Cloud Working Group

1. Motivation

In a cloud-based world as we are today and will be even more in the future, the facilitation of easy integration between digital solutions and cloud environments is essential.

2. Main Goals

- Automatization of Nodes Deployment
- Dynamic DNS
- Cloud Key Management (Orion)

3. Participants

- Tech Lead: Sergio Cerón Figueroa
- Coordinator: Antonio Leal
- Supervisor: Marcos Allende

4. Detailed description

4.1 Automatization of Nodes Deployment

Background

The current node provisioning mechanism makes it difficult to deploy LACChain nodes quickly and without errors in different computing environments. Despite performing a dependency installation automation, it is possible that there are requirements and situations where there are inconveniences for the correct start-up of LACChain nodes.

Running applications in containers is one of the fastest-growing technologies in the recent history of the software industry. At its heart lies Docker container engine, a platform that allows users to easily pack, distribute and manage applications within containers.

Virtualization, on the other hand, is not based on container technology. They are made up of user space plus kernel space of an operating system. Under VMs, server hardware is virtualized. Each VM has their own Operating System and It shares hardware resource from the host. However, and regardless of their differences, they both converge on one thing, automation and infrastructure scaling for platforms primarily in the cloud.

Proposal

The idea behind this proposal is to automate the deployment of LACChain nodes on cloud-based platforms, in order to make use of the advantages of both deployment paradigms (containerization and virtualization) to decrease the difficulty and time of installation and launching of services of nodes such as *Hyperledger Besu* and *Orion*.

Each of the LACChain nodes involves different installation and deployment mechanisms, so it is important to provide options for any member of the network who wants to join, either publicly or privately and regardless of the permissioning process.

In this proposal, two methods are described as an alternative to the current provisioning of LACChain nodes, showing their advantages and the general architecture of the implementation:

a) Containerize LACChain Nodes

The idea of dockerized LACChain nodes is basically generate containers for the main services of each of them, but that run within a single physical (or virtual) host, sharing and optimizing hardware resources. Below are the different advantages of using docker as a method of generating containers.

Advantages

- **Cost Saving:** Docker container engines can help cost reduction by dramatically reducing infrastructure resources. Because of Docker container engines reduced infrastructure requirements maintaining any integration with the LACChain network can decrease infrastructure costs.
- **Standardization:** Docker containers ensure consistency across multiple developments and cloud environments, release development and deployment cycles and helps standardizing development and production environment.
- **Rapid Deployment:** Docker technology helps to reduce deployment time to seconds. This is because it creates a container for every process and does not boot a full-fledged OS.
- **Multi-Cloud Platforms:** Docker containers can run inside an Amazon EC2 instance, Google Compute Engine instance, Rackspace server or VirtualBox, provided that the host OS supports the Docker container engine.
- **Isolation and Security:** from a security point of view, Docker technology ensures that applications running on containers are completely segregated and isolated from each other, granting you complete control over traffic flow and management. No container can access to processes running inside another container. From an architectural point of view, each container gets its own set of resources ranging from processing to network stacks.
- **Fault-tolerance:** if any of the blockchain nodes fail, blockchain data remains intact if stored outside of containers. This makes blockchain data synchronization more reliable than conventional instance servers.
- **Scalability:** Blockchain networks base their security and trust on the number of nodes that are part of them, increasing and expanding the network infrastructure is an important task. Docker allows to scale resources horizontally in a simple and efficient way.

Architecture

Figure 1 shows the general architecture of the LACChain nodes using docker as the engine for creating containers and cabinets as deployment orchestrator and autoscaling.

