Aladdin Oracle Technical Architecture analysis

Oracle scalability definition

The Oracle Problem is defined as the security, authenticity, and trust conflict between third-party oracles and the trustless execution of smart contracts. The digital world needs to know about the physical world.

To solve the issue with scalability we can create decentralized oracle system.

We can provide an intriguing decentralized solution to authenticating the data from oracles and the subsequent output data of smart contracts. Centralized oracle feeds as a single point of failure and offers a solution through a “middleware” comprised of a decentralized oracle network.

Nodes definition, reputation, order matching and aggregating

We need to have specific nodes that reply to data queries made by contracts:

- Reputation Contract
- Order-Matching Contract
- Aggregating Contract

Miguel De la Cruz
migueldelacruz.com

For business inquiries: miguelenru@outlook.com

The *Reputation Contract* uses a proprietary method for storing and tracking oracle service provider metrics.

The *Order-Matching Contract* takes a *service level agreement* (SLA) and logs the data parameters of the SLA while simultaneously taking bids from oracle providers.

The *Aggregating Contract* collects oracle provider's responses and calculates the final collective result of the initial ChainLink query.

Aggregating the oracle data provided by multiple sources helps ensure a more accurate view of the data supplied, reducing the reliance on a single entity (oracle). Oracle provider metrics are also fed back into the reputation contract for managing oracle accuracy through an incentive-driven reputation system.

SLA oracle selection process

The use of the SLA is vital for the oracle selection process. Users requesting oracle data can identify explicitly the parameters and inputs they are looking for as well as how many oracles they would like to use. The reputation of oracle providers can also be added into the SLA proposal.

We can describe the next points for the system:

- ❖ A pool of oracle providers is eventually selected by our system who are notified of the task required.
- ❖ Providers (which are off-chain) subsequently report the requisite data on-chain.
- ❖ The resulting data is fed into the aggregating contract where a weighted answer is calculated.

- ❖ The weighted response is returned to the specific smart contract function as the trigger for the contract's relative execution.
- ❖ Further, the provided data accuracy from the oracles is fed into the reputation contract as part of the larger reputation system.

Aladin native token and DApp Oracle integration

Aladin should have a native token that is used for compensating oracle providers that supply accurate information. The architecture of the platform also consists of oof-chain components including external adapters, subtask schemas, and a core node software for interfacing with the blockchain or other systems.

- ❖ The Smart Contracts utilized by the DApps will call a function of the Oracle. DApp devel.
- ❖ The Smart Contracts utilized by the DApps will call a function of the Oracle.

DApp developer will provide the parameters like API endpoints and they would also need to provide the name of the field that they require in each API, and the data format of that API. The Aladin oracle will be as generic as possible, but for handling the different data types of the parameters, we will provide different interfaces/adapters which will help us to bring data from the data providers.

The number of interfaces/adapters will be dependent on the use cases/verticals that we want to cover under Aladin oracle. Adapters would enable the Aladin network to have easy integration of custom computations and specialized APIs. Adapter would be needed in case of a special operation for e.g A Smart Contract has made request to the APIs which gives the response in the numeric form and comparison operations like equals, not equal, greater than, less than, greater than equal to and less than have to be performed on that data, in that particular scenario adapter would be used and if in case API requests are made, which return boolean result then a separate adapter for boolean operations will be needed to perform the task and likewise. Therefore there would be separate adapters for different use cases.

Authentication methods

We can describe authentication methods:

- Login/Password
- OAuth
- OAuth 2
- Biometrics
- Two-factor authentication [2FA]

For all of these methods, we should *define security store to save sensitive user information.*

Aladin oracle will be able to fetch the data from the secured API endpoints (https) as well as IoT endpoints and bring them onto the network for the DAPPs to use.

File Transfer Protocols for Businesses

List of different protocols. Each of them will have an impact on your project and needs to be implemented for coverage and 3rd party implementation. The simplest and older ones, like FTP, HTML and SCP have a long history and development community, although lacks on security, when FTPS, HTTPS, SFTP, etc. will have security but then more development complexity.

For the unsecured protocols a custom DNS server instance is recommended for ensuring security.

FTP (File Transfer Protocol)

When it comes to business file transfers, FTP is probably the first that comes to mind.

