Versatus: A Tech Stack to Power a Decentralized Web

By: Andrew Nicholas Smith & Matthew Geddes
Organization: Versatus Labs
Email: Andrew Nicholas Smith <as@versatus.io>, Matthew Geddes <mg@versatus.io>
Organization Email: info@versatus.io
Website: versatus.io

Abstract

The modern digital era demands internet infrastructure that is both resilient and free from centralized control and vulnerability. Unfortunately, today, the web is captured by a small group of centralized mega-corporations with incentives that differ from the users of the web. This paper introduces the Versatus network. Versatus is a pioneering architecture designed to deliver on the promises of what has been dubbed “web3”. Versatus combines the speed and security of contemporary internet stacks with the benefits of decentralization and censorship-resistance. At the core of the architecture are three innovative layers: a cutting-edge peer-to-peer networking layer, a versatile application specific nano runtime, and an executable oracle that simulates a stateless rollup for seamless on-chain, off-chain and cross-chain integration. Together, these components offer a holistic solution addressing the challenges of modern internet usage, from communication protocols to decentralized compute and smart contract execution. With its forward thinking design and emphasis on flexibility and extensibility, the Versatus network stands as a beacon for the future of the web, promoting an open, secure, and inclusive digital landscape.

Introduction

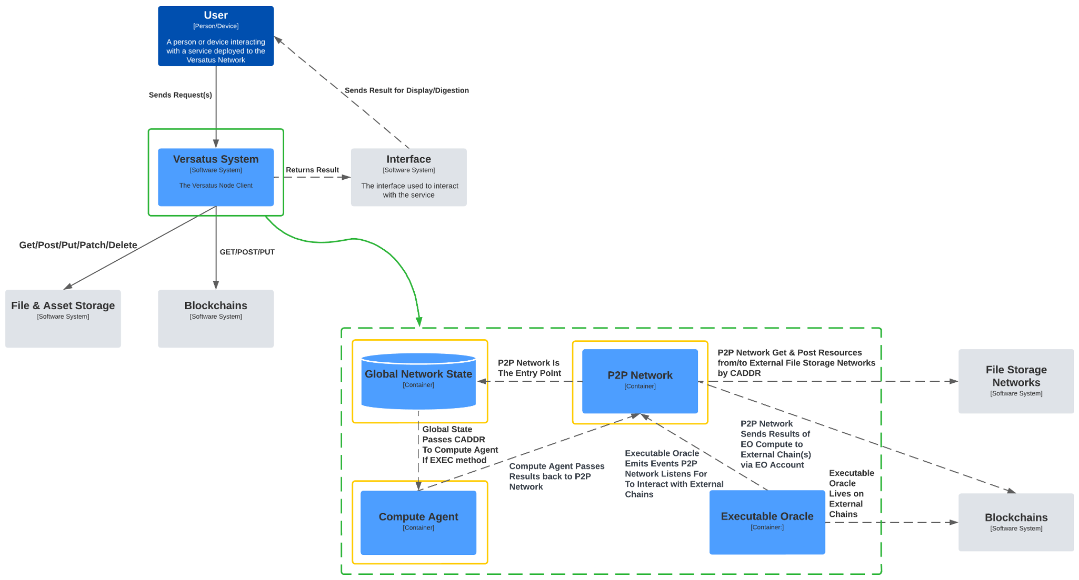
Herein we present an architecture to power a new generation of internet, one free from censorship, free from predatory data capture with the speed and security of the modern internet stack we have grown to love. The architecture leverages tools that already exist to accomplish this feat, while introducing a few new concepts to glue together existing tools under a modern architecture. The architecture enables developers to build full-stack web and mobile applications, websites, host on-demand long-running compute jobs, and build smart contracts that are language agnostic, and can use any public blockchain or distributed ledger for settlement. The architecture, at a high level, consists of 3 layers:

First, a modern peer to peer networking layer built to facilitate the stack. The network layer not only consists of and facilitates multi-protocol transport with TLS, but also incorporates a decentralized Domain Name Service, a decentralized Content Delivery Network, the ability to persist the state of the network, and is compatible with all communication protocols used by modern applications[1][2]. It is designed to be extensible so that proprietary communication protocols, as well as future standard protocols can be integrated seamlessly[1].

Everything other than HTTP verbs and Versatus RPC methods are handled by the programs themselves. With this flexibility of communication protocols, it enables users to access browser based services, mobile app services, email services, file sharing services, streaming services, as well as for devices to communicate directly with the network and services deployed to the network[1].

Second, a multi-layer application specific nano runtime that enables on-demand, rapid instantiation functions and compute tasks in an environment that has layered containment for security, hardware and software virtualization for platform agnosticism across heterogeneous machines, and modular extensible runtimes for programming language agnosticism and verifiability, both through redundant execution as well as cryptographic techniques[3]. These app-specific nano runtimes are designed to operate in a similar vein as functions, that said, they are capable of doing more. They are turing complete, hyper lightweight full ISA virtual machines, with multiple layers of containment for security purposes.

Third, an Executable Oracle to simulate a stateless rollup to power language agnostic, settlement layer agnostic smart contracts[4][5]. The executable oracle works, in many ways, similarly to data oracles built on top of blockchains, but with programmability[6]. When a user interacts with a program in the network through a given chain, the contract emits an event, a relay node picks up the event and structures an RPC call into the Versatus network, passing the necessary data to the network. The network schedules the job with a quorum of worker nodes, and when the job is complete and verified by the collector quorum, the results are posted via a transaction from the Oracle account back to the oracle. The results provide the oracle with instructions on what to do next. This model moves the execution of compute tasks off chain, while ensuring verifiable results settle on chain[7]. Combining this with a network of "watchers" to ex-post validate the compute results and potentially slash any node(s) that post errant or malicious results further enhances the security of the network, providing economic incentives to participate honestly[8].


User
[Person/Device]
A person or device interacting with a service deployed to the Versatus Network
Sends Request(s)
Sends Result for Display/Digestion
Versatus System
[Software System]
The Versatus Node Client
Returns Result
Interface
[Software System]
The interface used to interact with the service
Get/Post/Put/Patch/Delete
GET/POST/PUT
File & Asset Storage
[Software System]
Blockchains
[Software System]
Global Network State
[Container]
P2P Network Is The Entry Point
P2P Network
[Container]
P2P Network Get & Post Resources from/to External File Storage Networks by CADDR
File Storage Networks
[Software System]
Global State Passes CADDR To Compute Agent If EXEC method
Compute Agent Passes Results back to P2P Network
Executable Oracle Emits Events P2P Network Listens For To Interact with External Chains
P2P Network Sends Results of EO Compute to External Chain(s) via EO Account
Executable Oracle Lives on External Chains
Compute Agent
[Container]
Executable Oracle
[Container:]
Blockchains
[Software System]

