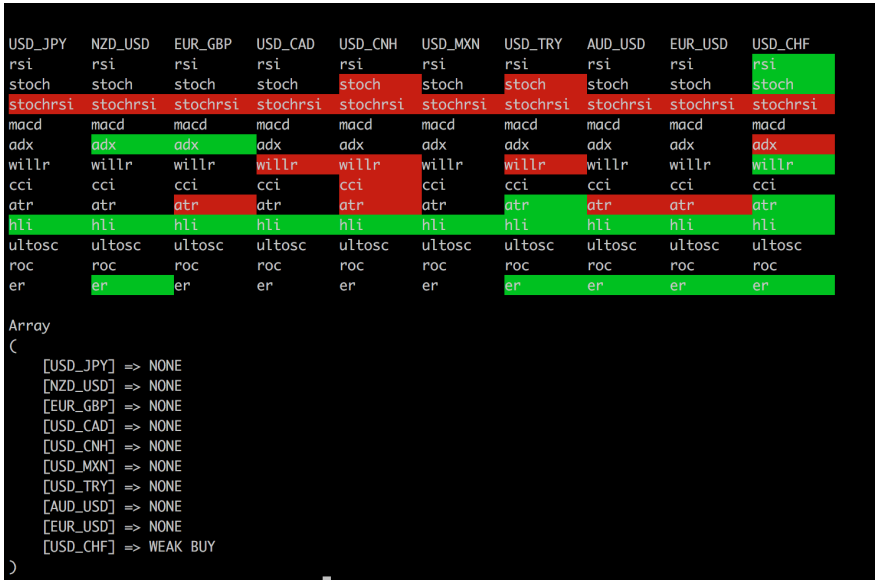
- Relative Strength Index (RSI)
- Stochastic Oscillator
- Moving Average Convergence Divergence (MACD)
- Average Directional index (ADX)
- Commodity Channel Index (CCI)
- Average True Range (ATR)

Combined, this a pretty decent mix of indicators used to trade currency pairs and will work equally effectively on cryptocurrencies as regular currencies.

These have all been compiled into a trait named 'Signals' in Bowhead. I have provided an example command to see how the Signals work.

```
php artisan bowhead:example_signals
```

To read this, the pairs are at the top and the indicators are in the column below, if one column is all green with one or two exceptions it is an indication that you should look at buying. If it is all red down, then it is an indication to sell or short. The more red it is, the stronger the signal is to sell and the more green, the stronger the signal is to buy.



In this example, that is running on a very incomplete database that is not up to date the only one that would be lukewarm to buy would be USD/CHF.

The trait has two options, a return and a compiled option, which you see above. The return option simply returns the raw data. The compile option returns all the pairs with potential recommendations of:

```
WEAK BUY, GOOD BUY, STRONG BUY, VERY STRONG BUY
WEAK SEL, GOOD SELL, STRONG SELL, VERY STRONG SELL
```

These can be used as one part , with strategies being the second part, to create a fairly sophisticated automated currency bot. Now lets put together some strategies for use with bots.

Strategies

Having a bot with *a single* strategy is okay I guess, but having a bot with many strategies hooked up to a signals library is better. This way we can have multi-confirmation of entry and exit and judge our strategies better, based on risk.

I figured that if we needed some strategies that *more* would be better. So I coded up the following twenty strategies to make sure you have a good start with your bot, these are all Forex/Cryptocurrency strategies for 1M, 5M and 1H periods.

Each strategy has three parameters $pair, $data and $return_full, with pair being the symbol, $data being the results of getRecentData($pair) and return_full being a return array used for testing with the direction (short/long) and the name of the function. Keeping the parameters always the same also makes it so it is easy to programmatically call these strategies and chain them together.

The strategies are as follows, with periods and basic descriptions:

- **bowhead_sar_stoch** —1M—if SAR is above a black candle and stoch or fast-stoch cross the upper 90 line going down we sell, if

SAR is below a green candle and stoch or fast-stoch cross the 10 line going up we open a long.

