REAP CHAIN Co., Ltd.

REAPCHAIN Yellow Paper

ReapMiddleChain

Ver 1.0.3

REAP CHAIN

목차 I

목차 II

그림 목차

1. 도입

1.1 문서의 목적

Reapchain Middleware의 목적은 사용자(분산원장 연결이 필요한 모든 소프트웨어)에 손 쉬운 분산원장 인터페이스를 제공하는 것이다. 사용자는 특정 분산원장의 사용 여부에 대해서 알지 못하며, 단지 데이터를 전달하는 인터페이스의 존재 여부와 어떤 데이터(데이터의 내용과 형식)를 전달해야 할 지만 알 수 있다.

본 문서의 목적은 Reapchain Middleware를 사용하는 사용자를 위한 인터페이스 및 내부적인 구조를 설명하는 것이며, 추가적인 확장과 서비스 구현을 위한 기본적인 정보를 제공하는 것이다. 필요하면 내부적인 구조를 설명을 위해서 개념적인 수준의 그림을 제공할 수 있지만, 실제 구현과 완전히 동일하지 않을 수 있으며 기능 추가 및 성능 개선을 위해서 공지하고 않고 변경될 수 있다.

1.2 문서의 관행

본 문서는 기술적인 내용을 다루게 되며, 기능적인 부분에 대해서 주요 내용을 포함할 수 있다. 사용하는 용어와 다루는 주제는 대부분 소프트웨어 개발에서 다루는 것들이며, 용어의 일반적인 의미와 다른 내용을 포함할 수 있다. 프로그래밍 분야의 표준적인 내용에 대해서는 별도의 설명을 추가하지 않지만, Reapchain Middleware자체에서 정의하는 용어에 대해서 부가적인 설명을 한다.

1.3 문서의 대상자

본 문서의 대상은 외주 개발자 및 Reapchain 개발 관련자와 Reapchain의 Middleware구현에 대해서 알고 싶어하는 내 외부 인력으로 한정한다. 본 문서는 일반 사용자(이용자)를 대상으로 하지 않으며, 서비스 개발에 필요한 내부 정보를 제공하는 것을 목적으로 한다. 하지만, 필요한 경우 본 문서를 읽는 대상을 추가할 수 있다.

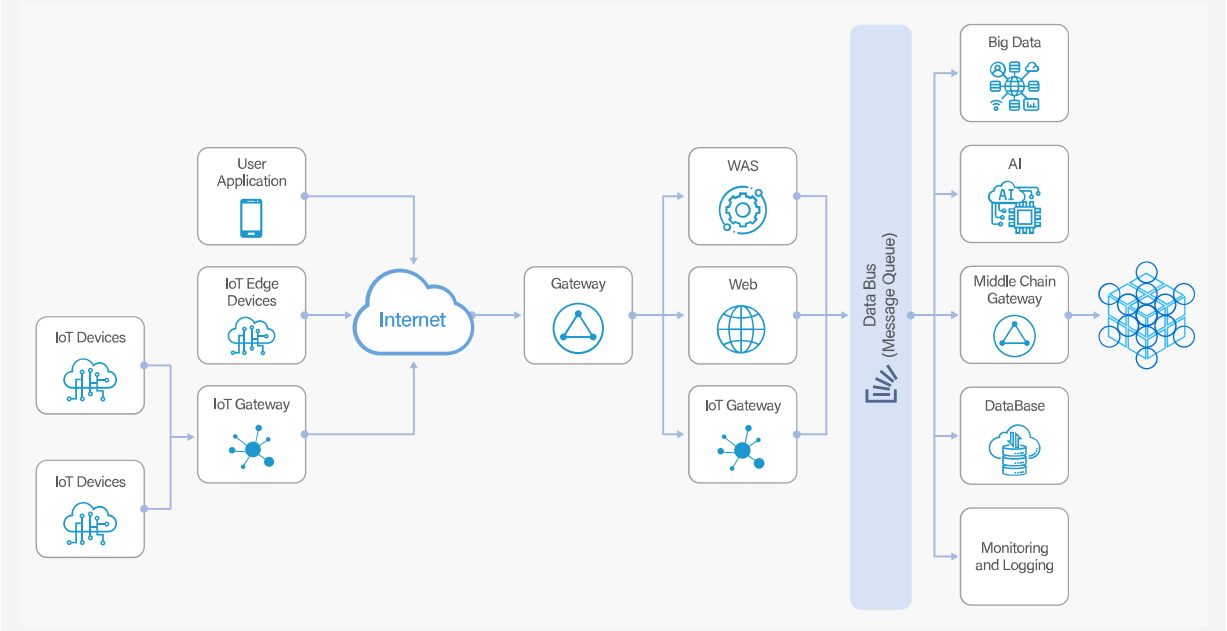
1.4 과제 범위

본 과제의 범위는 사용자 앱 및 장치(Device)에서 발생하는 요청(Request)을 분산원장으로 전달하는 중간 과정과 관리 측면에서 필요한 사용자 등록/삭제/변경, 시스템의 모니터링 기능에 한정한다. 필요하다면 테스트 목적의 사용자 앱(iOS, Android, PC, Tablet PC 포함)을 구축해서 제공할 수 있지만, 추가적인 별도 결과물로 관리하도록 한다.

2. 시스템 구조

2.1 ReapMiddleChain의 목적

Reapchain Middleware의 목적은 응용서비스 별 다른 형태의 Transaction(혹은 데이터)을 분산원장 네트워크와 연계하고, 사용자 서비스 품질 관점에서 유의미한 결과를 구현하기 위함이다. 기본적으로 Reapchain에서 고려하는 서비스는 IoT와 연계된 비즈니스 생태계이며, 이를 위해서는 IoT 기기에서 발생하는 대량 정보의 저장에 관련된 Transaction과 비즈니스 관점에서 필요한 사용자 결재 및 보상을 위한 Transaction으로 나누어 볼 수 있다. 하지만, 서비스 구축과 응용에 대해서 내용과 범위를 한정하지는 않는다. 각각의 서비스에 대해서는 별도의 인터페이스가 제공되어야 하며, 정보의 저장 방식과 활용도 다를 수 있다.

