Conceptual overview of the Logos Network and the sub0layer

Amar Čolaković, Angela Popa, Eugen Stoyanov

LogosLabs

February 15, 2024

Abstract

This document describes the concept of the Web3 sub0layer (computational layer), provided via the Logos Network, a network that provides the Web3 infrastructure (layer-0 and layer-1 blockchain nodes) with a *community provided and blockchain based autonomous computational core infrastructure*. The necessity for a sub0layer and the impact it has on the Web3 and Web2 ecosystems is explained in this document, as well as the technical possibilities, challenges and approaches for building the network. This document represents a conceptual overview and not a technical documentation. It introduces the general idea of the sub0layer and its implementation in the world wide web.

This concept is the foundation for the upcoming proof of concept (PoC) and for the technical specification paper. With this approach, we aim to ensure that the Logos network not only has a solid theoretical basis, but is also realizable in practice and can be adapted to specific requirements. This methodical approach should serve to evaluate the practicality and effectiveness of our concept with the highest possible accuracy.

Contents

1 Introduction

The fundamental idea behind the Logos project is to establish an ecosystem, which shall provide the Web3 community with compute power generated by the community. A lot of private PC-Hardware worldwide is either rarely used or in severe underuse. We are not talking about outdated workstations that are merely gathering dust in storage facilities, but rather daily used, up-to-date hardware, of which only a fraction of the overall potential gets utilized (Laptops, Home PC's, Home Servers).

Currently the cost for computational resources supplied by the big Cloud-Services is barely affordable and presents a big challenge for small-scale ventures as they expand. (For a detailed analysis of cloud costs, read article 'The Big Cloud Exit FAQ'. [36]) To counter this, the cost for compute-power shared by the *Logos ecosystem* will be defined, under a strict and fair set of rules, by the community itself.

The core focus of Web3 and blockchain technology is decentralization and to eliminate the need of reliance in a third party. Nowadays its evident that a large number of blockchain nodes, the backbone of decentralized structures, are hosted with big cloud providers (centralized structure (Azure, AWS, Google Cloud, ...)), thus defeating the purpose of true decentralization. Services that provide a platform to easily deploy Web3 nodes rely on such big cloud providers, consequently relinquishing security, price and trust of those networks back to big companies. Many who want to support those networks chose to use such services because of the complexity of node setups. The Logos network will aim to deliver a *sub0layer* solution, so that every willing participant will be able to take part in a truly decentralized and community-secured infrastructure.

Nowadays community members with the required know-how and resources operate individual blockchain nodes. The easier, more secure and more elegant solution would be to use computational resources of those members, in order to execute a genuinely trustless Web3 computational infrastructure. This infrastructure will be used as the computational foundation for the Web3 infrastructure.

Simply speaking, the *Logos network* strives to create a community computer that belongs to everybody and no one. An autonomous network to provide, receive and share compute-power. A decentralized computing environment, acting as an underlying layer (sub0layer) to a Web3 infrastructure (layer-0 and layer-1). With this approach LogosLabs want to make Web3 more secure, decentralized and let the community run the computation.

So called "layer-0 and layer-1" blockchains, which make up the Web3 infrastructure itself, are often hosted on conventional cloud services, reverting the infrastructure to a centralized one. Therefore this sub0layer is required, to secure and decentralize those hosted nodes again.

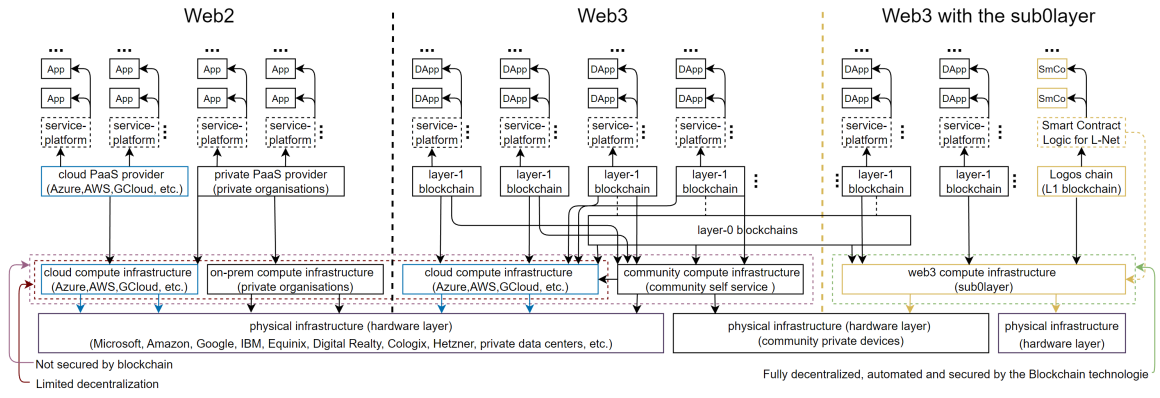

Figure 1: Overview of a Web2 approach in comparison to Web3 and Web3 with a new security and compute layer and how the Logos network can enhance the Web3 infrastructure.

After the release of Logos network 1.0, the community will play a significant role in the further development and direction of the project, which will be ensured and defined via the *Logos network philosophy*. The Logos network philosophy can be considered a kind of license. This means that the specifications of the Logos network philosophy can be "enforced" by defined regulations on the blockchain. While some decisions will be completely in the hands of the community, the basic management and supervision of the project will remain with LogosLabs for the time being to ensure the stability and security of the network until it is fully established.

The main focus of LogosLabs is to provide a *community enterprise computational environment*, based on the Blockchain technology, for the Web3 infrastructure. Although Web3 provides a wide

spectrum of security and development possibilities, the fact that participants are hosting their nodes with big cloud providers, contradicts a true Web3 decentralized approach.

2 Network

The Logos network will provide a secure *Web3 compute infrastructure* the resources of which are provided by the community. The infrastructure will be defined, secured and executed by a blockchain. The sub0layer (computational layer) will be the outcome of the Logos network. The Logos network will consist of four main components: the *DVCI (Distributed Virtual Computing Infrastructure)*, the *Logos chain*, the *Network Gatekeeper* and of a "computational distribution regulator" *W3bI*.

The DVCI will act as a "data center" where a blockchain will be utilized to ensure the security of operations. The DVCI consist of community provided computational resources and LogosLabs hosted dedicated root servers with predefined virtual environments.

A blockchain is simply a chain of data-blocks. Each data-block contains a list of transactions, securely stored on a multitude of nodes. All the information stored within those data-blocks is truthful and tamper-proof. This type of technology and the implementation of a *Smart Contract Logic* allow for the existence and operation of the aforementioned DVCI.

The network gatekeeper will ensure that each infrastructure component (existing and future) complies with all security protocols, handle the communication between the DVCI and the blockchain and also transfer new instructions/configurations to the underlying blockchain.

The W3bI will be the core "service" of the Logos network and the ecosystem and will provide the sub0layer for layer-0 and layer-1 blockchain infrastructures. This "service" will operate similarly to the current blockchain infrastructure providers, except that it will be defined on the blockchain (Logos chain) itself and the core infrastructure used in the backend (sub0layer) will be a full Web3 core infrastructure.

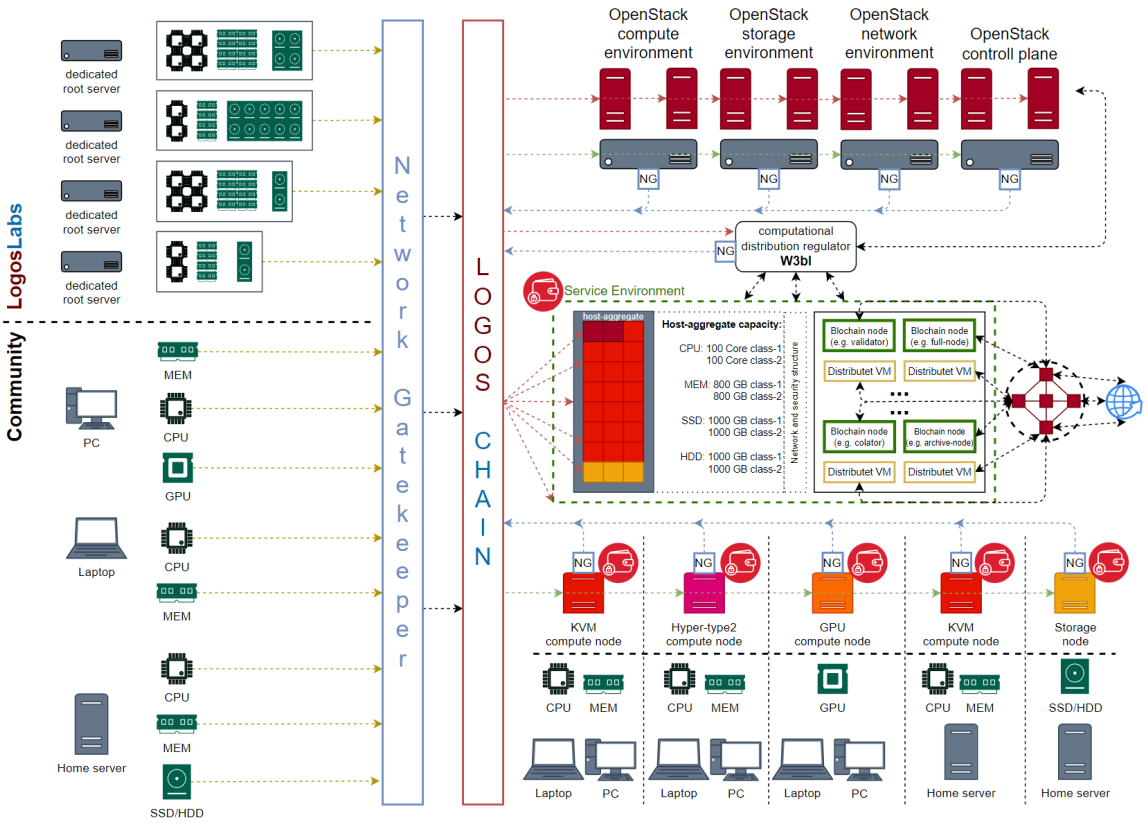

Figure 2: Conceptual overview of the Logos network and its benefits.

Blockchain nodes must be constantly synchronized to get a real-time overview of the blockchain-state. The main focus of LogosLabs is to further secure and decentralize the environment where these nodes are executed in. The development of the Logos network will incorporate, in addition to in-house developments, a variety of existing technologies, tailored to the specific requirements of this type of network. The main aspects which define the philosophy of the Logos ecosystem are: IaC (Infrastructure as a code), autonomous solutions, security and decentralization.

2.1 Distributed Virtual Computing Infrastructure

Big data centers these days can provide centralized computational infrastructures with ease. Because the DVCI is based on a decentralized approach, a variety of different issues will have to be addressed (e.g. network latency, virtualization in distributed environments, aggregated computation, ...). A cloud infrastructure, as DVCI aims to be, ultimately exists to productively manage the Web3 computational infrastructure, that is why response time is a crucial factor. Response time being the duration it takes for a request to be processed and returned to the requester. It is a unit of measurement for the "rate" of a system. It is important to note that this response time is not only defined by intern processes but also affected by external factors (connection quality, hardware performance, local conditions, ...).

A crucial component of such an infrastructure is a low-latency software defined network (SDN) to achieve acceptable response times. The challenge of such an SDN is, to connect all participants in a manner so that their resources can be connected and utilized as part of a "computational resource-pool" in a virtual compute environment. Each global region shall have its own computational network to ensure optimal resource management. In addition to *computational networks (CoN)* there shall be two additional network groups, namely *service networks (SeNe)* and *regional bridge networks (RBN)*. SeNe's will be used to divide resources in a global region, whereas RBN's will establish the connection between aforementioned regions.

Virtualization in distributed environments is another crucial part of this DVCI. By utilizing Hypervisor technology, especially type-1 Hypervisors, it is possible to create a virtual environment that can access hardware "directly" without an OS-layer. Its important to mention that those environments can be fully isolated using different methods. Even in case a user has root privileges, his access to KVM resources can be limited by certain configurations and security measures.

Since not every resource provider's OS is capable of running type-1 Hypervisors, the type-2 Hypervisor will also be utilized. The only downside being the increase of computational duration, due to the OS-layer. By using OpenCL (Open Computing Language) and ROCm (Radeon Open Compute Platform), GPU virtualization (for use cases requiring this specific type of resource) can also be accomplished. Due to not every provided hardware being of same quality, resources will have to be divided into classes (by efficiency, compute-periods, volume of processable date, ...) in order to task them with fitting services.