FTP is built for both single file and bulk file transfers. The large community and development life ensures that likely won't have problems with interoperability.

This old system as common raises the probability that the trading partner will be able to exchange information through it. Many client application exists so the end-users will easily interoperate.

FTP downsides and issues

File transfer protocol is not so strong on security. Hence, if you need to comply with data security/privacy laws and regulations like HIPAA, PCI-DSS, SOX, GLBA, and EU Data Protection Directive, stay away from it. Choose FTP if your business is NOT or does NOT:

Operate in a highly regulated industry like healthcare, finance, or manufacturing;

Send/receive sensitive files; or
Publicly traded (hence governed by SOX).

Another problem with FTP is its susceptibility to firewall issues, which can adversely affect client connectivity. Read Active v.s. Passive FTP Simplified to understand the problem and learn how to resolve it.

HTTP (Hypertext Transfer Protocol)

Like FTP, HTTP file transfer is a widely used protocol for business file transfers. It's easy to implement, especially for person-to-server and person-to-person file transfers (read Exploring Use Cases for Managed File Transfer for reference). Users only need a Web browser like Chrome, Firefox, Internet Explorer, or Safari, and they'll be ready to go.

No installation needed on the client-side.

HTTP is also less prone to firewall issues (unlike FTP). However, like FTP, HTTP by itself is inherently insecure and incapable of meeting regulatory compliance or securing data. Use HTTP if (lack of) security is not an issue for you.

FTPS (FTP over SSL)

FTP has FTPS, while HTTP has HTTPS. Both are protected through SSL. If you use FTPS, you retain the benefits of FTP but gain the security features that come with SSL, including data-in-motion encryption as well as server and client authentication. Because FTPS is based on FTP, you'll still be subjected to the same firewall issues that come with FTP.

Organizations in the Legal, Government, and Financial Services industry might want to consider FTPS as an option.

HTTPS (HTTP over SSL)

As mentioned earlier, HTTPS is the secure version of HTTP. If you don't like having to install client applications for your end-users and most of your end users are non-technical folks, this might be a perfect choice. It's secure and very user-friendly compared to FTP/S.

SFTP (SSH File Transfer Protocol)

Here's another widely used file transfer protocol that's perfect for businesses who require privacy/security capabilities. SFTP runs on SSH, a secure protocol that - like SSL - supports data-in-motion encryption and client/server authentication. The main advantage of SFTP over FTPS (which is usually compared to it) is that it's more firewall-friendly.

Pros: Firewall-friendly
Cons: Less user-friendly than HTTPS

SCP (Secure Copy)

This is an older, more primitive version of SFTP. It also runs on SSH, so it comes with the same security features. However, if you're using a recent version of SSH, you'll already have access to both SCP and SFTP. Since SFTP has more functionality, I would recommend it over SCP. The only instance you'll probably need

SCP is if you'll be exchanging files with a company who only has a legacy SSH server.

WebDAV (Web Distributed Authoring and Versioning)

Most of the file transfer protocols we've discussed so far are primarily used for file transfers. Here's one that can do more than just facilitate file transfers. WebDAV, which actually runs over HTTP, is mainly designed for collaboration activities. Through WebDAV, users won't just be able to exchange files. They'll also be able to collaborate over a single file even if they're (the users) working from different locations. WebDAV is probably best suited for organizations who need distributed authoring capabilities, e.g. universities and research institutions.

Pros: Not only file transfer, can run over HTTP, excellent for collaboration

Cons: Do not apply to this project except if there is a conditional payment involving collaboration

WebDAVS

WebDAVS is a secure version of WebDAV. If WebDAV runs over HTTP, WebDAVS runs over HTTPS. That means, it exhibits the same characteristics of WebDAV, plus the secure features of SSL.

Pros: Same as WebDav

Cons: Complexity. Specific for collaborative projects as WebDavs

TFTP (Trivial File Transfer Protocol)

This file transfer protocol is different from the rest in that you won't be using it for exchanging documents, images, or spreadsheets. In fact, you normally won't be using this for exchanging files with machines outside of your network. TFTP is better suited for network management tasks like network booting, backing up configuration files, and installing operating systems over a network.