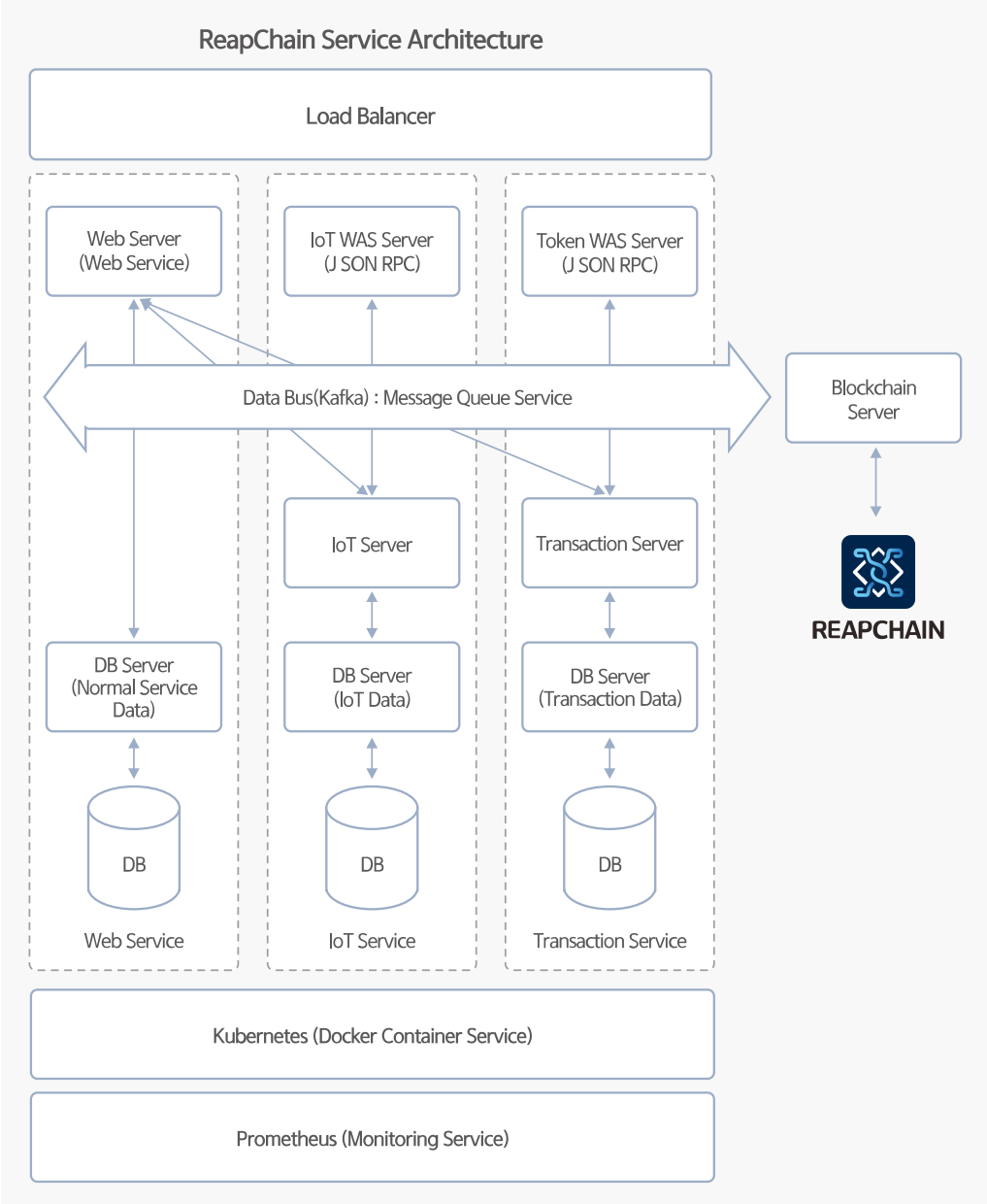


< 그림 1. IoT 데이터의 저장과 활용을 위한 구성 예 >

예를 들어, IoT 서비스는 대용량으로 시계열로 생성될 가능성이 높으며, 이를 분산원장 네트워크에 전체를 저장하는 것은 비용효율면에서 좋은 선택이 아니다. 사용자 Transaction의 경우에는 계정의 최종 상태 값과 그 과정에서 발생하는 모든 거래 데이터를 저장할 필요가 있지만, 대용량으로 같은 계정에서 연속적으로 발생하는 경우는 드물다(이상 Transaction의 경우 이를 검출할 수 있어야 한다). 따라서, 각각을 분리된 서비스와 인터페이스로 구축해 저장하는 것이 논리적으로 타당하며, IoT 데이터의 경우에는 추가적인 가공을 통해서 향후 의미 있는 정보를 제공해줄 수 있을 가능성이 높기에 별도의 DB로 관리할 필요가 있다. 향후 AI와 BigData를 위한 서비스로 확장하기 위해서 인터페이스 분리를 위한 데이터 버스(Data Bus)개념의 메시지 큐를 포함할 수 있다.

2.2 ReapMiddleChain의 구조

Reapchain Middleware는 Reapchain이라는 분산원장과 사용자 앱(IoT 및 지갑 등의 별도로 배포하는 형태의 응용프로그램을 포함)의 중간에서 효율적인 처리와 관리 및 모니터링 기능을 제공한다. 이를 위해서는 전체적으로 관리 기능을 담당하는 부분과 IoT(Internet of Things) 장치의 연결을 담당하는 부분, 사용자의 Transaction을 처리하는 부분으로 크게 나누어질 수 있다. 각각은 아래의 그림과 같은 형태로 존재할 수 있다.



< 그림 2. Reapchain Middleware 구성 >

그림 1. https://cloudmt.co.kr/?p=3339,[Azure IoT Reference Architecture]

소프트웨어 컴포넌트를 이루는 각각의 모듈(혹은 서버나 실행 단위)에 대해서는 아래와 같이 간략하게 설명할 수 있다.

▣ Load Balancer

사용자 어플리케이션에서 들어오는 요청을 각각의 서비스에 나누어 주는 역할을 한다. 이때 각각의 요청은 IoT나 Transaction, 혹은 관리 및 모니터링에 대한 요청으로 나눌 수 있으며, 대응하는 서비스나 서버를 한정해서 부하(Load)를 분산시켜 주는 역할을 한다. 만약 Load Balancer가 충분한 트래픽을 보장해 주지 못할 때는 추가적인 서비스 노드를 동적으로 운영할 수 있다.

▣ Web 서버

관리 및 모니터링에 대한 요청을 처리하는 주체이다. 사용자(단말기 및 실제 사용자) 관리 및 연결된 서버들의 상태와 내부적인 정보를 서비스 한다. 관리 업무내에는 사용자 및 단말기에 대한 인증을 포함할 수 있다. 또한, 사용자에 필요한 서비스를 웹으로 제공하는 역할도 수행할 수 있으며, 각각은 분리된 서버(혹은 분리된 Container형태)로 구현된다.

▣ IoT WAS(Web Application Server)

IoT와 관련된 요청을 처리하기 위한 서버이며, 주로 IoT 기기의 등록과 정보를 저장 및 제공을 목적으로 사용된다. IoT 트래픽이 증가하면 서비스를 담당하는 서버도 동적으로 확장할 수 있는 구조를 기본적으로 제공한다.

▣ Token WAS

Token관련 서비스 요청을 처리하기 위한 서버이며, 사용자 앱(Wallet App)과 연동해 사용자의 계정 상태를 변경하거나 알기 위해서 사용된다. 사용자 앱은 서비스에서 발생하는 보상 포인트나 토큰을 확인하기 위해서 해당 서버에 접속하며, 서비스 제공자는 해당 서비스를 이용료를 징수하기 위해서 해당 서버에 요청을 보낼 수 있다.

▣ IoT 서버

직접 IoT 관련 요청이나 데이터에 접근하기 위해서 사용되는 서버이다. Message Queue를 통해서 들어오는 Traffic을 받아서 처리할 수 있으며, IoT 전용 DB에 접근해 데이터를 저장 및 검색 등의 역할을 수행한다. 받아들인 IoT 기기의 정보 중에 분산원장에 저장해야 할 정보가 있는 경우에는 분산원장 서버에 요청을 보낼 수 있다. 이때 분산원장에 저장되는 데이터를 최소화하기 위해서 원본 데이터의 모음을 가지는 블록(Block)의 해쉬(Hash)값이나 기타 정보를 전달할 수 있다.