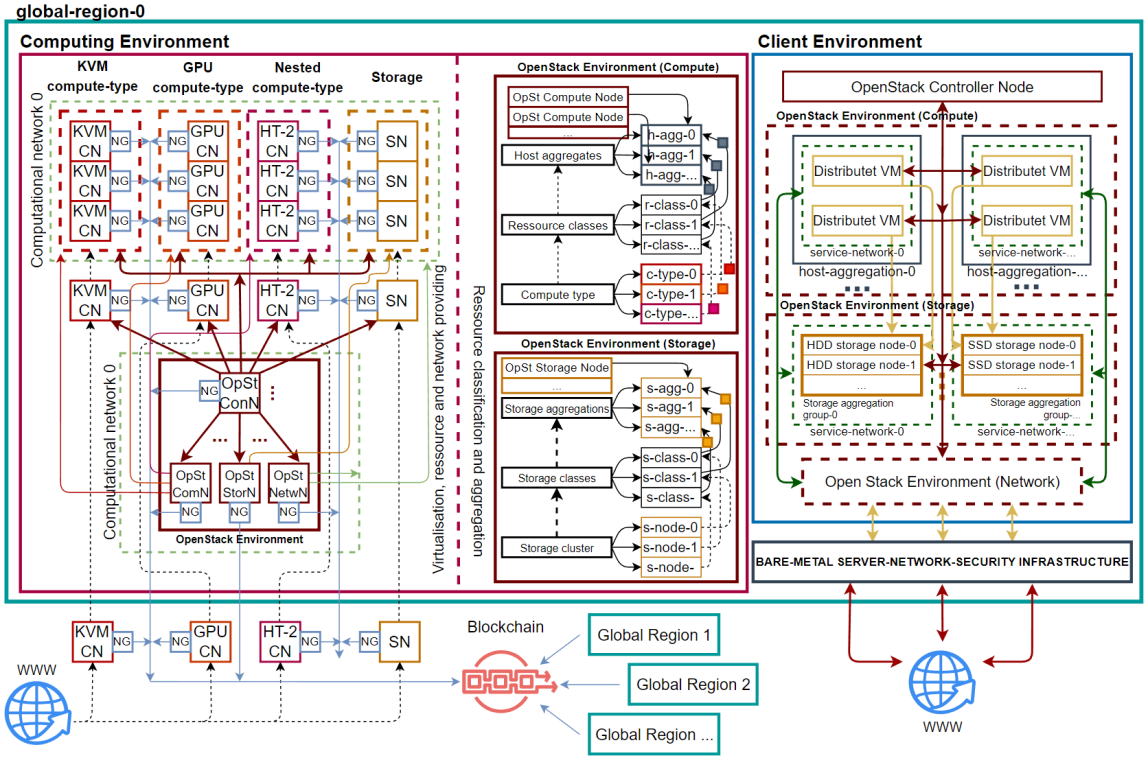

Figure 3: High-level overview of the Distributed Virtual Computing Infrastructure and its components.

Each global region gets assigned a minimum of four dedicated root servers with an *OpenStack environment*. This OpenStack environment will be subject to the *environment controle node (ECN)*.

The ECN consists of an OpenStack controller node and the corresponding network gatekeeper components. After the creation and configuration of suitable service environments (in the OpenStack environment) they will get assigned to so called *host aggregates*. Besides their own dedicated computational resources, the host aggregations will also get distributed resources assigned to them, in order to run corresponding VM's/containers. OpenStack offers functionalities in the field of IaC and automation, which will be used to improve accessibility and autonomous functions of the infrastructure. Each global region will have its own DVCI instance, which means every DVCI will be able to operate independently. The communication between all existing DVCI's will be carried out via RBN's and Geo-clustering. (chapter 2.1.4)

Every single component of a DVCI will be build, secured, configured and supervised by the blockchain. (chapter 2.2)

2.1.1 Architecture and components

The DVCI can also be laid-out as a reference model. Layer-0 is the physical layer. This layer combines dedicated root servers and distributed community hardware. Layer-1 on the other hand, is the layer, where basic hosting, virtualization, network and security infrastructures will be set up. (host-OS, host-Firewall, Hypervisor, networking and load balancing, failover mechanism, ...). Whereas layer-2 (computing environment) will be the one providing a cloud infrastructure for the OpenStack environment. Layer-2 will also be responsible for categorizing and bundling of provided computation resources. Within the last layer a.k.a client environment (layer-3) different service environments will be generated (cloud-service level) and configured.

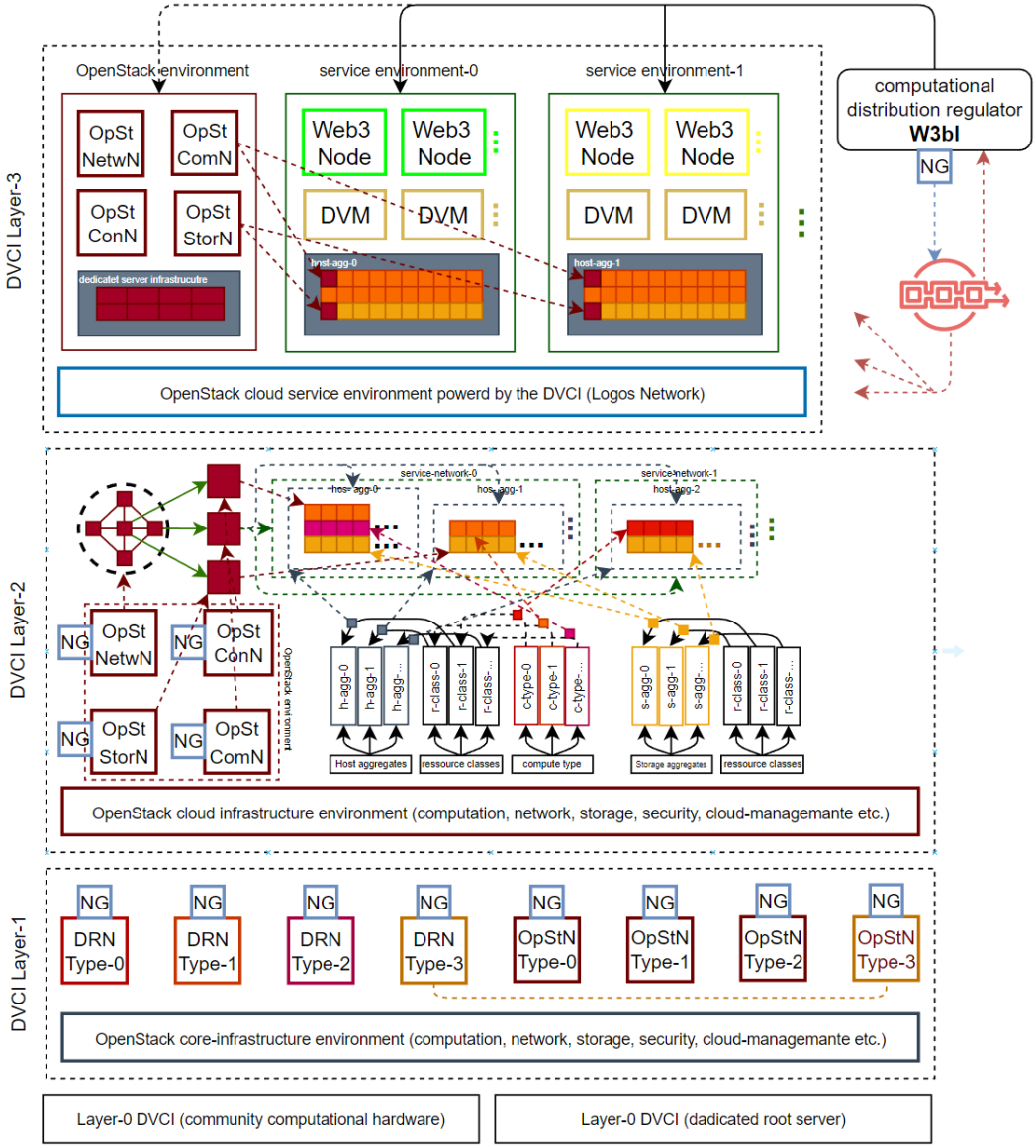

Figure 4: Overview of a Distributed Virtual Computing Infrastructure as a reference model.

Global Region: Computational resources will be divided into geographic (global) regions. The idea behind this division is to ensure optimal networking, virtualization and resource management on a local scale. The amount of such regions will heavily depend on the network components' overall global latency, and will increase if lower response times are needed.

Dedicated Root Server: These servers will be tasked with providing an OpenStack environment for the DVCI in a global region. Every global region, as mentioned earlier, will have four dedicated root servers assigned to them in order to facilitate a server failover mechanism.This will enable the migration of a service to another server, in case of hardware malfunction. A reliable method in "high-availability" and "disaster recovery", is to physically distance data centers from each other. Specifically the "active/passive failover" mechanism will be used. The advantage of this type of mechanism is, that standby-servers use only a small amount of computational resources, unless a service-migration has to be undertaken. All root servers will operate a host-OS under strict security precautions. The form of bare-metal network and LB-infrastructure will depend on the quality of the data center network infrastructure.

Region Environments: The term region environments is used as a collective term for layer-2 and layer-3. Region environments will separate the DVCI into phases, depending on field of activity, thus simplifying configuration management and oversight. Each global region will consist of a computing environment and a client environment.

- **Computing Environment:** The computing environment(layer-2) or back-end phase will take on layer-specific actions in managing, connecting and securing the DVCI. Besides providing a platform for the OpenStack environment, this layer-2 will also bundle provided resources, by resource class, into host-aggregates and deliver said resources to the service layer.
- **Client Environment:** The client environment will be the "execution-layer" for the DVCI and act as the access point for the computational distribution regulator, W3bI. Client environments will be created in this layer (via OpenStack) and aforementioned host-aggregates distributed to *DVM's (Distributed Virtual Machines)*. Virtual environments will be isolated through service networks, to ensure secure separation of different services.

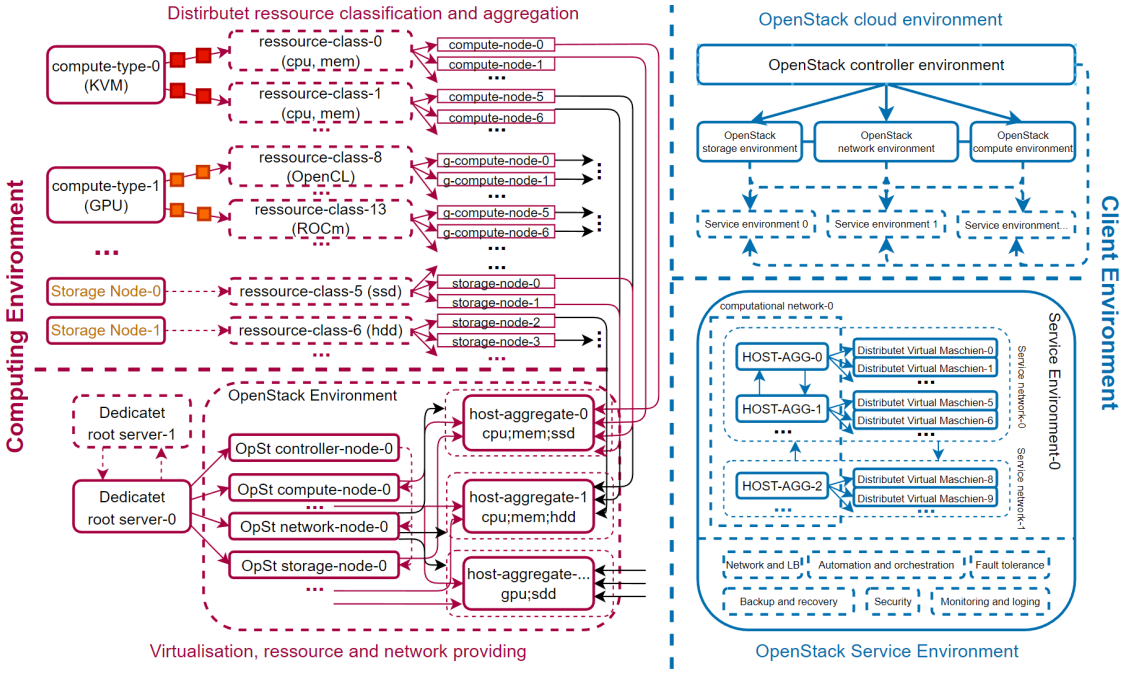

Figure 5: High-level overview of region environments and their workflows.

OpenStack Environment: OpenStack is a comprehensive cloud-infrastructure solution, that provides a wide variety of tools and resources for the deployment, scaling and management of virtual environments and other cloud resources. OpenStack allows for the creation of networks, able to meet the requirements of cloud-computing infrastructures. It also supports different virtualization technologies, storage and network solutions.