We do not recommend this system

AS2 (Applicability Statement 2)

AS2 is built for EDI (Electronic Data Interchange) transactions, the automated information exchanges normally seen in the manufacturing and retail industries. EDI is now also used in healthcare, as a result of the HIPAA legislation (read Securing HIPAA EDI Transactions with AS2). If you operate in these industries or need to carry out EDI transactions, AS2 is an excellent choice.

Pros: Designed for B2B so development and integration is solid

Cons: Market-specific, complex development, a small community

OFTP (Odette File Transfer Protocol)

Both OFTP and AS2 are inherently secure and even support electronic delivery receipts, making them perfect for B2B transactions.

Pros: Designed for B2B so development and integration are solid. Large European community

Cons: Market-specific, complex development.

AFTP (Accelerated File Transfer Protocol)

WAN file transfers, especially those carried out over great distances, are easily affected by poor network conditions like latency and packet loss, which result in considerably degraded throughputs. AFTP is a TCP-UDP hybrid that makes file transfers virtually immune to these network conditions. If you want to see the big difference AFTP makes, read the post Accelerated File Transfer In Action.

Aladin oracle would be able to give the data of Aladin blockchain to off-chain. We will accomplish this with the help of the APIs which will fetch the data from the smart contracts of the third-party developers on the Aladin network and provide it to the outside world. Oracle will make a request to the getter function of the smart contract which was used to store the data on the blockchain network and as soon as it gets a response from the smart contract, the data can be given to off-chain with the help of an API. Any developer or someone else who wants data off-chain will have to then call this API in order to access the data.

> *The smart contract will process requests from Aladin. The question arises where smart contracts will be launched. If smart contracts will be on the side of the user, somehow provide protection for data exchange.*

The channel (smart contract requesting API data from oracle) would be fully secure, means no malicious party can manipulate the data. Block Producers' nodes will be having a Trusted Execution Environment. Intel's recent Software Guard eXtensions (SGX) or ARM's TrustZone are good examples of TEEs. SGX allows an application to be executed in an environment called an enclave that confers hardware protection on user-level code. First, enclaves protect the integrity of the application (its data, code, and control flow) against subversion by other processes. Second, an enclave protects the confidentiality of an application, meaning that its data, code, and execution state are theoretically opaque to the rest of the operating system. It can however read and write memory outside the enclave region. SGX seeks to protect enclaved applications even against a malicious operating system, and thus against even the administrator of the node on which an application is running. Running an oracle in an enclave and distributing attestations can provide strong assurance that the oracle is executing a particular application because it can be proven remotely that a system is running a legitimate SGX system. Block producers would be required to have this hardware.

Pros: Security

Cons: Difficult implementation, needs specific hardware if using SGX

SGX and malware risks

Unfortunately, SGX is also a prime weapon for use in malware. For better or worse, it currently looks like Intel will not be giving the option for ‘trusted anti-malware vendors’ to access the contents of enclaves to make sure they are safe. Thus, malware can, in principle, freely create enclaves to prevent the operating system/hypervisor/ anti-malware from knowing what it is executing. Coupled with ubiquitous connectivity, the spectre of small loaders downloading sophisticated packages of malware remotely via an encrypted link rears its head.

The Aladin oracle would be deployed in the secure node as mentioned above. When the Aladin oracle takes any input from the smart contract or sends the request to any data provider that channels would be secure and impregnable. The data provided by data providers will be sent to the oracle and the oracle after aggregation and processing will pass the required result to the smart contract.

- There is no way to evaluate the security of interaction
- The smart contract once deployed cannot be modified.

A good point will provide versions for Smart Contract with back compatibility. The most important case upgrade smart contract with confirmation user:

- Smart contract developer release version 1.0
- The user confirms manifest permissions + User Agreement
- Smart contract developer release version 1.1
- User can decline updated version and will use old smart contract
- User can update the smart contract with confirmation user User Agreement

Also manipulating the input and output data will be possible only if the changes are made in the software package which is present in the block producer node. And If the package is changed by any block producer then in order for its changes to reflect, they will have to release a software patch with the changed software and hence other block producers will come to know about that malicious actors and then they can be taken out of the network.