▣ **Transaction 서버**

Transaction을 실행하기 위해서 필요한 서버이며, Message Queue를 통해서 들어오는 Transaction을 처리하고, 분산원장 서버에 접근해 Transaction 동기화를 수행해야 한다. 사용자 Transaction에 대한 사전(pre-confirm)을 담당한다. 최종적으로 Reapchain에 해당 Transaction의 기입 여부는 추가적인 Confirm을 통해서 가능하다.

▣ **DLT(Distributed Ledger Technology) 서버**

분산원장 네트워크 연계하기 위해서 사용되는 노드이다. 일종의 분산원장 네트워크를 구성하는 노드로서 동작할 수 있지만, 여기서는 분산원장에 Transaction 요청을 전달하고 상태(State) 정보를 얻기 위해서 사용된다. 분산원장 자체는 서비스 내부의 트랜잭션만을 처리하며, Reapchain 메인넷과 분리된 “Private Permissioned” 분산원장으로 구성한다.

▣ **Web Service DB 서버**

웹 서비스에 관련된 데이터를 저장하고 검색하기 위해서 사용되는 DB 서버이다. 인증된 IoT기기나 사용자를 관리하기 위해서도 사용될 수 있다. 서비스 접근을 위한 사용자 ID와 Password등의 정보를 포함할 수 있으며, 허가된 사용자가 등록한 인증된 IoT 기기의 등록정보도 확인할 수 있다.

▣ **IoT DB Service DB 서버**

IoT 기기에서 발생하는 데이터를 저장하기 위해서 사용하는 DB이다. 주로 NoSQL 계열의 DB를 사용해 시계열(Time-Series)로 발생하는 IoT 데이터를 저장한다. 저장되는 데이터 값의 최적화 및 암호화를 DB의 기본 기능으로 제공할 수 있다.

▣ **Transaction DB Service DB 서버**

분산원장에 기록되어야 하는 Transaction을 일시적으로 저장하고 검색하기 위해서 사용하는 DB이다. 분산원장은 상대적으로 Transaction 속도가 느리기 때문에 사용자 및 IoT 기기의 요청에 대한 빠른 반응을 제공하기 위해서 필요하며, 분산원장 네트워크의 상태 정보와 동기화를 제공한다.

최종적인 Reapchain Middleware의 결과물이 Docker 이미지 형태로 만들어져, Docker가 설치된 환경에서는 아무런 부가적인 노력(Network 설정은 제외)없이 동작할 수 있어야 하며, Kubernetes를 실행에 네트워크 Node들의 생성 및 실행을 관리할 수 있어야 한다. Prometheus는 서비스를 구성하는 각종 노드들에 대한 모니터링 기능을 제공하기 위해서 사용하며, 그래픽 UI(Graphana)를 제공하기 위한 도구와 이상 징후를 발견했을 때 이를 알리기(Notification) 위한 도구도 함께 제공한다.

2.3 Reapchain Middleware의 특징

Reapchain Middleware의 결과물에서 제공하고자 하는 제품의 특징으로는 분산원장과 실제 사용자 앱(App) 간의 연계성을 강화하고, 분산원장 서비스를 사용하는데 필요한 지식 수준을 완화하는 것이다. 즉, 서비스 개발자의 입장에서 사용자에게 제공하는 서비스 개발에 초점을 맞추고 부차적인 것들은 제공되는 솔루션 및 컴포넌트를 통해서 해결 책을 제공하는 것이다. 분산원장 서비스는 불변 데이터를 분산 저장하는 것을 목적으로 하며, 네트워크 참여자에게 신뢰를 구축하기 위해서 사용한다. 따라서, Reapchain Middleware는 아래와 같은 것들을 특징(Feature)이라고 할 수 있다.

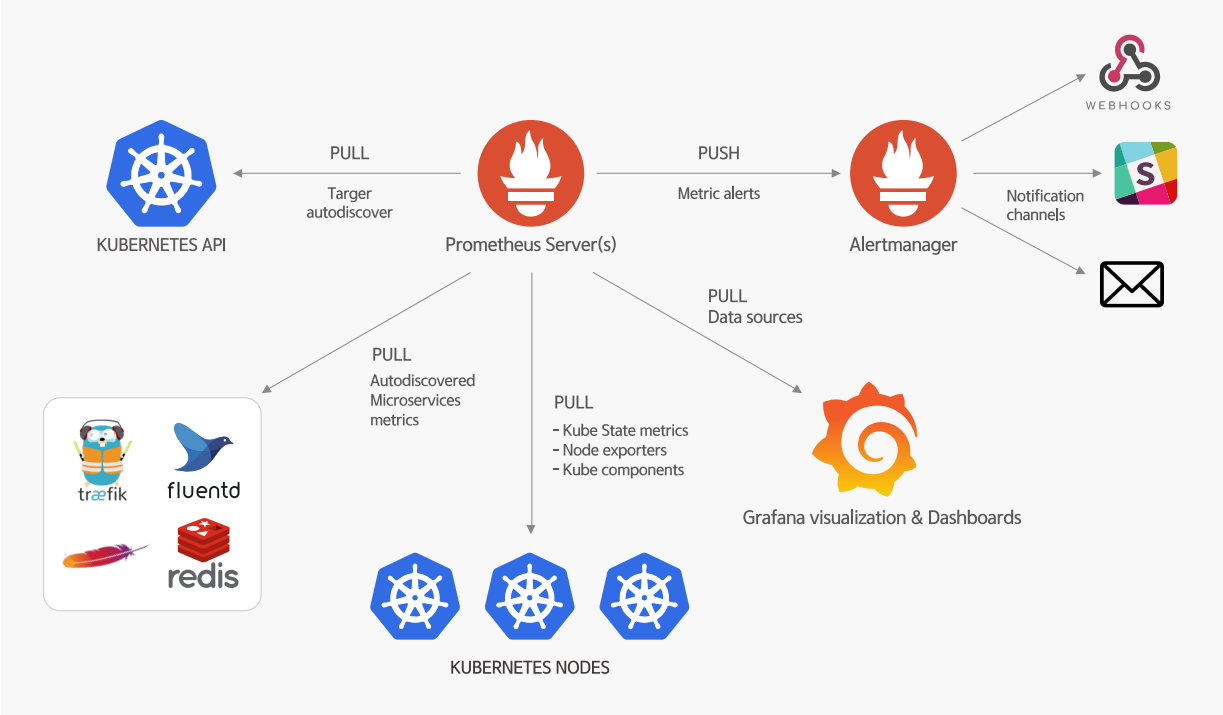
- ▣ IoT 장치에서 생성되는 데이터를 저장하고 조작되지 않았다는 것을 보장할 수 있다.
- ▣ 사용자 Transaction을 분산원장(Middle chain)에 저장해 장부(Ledger)의 조작이 발생하지 않았으며, 사용자가 가지고 있는 계정의 잔고를 실시간으로 확인할 수 있다.
- ▣ 관리 서비스를 활용해 등록된 사용자에 대해서 내부 시스템의 상태(계정의 상태 포함)를 조회할 수 있으며, 개인 정보 및 IoT 장치의 등록/수정/삭제 등을 요청할 수 있다.
- ▣ Reapchain Middleware 서비스를 활용하기 위해서는 인증된 IoT 장비와 사용자만 접근할 수 있으며, 각각의 서비스는 인증 절차를 준수해야 한다. 이를 위해서 개별 서비스는 각자 인증 절차를 구현할 수 있다. IoT 장비와 사용자 앱(혹은 사용자)는 인증 방법이 다를 수 있으며 권한 설정에도 인증 절차 및 방법은 유효하다.
- ▣ 제공되는 서비스 한계(Boundary)내에서 사용자 및 IoT 장치의 Transaction 처리를 실시간으로 보장한다. 여기서 말하는 실시간은 사용자의 관점에서 느낄 수 있는 범위내로 한정하며, 대략 3초 이내 수준으로 제공하는 것을 목표로 한다.[1] IoT 장치의 경우는 사용자가 바라보는 관점에서 새로운 데이터의 반영 여부가 실시간이라고 보장한다.