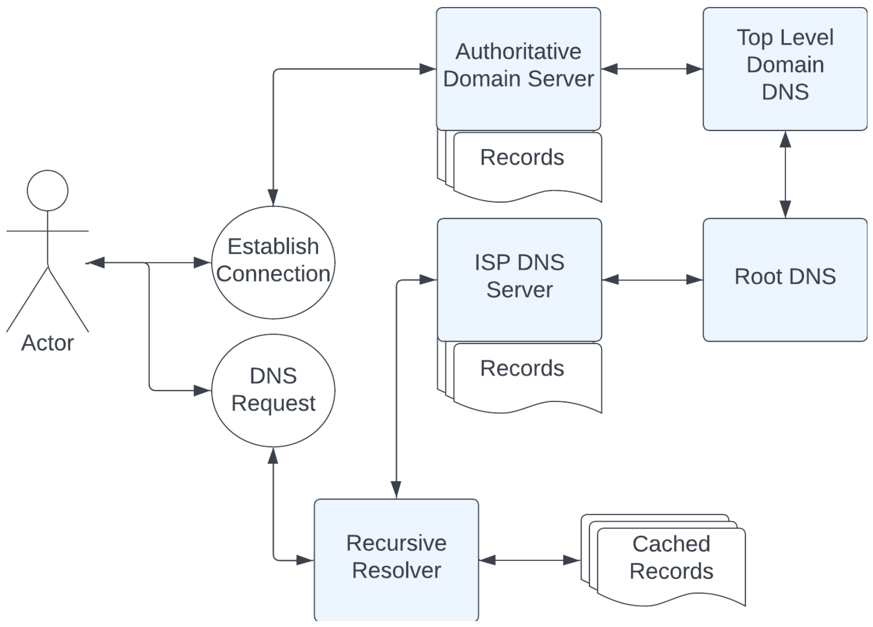
Network

The node client is built to be both a server and a client, as many Peer to Peer nodes are, however, it is also built to be modular and configurable[9]. An operator can run a full-node, which includes all functionality, or can choose which service within the network they would like to participate in[9]. For example, an operator may choose they only want to provide relay, and not do any compute, state persistence, or service hosting. On the other hand, they can opt to only provide compute or only state persistence. Nodes may also configure the types of compute jobs they want to participate in[10].

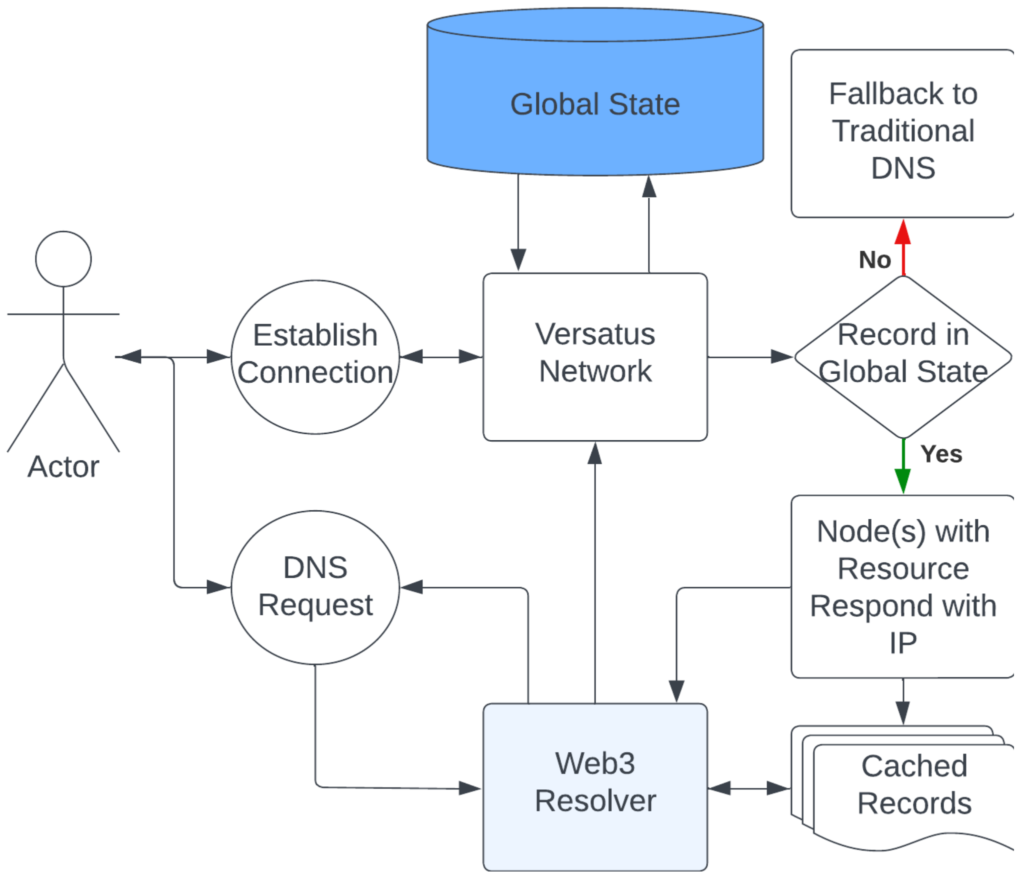
The node client is also designed such that operators can easily federate the services in their own operational domain, and for example, run a compute service on one node, while running a state persistence service[11]. This model ensures that node operators can choose the service(s) their hardware and infrastructure can best handle and separate roles in a way that provides both their organization and the network maximum benefits[1]. Nodes join the network over a peer to peer networking protocol, which establishes and maintains their connection to other nodes in the network[12].

Decentralized DNS

The Versatus DeDNS operates over a stateless DNS facade that can handle any DNS request for resource records for entities on the network. The DNS facade instead of recursively searching the hierarchical structure of traditional DNS, searches the Versatus protocol and uses the content address to locate the resources requested[12]. Under traditional DNS a request is sent to the ISP DNS resolver, which checks its records to see if the record attempting to be discovered is available in the ISP DNS cache. If so, it responds with the IP address of the server hosting the resource. If not, it recursively searches the Root DNS server, and then the Top Level DNS server. The IP address of the record’s Authoritative Domain server is returned to the user and the process ends. The diagram below shows the traditional DNS process.