The OpenStack architecture used for the Logos network will consist of four Node-types playing different roles. The first type (controller node) will host the main OpenStack services required for the DVCI. The second type (compute node) will take on the execution of VM's (DVM's). The third type (storage node) will facilitate storage and retrieval of data. The fourth and last type (network node) will coordinate network resources, configure network topologies as well as enable and manage communication channels between the VM's (DVM's) and external networks. Each OpenStack environment will have to retain at least two nodes of each type, for load-balancing and high-availability to function uninterrupted. The choice if a node will be a physical server or a virtual machine will depend heavily on the service type and the computational requirements. For a more comprehensive understanding of OpenStack, see [15].

- **Controller Node:** In an OpenStack environment the term controller node is associated with the key component of the OpenStack-system, serving the purpose of management different OpenStack services.(Keystone, Glance, Nova and Neutron (control components), Heat, ...). For detailed insights into the control-plane design, refer to [14]

- **Compute Node:** Compute nodes, included in the OpenStack system, will be responsible with the distribution of computational resources and the execution of the DVM's. The most important component operated by a compute node is "Nova" (OpenStack compute service). Every node should have its own Hypervisor (type1) to enable the creation and execution of VM's (DVM's). For further information on compute architecture, see [13]. The approach of node-type separation on different physical servers is aimed at improving the virtualization efficiency.

- **Storage Node:** These nodes will provide storage space in form of block storage (Cinder) and object storage (Swift). For a deeper understanding of the storage architecture, review [17]. Future research will determine if the efficiency of a single physical server hosting both storage and compute node is acceptable.

- **Network Node:** A network node (physical server or virtual machine) will be responsible for network traffic, run network services (Neutron, Octavia, Designate) and define the network topology inside an OpenStack environment. This node will play a vital role in the communication between different OpenStack environment components and external networks. To delve deeper into the topic of the network architecture, see [16]. If network nodes will have to be set up on separate physical servers or could be combined with compute/storage nodes, will also have to be determined in future research.

Future research (PoC) shall also define the amount of physical servers to be used per global region as well as the advantages and disadvantages of different server configurations (combined nodes or single node servers).

Distributet Ressource Node: This type of node will connect distributed resources to the DVCI. *Distributed resource nodes (DRN's)*, from a simple perspective, would have to be PC's and home-servers, from the networks perspective however, they will be virtual compute/storage nodes. Compute nodes will have the option to commit to SLA's (service-level-agreement) therefore increasing the DVCIs control over the provided resources. DRN's will have to be categorized by the provided resource type (CPU, RAM, GPU, HDD/SSD). Each DRN will also be equipped with its own client network gatekeeper. Subsequent to the separation into global regions, by the network gatekeeper, the DRN's will be connected to a virtualization network and assigned host-aggregates via the OpenStack environment. Each provider (of distributed resources) will have to be a separate, isolated environment. (SELinux, AppArmor, Cgroups, Libvirt Access Control, GPU-passthrough, ...) This should enable providers to lend resources to the network while using the remainder for their local applications. In order to deliver distributed resources securely and efficiently several factors will also have to be accounted for (network connection, network latency, compute-times, fail-safe measures).

- **KVM Compute node:** KVM compute nodes will be tasked with providing the network with CPU's and RAM. This type of node will only be applicable to hardware capable of virtualization (KVM). Contrary to OpenStack compute nodes, KVM compute nodes will only be equipped with "libvirt" used by NOVA, not the full service.

- **Hyper-Type2 Compute node:** As is the case with the KVM compute node, the Hyper-type2 compute node will also be tasked with providing CPU and RAM, but with a type-2 Hypervisor. The necessity of this type of node arises from not every operating system being able to support type-1 Hypervisors. (PoC: Efficiency of different types of virtualization environments needs to be analyzed)

- **GPU Compute node:** The GPU compute node will be tasked with providing GPU's. Based on the type of chip architecture (NVIDIA, AMD) the provided resources will have to be collected by the network by using either OpenCL or ROCm. Like the previous types of nodes, the GPU compute node will also be managed and configured via an OpenStack environment.

- **Storage node:** Storage nodes (full OpenStack node) will be tasked with providing data storage. These nodes should be bound by at least a 24-hour SLA and have server-like hardware requirements, to ensure more efficient data management within the DVCI.

2.1.2 Virtualization and cluster computing

A couple of different hardware virtualization technologies exist nowadays. The task at hand is to choose the most suited for a distributed virtualization environment.

The physical node of type-0 will be the provider (of computational power) on a Linux Kernel, which enables the utilization of the KVM technology. In this case the type-1 Hypervisor can be used to directly access the hardware without an intermediate layer. This approach is only possible on Linux based systems. Most Windows based systems on the other hand, are not capable of type-1 Hypervisor features. Possible solutions for these types of systems would a type-2 Hypervisor (nested) approach or the exclusion of those systems from CPU and RAM pools (only GPU). (PoC: Efficiency of nested virtualization against the GPU-only approach will be compared)

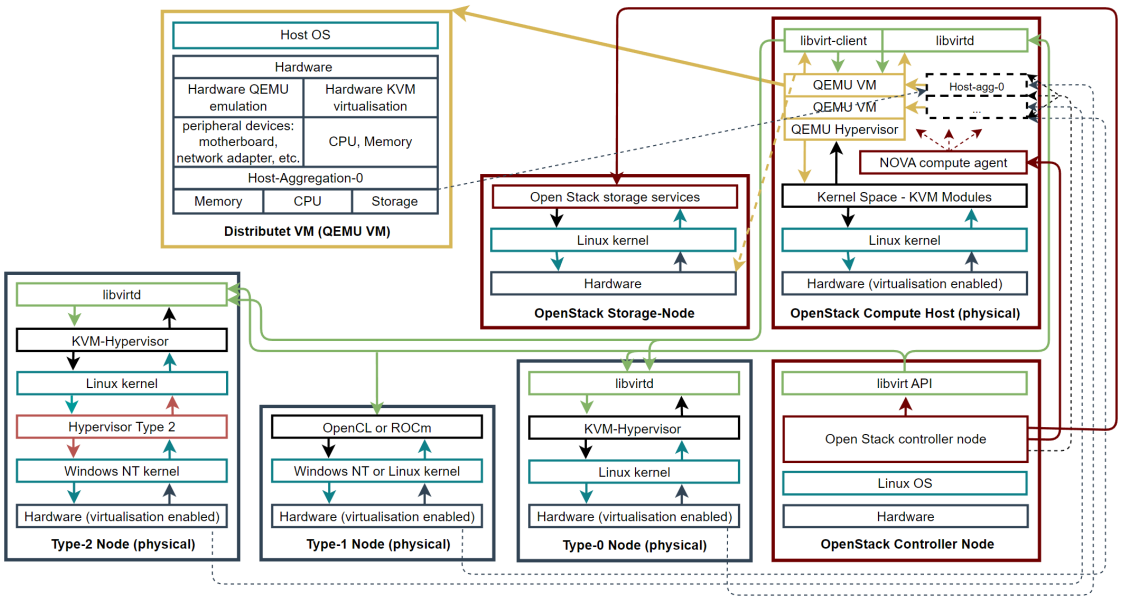

Figure 6: High-level overview of virtualization and cluster computing.

Hypervisor: The term hypervisor is used to describe a type of computer software, firmware or hardware responsible for the virtualization of computational resources. Hypervisors act as a transitional layer between physical hardware and future guest operating systems. Two types of Hypervisors (type-1, native or bare-metal and type-2 or hosted Hypervisors) exist and are implemented in different ways. See documentation [39].

- **Type-1, native or bare-metal Hypervisors:** These Hypervisors (hardware) run directly on the host's hardware, control the hardware and manage guest operating systems. The type-1 Hypervisor to be used for the provision of distributed virtualization environments, based on Linux systems, is the KVM Hypervisor.

- **Type-2 or hosted hypervisors:** Type-2 Hypervisors (software) run on a conventional operating system (Windows, Debian/GNU, macOS, ...) just as other computer programs do. This

type of Hypervisor enables the virtualization of resources at the software level without the need for special hardware.

KVM and QEMU: The Kernel Virtual Machine is a technology and a Linux kernel extension that makes it possible to support hardware virtualization, allowing the kernel to act as a type-1 Hypervisor. For a more comprehensive understanding of the KVM technology, see [6]. The Quick Emulator or QEMU (type-2 Hypervisor) is a virtualization software that uses various virtualization techniques to virtualize (guest) operating systems on non-identical hardware. For a deeper understanding of QEMU, review [24]. QEMU is very often used in combination with KVM as backend. While KVM provides the resources such as CPU and memory, QEMU emulates the required peripheral devices such as network cards, motherboard, etc.

libvirt: Libvirt is a set of libraries and a toolkit that serves as an interface for managing various virtualization technologies and Hypervisors. It is designed to facilitate the management and automation of virtualization tasks and provides a unified API for interacting with different virtualization platforms. For a thorough description of the libvirt project, reference [8]. The libvirt project can be used independently with its CLI-tool virsh. However, libvirt is primarily intended to be used via the OpenStack environment.

- **libvirt-APIs:** The libvirt-API is a central component of libvirt. It enables standardized interaction with a variety of Hypervisors (KVM, QEMU, Xen, Hyper-V, ...), which facilitates the development of virtualization management tools and applications. Libvirt-API is actually just a collective term, as it offers a variety of APIs. For further informations on libvirt-APIs, see [10].
- **libvirtd:** The libvirt daemon is responsible for receiving instructions from the libvirt-api or the libvirt-client and forwarding them to the Hypervisor. For detailed insights into the libvirt daemon, refer to [9].
- **libvirt-client:** The libvirt client allows direct access to the libvirt daemon on remote hosts, which results in the OpenStack Compute Nodes being able to access and manage the remote resources efficiently. For a better understanding and the utilization of the libvirt-client, consider [11] and [7].

OpenCL and ROCm: OpenCL provides a cross-platform API for heterogeneous systems, while ROCm is a platform from AMD that relies on various open standards and programming interfaces to utilize the computing power of AMD GPUs and accelerators. Both technologies are designed to facilitate parallel programming on heterogeneous systems. To delve deeper into the topic of the OpenCL and ROCm project, see [12] and [1]. The basic idea behind the use of OpenCL and ROCm is to write so-called kernels to be executed on remote graphics cards. These kernels are then send over the network to the graphics card, which performs the calculations and sends the results back to the application system.

This can be useful if the computing power of remote GPUs is to be used without the GPU being physically installed on the same system as the application.

2.1.3 Storage and volume clustering

A distinction must first be made between ephemeral and persistent storage in the provision of storage resources in a DVCI. The difference between ephemeral and persistent storage lies in the way data is stored and handled, especially in the context of virtual machines or cloud instances. Simply put, ephemeral storage is temporary, while persistent storage is used to store data permanently. The persistent storage will be provided in the form of object or block storage via the distributed storage nodes (DRN's), where ephemeral storage will be provided through the dedicated root servers.

Block and object storage are two different approaches to storing data, and they are used depending on the requirements.

Block Storage: Block storage is a type of storage that organizes data in blocks, where each block is considered a separate unit. Block storage is flexible and can be used for various use cases, including being used as hard disks for virtual machines or for storing databases.

Object Storage: Object Storage organizes data in objects that are assigned a unique identifier (ID). In addition to the actual data, each object also contains metadata and a unique address that

can be used to access it. Object storage is highly scalable and is well suited for large volumes of unstructured data, such as those used in cloud storage solutions or content delivery networks (CDNs).

In summary, block storage is suitable for use cases where you need file system access, such as operating systems or databases that rely on hard disks. Object storage, on the other hand, is suitable for large amounts of data, such as media files, backups and other unstructured data. Both types of persistent storage will be provided within the DVCI.

It is also important to differentiate between storage types and file systems. Fundamentally the DVCI will provides storage (block and object) but depending on the use case different file systems (IPFS, NFS CephFS, ...), will be used in the interaction with that storage.

The storage (persistent) is to be provided on a technical level by the OpenStack services SWIFT and CINDER. This means, as already mentioned before, that every Distributed Storage Node will be a full OpenStack Storage Node (block or object). For detailed insights into the storage concept, refer to [18].

2.1.4 Network groups and security

For an efficient and secure infrastructure to be established, the components of which must fulfill security requirements and be able to communicate with each other. To provide the infrastructure, specific implementations/configurations should be realized on all *DVCI layers* (exception: DVCI layer-0) in terms of network and security. All security measures and network configurations will be carried out entirely by executing smart contracts on the blockchain.

