GraphLinq Lite Paper

graphlinq.io

The automation of decentralized datas monitorization and external executions over multi-chains applications.

January 24, 2021

Disclaimer

THIS DOCUMENT PROVIDES AN INITIAL SUMMARY OF THE GRAPHLINQ PROJECT. AS THE PROJECT PROCEEDS, THIS DOCUMENT IS EXPECTED TO EVOLVE OVER TIME. THE GRAPHLINQ TEAM MAY POST MODIFICATIONS, REVISIONS AND/OR UPDATED DRAFTS UNTIL THE FINAL DOCUMENT IS PRESENTED PRIOR TO THE DATE OF THE PUBLIC BETA.

THIS WHITEPAPER SETS FORTH A DESCRIPTION OF THE PLANNED USE OF THE GRAPHLINQ TOKEN. THIS IS BEING PROVIDED FOR INFORMATION PURPOSES ONLY AND IS NOT A BINDING LEGAL AGREEMENT. THE GRAPHLINQ BETA WILL BE GOVERNED BY SEPARATE TERMS & CONDITIONS.

IN THE EVENT OF A CONFLICT BETWEEN THE TERMS & CONDITIONS AND THIS WHITEPAPER, THE TERMS & CONDITIONS GOVERN. THIS WHITEPAPER IS NOT AN OFFERING DOCUMENT OR PROSPECTUS, AND IS NOT INTENDED TO PROVIDE THE BASIS OF ANY INVESTMENT DECISION OR CONTRACT.

1. Introduction

1.1 The problem

In the current blockchain world, we have a lack of simple interface and friendly way to monitor our blockchains and information attached to them, without having special skills for reading or understanding the blockchain, it's complicated to create, or innovate for non-coders users.

Especially with the last major crypto evolutions, we need a lot of new tools to help and develop the ecosystem within decentralized finance. With the high cost of running nodes and executing transactions over mitigated or overloaded networks, we need solutions to read our chains infos efficiently and execute transactions smartly.

Over the past 10 years we had trouble doing blockchain easily to understand for users, and fast to build innovative projects, usable quickly in a production world moving really fast. We need to find multiple ways to create and handle the network flux of datas from our chains activities, to collect stats, users information and chain datas.

Similar solutions already exist for automatizing others type of companies, like creating a website, deploying a blog or a basic application, but for the crypto-world it all seems 'complicated' to get involved in and it's a major barrier for a lot of company that could tend to use the blockchain for their activities.

1.2 The Solution

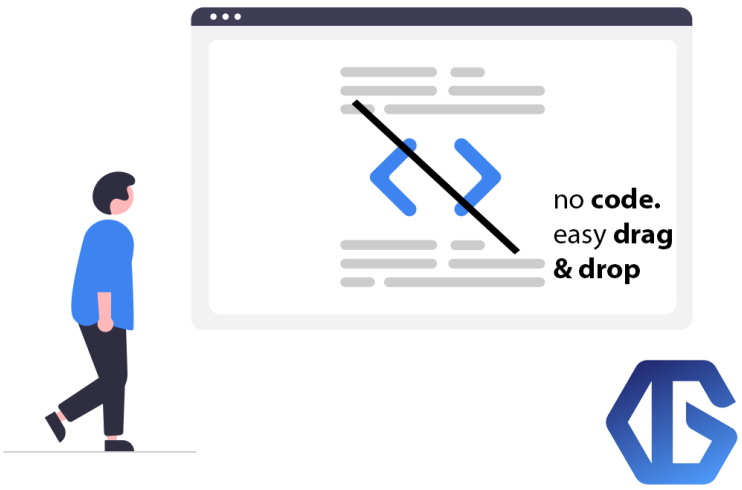
So GraphLinq is here to solve this problem, through an online interface, we create the ability of generating and mastering your own blockchain datas without the need of coding experience.

Imagine a platform on which anyone can create his own cryptocurrency with the least effort, or watch the activity of his smart-contract, generate a bot to handle subscriptions, or watch an **AMM*** pair activities, execute trades on centralized exchange based on decentralized datas, create **DEX*** arbitrage automatically through graphs executed on the GraphLinq engine.