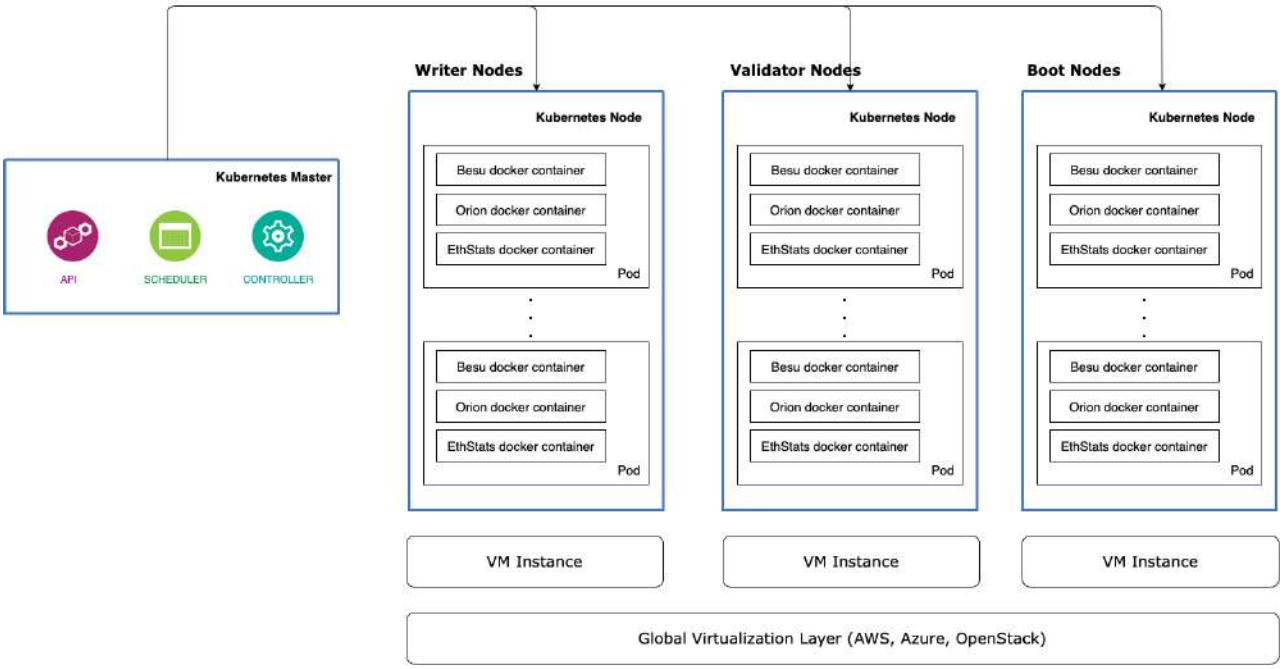

Figure 1. General Architecture of dockerized LACChain Nodes.

b) Generate LACChain Cloud Images

Additionally, and independent of dockerizing, it is important to maintain preconfigured virtual images of the LACChain nodes to optimize the deployment of nodes where direct access to the hardware is required without having the restrictions of container engines.

However, it is also possible to combine both architectures for a better use of virtualized resources.

This type of images allows to deploy nodes on different cloud computing platforms in a matter of minutes, with all the resources and services installed and preconfigured. The most important advantages of this deployment approach are described as follows.

Advantages

- **Deterministic Resources:** With virtual images we have the ability to quickly and efficiently determine what computing power, memory, storage, and other resources is required for LACChain nodes.
- **Faster Deployment:** the cloud image also speeds up configuration and deployment because the templates are well-known and defined for every LACChain node and their computing infrastructure requirements**.**
- **Bootstrapping:** Image creation is similar to a snapshot of a virtual machine, making it possible to initialize instances with old data, such as pre-synchronized blockchains.

Architecture

Figure 2 shows the use and deployment of nodes in LACChain using preconfigured virtual machines in different cloud computing services.