2.4 사용자 계층 및 특성

본 과제의 결과물의 사용자는 크게 IoT 장치의 인터페이스를 담당하는 프로그램(혹은 Framework)나 사용자 앱(지갑 및 기타 사용자가 직접 사용하는 어플리케이션)을 포함한다. 그 외의 사용자는 웹을 통해서 직접적으로 서비스에 접속하는 경우이며, 이때는 별도의 앱을 필요로 하는 것이 아닌 웹 인터페이스를 제공하는 것을 기본으로 한다.

2.5 운영 환경

본 과제의 최종 결과물은 운영체제나 하드웨어 환경에 무관하게 동작하는 것을 목표로 한다. 하지만, 기본적으로 Docker를 사용한 컨테이너(Container) 형태로 제공될 것이며, Linux와 같은 서버 운영체제[2] 와 가상환경(Virtual Machine기반)에서 동작하는 것을 기본으로 가정한다. 따라서, 개발 결과물을 제공하기 위해서는 컨테이너화 된 이미지에 배포물을 전부 취합해서 제공한다.



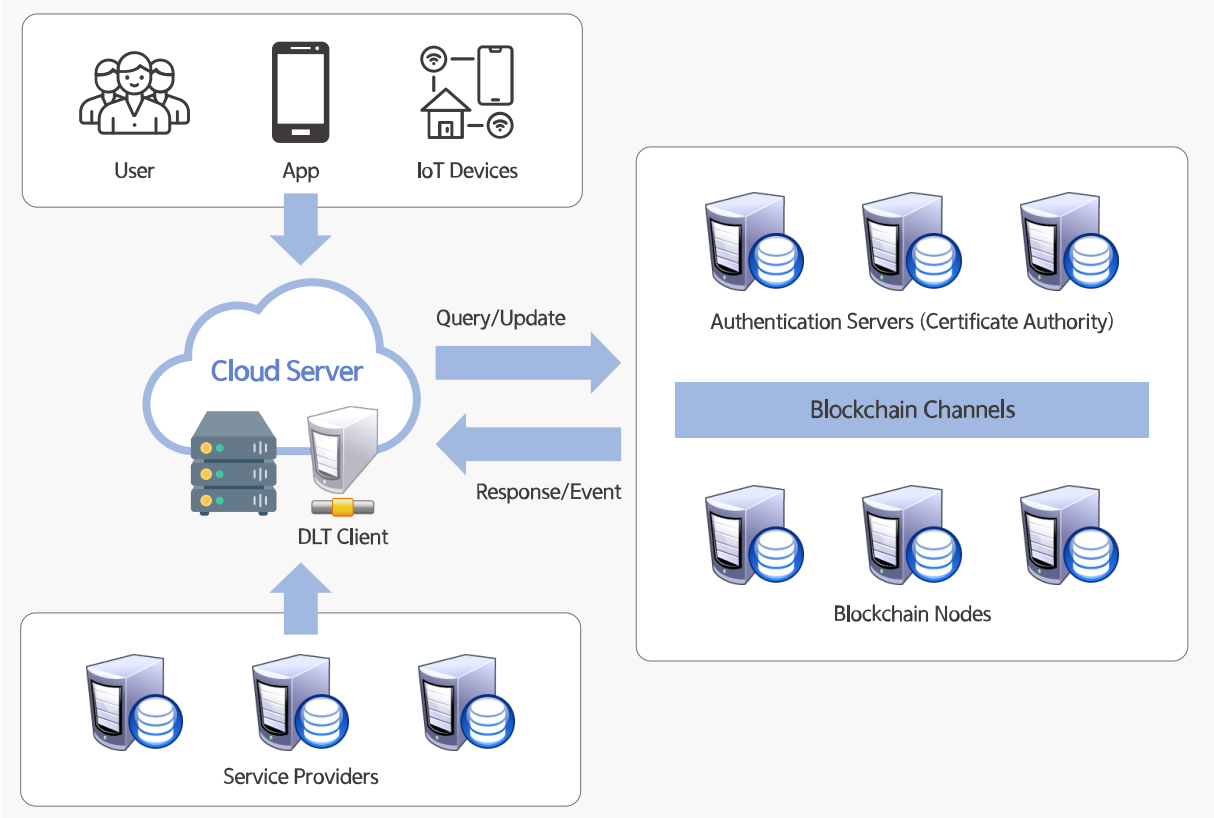
< 그림 3. Prometheus를 활용한 Kubernetes 노드의 관리 예 >

컨테이너화의 이점은 개발환경과 동작환경을 분리하지 않고 유지할 수 있다는 점이며, 동적인 확장(수평 확장)의 경우도 쉽게 가능하다는 것이다. 만약 추가적인 서버의 증설이 필요하다면, 해당하는 서비스를 제공하는 컨테이너의 인스턴스(Instance)를 제공하는 것으로 대응할 수 있다.

1. 일반적으로 웹에서 사용자의 관심이 한 사이트에 머물 수 있는 한계가 3초로 알려져 있다.
2. 기본적으로 Ubuntu 최신버전을 사용하도록 한다. CentOS 최신버전을 사용해도 무방하다.
3. http://onem2m.org 참조
4. https://omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/ 참조

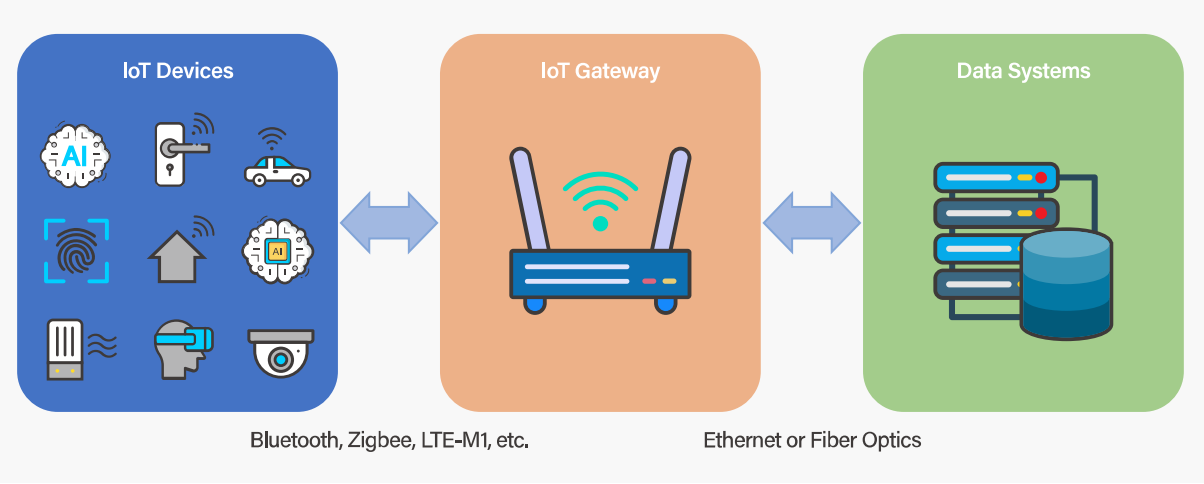
추가적으로 Docker 이미지로 만들어진 결과물은 Kubernetes를 활용해 서버 부하가 많을 경우 여러 인스턴스(Instance)를 만들어 서비스 할 수 있다. 따라서, 기본적으로 Docker와 Kubernetes를 기본 운영환경에 포함하도록 했다. 로드 밸런서(Load Balancer)는 각각의 가동중인 서버 노드(Node)에 대해 적절한 Traffic 분산될 수 있도록 만들어 준다. Docker 이미지는 배포를 위해서 사용하지만, Kubernetes를 배포된 이미지를 네트워크 상에 동적 노드로 관리하고 운영할 수 있는 환경으로 사용한다. Prometheus는 반드시 구성할 항목은 아니지만 네트워크 상황을 감시(Monitoring)하고 문제가 발생시 신속하게 대응하기 위해서 구비되어 있다. 시각적인 관리를 위해서 부가적인 도구(Tool)들을 결합해서 설치 및 운영 편의성도 제공한다.