With GraphLinq you can generate a set of nodes (blocks) that receive an input and output to a single/multiple other nodes, so you create with a set of tools your 'structure' of code with a path of execution that will be launched on the blockchain or the GraphLinq Engine, then you can deploy it on the test net Engine or the main net Engine, once you tested and want to get in production.

One graph can for example track network pairs activities on Binance and report stats to webhook, or slacks, discord, telegram, twitter with any conditions you decide to trigger a possible results of your nodes.

A new way of designing chain applications needs to be thought of, with the use of protocols like Graph Protocol or ChainLink we are already starting to see major projects going into that direction.



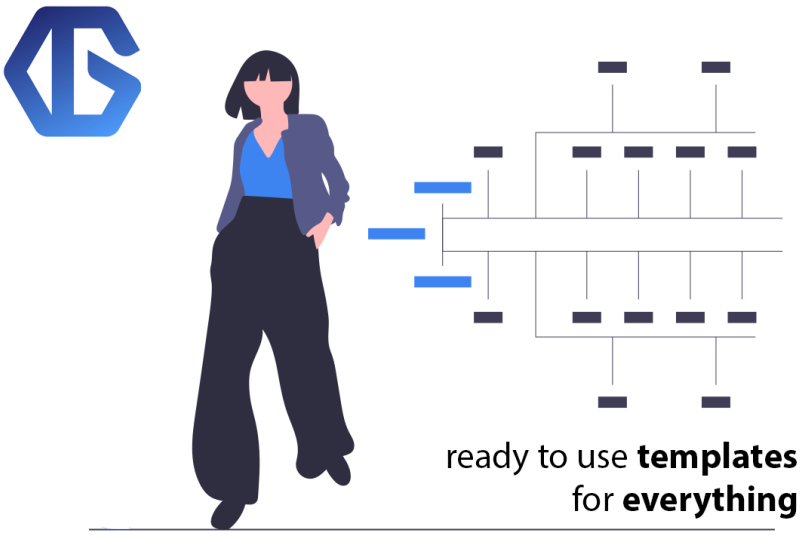
1.3 Conclusion & Our Vision

We believe that a lot of the job tasks runned on mainstream crypto projects can be automated and generated with node graphs, so that the datas can be listened to, triggered and saved to a safe place accessible for maintaining a service from an off-chain side.

While creating a way for innovative content creators to exploit the benefit of blockchains without getting stuck on technical matters, the goal of this way of thinking is mainly being able to reach all possible audiences on the Internet, to make the blockchain world accessible to every business.

GraphLinq remove the pain and struggle of chain implementation that can block new companies in the crypto sphere possibilities of expansions, while helping to connect on blockchains informations we also can propose to facilitate execution of centralized trades through API on nodes, bots for socials networks and even managing your entire asset through multiple graphs.

A graph can be created to watch a token activity, transactions, events and execute any available node orders type on the engine to send for example another version of an asset through a different type of chain or execute any third party, which mean that atomic swap can be made through one or multiple graphs, same as decentralized exchange, you can store your excess of information that will have a high cost and latency time on-chain, on the GraphLinq protocol off-chain base.



2. Technicals FOV

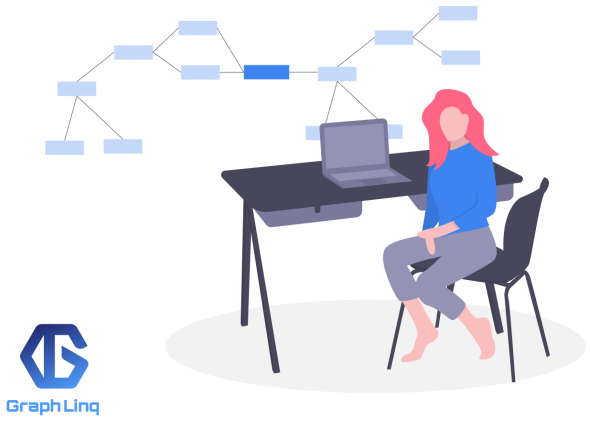
2.1 The Engine Perspective

From a coding perspective, the engine is developed and maintained into **.net core 3.1** (known as C# language) which allow for a fast execution and a cross-compatibility over any exploitation system.