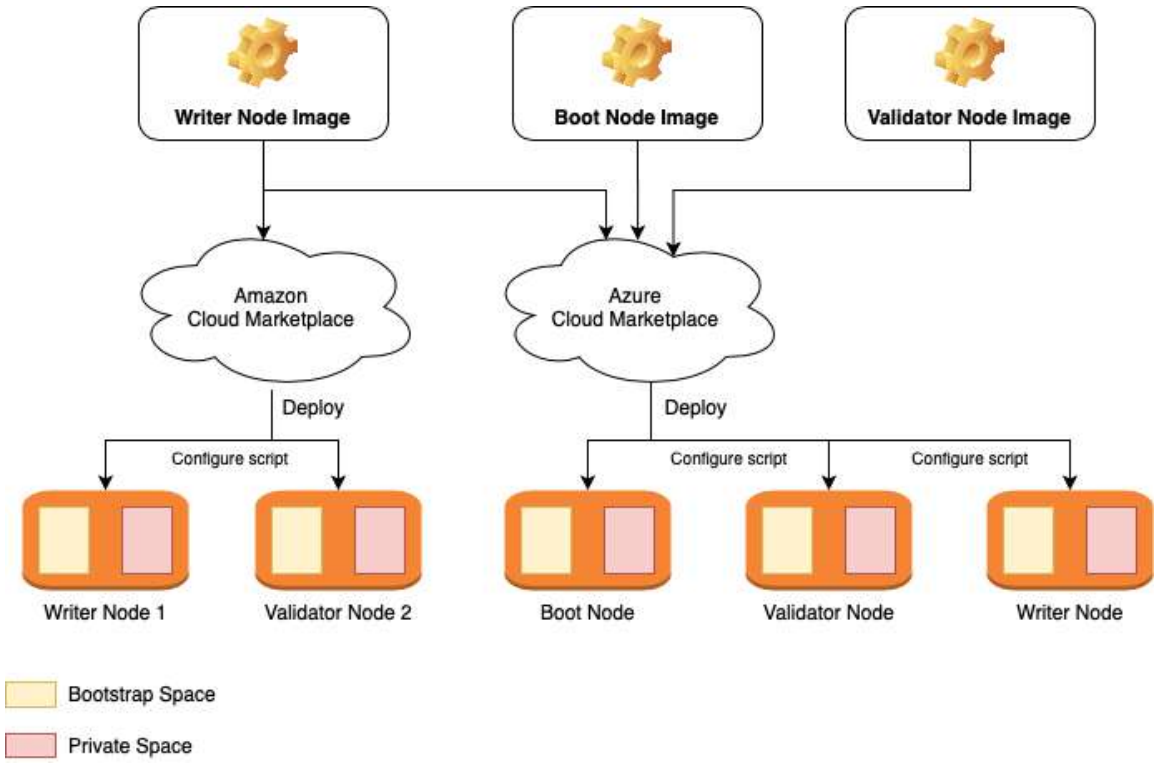

Figure 2. Using of AMIs to deploy LACChain nodes

Bootstrap space: This is the preconfigured space, includes the operating system and all the necessary services to run the LACChain node.
Private Space: This refers to a space created only for that instance and depends on the initial configuration of each deploy, also can be encrypted.

Technical milestones:

- Dockerize LACChain Nodes
  - Isolation of services
    - Isolate the services running inside each LACChain node (Hyperledger Besu, Orion, Nginx, EthStats and HealthCheck).
    - Establish and configure communication ports for each container.
    - **Deliverable**: Services and configurations files isolated.
  - Integration Test
    - Perform integration tests mainly between Hyperledger Besu and Orion for sending private transactions.
    - **Deliverable**: Testing Reports.

- Dockerization of services
    - Create flexible and persistent storage volumes for Besu and attach to the docker container, this in order to keep the blockchain synchronized, regardless of whether the container fails or is deleted.
    - Generate a container for each of the services and publish it within the docker registry.
    - **Deliverable**: Docker images and public registry ID.
- Automatization of deployment and autoscaling
    - Integrate Kubernetes to maintain load balancing, instance monitoring and auto-scaling in case of resources limitation or failure.
    - Create an orchestration service to manage different node types (validator, boot and writer).
    - **Deliverable**: Kubernetes Installed and configured with Docker images.

▪ LACChain Cloud Images
- Clean installation of LACChain Nodes (Boot, Validator, Writer)
    - Reset all preconfigured files with ansible provisioning scripts
    - Stop all services
    - Remove all generated files from installed services
    - Create a simple bash script to configure all files just like ansible, but interactively
    - **Deliverable**: Clean installation of LACChain (Writer, Boot and Validator) nodes.
- Virtualization of nodes
    - Create a virtual machine with minimal resources for each node type
    - Deploy and publish generated image inside the provider cloud marketplace
    - Export and import image from one cloud provider to other
    - **Deliverable:** Virtual image of LACChain (Writer, Boot and Validator) nodes.

4.2 Dynamic DNS

Background

The current method of interconnecting and permissioning nodes is done using the standard URLs known as *enode*. Each Hyperledger Besu node has an identifier, which is made up of its public key, IP address, and TCP port.