IoT 기기의 직접적인 분산원장에 대한 연결은 없다고 가정하며, 일단 IoT 기기들은 중간 게이트웨이(Gateway)역할을 하는 노드에서 Reapchain Middleware를 통해서만 서비스 내부의 허가형(Permissioned) 분산원장과 연결된다고 가정한다. 따라서, 인증되고 연결된 IoT 기기들은 중간에 Reapchain Middleware에 요청을 보내는 노드를 통해서만 요청을 전달할 수 있다. 추가적으로 Reapchain Middleware는 IoT 산업에서 활용하는 각종 Gateway 및 연결 Protocol을 제공하기 위해서 OneM2M[3] 이나 LWM2M[4] 등을 포함할 수 있다.



< 그림 4. 허가형 분산원장 클라이언트 사용 예 >

연계된 분산원장은 기본적으로 허가형(Permissioned)을 가정하며, Docker 이미지를 이용해 구축할 수 있다. 허가형 분산원장을 사용자나 IoT 기기가 직접적으로 연계하지 못하기에, 중간에 대리자(Agent) 역할을 수행하는 분산원장 클라이언트(Client) 노드를 사용하며, 이 노드의 수는 트래픽 양에 반응하기 위해 동적으로 증감할 수 있다. DLT을 기본으로 하지만 다양한 분산원장을 연계해서 사용할 수 있으므로, 정의된 인터페이스를 사용해 분산원장을 연계하는 방법을 제공한다.



< 그림 5. IoT와 연계된 서비스 구축 예 >

사용자 앱은 웹을 통해서(HTTP를 이용해서) Reapchain Middleware에 직접적인 요청을 보낼 수 있으며, 이를 위해서 Reapchain Middleware는 HTTP 요청을 처리할 수 있어야 한다. 따라서, 기본적으로 HTTP 요청을 처리하기 위한 Web 서버와 각종 동적인 요청에 대한 응답을 처리할 수 있는 WAS(Web Application Server)를 구비하고 있어야 한다. 추가적으로 다양한 IoT 기기의 연결 처리를 위해서 기존에 존재하는 Protocol의 연계(예를 들어, MQTT등)도 제공한다.

2.6 디자인 및 구현 제약사항

Web, IoT, Transaction 서비스 각각은 분리된 서비스로 구현되어야 하며, 각각의 서비스간 연계는 제공되는 API를 통해서만 가능하다. HTTP를 이용한 Restful API를 기본으로 제공하고 있으며, 추가적으로 아래와 같은 제약 사항이 더 있을 수 있다.

- ▣ 각각의 서비스는 내부에 Microservice 형태의 구현을 기본으로 한다. 따라서, 내부에 다시 분리되는 Docker 이미지 형태로 배포 및 유지보수 될 수 있다.
- ▣ 대량의 데이터를 전송하기 위해서 Message Queue를 사용하며, Publisher/Subscriber 형태의 구현을 기본적으로 채택한다. 따라서, 메시지 큐의 CFT(Crash Fault Tolerance)를 기본으로 제공한다.
- ▣ 각각의 서비스별 DB는 별도의 노드로 구축하며, 추가적인 증설과 오류 극복을 위해서 Cluster형태로 구축될 수 있다. 다만 PoC(Proof of Concept) 단계에서는 별도의 단일 노드로 구성하는 것으로 한정하도록 한다. 추가적인 서비스나 부하 분산이 필요하다면, 이는 다른 시스템의 구현에 영향을 주지 않고 독립적으로 처리할 수 있도록 한다.
- ▣ 인증된 사용자와 IoT 기기(혹은 IoT Gateway)만 Reapchain Middleware에 요청을 보낼 수 있다. 이를 위해서 사용자 인증 및 IoT 기기 인증에 대한 서비스를 API및 Library형태로 제공한다.
- ▣ 서비스 내부의 분산원장 연계를 위해서 별도의 Gateway역할을 하는 노드(분산원장 클라이언트 노드)를 정의해야 하며, 서비스 내부에서 사용하는 분산원장은 특정 분산원장으로 한정하지 않는다. 다만 PoC에서는 허가형 DLT를 기본으로 가정한다.
- ▣ 구현언어와 사용하는 프레임워크의 제약은 없다. PoC에서는 간편한 구현과 검증을 목적으로 특정 프레임워크를 사용할 수 있으나, 서비스 제공자는 자신이 사용하는 개발 환경을 Reapchain Middleware와 결합시키는데 어렵지 않도록 각각을 분리해서 제공하도록 할 것이다. IoT 서비스 및 Transaction 서비스는 구현의 편의와 성능을 고려해서 적절한 구현언어와 프레임워크를 선택할 수 있으며, 별개의 서비스로 나누어 구축될 수 있도록 한다.
- ▣ IoT 서비스의 경우 대용량의 DB를 구축하고, 향후 AI(Artificial Intelligence)와 같은 연계를 위해서 Big Data연계를 염두에 두고 구축되었다. 예를 들어, Hadoop과 같은 것을 서비스 내부에 사용하고, 외부로 드러나는 인터페이스를 위한 추가적인 서비스용 서버를 운영할 수 있다. 서비스 내부의 추가적인 구현이 쉽도록 만들기 위해 메시지 큐를 시스템간 연계를 위해서 기본으로 사용하고 있으며, 이때 데이터 제공자는 Publisher가 되고, 추가 서비스는 Subscriber로 등록되도록 하고 있다.
- ▣ 사용자 인터페이스는 웹과 IoT 기기를 위한 Protocol에 한정한다. 사용자 App을 이용해서 Transaction을 발생시킬 수 있으나, 별도의 iOS나 Android 앱은 별도로 제공하지 않고 인터페이스만 제공하도록 한다. 따라서, Web 기반 인터페이스를 기본으로 3rd Party에서 앱을 개발해서 서비스와 연계할 수 있으며, 파트너 사인 경우 협의를 통해 확장 API를 이용해서 서비스를 이용할 수 있도록 한다.