- **bowhead_awesome_macd**—1M—if MACD<0 and AO < 0 then sell, if MACD >0 and AO > 0 then long.
- **bowhead_adx_smas** —1M—if ADX > 50 and SMA6 crosses SMA40 going up, we sell, if ADX > 50 and SMA6 crosses SMA40 going down we open a long
- **bowhead_rsi_macd** —1M—if RSI < 30 and MACD_raw-MACD_signal < 0 then we sell, if RSI > 70 and MACD_raw-MACD_signal > 0 then we open a long
- **bowhead_sar_rsi**—1M—if SAR over black candle and RSI > 70 sell, if SAR under a green candle and RSI < 30 long
- **bowhead_stoch_adx**—5M—if ADX > 50, Stoch < 10 and last two candles were black then sell, if ADX > 50, Stoch > 90 and last two candles were green open a long
- **bowhead_cci_scalper**—5M —if CCI200<0 and EMA10 < EMA21 and EMA10 < EMA50 then sell and if CCI200>0 and EMA10>EMA21 and EMA10 > EMA50 then long *note: this strategy needs 200 points of data.*
- **bowhead_ema_scalper**—1M —red = EMA(2, 3, 4, 5 ,6, 7, 8, 9, 10, 11, 12, 13, 14, 15) green = EMA(44, 47, 50, 53, 56, 59, 62, 65, 68, 71, 74) when red crosses green from the top open a long, when red cross green from the bottom, sell.

- **bowhead_ema_stoch_rsi**—1H—if EMA5 crosses EMA10 from the bottom, RSI>50 and Stoch not above 80 yet then long. If EMA5 cross EMA10 from the top, RSI < 50 and Stoch not below 20 yet then sell.
- **bowhead_double_volatility**—15M—if SMA5(high)>SMA20(high) and RSI > 65 then long, if SMA5(high)<SMA20(low) and RSI < 35 then sell.
- **bowhead_adx_momentum**—5M—if ADX > 25 and MOM > 100 and SAR below a green candle then long, if ADX > 25 and MOM < 100 and SAR above a black candle then sell.
- **bowhead_base_150**–1H—MA6, MA35 cross MA150 and MA365 from the bottom then long. MA6, MA35 cross MA150 and MA365 from the bottom then sell.
- **bowhead_breakout_ma**—1D/1H—if EMA34>SMA20(low) and ADX >25 then long, if EMA34 < SMA20(low) and ADX > 25 then sell
- **bowhead_sar_awesome**—30M—if SAR above a black candle and AO has a bearish cross and EMA 5 < current price then long, if SAR below a green candle and AO has a bullish cross and EM5 > current price then sell
- **bowhead_cci_ema**—1H—if EMA8 crosses EMA28 from below and CCI30 > 0 then long, if EMA8 crosses EMA28 from above and CCI30 < 0 then sell
- **bowhead_bband_rsi**—15M—if RSI > 70 and prices touches BBAND lower then long, if RSI < 30 and prices touches BBAND

top then sell

- **bowhead_ema_adx_macd**—4H—if EMA4 crosses EMA10 from the top and MACD < 0 then long, if EMA4 crosses EMA10 from below and MACD > 0 then sell
- **bowhead_mov_avg_sar**—1H—if EMA10 crosses EMA25 and EMA50 from below and SAR below a green candle then long, if EMA10 crosses EMA25 and EMA50 from above and SAR above a black candle then sell
- **bowhead_momentum**—1H—if RSI > 0 and MOM>100 and SMA11>SMA21 and current price > SMA21 and SMA11 then long.if RSI < 0 and MOM greater than 100 and SMA11<SMA21 and current price <SMA21 and SMA11 then sell.
- **bowhead_sma_stoch_rsi**—1H—if current price > SMA150 and RSI < 20 and StochK > 70 and StochK>StochD then long, if current price < SMA150 and RSI > 80 and StochK > 70 and StochK<StochD then sell.

> See the code for the actual strategies, and feel free to add to this list.

This should be a good start and should offer plenty of examples for how to code up strategy libraries that are quick and easy to use.

Part 3 will provide another big batch of strategies that we can automate and those will make use of TALib candle patterns for automated chart reading that you can combine together to replicate whatever favorite strategy you have. If you are interested, the library is already in Bowhead.