The hexadecimal node ID is encoded in the username portion of the URL, separated from the host by an @ sign. The hostname can only be given as an IP address, DNS domain names are not allowed. The port in the host name section is the TCP listening port (https://eips.ethereum.org/EIPS/eip-1459).

The on-chain permissioning mechanism used by Besu, registers the enode identifier using a smart contract, and receives the required parameters, and that are mainly used by the validator and boot nodes to allow the connection to the blockchain network.

Proposal

The identification method for each of the nodes in the LACChain network requires all the parameters be constant over time. While the public key of the node never changes, the TCP port or the IP address can be dynamic, causing a problem mainly in the on-chain permission mechanism of the nodes that require access to the network.

The use of DNS domain names is a very common way to replace IP addresses, especially when the IP addresses are not static. However, the use of qualified names is not allowed, even more in permissioning contracts. Therefore, the solution to this problem can be approached in 2 ways:

1. Use a cloud service that stores in a key-value database, and additionally a process that runs within each node to publish and update the IP addresses of all nodes on the network.
2. Modify the node identifier with a domain name instead of its IP address and modify the on-chain permission contracts to allow the use of a string instead of a byte16 value for the node address.

Clearly, the first option requires more effort since it requires to develop two services, one that acts as a DNS and the other that updates the addresses on each node. However, it is less intrusive than option number 2, although with this approach its more complex to update the permission smart contracts.

The second option can be combined with the first one to solve the problem of node permissioning, since any change in the IP address of the nodes would require an additional call to the smart contract.

Advantages

- **Less Maintenance:** Allow connection of nodes with dynamic public IP.
- **Node permission:** Minimize the number of calls to permissioning contracts in the node registry.

Architecture

Method 1. Dynamic DNS (DDNS) Cloud Resolver

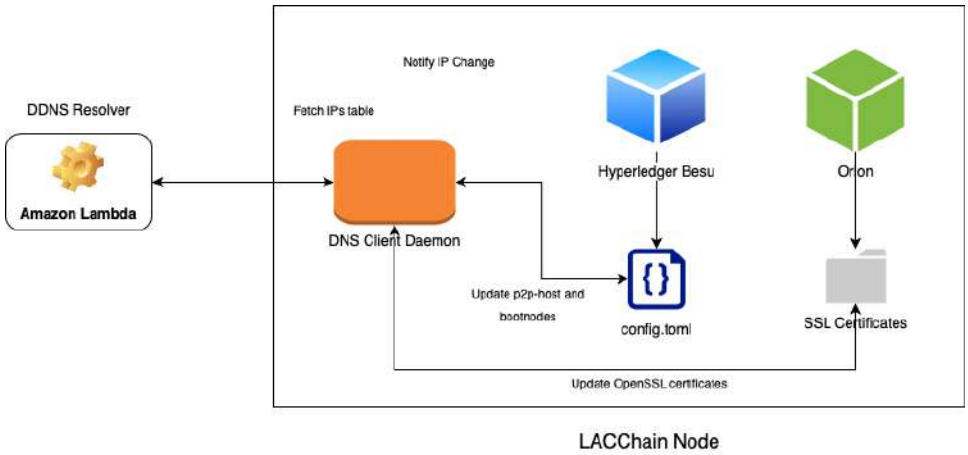

Figure 3. DDNS for peering LACChain nodes.

Method 2. Update enode definition into NodeRuleList (Onchain Permission Contract)

```
// struct size = 82 bytes
struct enode {
    bytes32 enodeHigh;
    bytes32 enodeLow;
    bytes16 ip;
    uint16 port;
}
```

Technical milestones:

- DDNS Function
  - Program a function with Amazon Lambda to update and store DNS node records, as well as regenerate the Orion SSL certificates.
  - **Deliverable**: Amazon Lambda Function.
- DNS Synchronizer
  - Create a daemon which detects any change in the public IP address of the current node and at the same time notifies changes in the Amazon Lambda service.
  - Allow the daemon to periodically download the IP address table and update the Besu *bootnodes* list in config.toml file.
  - **Deliverable**: Daemon to synchronize DNS Tables primary for boot nodes.
- Modify Permissioning Contracts
  - Change the definition of enode in Pegasys permissioning contracts, as well as the functions to add, modify and remove nodes from the permission mapping.
  - **Deliverable**: Deployed Permissioning Smart Contracts

4.3 Cloud Key Management (Orion)

Background

Hyperledger Besu can implement a third-party component known as Orion, this component allows transactions to be executed privately with data only visible to a subset of the network. Internally Orion makes use of a key-pair to sign the transactions, as well as a set of SSL certificates for communication.