Components that should become part of a DVCI must first be configured appropriately. A component cannot become part of a DVCI, unless it complies with the configurations and security measures defined on the blockchain. Before these components join the network, they must fulfill a number of requirements. These requirements can be relevant security configurations, such as firewall settings, SELinux configurations, libvirt access control configuration, ...

An important component of the response time for a system is the network latency. Network latency is the delay that occurs in the transmission of data between two points in a network. It depends on various factors (physical distance, network infrastructure, routers and switches, packet loss, ...), which influence the speed and efficiency of data transmission. In DVCI, there should be three groups of network approaches (computational, service and region-bridge) to be utilized and implemented at different levels.

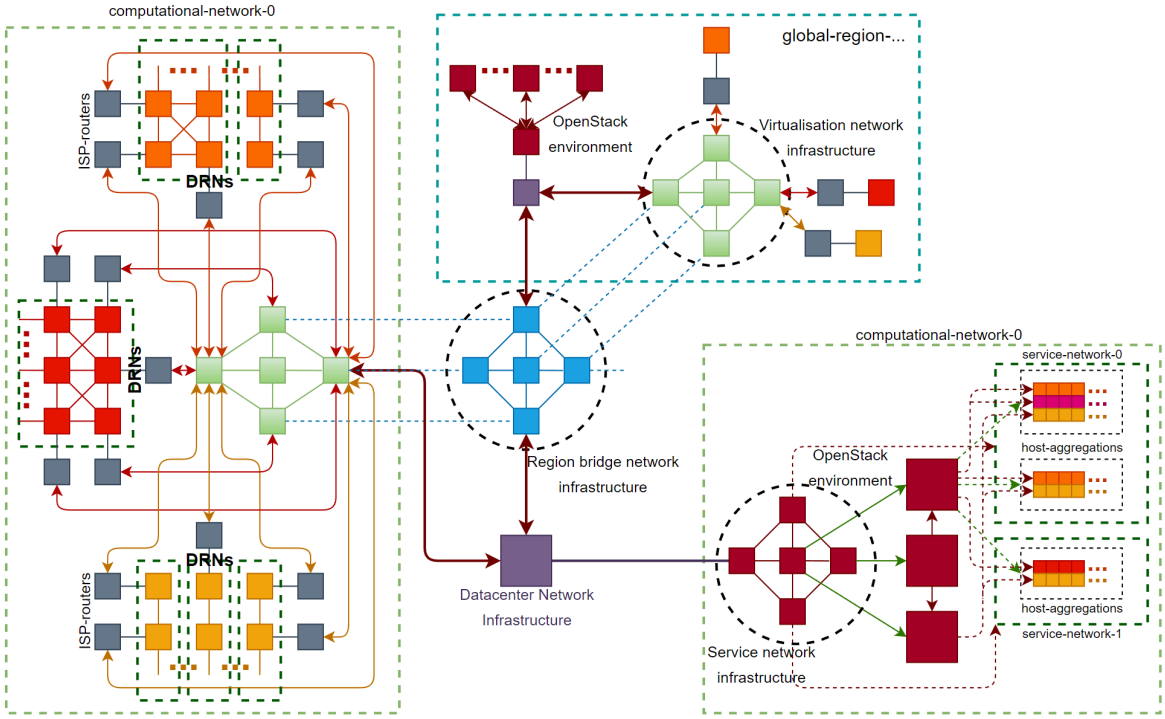

Figure 7: High-level overview of the DVCI Network approaches.

The fundamental idea here is to establish communication in the Logos network through an efficient low-latency network structure (external network service provider) with specific (Logos network) Software Defined Networks (SDNs). (What is Software-Defined Networking (SDN)? [31]) These SDNs will be defined on the blockchain in order to "bring" the network sector onto the blockchain. Since very low response times are not required for the Logos ecosystem and the network infrastructure is supported by SDNs, neither a very expensive nor a highly advanced network infrastructures (low-latency network structure) should be necessary.

Computational Network: These should be the main networks in a DVCI network environment. A low-latency software-defined network (computational network) should enable internal networking in a DVCI. Virtualization of physical hardware only makes sense and will function with optimum performance if the internal communication (DRNs with the dedicated root servers) runs with adequate latency times.

Service Network: These networks are to be provided by the OpenStack environment for the Logos ecosystem services. Service networks are basically virtual networks that are provided by the OpenStack Network Nodes (neutron service). The intention is to separate services that run in different service environments (one or a set of host aggregates) across the computational network into separate service networks and make them efficiently accessible from the external world.

Region Bridge Network: Region bridge networks (RBNs) are a concept in which services from one global region can access resources from another global region. The basic idea behind RBNs comes from the so-called Gaming Private Networks (GPN), which optimize connection stability between a player and the game's servers. This is achieved by routing network packets on an optimized path between the user and the game servers. This approach should make it possible to efficiently route and distribute DVCI resources on a global level.

All three of the network groups should be provided as SDNs at individual levels. Efficient (Sufficiently efficient) network routes need to be provided by external Network Service Providers (NSPs) (for compute and region-bridge). This hardware network node infrastructure (provided via NSPs) should be optimized with the principle of SDNs specifically for the Logos network and thus reduce the overall cost of the NSP route nodes.

It is important to notice, that this is a concept and only with the provision of the PoC the final proof can take place. The detailed architecture and how the provision of such networks will be realized as well as their integration into the Logos network will be provided via the PoC in collaboration with external network experts.

2.1.5 Configuration and operation management approach

The Logos network should be built as an automated or autonomous network. This can be achieved by using several existing technologies (Ansible, Terraform, ...), approaches and principles (GitOps, IaC CI/CD, ...).

Infrastructure as Code (IaC) [40] is an approach at managing and provisioning IT infrastructure resources through the use of declarative or imperative definitions. The basic idea behind IaC is to treat the infrastructure in the same way as an application code, resulting in more efficient and consistent management of resources.

In a declarative approach (Terraform [5]), the infrastructure is described, as it should look, without explicitly specifying how it should be created. The desired state is specified and the IaC tool ensures that the infrastructure corresponds to this state. In an imperative approach (Ansible [2]), explicit instructions are given on how the infrastructure should be created and the IaC tool follows step-by-step instructions to achieve the desired configuration.

What makes the Logos network different in relation to IaC is that the code that will be executed by the IaC tools is on the blockchain in form of smart contracts. Desired states or certain instructions of the infrastructure are defined on the blockchain.

The generation of transactions that trigger smart contracts is to be automated, which will be achieved by the Logos network gatekeeper component. Simply explained, the relevant network gatekeeper components should accept different instructions from the DVCI components, create transactions from them and execute them on the blockchain.

Another important strategy is the use of GitOps [4] principles with CI/CD (continuous integration/continuous delivery) [3] practices. These practices aim to automate the development process,

shorten development cycles, improve the quality of the code and create the possibility of fast and reliable deployments.

After the release of the Logos network version 1.0, we intend to involve the community itself in the development and decision process for the future of the Network and this approach (configuration and operation management) should enable transparency, controlled and easy access (for community authorized participants) and management of the network.

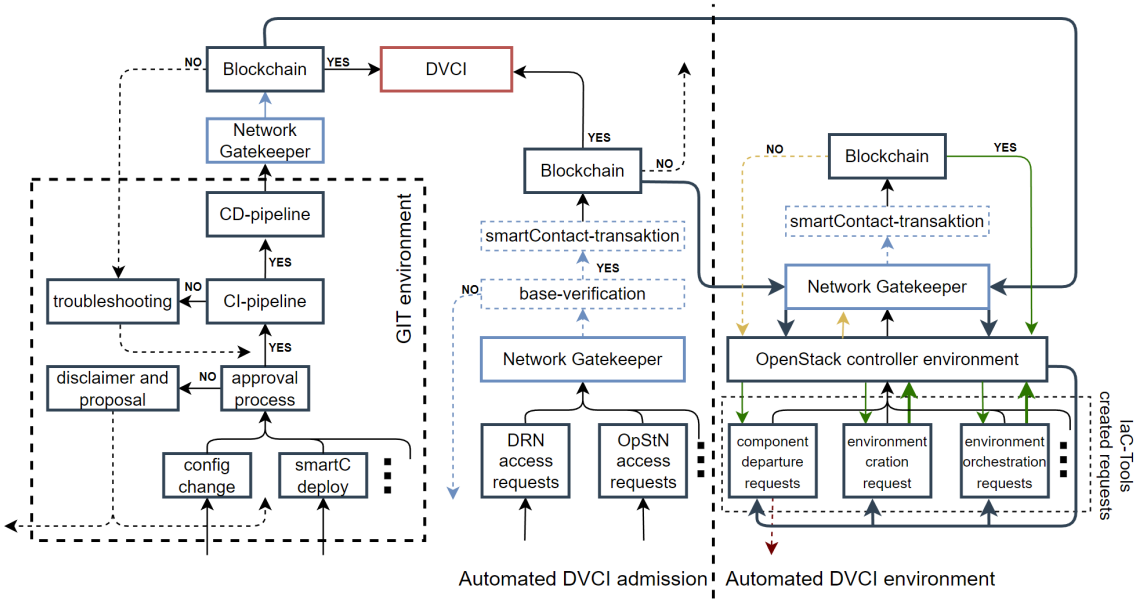

Figure 8: High-level overview of the configuration, operation and management approaches of the Logos network.

2.2 Logos Chain

The Logos chain (blockchain component) is the basis for the Logos network. Until this point, although the DVCI is already a decentralized infrastructure it still functions in a conventional way (not web3). Through the implementation of the DVCI into a blockchain, LogosLabs aims to create this additional security and computation layer (sub0layer). This layer will secure the DVCI by the means of blockchain technology, thus providing a proper core infrastructure, web3 deserves.

The Logos chain should be developed as a layer-1 blockchain and be used to create and secure the sub0layer utilizing Smart Contract Logic, also being responsible for the implementation of certain functionalities (access to LN, payment options, transparency, ...).

Smart contracts will be implemented in form of *Smart Contract Logic*. Smart contract logic should be an environment where different smart contracts are clustered by use case. The Logos chain shall not be a public smart contract platform (e.g. Moonbeam, Astar, Phala Network, Cardano, ...). An independent chain will have to be deployed to ensure specific smart contract implementations (type of SC, privacy features, SC configurations, ...). In order to create an efficient "system chain" all smart contracts on the Logos chain will have to be dedicated exclusively to the Logos network. In principle, smart contracts on the Logos chain are not different from contracts on other chains, but the logic operating those contracts is.

The execution times of smart contracts do not play a major role in the overall network efficiency, since at this service layer participating nodes are not affected. Should such a participating node get corrupted or become of a malicious nature, it will be subtracted from the DVCI (host aggregate or OpenStack environment). Depending on how fast resources for a failed node (DRN) could be redirected back to its DVM, participating nodes would only have very short to no downtimes. Each host aggregation will have an additional 30% backup buffer to further decrease possible downtimes (live VM-migration). Smart contracts should only be triggered by the network gatekeeper, thus requiring a hybrid approach to confidentiality (public and private).

This hybrid approach, which allows for public and private elements to coexist, can limit access to the blockchain to only authorized accounts and certain access privileges (role-based access control principle). Simply speaking, this approach would ensure a clear control over who has access to what and when. Therefore combining the merits of both private networks and public infrastructures.