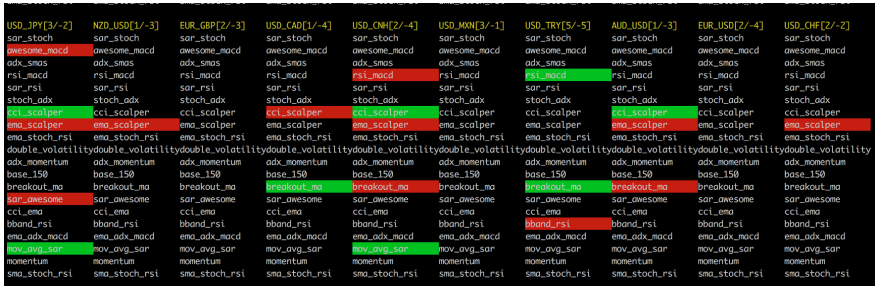
Testing

We have our strategies and our data collection, now we need to be able to test and see what works. Keep in mind that these strategies require a lot of data to work with, without enough data you may see false positives/negatives. Make sure that your Oanada streamer is collecting data.

We are going to use Whaleclub to test because they have a nice demo account and their interface is simple and easy to understand, for purposes of this tutorial they are the easiest.

Now, lets test all of our strategies using ten Forex pairs with a demo account and see what happens. This will run 200 strategies per run with the following command.

```
php artisan bowhead:test_strategies
```



This image was on a short-run testing database. Most of these strategies require more than 30 data points, with 50, 150 or even 200–365 data points for some, your view should have much less that this. These strategies should be firing off like mad. Keep in mind that some of these strategies are meant to be run on one hour periods, not our 1 minute periods. Results may vary.

You can see the signals positive/negative next to the currency pairs in the top row, these can be used as a secondary verification for position entry or exit.

If you are ready to start a live test of these strategies, you will need to make sure that you have your .env file up to date and run the following:

> NOTE: This is a blanket TEST, DO NOT under ANY circumstances, use real money on this. The entire idea is to see what does NOT work.. What does work is a super-small percentage that is base on time and what pair you are looking at.

```
php artisan bowhead:test_strategies test
```

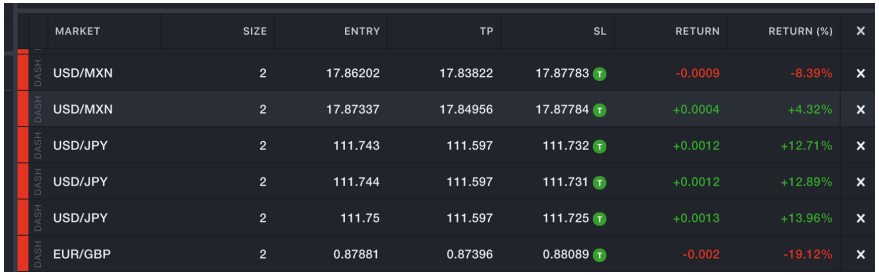
This will fill in the database table named **bowhead_strategy** as it creates longs and shorts for each strategy and you will see the strategies being run in the site interface. We are not interested in managing the positions for this test, we set a take profit at 20% of invested BTC value at 200x leverage and we set a stop loss at 10% of invested BTC value at 200x leverage and then just let the trades ride.

A note about how percentages are computed with leverage:

```
// note: we round to five decimal points
amount = price + round((price * (percentage / leverage)) /
100, 5)
```

```
// so if we want to do a 45% Take Profit on USD/JPY with 200
leverage // at price 111.702 we would set it to:
$take_profit = 111.702 + round((111.702 * (45/200))/100, 5);
// 0.25133 is added to 111.702 and TP is set to 111.95333
```

Once we are up and running and see some trades being made on strategies we can swap over to the web interface and take a look.



	MARKET	SIZE	ENTRY	TP	SL	RETURN	RETURN (%)	
DASH	USD/MXN	2	17.86202	17.83822	17.87783 T	-0.0009	-8.39%	×
DASH	USD/MXN	2	17.87337	17.84956	17.87784 T	+0.0004	+4.32%	×
DASH	USD/JPY	2	111.743	111.597	111.732 T	+0.0012	+12.71%	×
DASH	USD/JPY	2	111.744	111.597	111.731 T	+0.0012	+12.89%	×
DASH	USD/JPY	2	111.75	111.597	111.725 T	+0.0013	+13.96%	×
DASH	EUR/GBP	2	0.87881	0.87396	0.88089 T	-0.002	-19.12%	×

This is what the demo interface will look like while testing.

The script as it runs will update the currently active and closed positions with the closed reason (*at_market, at_stop, at_target, or liquidation*) and the profit column will display the amount in negative or positive satoshi's which we will convert back to BTC values.