The engine share the state of the differents network streams through his **threads*** which are actually graphs running over their own context separated from their own memory access to any others of the running graphs, then, the connections streams are shared through **singleton*** over the entire process to assure the integrity of the external datas (ethereum, other chains…) and are shared to every graph running on the engine.

Attributes are attached to and nodes linked to utilities, which means that any type of node (execution and monitoring) can be implemented quickly as the community asks for it.
A smart system is used for having the possibility to do interoperability over different libraries and blockchains through available **NuGet*** packages.

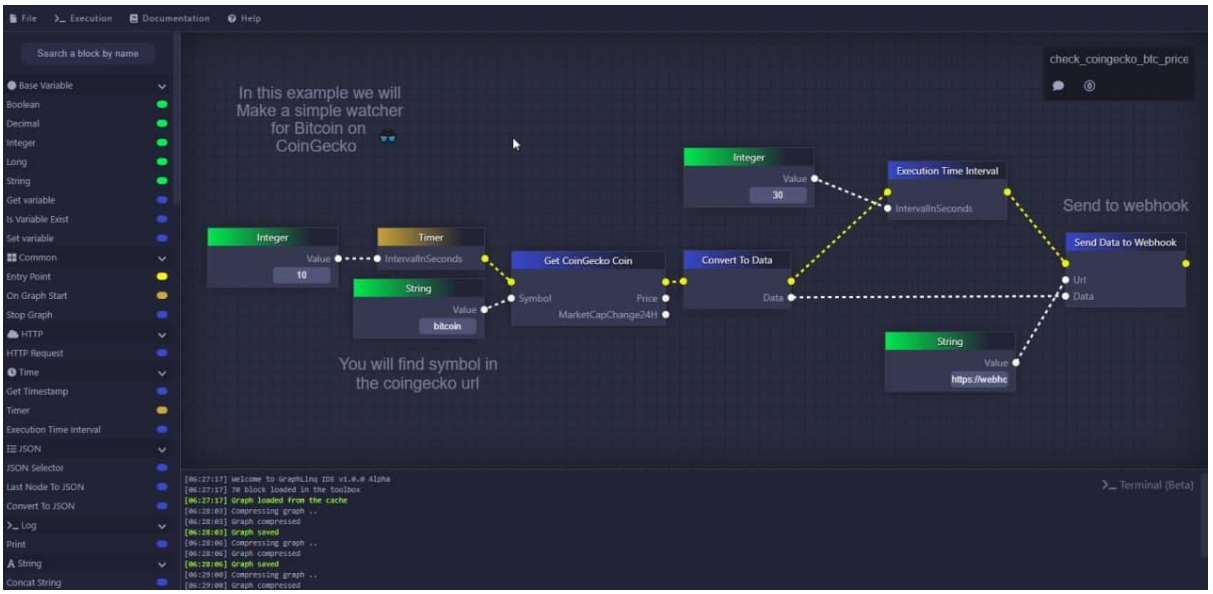
All the source code and information on the engine is completely open-source and available on our github, you can decide to run your self hosted GraphLinq engine, or use ours with the online interface to handle your entire workflow.



2.2 Graphs utilizations

All graphs can be created on any developed interface that generate a comprehensible set of bytes by our engine, all graphs are saved as **json*** instructions (nodes) that are compressed into raw bytes, an unique hash is also linked to your graph execution, which means that you can use this hash to follow up exactly your graph state and his execution, and manage it.

One graph has a path of execution and a path of instruction parameters, you need to link both accordly to have the pattern of each one correctly before deploying your graph. Here is a sample on our development interface of a graph that watch transaction of a specific ethereum smart contract and send any new events to a Telegram bot:



The yellow line represents here the execution pattern where the blank lines are the links parameters between nodes. Through our online interface you can create your own graph made of all the availables nodes in our documentations.