The Logos chain will be developed as a "standalone chain" based on a substrate framework. The initial idea was to develop the Logos chain as a parachain in the Polkadot Ecosystem. Polkadot is a blockchain platform, the main goal of which is to operate an interoperable and scalable platform for the Web3. The basic idea behind Polkadot is to create a network of interconnected blockchains that promote scalability, security and interoperability. Polkadot, as a layer-0 blockchain, is designed to overcome fragmentation in the blockchain ecosystem by connecting different blockchains (layer-1 blockchains). It enables secure and scalable interaction between them and promotes the flexibility and upgradeability of the overall system. Each parachain [22] (layer-1 blockchains) has its own specific rules and consensus mechanisms, independent of the relay chain, while it (relay chain) acts as a link to enable overall security and interaction between the parachains. From LogosLabs' perspective is this approach (Relay chain with parachains) "the way to go" for the future

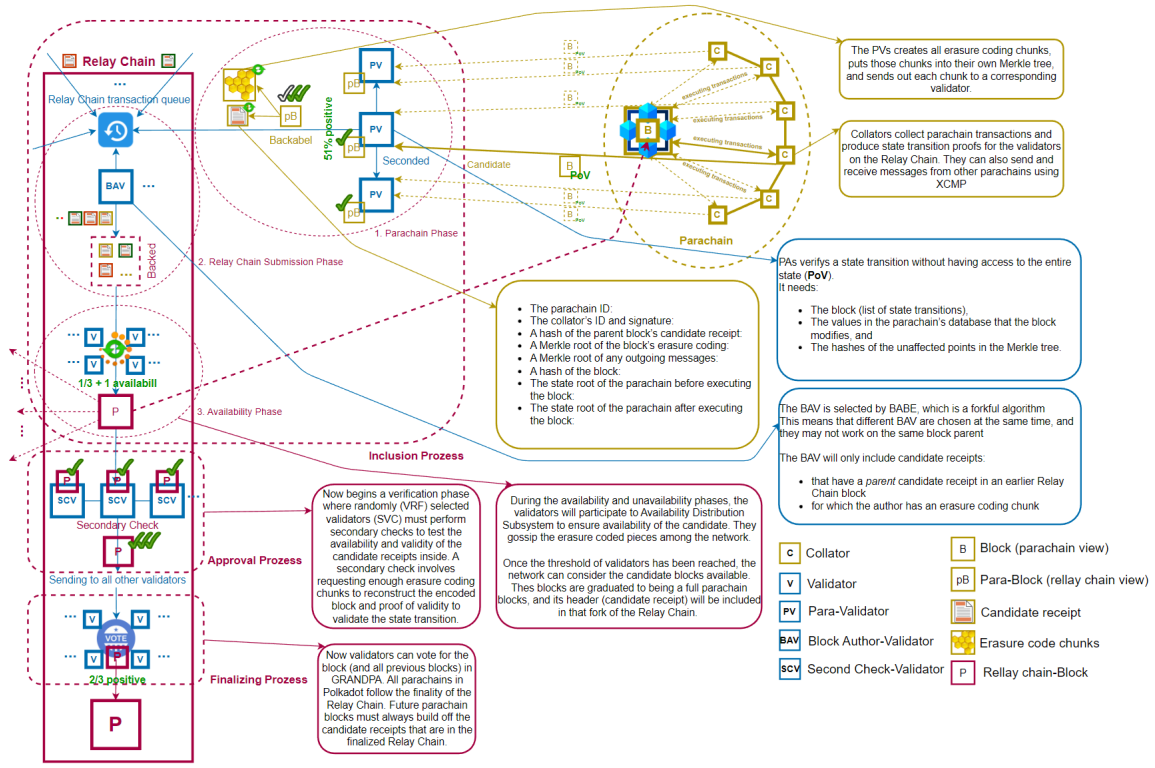

Figure 9: Overview of a Parachain Block Production in Polakdot 1.0. [35]

Apart from interoperability (no focus of Logos network) and the ability to communicate with other parachains, Polkadot offers a number of other benefits and features (shared security, scalability, upgradeability, heterogeneity, community-driven governance, ecosystem diversity, ...). For a more comprehensive understanding of Polkadot, read the original whitepaper [42].

Since Polkadot's new approach has been formed with so-called system or common-good parachains, the Logos chain may be classified as a system parachain in in the future. *System-chains*, a term coined by Polkadot [23], are chains containing the core protocols of the main system, and are exclusively tasked with network support (e.g. Asset Hub, Encointer, Bridge Hubs, ...). The sub0layer to be created by the Logos network is not technically limited to Polkadot, as the blochain nodes could use the computational layer regardless of the blockchain technology/frameworks (Substrate, Tendermint, Ethereum Virtual Machine, Ouroboros, Sealevel, ...) on which they are based.

At this point, however, it is not yet clear how this implementation would work as a system parachain and whether the Polkadots approach of the implementation would be the same as ours. For this reason, we have decided to develop and provide the Logos Chain as a standalone Substrate Chain for the initial period. Since the chain will be developed with the Substrate Framework, the migration to a parachain should not be a huge effort, in case other possibilities and ideas come up at that time.

2.2.1 Architecture and components

The Web3 infrastructure is the technological basis for the development and provision of decentralized applications and the communication between them. It is a decentralized, interoperable and

programmable infrastructure derived from blockchain and other distributed technologies. A new web infrastructure is created by "designating" blockchain nodes to layer-0 and layer-1 blockchains.

Substrate is an open-source framework developed by Parity Technologies, that enables the creation of custom blockchain networks for the Web3 era. Substrate provides a set of tools and libraries, that allow to design and implement blockchain protocols. It is specifically designed to facilitate the development of blockchain networks, while offering high flexibility and customizability. For a better understanding of Substrate, consider [30].

Blockchains in Substrate environments are defined by the runtime composition. Because the implementation of the DVCI will be carried out by smart contracts, the main focus of the runtime will have to be the provision of a infrastructural "Smart Contract Logic".

Because we aim to develop a standalone chain, the architecture has to be able to forward block candidates with valid proofs, created by the validators, to another validator in the network to validate this blocks. Theoretically only validators have to participate in candidate block production and validation. Different types of substrate nodes also exist (full node, archive node, light clients, boot nodes, ...) and will be responsible for different functionalities (transaction execution, transaction validation, interaction with the blockchain, ...). These types of nodes (full, archive, light and boot) augment the performance, security and accessibility of the main network participants (validators).

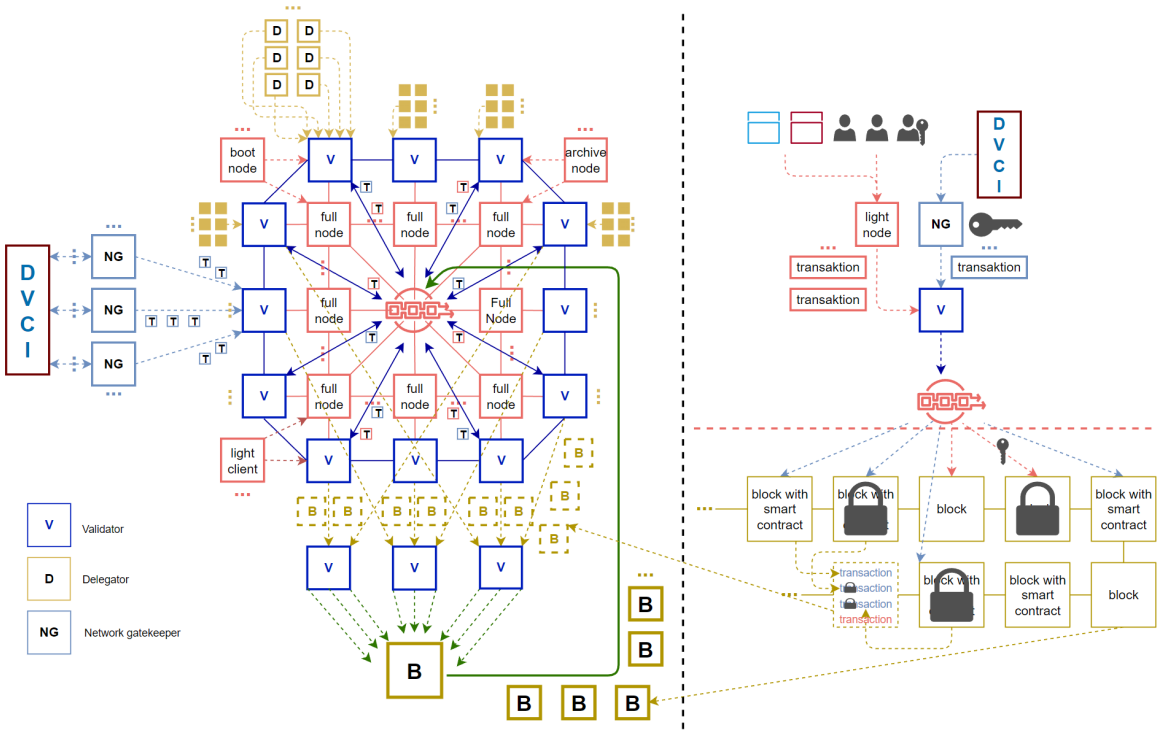

Figure 10: High level overview of the Logos Chain and its components.

BABE (Blind Assignment for Blockchain Extension) [19] in combination with **GRANDPA** (GHOST-based Recursive Ancestor Deriving Prefix Agreement) [20] will be used as consensus mechanisms for the Logos chain. BABE is a block-producing mechanism that relies on randomization to determine which validator is allowed to produce a new block next. This mechanism is designed to be fair and efficient and enables a high transaction processing rate. GRANDPA is used for the finalization of blocks, which means that it marks blocks as final after a certain consensus has been reached. This ensures security within the network by guaranteeing the irreversibility of transactions, even if there are network forks. By combining BABE and GRANDPA, blockchains can achieve a high transaction processing rate while providing robust security through fast finalization of blocks. This combination offers a good balance between performance and security. Since BABE and GRANDPA are already successfully used in a large network such as Polkadot, it can be presumed that this is a proven and tested solution.

Nominated Proof of Stake (NPoS) [21] is a staking model that determines how validators are selected to participate in the consensus mechanism. These validators then participate in block production and finalization. NPoS aims to maximize network security by promoting a broad distribution of tokens responsible for validation while ensuring that only the most reliable and best performing validators are selected. For more details on BABE, GRANDPA and NPoS see [26].

The combination of NPoS with GRANDPA and BABE in a hybrid (private and public elements) blockchain environment is usually not a preferred solution (e.g. AURA and Proof of Authority). The reason for this decision is that the Logos network itself should be public network but due to security risks some information and configurations cannot be public. There are several ways to solve this Problem. Approaches such as private transactions, access control lists (ACLs), encryption of data, off-chain data storage (only hashes of this data published in the blockchain), etc. will be evaluated and specified in detail in the upcoming PoC.

Client and runtime: A Substrate Node basically consists of two main parts, the core client and the runtime.

The Substrate core client is the component that runs on a computer or server to interact with the Substrate network. It is a user interface that makes it possible to communicate with the blockchain, create transactions and retrieve information from the network.

Runtime is the heart of Substrate and embodies the logic that drives the blockchain. The runtime determines whether transactions are valid or invalid and is responsible for processing changes to the blockchain state. External requests are forwarded to the runtime via the client, and the runtime is responsible for state transition functions and storing the resulting state.

The Substrate Runtime is designed for compilation in WebAssembly (Wasm) bytecode. WebAssembly (Wasm) [32] is a technology designed to compile code in a form that works efficiently, both on the web and on servers. A significant benefit of using Wasm in Substrate is the ability to update the blockchain runtime without hard forks. As the runtime is available as a Wasm binary, it can be updated by the blockchain's governance system. This enables flexible and agile further development of the blockchain.

Substrate libraries: There are three types of libraries in Substrate that relate to the development of Substrate-based blockchains. The Core Libraries, the FRAME Libraries and the Primitive Libraries.

- **Core client libraries:** These are libraries that enable a Substrate node to perform its network tasks, including consensus and block execution, as well as building the network layer, managing it and enabling communication between network participants and the transaction pool.
- **FRAME libraries:** The libraries that make it possible to create the runtime logic and to encode and decode the information transferred from the runtime. The FRAME libraries provide the infrastructure for the runtime. In addition to the infrastructure that is provided, the runtime can contain one or more "pallet" libraries.
- **Primitive libraries:** At the lowest level in the Substrate architecture, there are primitive libraries that provide control over the underlying operations and enable communication between the central outer client services and the runtime.

For detailed insights of the core-client, runtime and libraries, refer to [25]

Composition and usage of FRAME, appropriate PALLETS as well the definition of the runtime-logic itself are subject to future research, hence the present shortage of specifics.

Nodes and Roles: Every node in a Substrate environment can be configured individually. The access to nodes can be limited, nodes can be excluded from block production, communication restrictions can be imposed, etc. The prospect of individual node configuration will play a major role in the implementation of the Logos network. In order to simplify the maintenance of security and integrity of the Logos chain, five types of nodes, able to interact with the network, will be utilized.

- **Validator Node:** In a Substrate-based standalone chain, validators play a crucial role in maintaining security, integrity and consensus within the network. As key nodes, they are responsible for processing and validating transactions as well as creating and adding new blocks to the blockchain. Their tasks include verifying the validity of transactions according to network rules, ensuring compliance with smart contract logic and maintaining a consistent blockchain state to prevent malicious activities. By participating in the consensus process, validators contribute significantly to network security and help ensure a consistent state of the blockchain.
- **Boot Node:** Boot nodes, also known as bootstrap nodes, play a crucial role in the initialization process and network connection in a blockchain They are accessible at known, stable addresses

and are operated by trustworthy actors. Simply speaking, boot nodes serve as initial connection points in the network, facilitate peer discovery and support blockchain synchronization.