For this testing: leverage is computed based on the signals, so for each *negative* signal there is for a *long* position we take 20 leverage away with the alternate being true for a *short* where we multiply the *positive* signals times 20 and subtract that amount of leverage. This attempts to adjust risk for each trade.

If you would like to run more detailed stats on your strategies from the database you can run the following SQL query to see which strategies are net positive or negative per strategy and the number of positive trades vs the number or negative trades per strategy.

```
SELECT
sum(`profit`)/100000000 AS `profit`
, strategy
, direction
```

```
, count(*) AS cnt
, count(CASE WHEN `profit` < 0 THEN 1 END) AS neg
, count(CASE WHEN `profit` > 0 THEN 1 END) AS pos
FROM bowhead_strategy
WHERE pair IS NOT NULL
GROUP BY direction, strategy
ORDER BY `profit` DESC
```

To dig in deeper and examine certain strategies on certain pairs you can modify the query. The table also keeps track of the signal data from the signals library. The test runs the signals and it marks down, for each trade, the number of positive signals and the number of negative signals.

All of this data combined should give you a fairly comprehensive view of how these strategies can work, and what you need to do to test and build your own winning strategies.

Further testing

To further refine this test the command TestStrategiesCommand in

```
app/Console/Commands/TestStrategiesCommand.php
```

is commented near the bottom where you would make changes. Modify and test different combinations of these strategies. Or use the **getRecentData** helper method with the ***periodSize*** parameter to set

larger periods (1m, 5m, 15m, 30m, 1h), all the strategies are organized by period at the top of app/Traits/Strategies.php so you can run just a subset of 1 hour strategies or only 5 minute strategies on 5 minute periods. Etc.

Another approach, that I will leave implementation of to the reader, is to take a subset of the strategies and combine them up in multiples of two and three and see how multiple strategies combined might stack up. In some of my testing I have done this with different strategies and candle indicators and signals trying various combinations to see what I have found. In one week I opened over 50,000 demo binary options testing various strategies and in part 3 I can share that data.

Here is a rough bit of code to get you started that loops through the strategies and combines then and tests them.

```
foreach ($strategies as $st1) {
    foreach($strategies as $st2) {
        if ($st1 <> $st2) {
            if (${$st1} > 0 && ${$st2} > 0) {} // long
            if (${$st1} < 0 && ${$st2} < 0) {} // short
            foreach ($strategies as $st3) {
                if ($st2 <> $st3 && $st1 <> $st3) {
                  if (${$st1} > 0 && ${$st2} > 0 && ${$st3}
> 0){}

                  if (${$st1} < 0 && ${$st2} < 0 && ${$st3}
< 0){}

                }
            } //*/ st3
        }
    }//*/ st2
}//*/ st1
```

Basic GDAX straddle bot

Note: As I was initially working on this, GDAX and Coinbase crashed, which seems to be happening with more frequency. Due to these disruptions time I had set aside to work on this part of the article and code was wasted so I will pick this back up in greater detail in Part 3.

— That said —

First we need to understand how GDAX works, that it is a real-time exchange and what "Time in force" means when opening an order on the GDAX exchange.

> *from the GDAX API docs*

> *Time in force policies provide guarantees about the lifetime of an order. There are four policies: good till canceled* `GTC`*, good till time* `GTT`*, immediate or cancel* `IOC`*, and fill or kill* `FOK`*.*

> `GTC` *Good till canceled orders remain open on the book until canceled. This is the default behavior if no policy is specified.*

> `GTT` *Good till time orders remain open on the book until canceled or the allotted* `cancel_after` *is depleted on the matching engine.* `GTT` *orders are guaranteed to cancel before any other order is processed after the* `cancel_after` *timestamp which is returned by the API. A* `day` *is considered 24 hours.*

> `IOC` *Immediate or cancel orders instantly cancel the remaining size of the limit order instead of opening it on the book.*

> `FOK` *Fill or kill orders are rejected if the entire size cannot be matched.*

We are most interested in the non-default order types for what we are doing.

Our GDAX bot has a couple requirements:

1. We do not want to have to 'manage' orders, we want them to have limited life, or be filled quickly without having to check them and/or close them.
2. "Good till time" scalping orders with very short windows.