The use of private keys stored on any machine locally makes the signature vulnerable, giving way to any case of identity theft.

Currently there is a set of cloud services that allow sensitive data such as keys and certificates to be stored securely, this data is known as secrets. These secrets stored in cloud environments make their use inside applications safer, preventing their loss or change, and even allowing their use in a centralized way.

Proposal

Considering the use of key pairs in the Orion private transaction manager, it is important to use a key storage service to improve security. One of the most important advantages of these services is the data persistence, since in the event of a node failure, it could be restored with the same keys, thus preserving the same identity.

The integration of Orion with any KMS will consist of modifying the source code of the transaction manager internally to use these services through its API instead of simple text files. The following figure contains a general diagram of the architecture for communication between Orion and a Cloud-Based KMS.

Services such as Amazon or Google Key Management Service (KMS) provides a profitable and efficient option for storing secrets, the most important advantages of these services are listed below:

Advantages

- **High Availability:** These services are available in several global locations and across multi-regions, allowing low latency and high availability.
- **Automatic Destruction:** Secrets can be programmed for key material destruction, to prevent accidental or malicious data loss.
- **API**: A REST API that can use a key to encrypt, decrypt, or sign data such as secrets for storage.
- **Key Rotation:** Allows to rotate a key at will and set a rotation schedule for symmetric keys to automatically generate a new key version at a fixed time interval.
- **Compliance:** The security and quality controls are certified under multiple compliance schemes to simplify every compliance obligation. Some vendors provide the option to store the keys in a Hardware-Based Machines known as HSMs instances, validated under FIPS 140-2.
- **Integration with Kubernetes:** Encrypt Kubernetes secrets at the application-layer.
- **Fine Granularity:** Ability to define permissions to use keys, user-level permissions on individual keys for granting access to both individual users and service accounts.

Architecture

In figure 4 we diagram the general architecture for Orion key management, using the Amazon key management service (KMS).

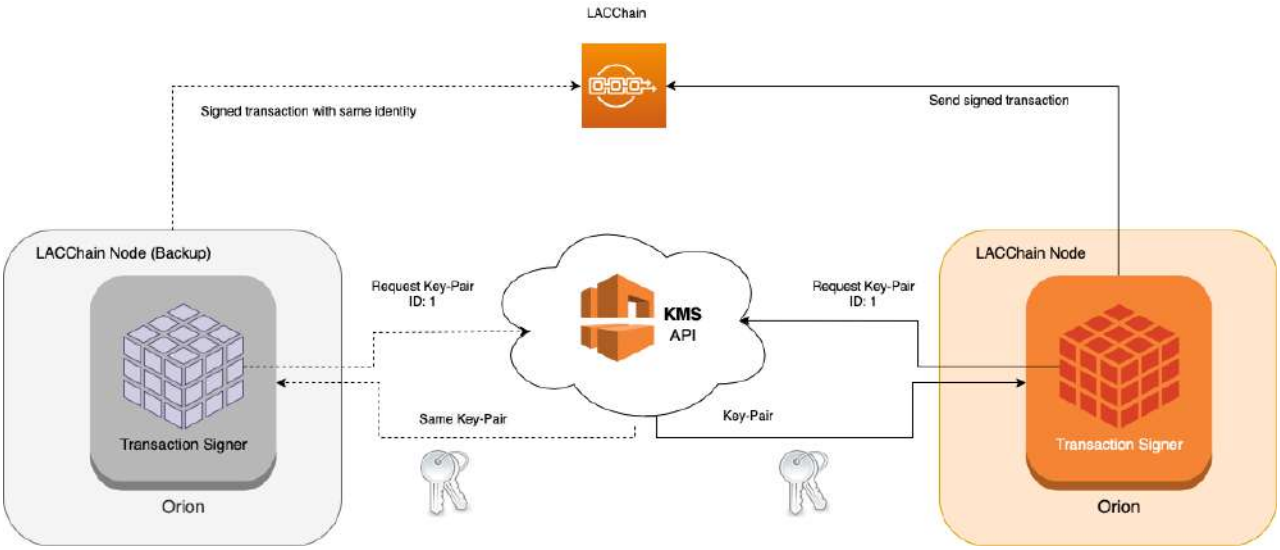

Figure 4. Orion Key Generation/Consumption using Amazon KSM.