▣ IoT 기기의 경우 별도의 앱을 제공하지 않으나, 인증 절차를 거쳐 Reapchain Middleware를 사용할 수 있도록 한다. IoT 기기의 인증은 사용자 인증과 별도로 이루어지며, 등록은 허가를 얻은 사용자와 관리자로 한정한다. 즉, 인증된 사용자가 인증된 IoT기기를 등록할 수 있도록 한다.

▣ IoT 기기와의 연결을 위해서 Message Queue의 구현을 위해 MQTT와 같은 것을 사용할 수 있으나, 특정 Message Queue 구현에 의존하지 않도록 인터페이스를 정의하고 구현을 분리하도록 한다.[5] OneM2M이나 LWM2M 등에서 제공하는 프레임워크의 경우 내부 혹은 외부에 분리된 형태로 별도로 제공할 수 있다.

▣ IoT 기기 트래픽에 대해서는 10,000 TPS이상을 제공하는 것을 목적으로 하며, 내부 분산원장 Transaction은 2,000 TPS이상을 제공하는 것을 목표로 한다. 이와 같은 차이가 발생하는 이유는 IoT 기기의 모든 트래픽을 분산원장에 기록하는 것이 아니기 때문이며, 특정 기간이나 주기 동안의 값을 모아서 해시 형태로 기록한다. 검증은 기록된 해시값과 실제 IoT에서 발생한 데이터의 해시값 비교를 통해서 이루어질 수 있다.

5. 추가적인 Message Queue 구현으로 Kafka를 사용할 수 있다.

3. 시스템 특성

Reapchain Middleware는 IoT 기기 및 사용자 거래를 위한 중간 계층 소프트웨어를 제공해 사용자 측(혹은 서비스 이용자 측)의 사용 편의를 도모하고, 빠른 응답(Response)을 주는 것을 특징으로 한다. 이를 위해서 아래와 같이 Web, IoT, Transaction이라고 불리우는 세개의 서비스를 기본으로 제공하는 Subsystem으로 구성되며, 각각은 다음과 같은 특징을 가진다.

3.1 Web 서비스 서브시스템 특성

Web 서비스 서브시스템은 Reapchain Middleware의 운영과 사용자 및 IoT 기기의 등록을 위해서 제공된다. Web 서비스는 관리자를 위한 UI(User Interface)를 가지고 있으며, 사용자 및 IoT 기기의 등록, 수정, 검색, 삭제 등의 기능을 제공한다.

- ▣ 관리자 이외의 일반 사용자에 대해서는 사용자에 특화된 UI를 제공하며, 개인 정보(혹은 Profile)의 등록과 수정, 계정 삭제 등의 요청을 처리할 수 있다.
- ▣ 관리자의 경우 사용자 및 IoT 기기의 등록, 수정, 검색, 삭제 등이 가능하며, 각각에 대해서 별도의 서비스로 분리해서 구현하고, 이를 별도의 인터페이스를 제공한다.
- ▣ 일반 사용자의 경우에는 자신이 등록한 IoT기기 이외에 관련된 정보를 열람하지 못한다. 특정 IoT 기기의 인증(IoT Gateway에 대한 인증과 설정도 포함해서)은 관리자 권한(혹은 관리자 권한으로 실행되는 서비스)으로만 가능하며, 일반 사용자는 자신의 IoT 기기에 대한 관리 및 열람 권한만 한정해서 가진다.
- ▣ 사용자 등록과 인증에 대한 인터페이스를 제공하고, SNS(Social Network Service)를 활용한 인증과 등록도 가능해야 한다. 대상 SNS로는 Kakaotalk과 Facebook, Naver를 포함하지만, PoC 단계에서의 구현에서는 이중 하나만 포함할 수 있다. 특정 서비스의 구축에 사용할 경우 요구되는 사양에 따라 사용자 및 기기 인증에 대한 방법은 달라질 수 있다.

< 그림 6. Web Dashboard(Graphana)의 사용 예 >

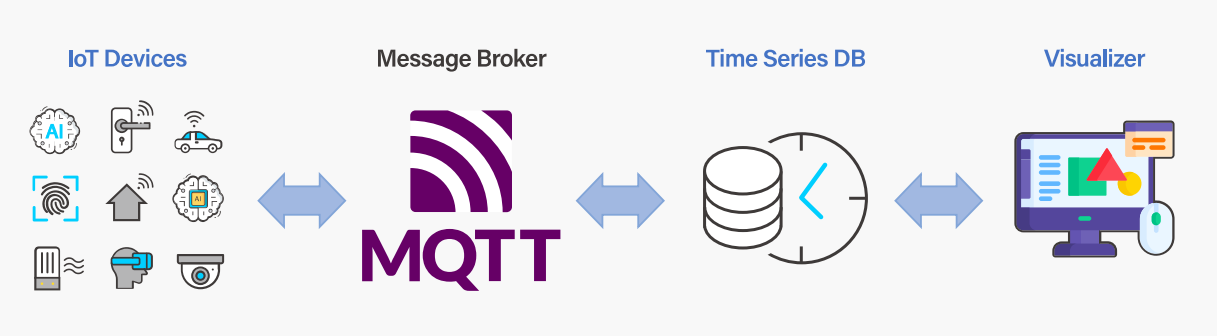
▣ 관리자 계정에서는 현재 서비스 상황(IoT 및 Transaction)에 대해서 모니터링을 위한 UI를 제공해 주며, 각각의 Traffic에 대한 추이를 시간의 흐름에 따라 보여줄 수 있어야 한다. 중요한 이벤트(Event)에 대해서는 로그(Log)를 저장해 보여주도록 한다. 서비스 현황에 대한 정보는 차트(Chart)를 활용해 시각적으로 웹을 통해서 보여줄 수 있다.

▣ 관리자 및 사용자 인증에 대해서는 간편 인증 방식과 ID/Password 방식 둘 다 적용할 수 있으나, PoC에서는 ID/Password 방식을 사용한다. 추후 간편 인증과 관련된 서비스를 활용해 IoT 기기 및 사용자 인증에 활용할 수 있도록 한다.

▣ PoC 단계에서는 보안에 대해서는 특별한 요구사항이 없으나, 향후 HA(High Availability)와 관련된 내용을 대응하기 위해서 이중화를 위한 방안은 제공하도록 한다. 아래에서 따로 기술하지 않을 경우 IoT 및 Transaction 서비스에 대해서도 마찬가지로 이중화 방안을 제공한다. 서비스에 대한 보안은 Legacy 시스템의 보안과 크게 다르지 않으며, 이를 위한 추가적인 솔루션은 별도의 사항으로 분리해서 다루도록 한다.