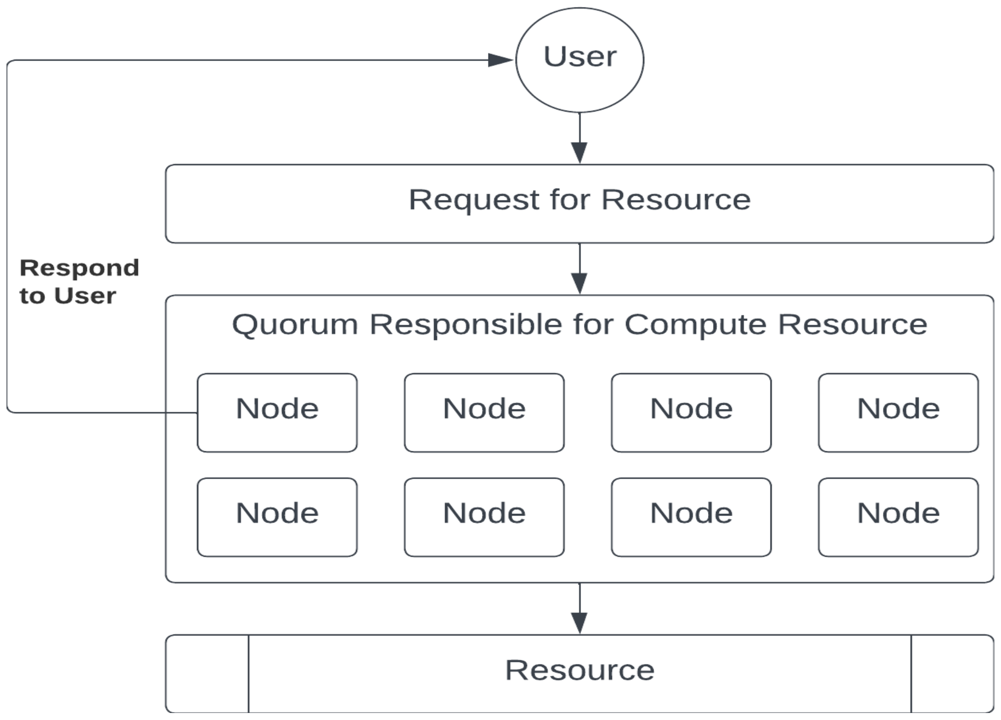


Under the Versatus DeDNS the process is similar but with some key differences. When the request begins a request to the configured web3 resolver is sent. The resolver forwards the request to the Versatus network. Node(s) pick up the request, search their local cache, if not available, check the global state. If the record does not exist in the global state, the resolver falls back to web2. For records that do exist in the global state of the network, the resolver responds with the IP address of the service, to which the user's browser initiates the connection using standard IP mechanics. Resources are redundantly cached on many nodes, and nodes rotate, over time, the resources that they store. Nodes earn by actively maintaining resources in accordance to the protocol and servicing requests. Typically, nodes rely on existing decentralized file storage protocols for file storage[12].



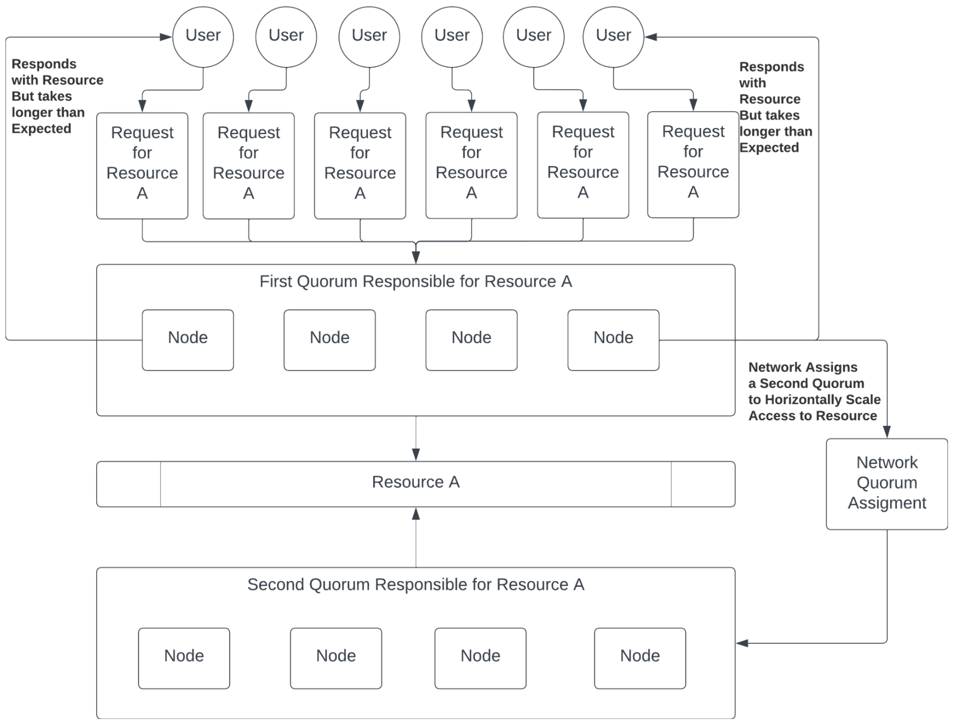
Auto-Scaling

As will be discussed in more detail in the section on our Multilayer App-Specific Nano Runtime model, all services are static and all compute tasks in the network are pure functions. Leveraging serverless architecture enables rapid auto-scaling of compute enabling even the most load heavy applications to utilize the Versatus network for deployments[11]. That said, a protocol for handling load balancing and auto-scaling is necessary. Fortunately, the way the DeDNS and the network at large are designed, programs in the network are highly redundant across the network.



While not every node is required to maintain a local version of every resource, quorums of nodes are required to maintain resources, and the number of quorums required to maintain a specific resource is auto-incremented as the need to scale that resource increases. When a given resource is being requested at a significant rate, and as a result, the time between request and initialization of the execution increases beyond 15 seconds for over 1,000 consecutive requests for the same resource, a new quorum is automatically introduced and required to maintain the resource and service requests for the resource.

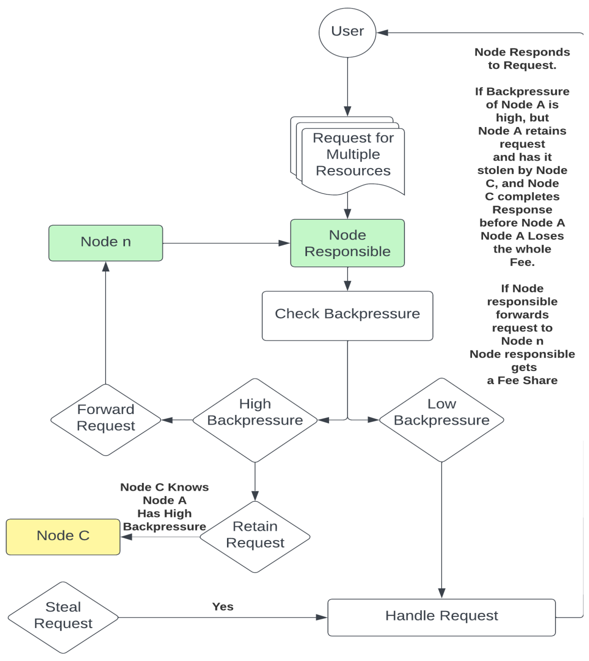
This auto-scaling mechanism, in a peer to peer network, is theoretically infinite. To accommodate the nodes in the quorum having to maintain an additional resource, a peak load pricing mechanism kicks in enabling nodes to earn additional income by accepting the responsibility. On the flip side of the same coin, when the time between request and initialization of execution is less than 50 milliseconds for 1,000 consecutive responses for the same resource, the quorum responsible for the resource with the largest average XOR distance from the resource can drop their local redundant copy of the resource, and no longer are required to maintain it. Under the peak load pricing scheme, the income earned for servicing the resource declines until an equilibrium is found. For most resources, a single quorum of nodes is all that will ever be required. However, for frequently accessed and requested resources, redundant availability on every node in the entire network may be required. This market mechanism works to ensure resource availability is maximized, response times are minimized and scaling is handled automatically[11].