Technical milestones:

- Configure KMS Service
  - Configure KMS API service from Amazon to generate and store Orion Keypairs.
  - **Deliverable**: KMS Service API.
- Modify Orion to use KMS
  - Fork Orion Private Transaction Manager from its repository to be able to generate and consume keypairs from Amazon KMS instead of local certificates.
  - Code a function to generate a secure token to identify nodes, using instance hash to consume KMS API.
  - Implement API calls to generate and send keypairs.
  - Implement a function to request keypairs.
  - Modify the signer function to use requested keypairs downloaded from KMS.
  - **Deliverable**: Modified Orion Service to use KMS.

4.4 Blockchain as a Service

Background

Consumers and businesses are increasingly willing to adopt blockchain technology. However, the technical complexities and operational overhead involved in creating, configuring, and operating a blockchain and maintaining its infrastructure often act as a barrier.

BaaS (Blockchain-as-a-Service) is a unique product where consumers can use cloud-based services to build, use, and host their blockchain solutions, features, as well as smart contracts. In summary, these offer a fully functional blockchain over a cloud service.

Usually, BaaS offers an external service provider to set up all the necessary blockchain technology and infrastructure for a fee. Once created, the provider continues to handle the complex back-end operations for the client.

Proposal

Renting blockchain infrastructure in the shape of BaaS allows technology managers to instantly acquire much of the skills and services required to operate blockchain infrastructure of LACChain.

Advantages

- **Documentation tracking**. Blockchain provides an immutable, distributed documentation tracking system. In other words, where documentation is key to blockchain transacting can ensure that all participants have equivalent access to the same information. Plus, blockchain is immutable, in other words, no participant can change records – everyone is assured that documents are immutable, correct and secure.
- **Data storage**. Many applications that require secure data storage that is not centralized can benefit from storing data on the blockchain. With data stored in a decentralized blockchain the risk of data loss is minimized, although it is important to consider blockchains are particularly scalable. Highly regulated industries benefit from the secure, immutable characteristics of storing data on the blockchain.
- **Contract execution.** Alongside smart contracts blockchain can provide a platform for contract execution that enables high levels of transparency. The distributed nature of blockchain implies all parties are equally informed as there is a “single source of truth”. As a result, contracts are processed more efficiently, allowing conclusion and settlement

to occur without delay. The flexibility of blockchain and its associated technologies lends itself to many applications that are yet to be discovered. BaaS provides an opportunity for enterprises to find these applications without making large technology commitments.

Architecture

Figure 5 shows the architecture of a blockchain solution provided as a service, this image depicts how a company may integrate into the LACChain network without the need of its own infrastructure, but with all the rights to own it. The architecture is similar to a PaaS (Platform as a Service), where the end point is the direct use of services without having to interact at low level, however, this level of access requires a more rigorous registration including a KYC (Know Your Customer) process.

Preconfigured applications and a public smart contract marketplace would allow customers access to specific functions of certain sectors within the network.