Our GDAX scalper code is at

```
php artisan bowhead:gdax_scalper
```

#1- The data

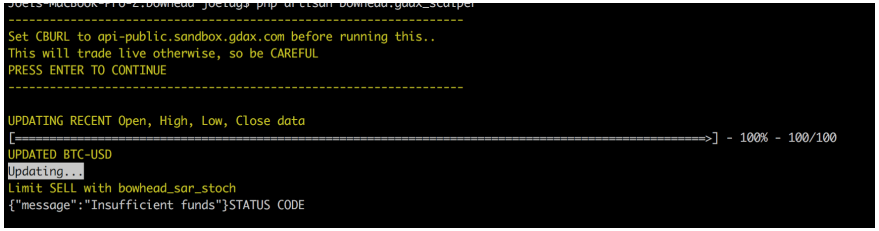
The first thing that the scalper does is to pull recent trades from GDAX, it snags our current list of positions (which we will not use in this example) and our current balances.

It updates the database with the pair 'BTC-USD' (which is distinct from 'BTC/USD' used on Bitfinex, it also utilizes a tick function to automatically call a data updater every thirty seconds or so.

#2-ordering

The ordering is all Good Till Time orders of one-minute, which will catch wicks and tails (shadows) in the candles.

Here is what the command will look like.



At the moment my money is tied up elsewhere.

You can find the code in

```
app/Console/Commands/GdaxScalperCommand.php
# in the fire() method
```

REST API

Setting up Bowhead as a REST API, this is not something you will need to ***do*** as Bowhead has it built in—Bowhead uses the Laravel package called 'Dingo' for handling it's REST API.

The key parts are.

RESTful interactions are such that we use HTTP GET/POST/PATCH and DELETE for 'data retrieval', 'creation', 'updates' and 'removals'—respectably. Bowhead has taken all of the API access to brokerages and market makers and placed them all behind an API which uses standardized calls to interact with each. So, opening a buy order on Bitfinex will be the exact same as opening a buy order on Poloniex, you just pass in which brokerage you want to open the order with.

This is powerful because it allows you to work on arrays of different sites and services to create applications that can work with many different services out of the box. Also, you are not stuck with only working in PHP with Bowhead, just set up the API keys and you can then use Bowhead as the interface for building your application in

Python, Node or Go—whatever language you want. All the heavy lifting is done behind the scenes so you can focus your energies on making trading apps.

Bowhead REST API endpoints for brokerages and sites. (work in progress)

- /api/account(s) the various account functions, balances, etc
- /api/market(s) information on instruments traded, market prices etc
- /api/position(s) information on open, closed or pending positions.

Bowhead REST API endpoints for Bowhead internal data. (work in progress)

- /api/signal(s) specific signals for instruments
- /api/candle(s) specific returns for candles for an instrument
- /api/indicator(s) specific returns for indicators for an instrument

Final words

Thanks for reading part 2, I have already started on part 3 and it should bring even more focus to using Bowhead with strategies and signals in a real environment as well as much more testing and logging your strategies. Additionally, I am hoping to have the REST API in a state

that we can provide a simple example of using it to open orders and test the API and use it via a web server or service like Valet. As well as finishing up the GDAX bot and having it in a better state.

I hope you find bowhead a useful tool to use to be able to write your own bots. I personally use it and am developing it for me and felt that because there was a fairly serious lack of real tutorials on creating cryptocurrency bots that I should open source my work and write up some tutorials for helping people along.

As always, PLEASE NOTE: Cryptocurrency and Forex trading constitute extreme risk, and automating that risk could mean that you compound your losses. Do not use money you cannot afford to lose. ALWAYS trade in DEMO mode with any strategy you build for a long time before you turn it loose with your money.

Continue to PART 3

Get social:

If you enjoyed this article, recommend by clicking on that heart icon on the left of this text, and feel free to share it, I work hard on these and feel they are useful.

Upvote this on Reddit please.

*I mention in my bio here that I "**Write articles I wish I found, instead of having to write**" and this is 100% the case. If there was an article about*

this topic online, I would NOT have taken the time to write it.

joel degan (@joeldg) | Twitter

The latest Tweets from joel degan (@joeldg). I like puzzles, games, beer and travel and any combinations of those...

twitter.com

Joel De Gan | LinkedIn

As Vice President of Technology I architected and built out the REST API and web systems used for RXMG for millions of...

www.linkedin.com

joel degan — Medium

Read writing from joel degan on Medium. I like puzzles, games, beer and travel and any combinations of those. Every day...

medium.com