User
User
User
User
User
User
Responds with Resource But takes longer than Expected
Responds with Resource But takes longer than Expected
Request for Resource A
Request for Resource A
Request for Resource A
Request for Resource A
Request for Resource A
Request for Resource A
First Quorum Responsible for Resource A
Node
Node
Node
Node
Network Assigns a Second Quorum to Horizontally Scale Access to Resource
Resource A
Network Quorum Assigment
Second Quorum Responsible for Resource A
Node
Node
Node
Node

Load Balancing & Scheduling

For load balancing in a network where a single node is handling responses to requests for multiple resources, auto-scaling applications with frequent requests or spikes in requests is not the only concern. A general load balancing mechanism is needed, and the Versatus network uses a leaderless decentralized task scheduler for both intra and inter quorum workload scheduling. Nodes that are already at maximum capacity of tasks can forward tasks to other nodes. When sharing the task with another node, the originating node shares their current backpressure, a measure of their current workload.

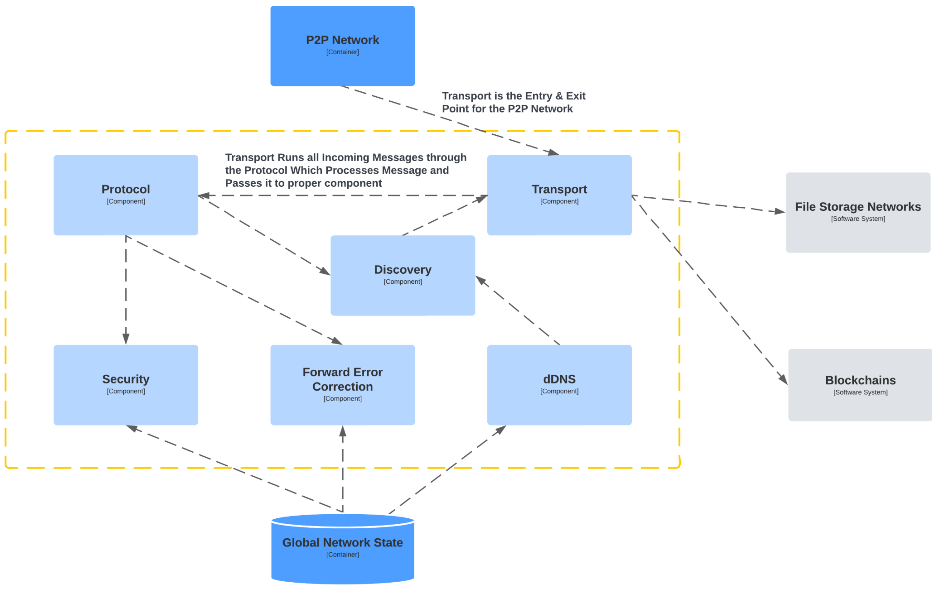
The receiving node(s) respond with their backpressure. Every time a workload is shared, proof that the originating node is indeed overwhelmed is provided, and as time passes, if that workload lightens, then the update to the back pressure will become stale and will be ignored. Once ignored, the node that was previously overwhelmed can have tasks shared with it again. Over time, the network works to relieve back pressure of nodes that are overwhelmed with tasks through this task sharing mechanism. Nodes that pass on executing a task still receive incentives, despite not executing the task themselves. On the other hand, if a node takes too long to respond to a request because it is overwhelmed, a node that is further away may “steal” the task by responding to it sooner than the overwhelmed node can. In this case the node does not receive any of the incentives related to executing the task. As a result, nodes have an inherent incentive to share work rather than have it stolen, facilitating a smooth and fast response time for any requests for any resources[9].


User
Node Responds to Request.
If Backpressure of Node A is high, but Node A retains request and has it stolen by Node C, and Node C completes Response before Node A Node A Loses the whole Fee.
If Node responsible forwards request to Node n Node responsible gets a Fee Share
Request for Multiple Resources
Node n
Node Responsible
Check Backpressure
Forward Request
High Backpressure
Low Backpressure
Node C Knows Node A Has High Backpressure
Node C
Retain Request
Steal Request
Yes
Handle Request

State of the Network

In order for the peer to peer network to know about the resources, and have made available said resources, the maintenance of a low cost, reasonably scalable, secure global state is necessary. Every resource deployed to the Versatus network is content addressed. This is the case whether it be files stored on third party decentralized file storage networks, programs stored on compute nodes in the network, or on-demand compute resources available to users and developers interacting with the network.

The content address is the unique ID of the resource, program package, or runtime package's original content that the node's compute agent uses to build the resource. In the case of programs, this could involve compiling the program according to the configuration file, or running it through a specific runtime in the case of higher level language programs. For static files, this could involve caching, sharing with peers and other processes to ensure rapid availability when needed. Those content addresses need to be made available to every node in the network, so a Global State and Data Availability layer that saves those content addresses, and any necessary mapping to/from records, human readable addresses and metadata is necessary to the operation of the network at web-scale. For this global state and data availability layer we use an innovative, modular and scalable blockchain architecture.



Security

The Versatus compute network uses a modified Proof of Stake algorithm called HoneSTake, which uses a reputation score driven dynamic staking protocol for centralization resistance. The network also employs stateless ultra-light clients, and where necessary, cryptographic proofs, such as ZK proofs, to ensure validity and implement stake slashing mechanisms. This ensures that nodes in the network can provide the resources and services they claim to, do so honestly, and are not errant, bug-ridden or otherwise faulty. For a decentralized, permissionless network, where anyone can participate, the stakes are higher. Avoiding the delivery of malicious services or resources is mission critical. The HoneSTake algorithm is also restaking compatible, and can use restaking protocols to leverage existing networks of trust and security[13]. By participating honestly in the network, node operators benefit from reduced stake, which ultimately means a greater return on their investment over time.

Multilayer App-Specific Nano Runtime

To power a developer experience that is seamless, and enables language agnostic, tooling agnostic services, compute, functions and smart contracts, Versatus has designed an app-specific nano-runtime that is specifically built for decentralized compute[14]. Decentralized compute is a challenging problem, not only because of the communication challenges, of which we address in the above section on the peer to peer network, but also because of heterogeneity of the nodes operating within the network[15]. Nodes in the network operate on different chip architectures, have different resources available, and are optimized for different workloads.

Advancement of containerization and library operating systems (unikernels) over the past 15 years, combined with modern knowledge of distributed and decentralized systems, a serverless-only architecture is now possible over a peer to peer network securely, at scale[16]. Techniques such as crypto token emission and restaking make bootstrapping such a network economically viable[17].