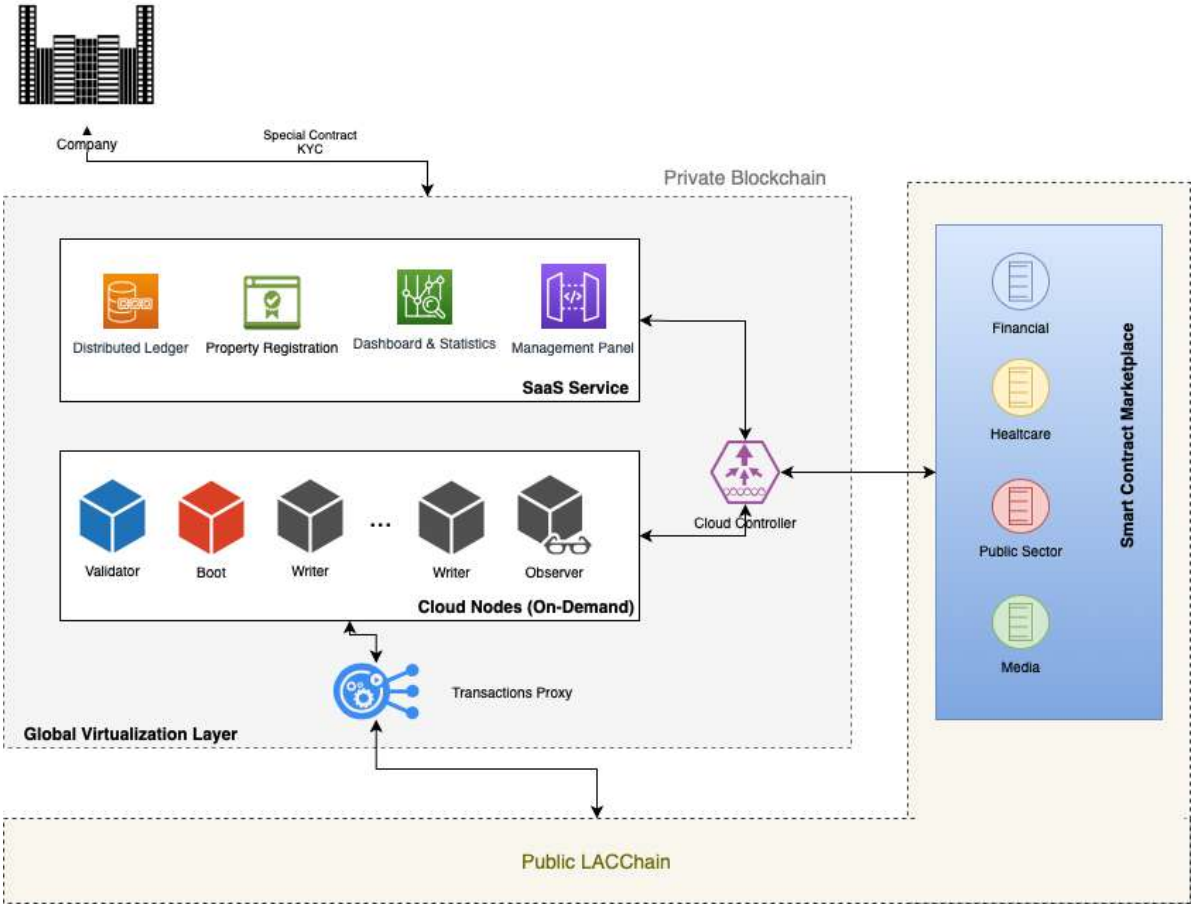

Figure 5. LACChain architecture provided as a BaaS

Technical milestones:

- Deploy a private cloud
  - Install and configure OpenCloud software to deploy a private cloud.
  - **Deliverable**: Private cloud using OpenCloud.

- Deploy LACChain test network
  - Install a minimal LACChain topology nodes inside the private cloud (3 validators, 2 boots, 2 writers)
  - **Deliverable**: LACChain Private Blockchain.
- Transaction Proxy
  - Develop a transaction proxy server to allow private LACChain to communicate to public LACChain and broadcast transactions
  - **Deliverable**: Transaction Proxy Server.
- Contracts Marketplace
  - Develop and implement a platform (DApp) to publish Smart Contracts, from which programmed smart contracts can be automatically viewed, searched and deployed in the LACChain network.
  - **Deliverable**: Smart Contract Marketplace DApp.
- Cloud Controller
  - Code a Cloud Master Controller to manage resources such as: Nodes, Applications, Contracts, Accounts and Services in a private LACChain test network. This controller should be able to deploy smart contracts from Smart Contracts Marketplace.
  - **Deliverable**: Cloud Controller Server
- KYC Register
  - Design and develop a KYC (Know Your Customer) API and registration page to allow contract and deploy a private LACChain.
  - **Deliverable**: KYC Registration Portal (API).