The Aladin oracle can take up to 5 APIs in a single call. The Aladin oracle will work whether any smart contract will request a single API or if any smart contract will request multiple APIs at a time. Once the responses from all the data providers have arrived, they will be sent back to the aggregation contract for deriving the required results from it. By consuming data from multiple sources, the contract is able to have additional assurance that the data provided is indeed accurate. For eg. Data which will be received from multiple data sources will be compared. If the majority of the APIs give similar response then only the result will be considered as correct.

Suppose if we make a call to 5 APIs and 3 of them don't provide similar response or timeout then the request will have to be made again. This will help us to eradicate the chance of the data provider corruption. It is important to note that multiple data sources for the same information are included to prevent a single point of failure risks.

5 API in one call, no explanation is given for choosing this number 5.

If there is 6, will something change?
What should be update in the system to make call N requests ? WHat is maximum of N ?

If the oracle node does not respond, then the request will be sent to another node.

Top 21 block producer would be active as an oracle node all the time, but requests will be sent to separate node every time. Oracle is deployed on all the top 21 Block producers so any block producer from top 21 is able to handle the input request coming from smart contract. No one node receives consecutive requests.

There is a risk that the top 21 block producer may be heavily loaded to perform additional operations
You must define the definition function of the Top 21 block producer.

For example:
MEM_MAX = max (BlockProducerMem_i), i = 1, N
CPU = max (BlockProducerCPU_i), i = 1, N
NET_MAX = max (NetworkAVALIABILITY_i), i = 1, N

BEST21 = max (InstanceMEM_i / MEM_MAX + InstanceCPU_i / CPU_MAX + NET_INSTANCE_i / NET_MAX), get top 25

The block explorer will show the metadata for the audit trail when the final result arrives from the oracle. The Audit Trail will have information about the oracle functions called from the smart contract in human-readable form. Here the flow would go like this, the smart contract would send the request to oracle node, oracle node would pass this request to the data provider, data provider will send the result to the oracle, oracle processes the results (aggregation) and the aggregated results will be sent by the oracle to the smart contract audit trail would include Transaction ID, Smart Contract Address, Name of Function of Smart Contract Called, API URL.

Data provider current infrastructure and risks

Integration with 3º party API - High risk

Creating the right structure for 3º party API will ensure high speeds to get data and security measures where user and the system will be safe from losing sensitive user data.

Data security is not defined and therefore is no information about the protection of private data and tokens. When using 3º party API there is a use of login-pass or token. When a user goes to any Google App (Gmail/GDrive/GAnalytics, etc), the system will get the Google API for the analytics data, for that user of this system will auth in Google, the user will provide permissions to the system and this needs webhooks implementation, not defined in the original documentation, as well the app token and user token interaction or where is stored and how is encrypted is not determined. This is a clear case of high risk.

High risk: No information where System will store privacy data. I did not find any information about encryption for 3rd party access tokens.

Risks:

- The system will be down
- The system will be banned
- The system will lose sensitive user data
- The system will not handle all user data
- The system will not reprocess user data

To integrate different API system should know how the system connects, store and encrypt:

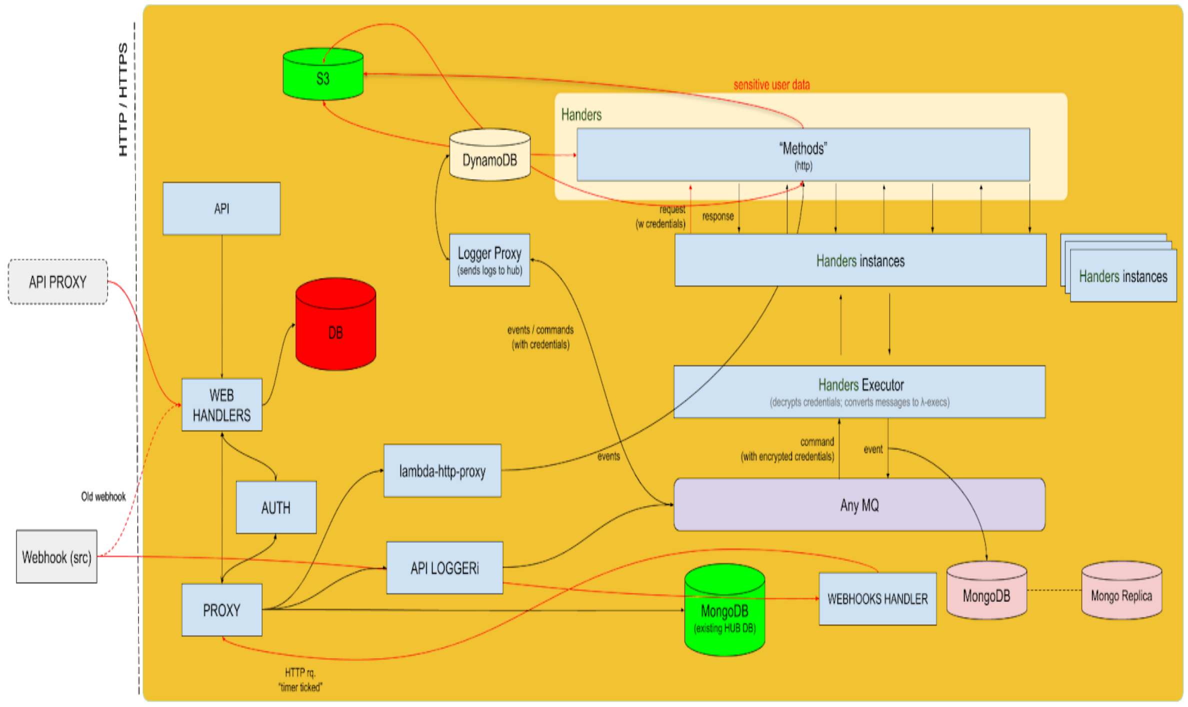
- To Get data
- To Save data
- To Handle data

Solutions:

- **To get data**, user should provide credentials for the system example google.com
- **To get data** from google.com user should provide credentials and OAuth tokens [OAuth token recommended]
- **To get OAuth tokens System,** should handle webhooks or handle requests from google.com.

This infrastructure will allow restructuring how 3º party API interactions are performed and will ensure security and scalability.

Architecture proposal



Smart Contract

Smart Contract functionality needs to be three things:

- ❖ Deterministic.
- ❖ Terminable.
- ❖ Isolated.

Deterministic

A program is deterministic if each time it gives the same output to a given input. For example. If 3 + 1 = 4, then 3 + 1 will ALWAYS be 4 (assuming the same base). Therefore, when a program produces the same output on the same set of inputs on different computers, the program is called deterministic.

There are various points when a program can act in a non-deterministic way:

- Calling un-deterministic system functions: When a programmer calls an un-deterministic function in their program.
- Un-deterministic data resources: If a program acquires data during runtime and that data source is un-deterministic then the program becomes un-deterministic. Eg. Suppose a program that acquires the top 10 google searches of a particular query. The list may keep changing.
- Dynamic Calls: When a program calls the second program it is called dynamic calling. Since the call target is determined only during execution, it is un-deterministic in nature.

Feature #2: Terminable

In mathematical logic, we have an error called “halting problem”. Basically, it states that there is an inability to know whether or not a given program can execute its function within a time limit. In 1936, Alan Turing deduced, using Cantor’s Diagonal Problem, that there is no way to know whether a given program can finish in a time limit or not.

This is obviously a problem with smart contracts because, contracts by definition, must be capable of termination in a given time limit. There are some measures taken to ensure that there is a way to externally “kill” the contract and do not enter into an endless loop which will drain resources:

- Turing Incompleteness: A Turing Incomplete blockchain will have limited functionality and not be capable of making jumps and/or loops. Hence they can’t enter an endless loop.
- Step and Fee Meter: A program can simply keep track of the number “steps” it has taken, i.e. the number of instructions it has executed, and then terminate once a particular step count has been executed. Another method is the Fee meter. Here the contracts are executed with a pre-paid fee. Every instruction execution requires a particular amount of fee. If the fee spent exceeds the pre-paid fee then the contract is terminated.
- Timer: Here a pre-determined timer is kept. If the contract execution exceeds the time-limit then it is externally aborted.

Feature #3: Isolated

In a blockchain, anyone and everyone can upload a smart contract. However, because of this the contracts may, knowingly and unknowingly contain virus and bugs.

If the contract is not isolated, this may hamper the whole system. Hence, it is critical for a contract to be kept isolated in a sandbox to save the entire ecosystem from any negative effects.