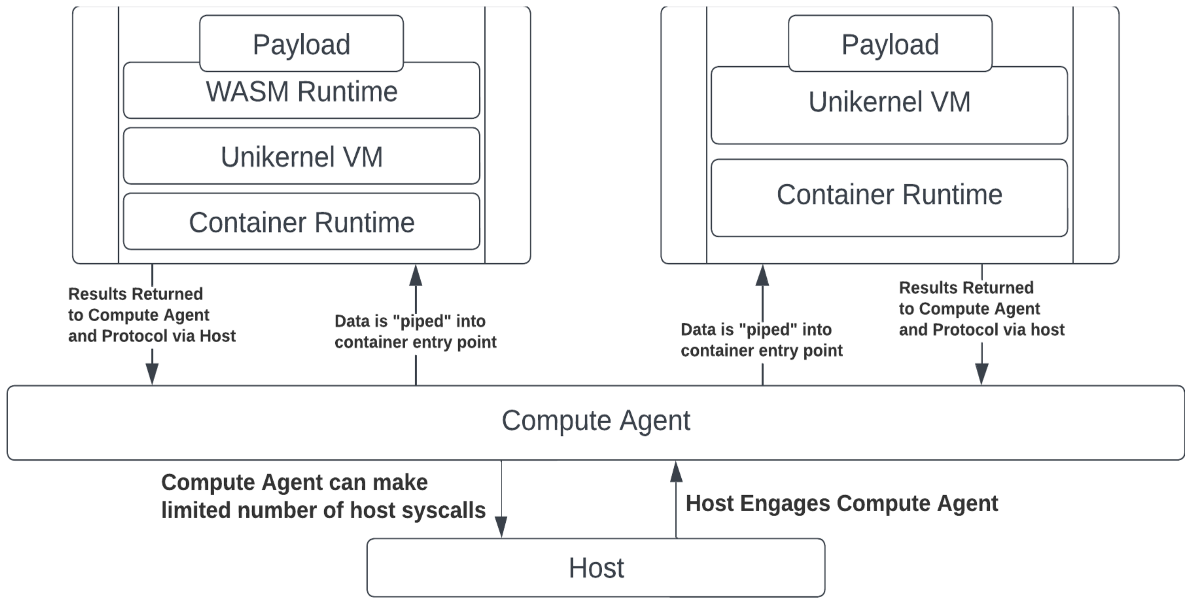
The Versatus app specific nano-runtime consists of 3 to 4 layers of containment under different scenarios. For long running compute jobs, a language runtime embedded in a unikernel runtime, wrapped in an OCI compliant container, provides all the functionality necessary in a lightweight package that can be booted on demand in milliseconds, run to completion, and have results returned. Pure functions can be built with the exact same stack, in any language, except with shorter run cycles. Under both of these paradigms, the application executed within this runtime stack can be as small as the developer likes or as large as they need.

On the flip side, smart contracts, which necessarily need restrictions imposed on them, are run in a runtime stack that has all of the 3 previous components mentioned, but also contains a modified WASM/WASI layer to eliminate potentially dangerous syscalls that could cause trouble under the conditions required by a smart contract[18]. Among those conditions include redundant execution across a quorum of nodes, consensus on the results of the execution, and the ability, therefore, to direct significant resources from a significant number of nodes in the network during spikes in requests. The modified WASM/WASI layer offers an additional layer of containment, making exploiting the contract more difficult, and when combined with the other layers of containment eliminates the ability to open sockets, access host file systems, and enable system standardized timeouts to be implemented, among other things.

Additional benefits to the introduction of a WASM/WASI layer include:

- Optimal abstraction and extensibility
- Total architecture/hardware independence
- Clear interfaces for fostering modularity

Under this architecture, and given the layers of containment, a model in which data is piped into and out of the entry point of each program provides nodes in the network with a standardized way to execute any request any user, device or other program may have. It also provides a very simple and elegant design for integrating multiple services and programs deployed to the network with one another. Execution of a program that relies on the execution of other programs becomes inherently atomic. Either the entire sequence of programs execute and return valid results or the entire sequence of tasks is considered invalid. This model also provides great transparency into any errors, as the return results can be returned to stdin, stdout or stderr. Whether debugging a program before deploying, or gathering information to report a bug to a developer through the proper channels, users and developers alike benefit from verbose failures that are returned to the interfaces from which the program is being executed.

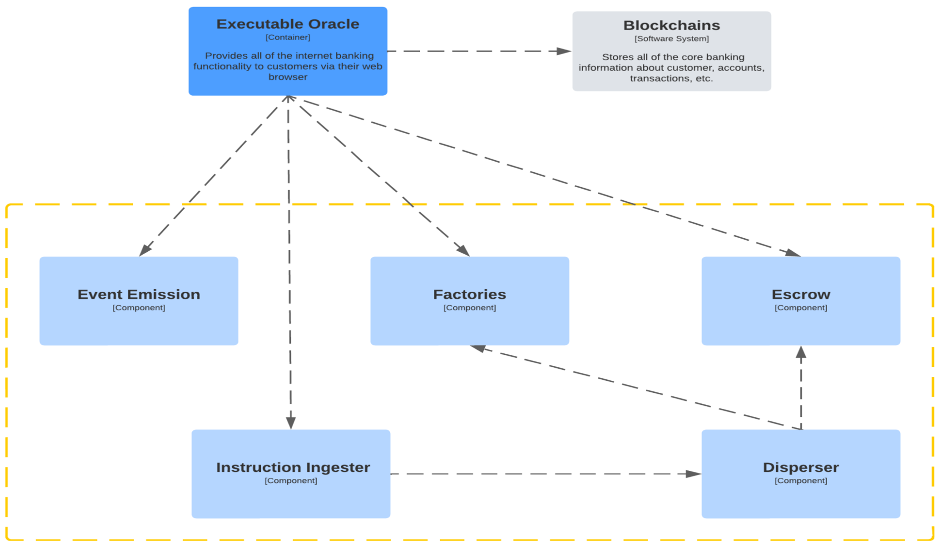

Payload
WASM Runtime
Unikernel VM
Container Runtime
Payload
Unikernel VM
Container Runtime
Results Returned to Compute Agent and Protocol via Host
Data is "piped" into container entry point
Data is "piped" into container entry point
Results Returned to Compute Agent and Protocol via host
Compute Agent
Compute Agent can make limited number of host syscalls
Host Engages Compute Agent
Host

Compute Agent

The Compute Agent in the Versatus node client acts as a container builder, execution engine, and a communicator with the host resources. The compute agent serves as an orchestrator and executor of the nano-runtimes on behalf of the protocol. When a request to execute a compute job or smart contract is received, the compute agent rapidly assembles, executes, and discards the container with the embedded payload. The developer can provide references to dependencies, job parameters and execution environments in package metadata. The compute agent can schedule execution of tasks in sequence, if necessary, or where possible, execute multiple payloads at the same time, or in parallel.

Executable Oracle

In order for the Versatus network to accept execution instructions from blockchain networks for smart contracts deployed to Versatus, and to return results that ultimately alter the state of the blockchain network, a novel design of a stateless rollup that we have coined an “Executable Oracle” is implemented as a smart contract on the base layer to which the smart contract deployed on Versatus is configured to settle to[19]. The executable oracle consists of 5 core components: Oracle Event Emission, Escrow, Factories, Instruction Ingestor and a Disperser.