- **Full Node:** Full nodes store blockchain data and typically participate in common blockchain operations, such as creating and validating blocks, receiving and verifying transactions, and providing data in response to user requests. They support the network by ensuring consistency and reliability.
- **Archive Node:** Archive nodes are similar to full nodes, with the difference that they store all past blocks with the complete status for each block. Archive nodes are most commonly used by utilities such as block explorers, wallets, discussion forums and similar applications that require access to historical information.
- **Light client node:** Light Client Nodes enable interaction with a substrate network with minimal hardware requirements. They are not part of the blockchain or impact network operations. They can be embedded in web-based applications, browser extensions, applications for mobile devices, ...

For a better understanding of the base substrate nodes and roles, consider [28] and [29].

2.2.2 Smart Contract Logic

It is important to have a general understanding of the term smart contract [27] before explaining the term Smart Contract Logic further. Smart contracts are protocols that automatically, transparently and immutably execute contracts when predefined conditions are met. These contracts are written in code and are executed on a blockchain. Smart contracts automatically execute actions as soon as the predefined conditions are met. There is no need for intervention once the code has been deployed. Once deployed on a blockchain, the code of a smart contract cannot be changed without the consent of the network participants. This ensures the integrity and reliability of the contract.

The *Smart Contract Logic (SCL)*, simply speaking is a comprehensive set of different types of smart contracts, tasked with the execution of case specific instructions and measures as the interaction between those contracts. The term and the definition of "Smart Contract Logic" as such is only used in the Logos Ecosystem and should not be confused with the general term.

Three types of contracts should be defined on the Logos chain: *configuration smart contract (CSC)*, *operation smart contract (OSC)* and *privacy smart contract (PSC)*. The first two smart contract types (configuration and operation) will be used to implement, secure and provision the DVCI, while the third type makes it possible to deploy and manage specific private data (e.g. critical configuration data) on the blockchain without the use of private validators in the network.

The configuration smart contracts are intended to store the DVCI configurations in several central smart contracts. This makes it easier to manage and simplify the configuration data. In a smart contract that manages configurations, each key-value pair represents a specific configuration or setting (important parameters or options for the system).

The OSCs perform certain operations or execution logic based on the configurations. They will access CSCs in order to retrieve relevant configurations.

This type of structure allows a clear separation between the configuration and the operations elements, the key (key-value pair) plays a crucial role to identify the configurations. Through this approach, we want to define the DVCI with one type of smart contract (configuration) and use the others (operations) to create this infrastructure and keep it operational. Operations contracts can be used for each DVCI but the configuration smart contracts must be specific per DVCI.

The Substrate contract pallet [21], as provided by default, focuses primarily on the creation and management of smart contracts within the Substrate framework. Its core functions include the development, deployment and execution of smart contracts, the management of storage and state of the contracts, interactions between different contracts and fee management for the execution of contracts.

What the Substrate contract pallet does not feature directly, however, are functions for use cases such as the separate storage and management of configuration data in standalone smart contracts. If configurations are stored in separate smart contracts, they can be implemented modularly and reused by various other smart contracts. This enhances the reusability of code and configurations. Separate smart contracts for configurations make it also possible to update configuration data independently of the main contract (operations smart contract). This can be useful if configurations need to be modified.

To implement a system for storing configurations in independent smart contracts, separate solutions must be developed within the contract pallet. This may involve the creation of separate smart contracts for the management of configuration data or the development of special interfaces and protocols for communication and interaction between different contracts.

The privacy smart contract type is intended to enable private transactions and strict role-based access control in public substrate-based blockchain networks. This approach is essential for the development of the Logos chain since some configurations cannot be public for security reasons. The implementation of private validators in a public network can be challenging and adds to the complexity of the overall network. Centralizing block production in a small set of nodes also introduces security risks, especially if the nodes are compromised or act in coordination to compromise the network.

The PSC as well as the CSC and OSC type will be developed as separate substrate modules and can be included in the FRAME catalog after successful implementation in the Logos Chain, in case the community requests for a proposal.

In the smart contract logic, terms *groups*, *layers* and *sections* should be differentiated. In an infrastructure, there are various sectors such as network, security, computation, ..., which are divided here into so-called smart contract groups. The terms layer and section are very closely related. The DVCI is divided into different layers. In each of these layers, the DVCI is managed at a different level, so that, for example, the network is provided at different levels. This creates these different sections (network-section-1, network-section-2, ..., compute-section-1, compute-section-2, ...) which are organized into different smart contracts.

The smart contracts will be developed with Rust [37] and the ink! Framework and compiled in Wasm bytcode so that they can be executed on the Substrate-based Logos chain. This is usually done with the Rust compiler, which is configured to compile Rust code in WebAssembly. The ink! Framework facilitates this process by providing the necessary tools and configurations. For detailed insights of the ink! framework , refer to [38]

The entire implementation of the smart contract logic will be provided with the PoC and the technical specification paper, as the implementation solutions still need to be clarified.

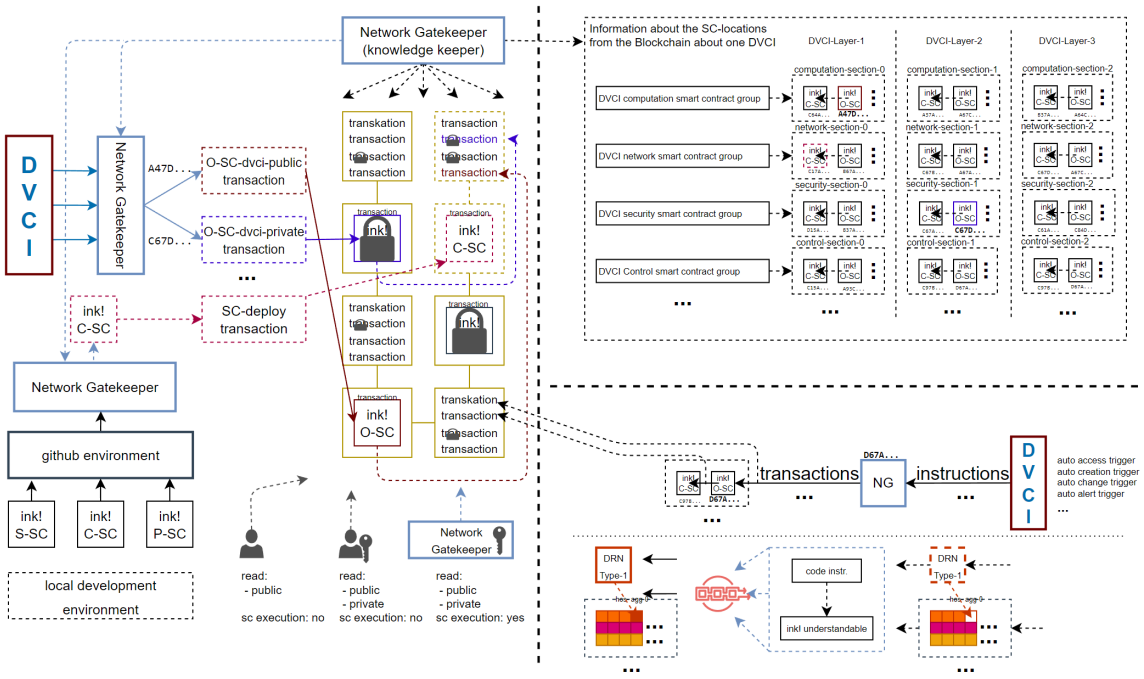

Figure 11: High level overview of the the smart contract logic implementation.

It is important to underline again that the approach is not the provision of an oracle, it is an attempt to operate the DVCI on-chain (smart contract execution time not relevant). From our perspective, the Logos network is the manifestation of what is defined on the blockchain.

2.3 Network Gatekeeper

The network gatekeeper(NG), one of the most important components of the Logos network, will take on a multitude of tasks and will have to consist of multiple components. Communication between

the DVCI and the blockchain will entirely depend on the NG. Another crucial duty of the NG will be the oversight of node behavior, in case of deviation of default node behavior, exclusion of affected participants and determination of the future procedure regarding those participants. Simply speaking,this kind of *middleware* (NG) will be used to define the DVCI and ensure the validity of its operations on the blockchain.

The DVCI components will require different NG components to communicate with the blockchain. Not all of the DVCI components require all NG functionalities, which is why a distinction is made here between the *Main NG* and the *Client NG*. The difference is that the Main NG includes additional components (*DVCI and W3bI validators*, *Knowledge Keeper* and *Reference Libraries*). These components are required for the provisioning of the Logos network and will be only installed on the OpenStack nodes since they are only required there.

The network gatekeeper (entirely) will be defined and executed on the Logos chain in the form of OSCs and CSCs. For example, when a component such as a DRN wants to participate in the network, the NG (client) that will enable its interaction with the blockchain will be first deployed on the DRN. The deployment of this NG on the DRN will be handled by blockchain. This means that the client NG (client NG in this case) will only be deployed after the successful execution of the transaction and will be used to submit requests to the blockchain.

The reason for the development of the Network gatekeeper is primarily to enable such an automated infrastructure. If you look at the CNCF (Cloud Native Computing Foundation) landscape [33], you realize how quick the development (technologically) in the cloud-native environment is progressing. The Web3 world can benefit from some of these technologies (currently benefits but to a certain level) and that is why we want to make it easier to implement such technologies with the NG, allowing them to be defined and used more simply "on-chain". The network gatekeeper should enable automated communication of "cloud-native technologies" with the blockchain. This means that, for example, "native requests" (e.g. "kubeadm join ...") are understood by the blockchain via the gatekeeper (implementation in the reference libraries) and transactions can be triggered automatically to execute the request.

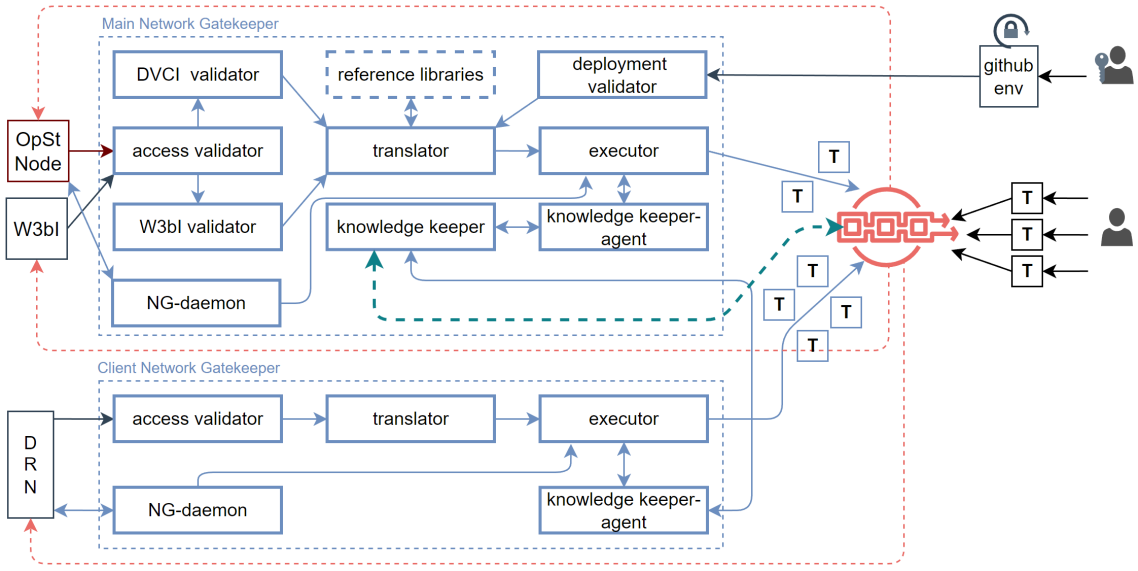

Figure 12: High level overview of the the Network Gatekeeper.

Access validator: The *Access validator* will be responsible for the validation of incoming requests (transactions). Since not every random request should be forwarded to the blockchain, the *AV* will have to perform a simple evaluation of the type of request and verify its compliance with prerequisites to trigger a transaction. In case a request is valid, and came from a DRN it will be send to the "Translator" directly. If this request comes from the DVCI or from the computational distribution regulator W3bI, the access validator will provide this evaluated transaction to the dedicated validator.

DVCI validator: The *DVCI validator* will process requests explicitly from the DVCI itself. Whereas the "AV" is appointed to DVCI-layer-1, *DVCI-V* will be directed at layers 2 and 3.

W3bI validator: The *W3bI validator* will process requests from the computation distribution regulator. (chapter 2.4)

Deployment validator: The *Deployment validator* will perform a simple evaluation of the Logos chain deployment requests (chain and SCL configuration deployments).