Now that we have seen these features, it is important to know how they are executed. Usually, the smart contracts are run using one of the two systems:

- Virtual Machines: Ethereum and Neo use this
- Docker: Fabric uses this.

We can compare these two and determine which makes for a better ecosystem. For simplicity’s sake, we are going to compare Ethereum (Virtual Machine) to Fabric (Docker).

Virtual Machines provide better Deterministic, terminable and isolated environment for the Smart contracts.

	Virtual Machines	**Docker**
Deterministic	The contracts have no un-deterministic functions and the data is limited to on-chain information only. However, it executes dynamic calls which can be un-deterministic in nature. Thankfully the data accessible is deterministic	Because of the design of the docker, the system is reliant on users to create contracts which are deterministic. That is not really the best of solutions.
Terminable	Ethereum uses the "Fee-meter" for termination. Every step in the contracts costs "gas" and once the gas cost exceeds the pre-paid fee, the contract is killed.	Fabric uses the timer. However, since the timer can change from node to node because each node has its own computational power there is a risk to the consensus process.
Isolated	Has good isolation properties.	Is namespace-reliant and not capable of proper isolation

Ok, so now we know what smart contracts are and the fact that Virtual machines are better platforms for smart contracts. Let's look at what exactly do Dapps require to run efficiently.

The team chosen SGX technology to run smart contract application

Intel® Software Guard Extensions (Intel® SGX) is a name for Intel Architecture extensions designed to increase the security of software through an "inverse sandbox" mechanism. In this approach, rather than attempting to identify and isolate all the malware on the platform, legitimate software can be sealed inside an enclave and protected from attack by the malware, irrespective of the privilege level of the latter.

So in short this means we can create a secure enclave (or a Trusted Execution Environment – TEE – if you wish) at the CPU level which is protected from the OS upon which it is running.

Architecturally Intel SGX is a little different from ARM TrustZone (TZ). With TZ we often think of a CPU which is in two halves i.e. the insecure world and the secure world. Communication with the secure world occurs from the insecure world via the SMC (Secure Monitor Call) instruction. In Intel SGX model we have one CPU which can have many secure enclaves (islands)

Uses of SGX

Now that we know a little more about the SGX technology, it is worth taking a look at how people might use it. As is so often the case, uses range from the good to the bad, and, alas, the downright ugly.

The good

In the right hands, SGX can be a very powerful tool, assuring privacy and protection from malware even when running on an insecure system. For example, running a web browser inside an enclave would prevent even privileged malware from gaining easy access to all your information (though malware can still simply take snapshots of the rendered window). Enclaves would make it harder for malware to take key ring passwords out of memory. VMs could use enclaves to prevent the hypervisor viewing some critical information that only gets decrypted after attesting to a remote server. Video games could put most of their logic code inside an enclave in an attempt to stop some forms of wallhacks/aimbots/etc. Kernels could be made massively more resistant to tampering and hooking. The possibilities are endless.

The bad

Unfortunately, SGX is also a prime weapon for use in malware. For better or worse, it currently looks like *Intel* will not be giving the option for ‘trusted anti-malware vendors’ to access the contents of enclaves to make sure they are safe. Thus, malware can, in principle, freely create enclaves to prevent the operating system/hypervisor/anti malware from knowing what it is executing. Coupled with ubiquitous connectivity, the spectre of small loaders downloading sophisticated packages of malware remotely via an encrypted link rears its head.

On the bright side, as enclaves are not able to handle exceptions inside themselves, anti malware products might still be able to determine if there is malware running inside them from file IO and other IO. Furthermore, operating systems could choose only to give whitelisted programs permission to create an enclave from the enclave creation API. However, should a piece of malware successfully burrow down to Ring 0, the entire range of SGX functionality would become available to the malware author.

Unfortunately the team did not define any information about system limitation to run smart contracts. If Aladin system will run smart contract inside thier instances system should control how much memory smart contract uses and how long smart contract uses Aladin resources. In this case Aladin should introduce sandbox system to controll CPU, Memory, Timeusage good point for that BSD Jails system.

Block Explorer

This is specific log management system we can describe some time of possible architecture with already exists 3rd party systems.