2.3 Nodes executions fees

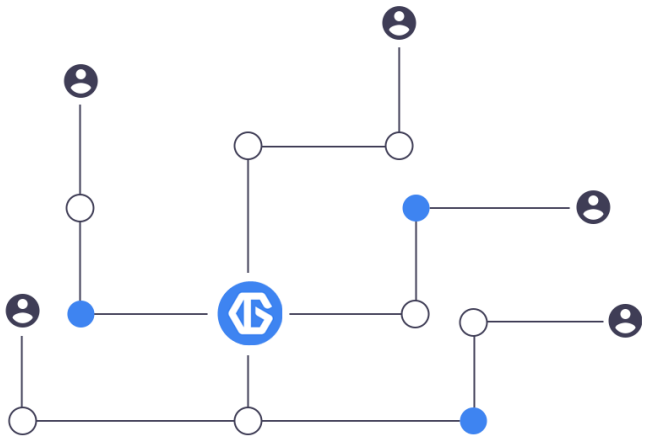
Graphs are running without interruption except if you don’t have set cycle, timer, or network flux streams (ex: connected to an Ethereum connection) then it will just execute your graph and exit once the output is returned. You can also schedule for multiple auto runs by a time in the day.

We are running the engine at our cost, maintaining the availability of the protocol which means that it has a cost of execution for each graph that is calculated by block price.

All specific block utilisations (EthConnector, TimerNode, HttpNode..) have a fixed really low cost fee for being executed into a graph that will update over time as the price of the token does major increases/decreases.

The amount of GraphLinq token used to maintain a graph is then used as fuel to run the different started graphs, all of the token spent for the running cost will be burned from the total supply which will reduce the token total amount with time, we will maintain a fixed level of dollars worth of execution.

The estimated price for the execution of one graph is available through our online interface, or you can calculate it manually by cumulating all the nodes prices from your graph and getting it.
We do this to prevent flood or overload of the network and to assure a high availability of your graph and the GraphLinq protocol.



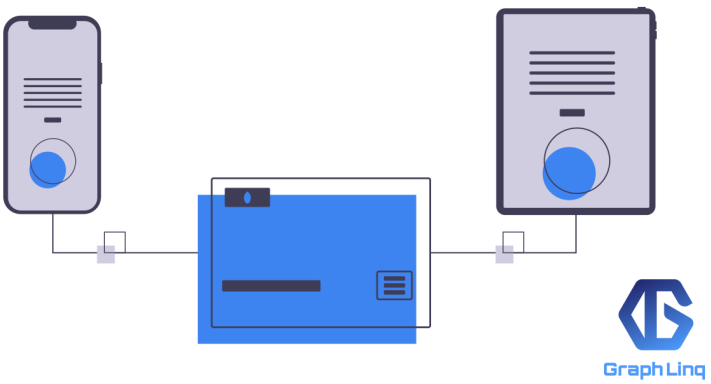
2.4 The GraphLinq Wallet

To use our platform and the engine protocol you need to have a Metamask (or web3 compatible) wallet loaded on your browser to connect on our web interface, by signing an unique ethereum transaction, we authenticate your wallet and authorize it to use the online interface. Once you sign up and that the session is automatically started into the online application (through **JWT***), you need to get some GLQ token.

Once you have tokens up and ready on your wallet to use, you need to deposit the amount you want in the specific smart-contract that manages the cloud balance (it works like any online cloud paid services).
You will be able to withdraw from the contract at any time contacting our API, minus the execution cost of the graph you already started on the Mainnet engine.

You can start deploying your graphs on the protocol, the test net engine cost no fees so it can be tested there for free but have limited possibilities and activities (as of execution time)

The balance within the smart-contract that handles the execution cost will automatically use the amount needed to fuel the engine for any graphs running on the Mainnet, once you stop or pause a graph, you wont get any cost for saving it into the protocol as long it’s offline.