Translator: The *Translator* is a categorized set of data points with assigned key-values, created, sorted and validated by the validators. The correct representation of these data points is necessary to produce a transaction, which will be the main task of the "TL". Simply speaking, the *TL* will generate a request, in the required format (in form of a transaction) to trigger smart contracts, and forward it to the *Executor*.

Reference libraries: The *reference libraries* is an additional data-source (set of rules and guidance's) that the *TL* use to produced specific (DVCI and W3bI) transactions. The reference libraries will be developed in such a manner, that the NG can be used for other use cases like the implementation of other cloud native technologies. For Example if a library for Kubernetes is defined in the NG, the translator will be able to provide a native call to trigger a Operations Smart Contract. Since we will not only use one existing technology ourselves, the idea is to implement the reference libraries as a type of a "configuration language libraries".

Executor: The *Executor* will deliver the request, with guidance from a *Knowledge Keeper*, to the blockchain.

Knowledge keeper and the agent: The *Knowledge Keeper* will contain data regarding the Smart Contract Logic. Furthermore the *KK* will be tasked with the collection, organization and local storage of block chain data (SC location and metadata). In order for the Executor to receive the requested contract addresses (guidance) as fast as possible, an *Agent* will be implemented to search the stored data.

Network gatekeeper daemon: The *network gatekeeper daemon* will be the component responsible for security, thus playing a major role in the overall infrastructure. The *NGd* will run on the host-OS of each component and ensure that the configurations "uploaded" to the node by the blockchain are intact and unchanged, otherwise excluding the node from the infrastructure under a zero-tolerance policy.

The middleware (Network Getkeeper) will be developed as open source. The focus of LogosLabs are "technologies" (OpenStack, Terraform, Ansible, ...) to be utilized for the Logos network. Once the middleware is developed and functional, there is always the possibility to use the code to implement more solutions (implementation in the reference libraries).

2.4 Computation Distribution Regulator W3bI

The W3bI will be a computation distribution regulator in the Logos network and of the ecosystem and will be primarily utilized to "host" blockchain nodes. The regulator will additionally be used to provide alternative computation service environments that can be used for community Web3 projects in compliance with the Logos Network philosophy. The main approach, however, is to use the regulator to provide the sub0layer, created by the Logos Network, for the Web3 by enabling automated provisioning of blockchain nodes (layer-0 and layer-1 blockchain infrastructures) on the sub0layer.

The service will operate similarly to the current blockchain infrastructure providers, except that it will be defined on the blockchain itself and the core infrastructure used in the backend will be a full Web3 compute infrastructure. The W3bI will be developed and operated by LogosLabs in accordance with fair-price principles defined in the Logos network philosophy. The regulator will utilize the fairly-priced (community governance) payment model (*pay-as-you-use*) [41] for compute-units. The computation distribution regulator will be defined on the same layer-1 blochain (Logos chain) as the Logos network itself, as it will handle the distribution of the Logos network resources and is therefore an important element of the ecosystem.

The W3bI service will be provided in a service environment made available by the Logos network. An entire platform (Web3 IaaS) is to be developed and provided in this service environment. For the provision of the W3bI service, the same implementation strategies will be used as for the Network

Gatkeeper (OSCs and CSCs). Definition and execution on the blockchain to perform multiple functions. The important thing to note here is that the execution time of the smart contracts on this layer has an impact on the services. For this reason, "off-chain" elements will be implemented here to make the platform more efficient.

The terms "on-chain" and "off-chain" fundamentally refer to different types of data processing and storage. "On-chain" refers to activities or transactions that take place directly on the blockchain. "Off-chain", on the other hand, refers to activities that take place outside of the blockchain and are not stored on the blockchain.

Off-chain transactions are related to the blockchain, even if they take place outside the blockchain. They are part of the broader blockchain ecosystem and often interact in some form (complementary function, on-chain anchoring, trust anchor, ...) with on-chain data. In practice, combinations of on-chain and off-chain solutions are often used to leverage the benefits of both approaches, especially in terms of performance, cost, scalability and privacy.

The W3bI will be developed and operated by LogosLabs in accordance with fair-price principles defined in the Logos network philosophy. The approach of W3bI is to facilitate the provision of computational resources (Logos network resources) and not to be a profit oriented service. Like the Logos network itself, W3bI will be part of the *common-good* initiative to encourage and contribute to the development of Web3.

This service will not be economically independent, since it is part of the network itself. Even if the service does not function completely autonomously and is maintained by LogosLabs, it must be regulated in order to comply with the "limited-profit" approach. The "limited-profit" approach will become part of the Logos governance system. This means that the provider (LogosLabs) of the W3bI service is obliged to comply with the pricing decided by the community in compliance with the Logos network philosophy.

2.4.1 Cloud computing

The type of "platform" (W3bI) that will be developed is generally referred to as a "Blockchain-as-a-Service" (BaaS) [43] platform. It is equivalent to other "as-a-Service" models in the cloud sector, such as Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) or Infrastructure-as-a-Service (IaaS) (IaaS/PaaS/SaaS definitions [34]), but specifically focused on blockchain technologies. This is a commonly used term that can be interpreted in many different ways. From the LogosLabs perspective, W3bI should be a *Web3 IaaS platform/infrastructure*.

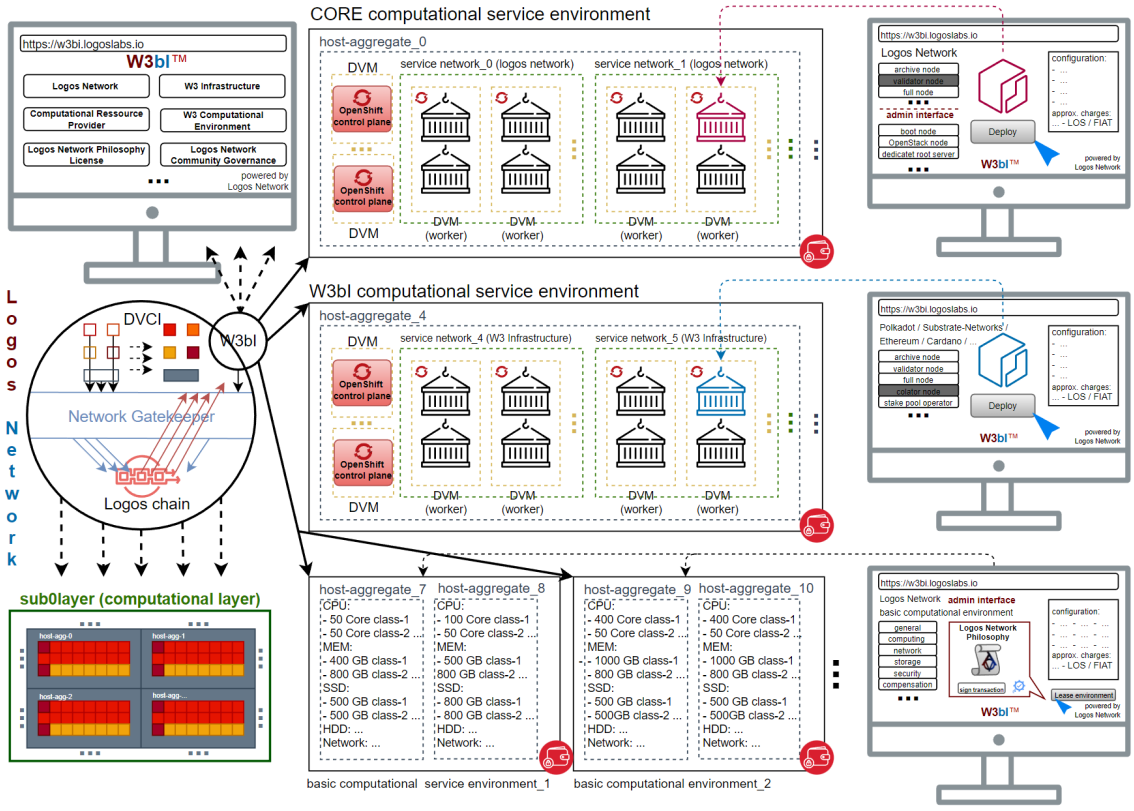

Figure 13: High-level overview of the computation distribution regulator; W3bI

The W3bI platform (primary utilization) is intended to offer a range of tools and services to simplify the process of deploying, maintaining and managing the Web3 infrastructure. The functionalities of the blockchain (smart contracts, transaction processing, consensus mechanisms, ...) will not be provided by W3bI. The focus will be on providing the necessary hardware and network resources (through Logos network) required to operate these blockchain functions. In short, the W3bI IaaS platform is a specialized form of cloud infrastructure that aims to simplify the setup and management of blockchain nodes, whereas the actual blockchain functionalities remain the responsibility of the developers of these blockchains.

The W3bI platform is generally intended to offer various options to further improve and facilitate the provision of the Web3 infrastructure. Users will be able to set up and operate blockchain nodes without requiring technical knowledge. This platform will also offer tools and services for monitoring, maintaining and updating the nodes to ensure their continuous functionality and security. Another important aspect is scalability. Users will have the flexibility to increase or decrease the number of nodes as required. These and other features make the W3bI platform an essential component for the efficient management and provisioning of the Web3 infrastructure (blockchain infrastructures).

Detailed specifications of the architecture will be provided with the technical specification paper.

3 Network realization

At this point, all key-components (DVCI, Logos chain, the network gatekeeper and the computational distribution regulator; W3bI), comprising the Logos network, have been described in detail. Before deep-diving into the realization of the network and its role in the World Wide Web, the Logos network itself has to be distinguished from the future "services" (additional compute service environments) (except W3bI) using it.

Simply speaking, the Logos network will be an autonomous infrastructure that provides service-specific (Logos Ecosystem) computational/service environments, the "services" on the other hand will be "customers" (consumers of computational resources) of the Logos network. This "customers" will be able to "rent" a computational environment for a specific *lease-time*. As described earlier, the main intention of LogosLabs is to further secure and truly decentralize the Web3 infrastructure. This shall be accomplished by the "sub0layer" and the advantages Web3 presents over the conventional tech-sphere (Web2).

Figure 14 shows the entirety of the Logos Ecosystem in three stages. The DVCI is fully defined on the blockchain(code) and will be supplied with enough physical hardware for its execution. The communication between the blockchain and physical hardware (via the network gatekeeper) is therefore required.

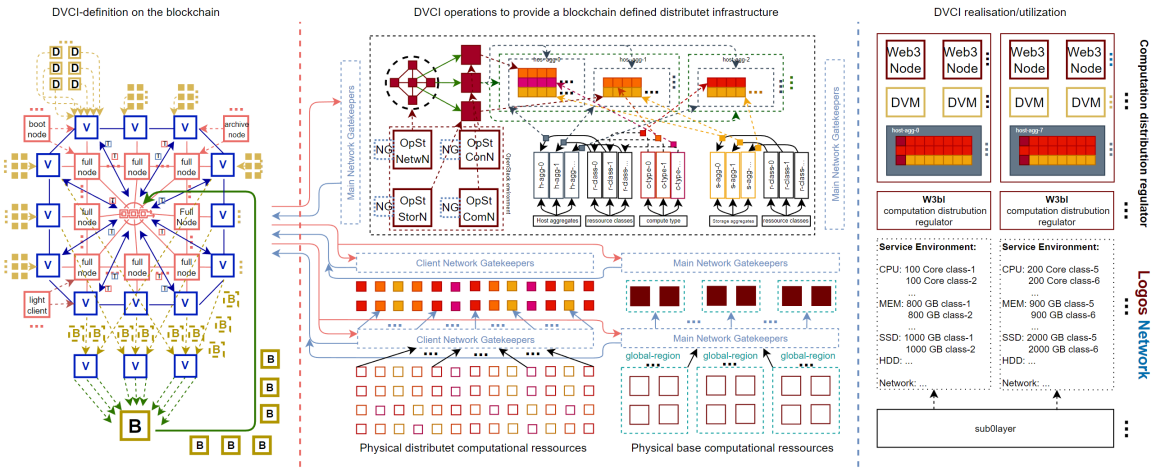

Figure 14: General overview of how the Logos network will be utilized

The "services" supported by the Logos Ecosystem will have access to computational resources provided by the DVCI (sub0layer), making the DVCI compatible with many Web3 based cloud services. The decision over the inclusion of a service into the network or a potential increase in compute power allocated to a service shall be, under the *Logos network philosophy*, ruled on by the community. The same Logos network philosophy requirements applied to W3bI will be applied to

all other services, but as the computation distribution regulator is the core service of the ecosystem, W3bI will not have a conventional lease time as it is intended to facilitate the implementation of the Logos network itself into the World Wide Web. The Logos ecosystem (web3 community projects) shall focus exclusively on three fields; Decentralization, education and science. Additional services could be implemented over time, provided they fit those fields.