Architecture

Project Hatohol develops a Fluentd plugin Hatohol output plugin. It pushes received messages to AMQP broker. Hatohol HAPI JSON pulls messages from AMQP broker and registers each message as an event. Hatohol HAPI JSON uses RabbitMQ as AMQP broker.

Zabbix uses Zabbix agent to collect logs. Hatohol uses Fluentd plugins to collect logs. For example, Hatohol uses tail input plugin for collecting Apache logs.

Zabbix agent sends logs on non secure connection. If you want to use secure connection, you need to create a secure tunnel by stunnel by hand. Hatohol can use secure connection for sending logs by secure_forwared input/output plugin.

Zabbix monitors logs on Zabbix server. Hatohol monitors logs on Fluentd with some Fluentd plugins. For example, Hatohol uses grep output plugin for selecting target log by regular expression.

You can't select log monitoring system architecture with Zabbix. You always need to select logs on Zabbix server.

Here is the log monitoring system architecture with Zabbix:

```
+--------------+ +--------------+ +--------------+ Monitoring
|Fluentd       | |Fluentd       | |Fluentd       | target
|with plugins  | |with plugins  | |with plugins  | nodes
+--------------+ +--------------+ +--------------+
collects,        collects,        collects,
selects and      selects and      selects and
pushes logs      pushes logs      pushes logs
|                     |                |
| secure connection   |                |
|                     |                |
\/                    \/               \/
+---------+
|RabbitMQ|
+---------+
|
| secure connection
|
\/
+----------+
|Hatohol   |
|HAPI JSON|
+----------+
pull logs
```

You can select log monitoring system architecture from some architecture patterns with Hatohol. Because Fluentd supports forwarding logs.

Here is a log monitoring system architecture with Hatohol for system that each monitoring target node has idle CPU:

```
+-------------+ +-------------+ +-------------+ Monitoring
|Fluentd      | |Fluentd      | |Fluentd      | target
|with plugins | |with plugins | |with plugins | nodes
+-------------+ +-------------+ +-------------+
collects,       collects,       collects,
selects and     selects and     selects and
forwards logs   forwards logs   forwards logs
|                     |               |
| secure connection   |               |
|                     |               |
\/                    \/              \/
+--------------------+
|Fluentd             | AMQP producer node
|with Hatohol plugin |
+--------------------+
receives all selected logs and pushes them
|
| secure connection
|
\/
+--------+
|RabbitMQ|
+--------+
|
| secure connection
|
\/
+---------+
|Hatohol  |
|HAPI JSON|
+---------+
pull logs
```

Here is a log monitoring system architecture with Hatohol for system that each monitoring target node doesn't have idle CPU:

```
+-------------+ +-------------+ +-------------+ Monitoring
|Fluentd      | |Fluentd      | |Fluentd      | target
|with plugins | |with plugins | |with plugins | nodes
+-------------+ +-------------+ +-------------+
collects and    collects and    collects and
forwards logs   forwards logs   forwards logs
|                     |               |
| secure connection   |               |
|                     |               |
\/                    \/              \/
+-------------+ +-------------+
|Fluentd      | |Fluentd      | Log select nodes
|with plugins | |with plugins |
+-------------+ +-------------+
selects and     selects and
forwards logs   forwards logs
|                     |
| secure connection   |
|                     |
\/                    \/
+--------------------+
|Fluentd             | AMQP producer node
|with Hatohol plugin |
+--------------------+
receives all selected logs and pushes them
|
| secure connection
|
\/
+--------+
|RabbitMQ|
+--------+
|
| secure connection
|
\/
+---------+
|Hatohol  |
|HAPI JSON|
+---------+
pull logs
```

Global Risks analysis

High risk for data provider: No information where System will store privacy data. I did not find any information about encryption for 3rd party access tokens.

Risks:

1. The system will be down
2. The system will be banned
3. The system will lose sensitive user data
4. The system will not handle all user data
5. The system will not reprocess user data

High risk for smart contract: usage sandbox for malware botnet

Risks:

1. The system will be usage for botnet
2. Without definition manifest and limitations for Memory Usage/CPU Usage/Time usage: price for instances will be huge.
3. Network protocol missing [VPN is the best way to protect communication]

High risk for block explorer: providing in logs sensitive user data. For example some urls includes in get arguments tokens and other system information because by default in https connection no one can review GET params.