3.2 IoT 서비스 서브시스템 특성

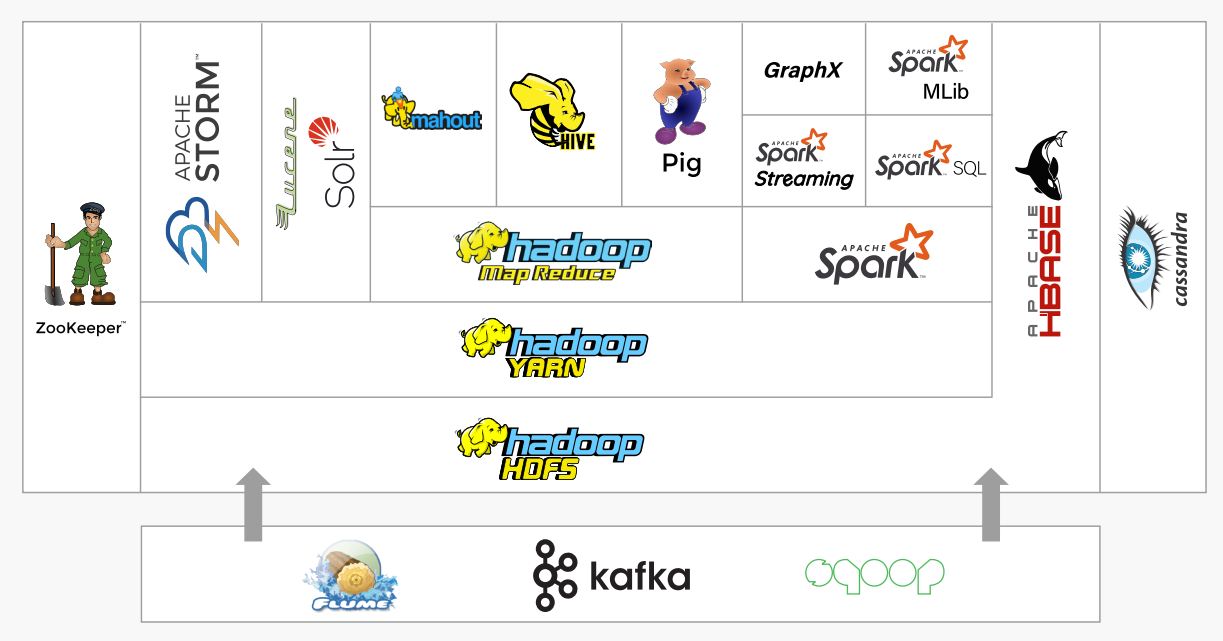
IoT 서비스 서브시스템은 대용량의 데이터 발생을 받아들일 수 있는 구조를 가져야 한다. 향후 IoT와 DLT의 연계를 통해서 조작되지 않은 데이터에 대한 신뢰를 확보하기 위해서, 데이터 묶음(Cluster 혹은 Block)단위의 해시(Hash) 값을 서비스 내부의 분산원장 네트워크에 저장한다.



< 그림 7. IoT 기기의 MQTT를 중심으로 한 데이터 연결 구성 >

▣ IoT 기기의 특성상 시계열 데이터를 대량으로 저장해야 한다. 따라서, 이를 위한 효과적인 저장/검색 구조를 가질 수 있는 DB를 사용해야 한다. 기본적으로 NoSQL 형태의 DB를 사용하는 것을 가정하지만, 데이터의 발생 형태에 따라 Time Series DB도 고려할 수 있다.

▣ 추가적으로 Big Data 처리를 위한 구성으로 Hadoop을 사용할 수 있지만, PoC 범위에서는 포함하지 않는다. 하지만, 시스템의 확장을 위해서 별도의 서비스 추가가 쉬운 구조를 제공하기 위해서 Message Queue를 기준으로 구현을 분리할 수 있도록 구축한다. 추가적으로 데이터 분석에 대한 도구가 필요하다면 포함하도록 할 것이다.



< 그림 8. Hadoop 내부 구조 >

▣ IoT 데이터를 실시간으로 분산원장에 저장하는 것은 비용에 따른 효과가 크지 않을 것으로 보기에, 적절한 주기로 데이터를 묶어 해시 값만 서비스 내부의 분산원장에 저장하고 검증에 사용하도록 한다. 원본 데이터의 저장을 위해서 File 서버를 별도로 구축할 수 있으며, 혹은 파일 단위 저장을 위해서 IPFS와 같은 시스템을 추가로 구성할 수 있다.

▣ 데이터 묶음에 사용되는 기기들은 각각이 고유한 ID를 사용하거나, 혹은 여러 기기의 데이터를 묶어서 한번에 해시 값을 구할 수 있다. 이는 PoC 구현의 편의성을 위해서 선택할 수 있지만, 향후 구축시에는 디바이스 각각에 대한 분리된 저장공간을 제공하는 것을 목표로 한다. 데이터의 저장 주기와 백업 및 삭제 등은 서비스 정책(Policy)에서 정하는 바를 따르도록 할 것이다.

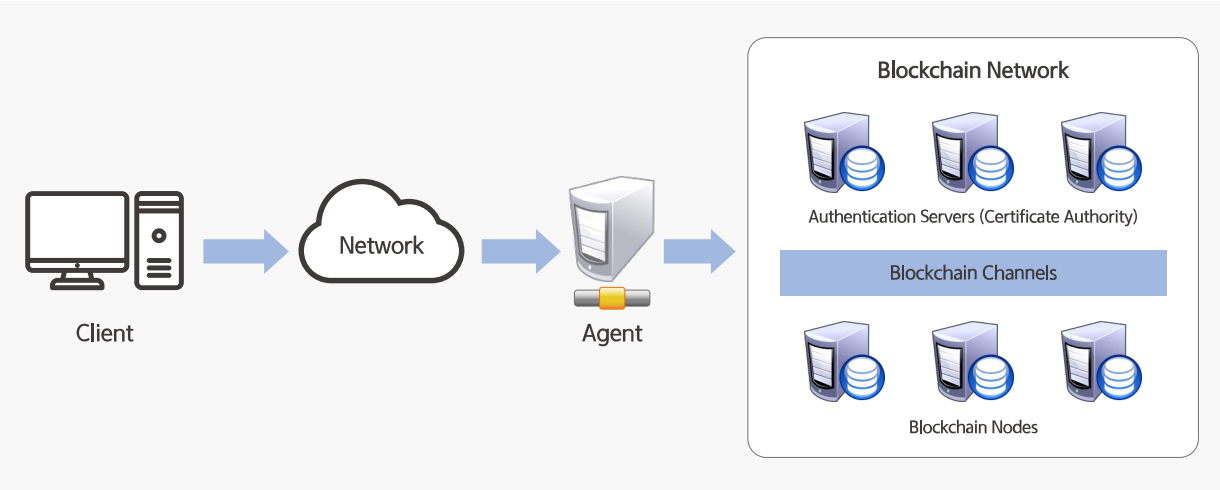
▣ IoT 서비스에 직접적으로 연결되는 기기는 PoC 단계에서는 가정하지 않는다(필요한 경우 연결할 수 있는 솔루션은 제공함). 따라서, IoT 기기의 Gateway역할을 하는 노드를 통해서 연결하는 것을 기본으로 한다. 이때 Gateway역할을 수행할 노드에 대해서는 기본적으로 OneM2M 이나 LWM2M 표준을 따른다고 가정한다. 따라서, OneM2M 이나 LWM2M 노드와 IoT 서비스 노드의 연결을 제공한다.

▣ HTTP를 사용한 Restful API를 제공하며, 기본 데이터 포맷으로는 JSON을 이용한다. 데이터의 내부 형식에 대해서는 IoT Gateway에서 연결에 제공하는 형식을 기본적으로 사용하도록 한다. 파트너(Partner) 사인 경우 해당 포맷 및 API에 대한 추가 및 확장을 제공할 수 있다.

▣ 연결된 IoT Gateway 노드들에 대해서는 별도의 고유 ID를 제공할 수 있다. IoT Gateway에 연결된 기기는 해당 IoT Gateway 내부에서 사용하는 고유 ID 및 외부에서 보여지는 ID가 달라질 수 있으며, 이는 개별 IoT Gateway 노드에서 제공하는 정보를 통해서 서비스를 활용할 수 있도록 한다.