For the Logos network to be able to provide the required resources (service environments), the network must be reimbursed. Such reimbursement should be carried out in the network's native currency. The exact implementation will be defined once a governance system has been established.

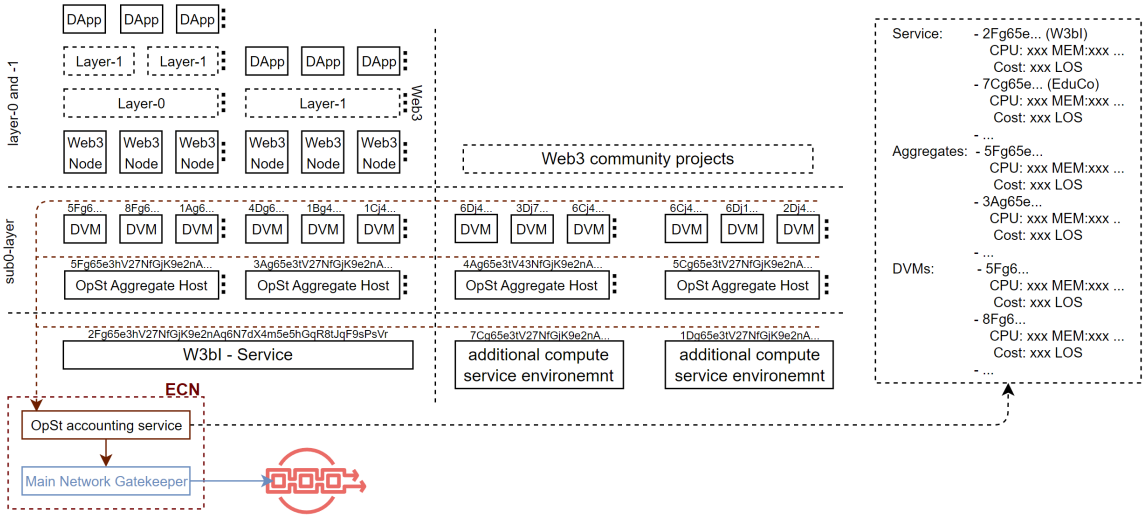

Figure 15: High level overview of the resource accounting implementation for Logos ecosystem services.

A community based compute service for educational purposes was the main intent behind LogosLabs. However during the development of the concept we realized, that a sub0layer would also be advantageous for Web3 altogether, and further help counteract its increasing centralization. For this reason, the main focus of LogosLabs shifted to the development of a sub0layer for all web3 infrastructures.

The main focus of this concept paper is to summarize and understand the idea behind the Logos network and the sub0layer. The project, its development and implementation is a challenge and may be different from the concept in the future. With this project we aim to create a community and blockchain based solution for Web3 computation.

References

[1] AMD ROCm official documentation. *AMD ROCm documentation*. URL: https://rocm.docs.amd.com/en/docs-6.0.0. (accessed: 2023).

[2] Ansible-Community official documentation. *Introduction to Ansible*. URL: https://docs.ansible.com/ansible/latest/getting_started/introduction.html. (accessed: 2023).

[3] GitLab official documentation. *What is CI/CD?* URL: https://about.gitlab.com/topics/ci-cd/. (accessed: 2023).

[4] GitLab official documentation. *What is GitOps?* URL: https://about.gitlab.com/topics/gitops. (accessed: 2023).

[5] Hashicorp official documentation. *What is Terraform?* URL: https://developer.hashicorp.com/terraform/intro. (accessed: 2023).

[6] KVM official documentation. *KVM*. URL: https://www.linux-kvm.org/page/Main_Page. (accessed: 2023).

[7] libvirt official documentation. *Connection URIs*. URL: https://libvirt.org/uri.html. (accessed: 2023).

[8] libvirt official documentation. *libvirt*. URL: https://libvirt.org/index.html. (accessed: 2023).

[9] libvirt official documentation. *Libvirt Daemons*. URL: https://libvirt.org/daemons.html. (accessed: 2023).

[10] libvirt official documentation. *Main libvirt APIs*. URL: https://libvirt.org/html/#main-libvirt-apis. (accessed: 2023).

[11] libvirt official documentation. *Remote support*. URL: https://libvirt.org/remote.html. (accessed: 2023).

[12] OpenCL official documentation. *OpenCL Resource Guide*. URL: https://www.khronos.org/opencl/resources. (accessed: 2023).

[13] OpenStack official documentation. *Compute architecture*. URL: https://docs.openstack.org/arch-design/design-compute.html. (accessed: 2023).

[14] OpenStack official documentation. *Control plane architecture*. URL: https://docs.openstack.org/arch-design/design-control-plane.html. (accessed: 2023).

[15] OpenStack official documentation. *Design*. URL: https://docs.openstack.org/arch-design/design.html. (accessed: 2023).

[16] OpenStack official documentation. *Network architecture*. URL: https://docs.openstack.org/arch-design/design-networking.html. (accessed: 2023).

[17] OpenStack official documentation. *Storage architecture*. URL: https://docs.openstack.org/arch-design/design-storage.html. (accessed: 2023).

[18] OpenStack official documentation. *Storage concepts*. URL: https://docs.openstack.org/arch-design/design-storage/design-storage-concepts.html. (accessed: 2023).

[19] Parity official documentation. *BABE Pallet*. URL: https://paritytech.github.io/polkadot-sdk/master/pallet_babe/index.html. (accessed: 2024).

[20] Parity official documentation. *Grandpa Pallet*. URL: https://paritytech.github.io/polkadot-sdk/master/pallet_grandpa/index.html. (accessed: 2024).

[21] Parity official documentation. *Staking Pallet*. URL: https://paritytech.github.io/polkadot-sdk/master/pallet_staking/index.html. (accessed: 2024).

[22] Polkadot official documentation. *Parachains*. URL: https://wiki.polkadot.network/docs/learn-parachains. (accessed: 2023).

[23] Polkadot official documentation. *Parachains*. URL: https://wiki.polkadot.network/docs/learn-system-chains. (accessed: 2023).

[24] QEMU official documentation. *QEMU*. URL: https://www.qemu.org/docs/master/about/index.html. (accessed: 2023).

[25] Substrate official documentation. *Architecture and Rust libraries*. URL: https://docs.substrate.io/learn/architecture. (accessed: 2023).

[26] Substrate official documentation. *Consensus*. URL: https://docs.substrate.io/learn/consensus/. (accessed: 2024).

[27] Substrate official documentation. *Decide what to build*. URL: https://docs.substrate.io/design/decide-what-to-build/. (accessed: 2023).

[28] Substrate official documentation. *Networks-and-nodes*. URL: https://docs.substrate.io/learn/networks-and-nodes. (accessed: 2023).

[29] Substrate official documentation. *Node roles and responsibilities*. URL: https://docs.substrate.io/deploy/prepare-to-deploy. (accessed: 2024).

[30] Substrate official documentation. *Welcome to Substrate*. URL: https://docs.substrate.io/learn/welcome-to-substrate/. (accessed: 2023).

[31] VMware official documentation. *What is Software-Defined Networking (SDN)?* URL: https://www.vmware.com/topics/glossary/content/software-defined-networking.html.html. (accessed: 2023).

[32] Webassembly official documentation. *Webassembly*. URL: https://webassembly.org/. (accessed: 2023).

[33] Cloud Native Computing Foundation. *CNCF Cloud Native Interactive Landscape*. URL: https://landscape.cncf.io/. (accessed: 2023).

[34] Geeksforgeeks. *Cloud Based Services*. URL: https://www.geeksforgeeks.org/cloud-based-services/?ref=lbp. (accessed: 2023).

[35] Parity official guide. *The Polkadot Parachain Host Implementers' Guide*. URL: https://paritytech.github.io/polkadot-sdk/book/index.html. (accessed: 2023).

[36] David Heinemeier Hansson. *The Big Cloud Exit FAQ*. URL: https://world.hey.com/dhh/the-big-cloud-exit-faq-20274010. (accessed: 2023).

[37] Steve Klabnik and with contributions from the Rust Community Carol Nichols. *The Rust Programming Language*. URL: https://doc.rust-lang.org/book/ch00-00-introduction.html. (accessed: 2023).

[38] Michi Müller. *What is Parity's ink!?* URL: https://www.parity.io/blog/what-is-paritys-ink. (accessed: 2023).

[39] Wikipedia. *Hypervisor*. URL: https://en.wikipedia.org/wiki/Hypervisor. (accessed: 2023).

[40] Wikipedia. *Infrastructure as code*. URL: https://en.wikipedia.org/wiki/Infrastructure_as_code. (accessed: 2023).

[41] Wikipedia. *Pay-as-you-use*. URL: https://en.wikipedia.org/wiki/Pay-as-you-use. (accessed: 2023).

[42] Dr. Gavin Wood. *Polkadot original whitepaper*. URL: https://assets.polkadot.network/Polkadot-whitepaper.pdf. (accessed: 2023).

[43] Jie Song; Pengyi Zhang; Mohammed Alkubati; Yubin Bao; Ge Yu. *Research advances on blockchain-as-a-service: architectures, applications and challenges*. URL: https://www.sciencedirect.com/science/article/pii/S2352864821000092. (accessed: 2023).

Glossary

Name	Acronym	Description	Definition
Access validator (NG)	AV	The Access validator (NG) is responsible for the validation of incoming requests (Logos network).	2.3
Computing environment	CoE	The DVCI-layer-2 or back-end phase takes on layer-specific actions in managing, connecting and securing the DVCI.	2.1.1
Computational network	CoN	A low-latency software-defined network that enable internal networking in a DVCI.	2.1.4
Configuration smart contract	CSC	This smart contracts store the DVCI configurations.	2.2.2
Client environment	ClE	This environment is the "execution-layer" for the DVCI and act as the access point for services.	2.1.1
Dedicated root server	DRS	These servers are used for providing an OpenStack environment for the DVCI in a global region.	2.1.1
Deployment validator (NG)	DV	The DV (NG) process the blockchain-configuration deployment requests	2.3
Distributed Virtual Computing Infrastructure	DVCI	A Web3 cloud infrastructure.	2.1
Distributet virtual machine	DVM	VMs created in a DVCI host-aggregate.	2.1.1
Distributed resource Node	DRN	Provider of distributed resources.	2.1.1
DVCI book (NG)		Data-source for the Translator (NG).	2.3
DVCI validator (NG)	DVCI-AV	The DVCI-AV (NG) process requests explicitly from the DVCI.	2.3
Environment controller node	ECN	The ECN consists of an OpenStack controller node and the main network gatekeeper.	2.1
Executor (NG)		The Executor (NG) deliver requests to the blockchain.	2.3
Global region		Computational resources (DVCIs) divided into geographic (global) regions.	2.1.1
Knowledge keeper (NG)	KK	The Knowledge keeper (NG) contains data regarding the smart contract logic.	2.3
LN computation distribution regulator	W3bI	Core service of the Logos network and the ecosystem which provides the sub0layer, created by the Logos Network	2.4
Logos network	LN	Web3 computational infrastructure.	2.0
Logos chain	LC	Substrate based smart contract infrastructure.	2.2
Network gatekeeper	NG	Middleware for the Logos network.	2.3
Network gatekeeper daemon	NGd	The daemon (NG) is the component responsible for security monitoring.	2.3
OpenStack environment		OpenStack (environment) is a comprehensive cloud-infrastructure solution.	2.1.1
Operation smart contract	OSC	This smart contracts perform certain operations or execution logic based on the configuration smart contracts.	2.2.2
Region bridge network	RBN	Is a network architecture approach (concept) in which services from one global region can access resources from another global region.	2.1.4
Region environments	RE	The term region is used as a collective term for layer-2 and layer-3.	2.1.1
sub0layer		A decentralized core computing environment acting as the underlying layer to a Web3 infrastructure(layer-0)	1

Name	Acronym	Description	Definition
Service computational environment	SCE	Environments that provided by the Logos network (computation utilization)	2.1
Service network	SeNe	This networks are virtual networks that are provided by the OpenStack network nodes (neutron service)	2.1.4
Smart contract logic	SCL	Comprehensive set of smart contract‘s, tasked with the execution of case specific instructions	2.2.2
Privacy smart contract	SSC	Contract type that defines and implement the network gatekeeper and W3bI on the chain	2.2.2
Translator (NG)	TL	Generate a requests, in the required format (in form of a transaction) to trigger smart contracts, and forward it to the Executor (NG)	2.4
W3bI validator (NG)	W3bI-V	W3bI-V (NG) process requests from W3bI.	2.3

Table 1: Glossary for the Logos network