Executable Oracle
[Container]
Provides all of the internet banking functionality to customers via their web browser
Blockchains
[Software System]
Stores all of the core banking information about customer, accounts, transactions, etc.
Event Emission
[Component]
Factories
[Component]
Escrow
[Component]
Instruction Ingester
[Component]
Disperser
[Component]

Oracle Event Emission

When a user interacts with an endpoint that calls a smart contract on Versatus, a transaction is routed to the executable oracle on the settlement layer chain[20]. The transaction does 2 things, it escrows funds committed by the transaction, and emits an event using the oracle function. The event consists of the data that will eventually be fed, by the Versatus compute agent, into the smart contract's nano-runtime entrypoint. The data will inform the smart contract which program to execute, which operation(s) to call, and what the parameters for the operation(s) will be.

Escrow

The oracle contract temporarily escrows the funds to ensure they will be available for the completion of the transaction[21]. Only the disperser component of the executable oracle can allocate funds from escrow or from a transaction posting results to the contract, and only the oracle account itself can call the instruction ingestor, which is the only component that can call the disperser.

Factories

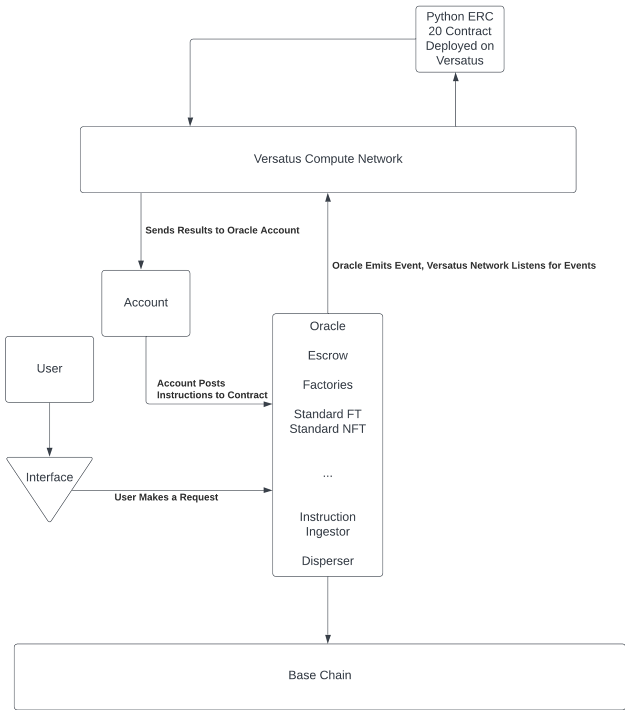
A series of factory contracts that can enable the creation of a variety of standardized contracts on the base layer are provided[22]. These factory contracts also allow the executable oracle to deliver the contract address(es) when created to the Versatus network so that they can be content addressed, and engaged with in the future via the network, without retaining the contract addresses in the oracle contract itself.

Instruction Ingestor

The instruction ingestor is what it sounds like. It takes a transaction which consists of results produced by the off-chain execution of the root smart contract deployed to the Versatus network, parses and creates parameters to be passed into the disperser to complete the transaction[23].

Disperser

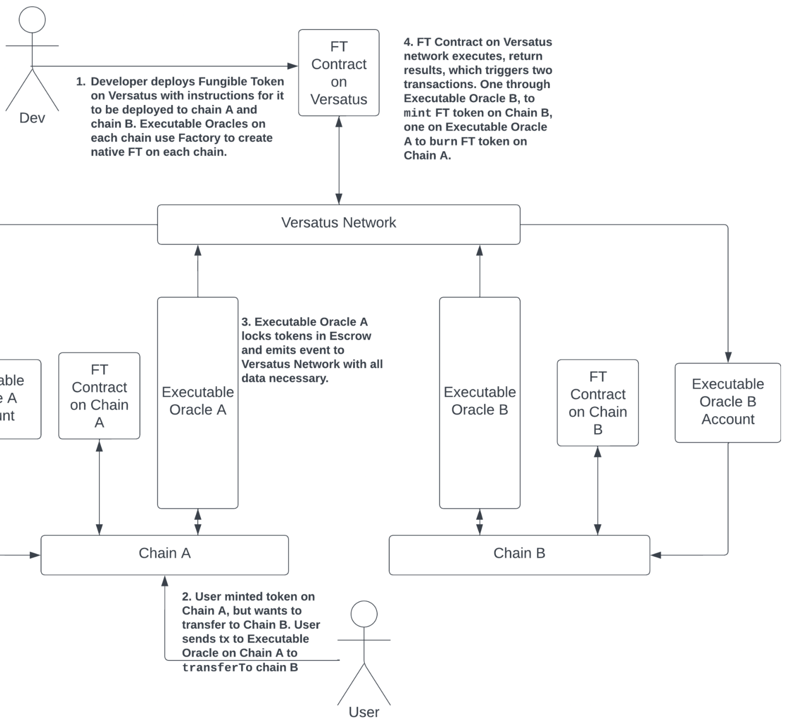
The disperser takes in parameters passed to it by the instruction ingester and can either make calls to external smart contracts on the base layer, dispersing the funds held in escrow to that contract for further execution, or can disperse the funds to wallet addresses provided as parameters by the instruction ingestor. The disperser cannot be called by any external entities, and can only be called by a single component, the instruction ingestor, which can only be called by the oracle account that passes in the results of the execution[19].


Python ERC 20 Contract Deployed on Versatus
Versatus Compute Network
Sends Results to Oracle Account
Oracle Emits Event, Versatus Network Listens for Events
Account
User
Account Posts Instructions to Contract
Oracle
Escrow
Factories
Standard FT
Standard NFT
...
Instruction Ingestor
Disperser
Interface
User Makes a Request
Base Chain

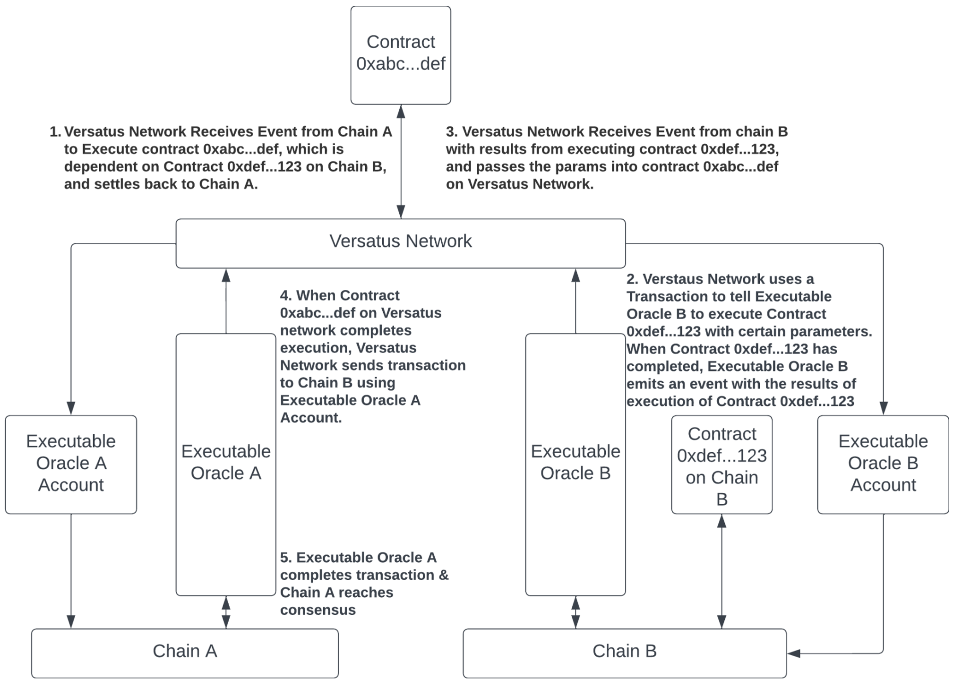
By moving execution off-chain, without state maintenance being moved off chain, the executable oracle, when combined with the Versatus network and nano-runtimes, enables language agnostic, chain agnostic, stateless smart contracts, bringing a flexible and seamless developer experience to any chain it is integrated on[21]. Beyond language agnosticism, the executable oracle model also enables bridgeless interoperability and interchain communication without having to encode and decode messages from one chain format to another[20]. It also doesn't have to worry about cross chain sequencing and differences in views of state between account(s) on one chain and account(s) on another. Further, the Versatus network enables "one deployment, many chain" capabilities. Any chain with oracle capabilities and factory contract capabilities can have the power of the Versatus network built on top of it.