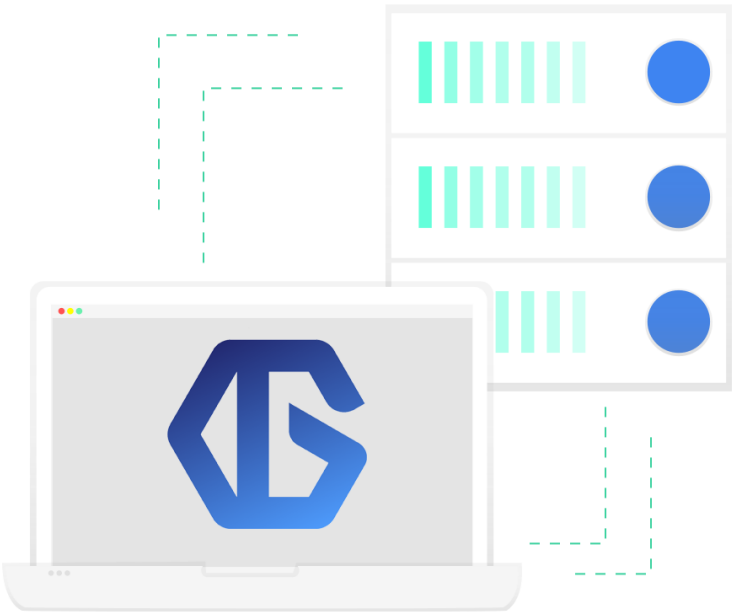


2.5 The architecture

The engine infrastructure is deployed as Kubernetes pods running through multiple servers based on **AWS*** clouds, which mean that it runs 24h/24 and has high availability access and 99.99% uptime.

A rest HTTP API is exposed from all the running engine nodes to accept incoming requests, to start/stop, deploy or remove a graph, you can manage a graph on POST request with the hash of your graph and key of your wallet to access the graph and manage its state.

You also have the possibility to run your own local engine if you want to manage it fully and handle the server's architecture as your graph execution. By time GraphLinq will be pushed to go on a fully decentralized way of doing the engine and transfer the ERC-20 asset to an engine-chain Tendermint based.



Conclusion

GraphLinq aims to be the leader in facilitating the way of communicating and manipulating blockchain datas. It gives people a way to set up their crypto pipeline and workflow, easily through a fast and easy to use interface.

We are a base team of three software engineers, one community manager passionate since years on blockchains and we struggled a lot on issues like reaching information on blockchains, executing simple tasks, doing bot creation...

Mainly, GraphLinq is made to create, update, and view blockchains datas and third party like exchanges, but it can be even more than that:
A marketplace for buying and selling your scripted graphs can be developed if the community has a need for it, a full-chain and so much more features helping the accessibility of chain information and the deployment of workflows.

At your graphs!

Marks

* **AMM:** An automated market maker (AMM) is a type of decentralized exchange (DEX) protocol that relies on a mathematical formula to price assets. Instead of using an order book like a traditional exchange, assets are priced according to a pricing algorithm.

* **DEX:** Decentralized exchanges (DEX) are a type of cryptocurrency exchange which allows for direct peer-to-peer cryptocurrency transactions to take place online securely and without the need for an intermediary.

*** THREADS:** A thread is the unit of execution within a process. A process can have anywhere from just one thread to many threads. It helps for multi execution over a CPU utilization.

*** NUGETS:** Provides the tools developers need for creating, publishing, and consuming packages. Most importantly, NuGet maintains a reference list of packages used in a project and the ability to restore and update those packages from that list.

*** SINGLETON:** In software engineering, the singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system.

*** JSON:** JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on object syntax. It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).

*** AWS:** Amazon Web Services is a cloud computing platform that provides customers with a wide array of cloud services. We can define AWS (Amazon Web Services) as a secured cloud services platform that offers compute power, database storage, content delivery and various other functionalities.

*** JWT:** JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties. This information can be verified and trusted because it is digitally signed..