Bridgeless Interoperability

Employing a mint and burn model, facilitated by Versatus Executable Oracles deployed on multiple chains enables assets to seamlessly be moved from one chain to another with no bridge locks and no “wrapped” versions of the coin[22]. By having a native smart contract deployed under the “one deployment, multiple chains” model, and providing update data to each contract upon the mint or burn of tokens to every network it is deployed to, each time it occurs via the Versatus network, the executable oracle can seamlessly enable interoperability across chains.

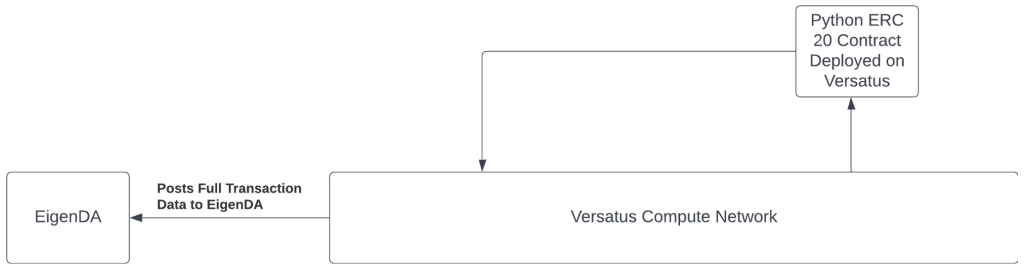

Dev
FT Contract on Versatus
1. Developer deploys Fungible Token on Versatus with instructions for it to be deployed to chain A and chain B. Executable Oracles on each chain use Factory to create native FT on each chain.
4. FT Contract on Versatus network executes, return results, which triggers two transactions. One through Executable Oracle B, to mint FT token on Chain B, one on Executable Oracle A to burn FT token on Chain A.
Versatus Network
Executable Oracle A Account
FT Contract on Chain A
Executable Oracle A
3. Executable Oracle A locks tokens in Escrow and emits event to Versatus Network with all data necessary.
Executable Oracle B
FT Contract on Chain B
Executable Oracle B Account
Chain A
Chain B
2. User minted token on Chain A, but wants to transfer to Chain B. User sends tx to Executable Oracle on Chain A to transferTo chain B
User

This goes far beyond simple asset transfers from one chain to another, however. The Versatus network, through its Executable Oracles can enable cross chain composable programs so long as the starting program is deployed to the Versatus network, the ending program is deployed to the Versatus network, and any sequence of cross chain executions include a program deployed to the Versatus network in the middle. Hopping directly from a contract on one chain to a contract on another chain is something the Versatus network does not enable, however, through programs deployed to the Versatus network, and the Executable Oracles deployed on multiple chains, the ability to communicate through said programs from one chain to another is possible.


Contract 0xabc...def
1. Versatus Network Receives Event from Chain A to Execute contract 0xabc...def, which is dependent on Contract 0xdef...123 on Chain B, and settles back to Chain A.
3. Versatus Network Receives Event from chain B with results from executing contract 0xdef...123, and passes the params into contract 0xabc...def on Versatus Network.
Versatus Network
4. When Contract 0xabc...def on Versatus network completes execution, Versatus Network sends transaction to Chain B using Executable Oracle A Account.
2. Verstaus Network uses a Transaction to tell Executable Oracle B to execute Contract 0xdef...123 with certain parameters. When Contract 0xdef...123 has completed, Executable Oracle B emits an event with the results of execution of Contract 0xdef...123
Executable Oracle A Account
Executable Oracle A
Executable Oracle B
Contract 0xdef...123 on Chain B
Executable Oracle B Account
5. Executable Oracle A completes transaction & Chain A reaches consensus
Chain A
Chain B

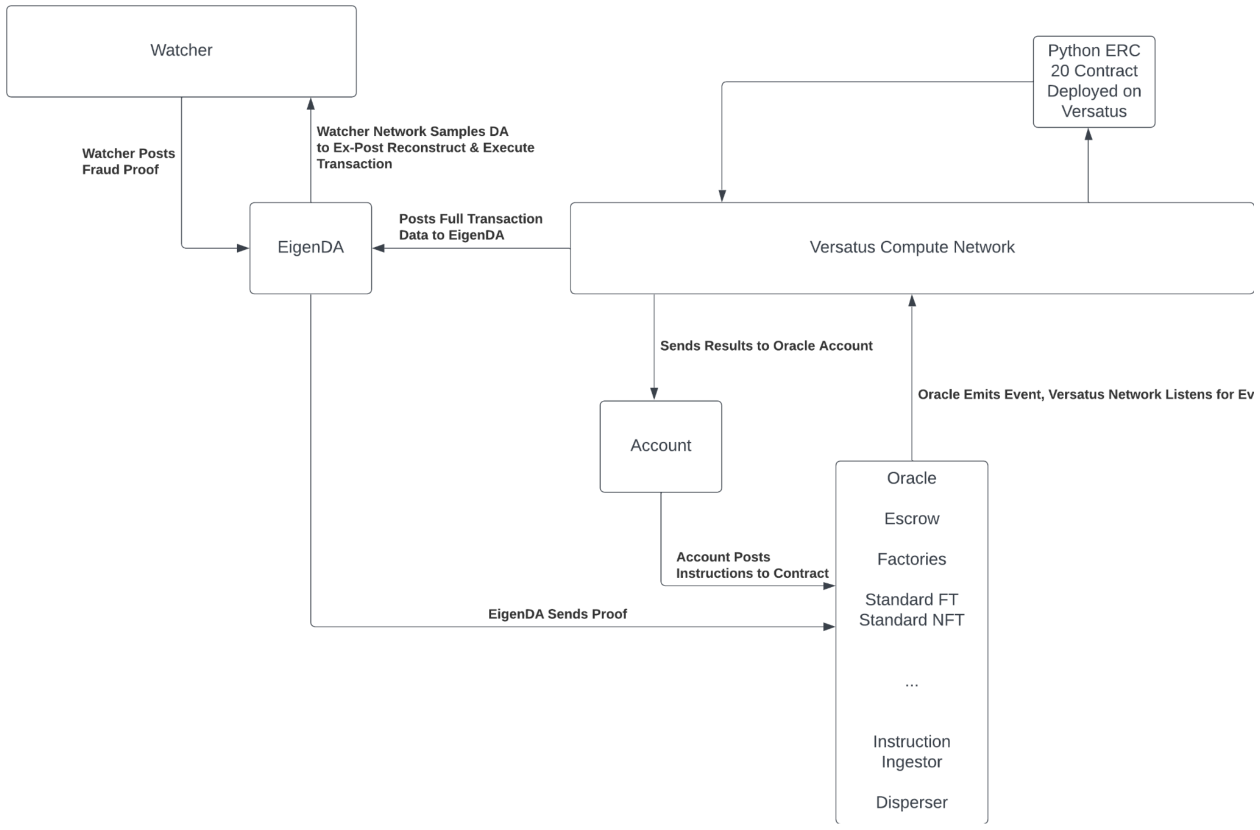
Data Availability Layer Integration

The Versatus network, while it does maintain its own global state of resources available to the compute nodes in the network, does not require transaction data be maintained on the Versatus global state or data availability layer[23]. Instead, the Versatus network is DA layer agnostic, and can integrate with any modular DA layer, as well as monolithic and hybrid blockchain networks to facilitate Data Availability to validators to recreate, reconstruct, and re-execute transactions to produce fraud proofs.


Python ERC 20 Contract Deployed on Versatus
EigenDA
Posts Full Transaction Data to EigenDA
Versatus Compute Network

Watcher Network

When integrating with external chains and DA layers, a network of ultra light validators that can sample from DA, reconstruct and re-execute transactions from the transaction data alone is a major benefit[19]. Scalable, modular DA layers such as EigenDA and Celestia enable this architecture. The Versatus watcher network employs a dual-staking model, just as the Versatus compute network does, enabling nodes to either stake in native Versatus tokens or to stake using ETH tokens using a restaking protocol that Versatus has integrated with.


Watcher
Watcher Posts Fraud Proof
Watcher Network Samples DA to Ex-Post Reconstruct & Execute Transaction
Python ERC 20 Contract Deployed on Versatus
EigenDA
Posts Full Transaction Data to EigenDA
Versatus Compute Network
Sends Results to Oracle Account
Oracle Emits Event, Versatus Network Listens for Ev
Account
Oracle
Escrow
Factories
Standard FT
Standard NFT
...
Instruction Ingestor
Disperser
Account Posts Instructions to Contract
EigenDA Sends Proof

Conclusion

The digital landscape evolves rapidly. With it, the need for a more decentralized, secure and efficient web architecture becomes paramount to building a more informed, transparent and accountable global society. Versatus, as presented in this paper, offers a groundbreaking solution to many of the challenges faced by today's web infrastructure. Using existing tools, while introducing innovative concepts, Versatus promises a new generation of the web that is resistant to censorship and predatory data capture, while maintaining the speed and security of the modern web.

At its core, the Versatus architecture is built on three foundational layers: a modern peer-to-peer networking layer, a multi-layer application-specific nano runtime, and an executable oracle that simulates a stateless rollup. Each layer is meticulously designed to ensure seamless integration, scalability and robustness. The network's ability to facilitate a wide range of communication protocols, coupled with its modular and configurable node client design, ensures flexibility and adaptability to future technological advancements.

The introduction of the Versatus app-specific nano-runtime revolutionizes decentralized compute by addressing the challenges of heterogeneity among nodes and ensuring a seamless developer experience.

The executable oracle, which simulates the functionality of a stateless rollup, bridges the gap between off-chain execution and on-chain settlement. In doing so, it enables language agnostic, chain agnostic smart contracts, enables bridgeless cross-chain token compatibility, and albeit with a tradeoff of latency, cross-chain integrations.

The Versatus network is not just a theoretical concept, but a tangible solution that holds the potential to reshape the future of the web. Its emphasis on decentralization, security and scalability positions it to lead the next iteration of the web. As the digital world continues to grow and change, Versatus will be instrumental in ensuring the web remains open, secure and accessible to all.

[1] Jacobs & Harwood, A Peer to Peer Browsable File Index using a Popularity Based Global Namespace, 2007
[2] Kashaf, et. al., Oh, What a Fragile Web We Weave: Third-party Service Dependencies In Modern Webservices and Implications, 2018
[3] Inselvini, Spam Prevention Using zk-SNARKs for Anonymous Peer-to-Peer Content Sharing Systems, 2021
[4] Nazirkhanova, Neu & Tse, Information Dispersal with Provable Retrievability for Rollups, 2021
[5] Tas, et. al., Accountable Safety for Rollups, 2022
[6] Motepalli, Freitas & Livshits, SoK: Decentralized Sequencers for Rollups, 2023
[7] Adler & Quintyne-Collins, Building Scalable Decentralized Payment Systems, 2019
[8] Ye, Misra & Song, Specular: Towards Trust-minimized Blockchain Execution Scalability with EVM-native Fraud Proofs

[9] Thoren, Taheri-Boshrooyeh & Cornelius, Wako: A Family of Modular P2P Protocols For Secure & Censorship-Resistant Communication, 2022
[10] Ishii & Inoie, An initial peer configuration algorithm for multi-streaming peer-to-peer networks, 2012
[11] Dogra, et. al., Integrated Modular Solution for Task Oriented Manipulator Configuration Design, 2021
[12] Anwar, et. al., Leveraging Social-Network Infrastructure to Improve Peer-to-Peer Overlay Performance: Results from Orkut, 2005
[13] Smith, HoneSTake: A Dynamic, Centralization Resistant, Reputation Driven Staking Protocol for Enhanced Security and Governance in Permissionless Peer-to-Peer Blockchain Networks, 2023
[14] Hong & Verghese, Resource Management in Fog/Edge Computing: A Survey, 2018
[15] Tordera, et. al., What is a Fog Node A Tutorial on Current Concepts towards a Common Definition, 2016
[16] Cesarano, Security Assessment and Hardening of Fog Computing Systems, 2023
[17] Bouachir, et. al., Blockchain and Fog Computing for Cyberphysical Systems: The Case of Smart Industry, 2020
[18] Wilhelmi, et. al., On the Decentralization of Blockchain-enabled Asynchronous Federated Learning, 2022
[19] Beniiche, A Study of Blockchain Oracles, 2020
[20] Xian, et. al., A Distributed Efficient Blockchain Oracle Scheme for Internet of Things, 2023
[21] Pasdar, Dong & Lee, Blockchain Oracle Design Patterns, 2021
[22] Adler, et. al, Astrea: A Decentralized Blockchain Oracle, 2018
[23] Goel, et. al., Infochain: A Decentralized, Trustless and Transparent Oracle on Blockchain, 2019