3.3 Transaction 서비스 서브시스템 특성

Transaction은 주로 거래(Contract)에서 발생한다. TPS(Transaction Per Second)를 높이기 위해서는 분산원장 네트워크를 통해서 거래가 확정되기 전에 사용자에게 확정되었다고 알려주어야 한다. 하지만, 이럴 경우 서비스를 벗어난 범위에서 Transaction의 최종성(Finality)는 보장되지 않는다. 즉, ReapMiddleChain에서는 하나의 서비스 내에서만 Transaction의 최종성(완결성)을 보장한다. 또한, 사용자 거래 요청이 대량으로 발생하는 경우를 대비하기 위해서 수평적인 서버의 확장도 고려한 방안을 고안할 필요가 있으며, 이는 Transaction의 처리시 일부 중앙화 된 동기화를 거쳐야 한다는 것을 의미한다.



< 그림 9. 대리자(Agent)를 통한 블록체인 네트워크의 접근 예 >

네트워크에 연결된 노드들이 분산원장 서비스를 활용하기 위해서는 각각 대리자(Agent) 노드에 연결될 수 있다. 이때 각각의 대리자 노드들이 사용자의 요청을 받아 분산원장 네트워크에 접속하고, 요청 결과를 다시 사용자 측에 전달하게 된다. 대리자 노드들은 분산원장 네트워크를 구성하는 노드일 수도 있으며, 서비스 내부의 분산원장 노드에 연결하는 API만 제공할 수도 있다. 내부 노드 간의 의존성을 분리하기 위해서 각각의 노드는 Messaging 방식으로 동작하는 것을 기본으로 한다.

▣ API를 활용해Transaction을 서비스 내부 허가형 분산원장 네트워크의 클라이언트로 전달한다. 이때 클라이언트는 도착한 요청에 대해서 분산원장에 대한 처리를 대행하는 역할을 수행한다. 따라서, 클라이언트는 분산원장 노드에 접근하는 권한을 가진다고 볼 수 있다.

▣ 사용자 요청에 대해서 빠르게 반응하기 위해서 서비스 내부 처리가 완료되기 전에 사용자에게 요청의 처리 상태에 대해서 알려줄 수 있어야 한다. 이때 이중 지불(Double Spending)과 같은 문제가 생기지 않도록 충분한 상태에서만 Transaction의 결과를 사용자에 보고한다.

▣ 서비스 내부의 분산원장 네트워크의 속도와 도달하는 요청 사이의 속도차에 대한 대응방안을 제공할 수 있어야 하기에 트랜잭션을 일정 시간동안 버퍼링(Buffering)할 필요도 있다. 여기서 말하는 버퍼링은 트랜잭션을 일정 주기동안 모아서 처리하거나 특정 길이(개수)가 되었을 때 Batch형식으로 처리하는 것을 포함한다. 따라서, 이때 설정되는 파라미터(Parameter)에 따라 사용자 반응 시간이 영향을 받을 수 있지만, 지나치게 긴 시간을 기다리지 않도록 적절한 값으로 설정하도록 한다.

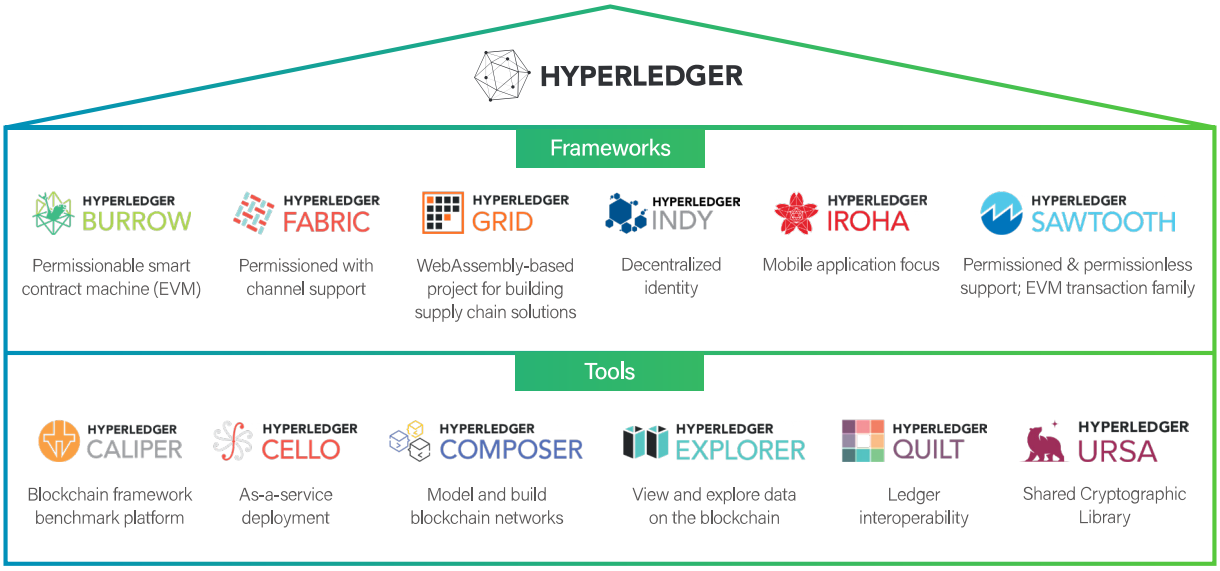
▣ 사용자 계정의 상태를 분산원장에서 읽어 들일 수 있어야 한다. Transaction 서비스 노드의 기동 시 이미 만들어져 있는 사용자 계정의 상태를 동기화 한다.

▣ 새로운 사용자 계정을 생성 및 변경, 삭제할 수 있다. 이때 관련된 분산원장 내부 데이터는 삭제되지 않지만, 사용자 계정을 이용한 추적은 할 수 없는 상태가 된다. 또한, 사용자가 등록하거나 인증한 IoT 기기에 대한 정보도 영향을 받을 수 있다.

▣ 거래 요청에 사용된 사용자 계정이 이미 만들어진 계정이 아닌 경우, 이를 확인하고 거래를 진행할 수 있어야 한다. 이때는 서비스의 사용자 계정 등록 및 인증 과정을 통하도록 하며, 서비스 내부의 분산원장 계정 생성도 연계하도록 한다.

3.4 분산원장 서브시스템 특성

분산원장 서브시스템은 IoT 데이터 및 Transaction이 분산원장 네트워크에 블록으로 생성하고 데이터의 변경 여부 및 계정의 최신 상태(Account 방식)를 검증하기 위해서 사용된다. 특정 분산원장을 가정하지는 않지만, PoC에서는 허가형 분산원장을 구축하도록 한다. 참고로 대표적인 허가형 분산원장 기술 프로젝트로는 Hyperleger Fabric이 있으며 Linux Foundation에서 관리하고 있다.



< 그림 10. Hyperledger Project 구성 >

▣ 허가형 DLT를 PoC 검증에 사용한다. DLT은 Docker 이미지 형태로 설치 및 설정이 가능하며, 별도의 배포 방식(Docker Repository)으로 제공될 수 있는 오픈소스 프로젝트다. 즉, 구축된 솔루션은 분리된 형태로 제공 및 운영 될 수 있다.

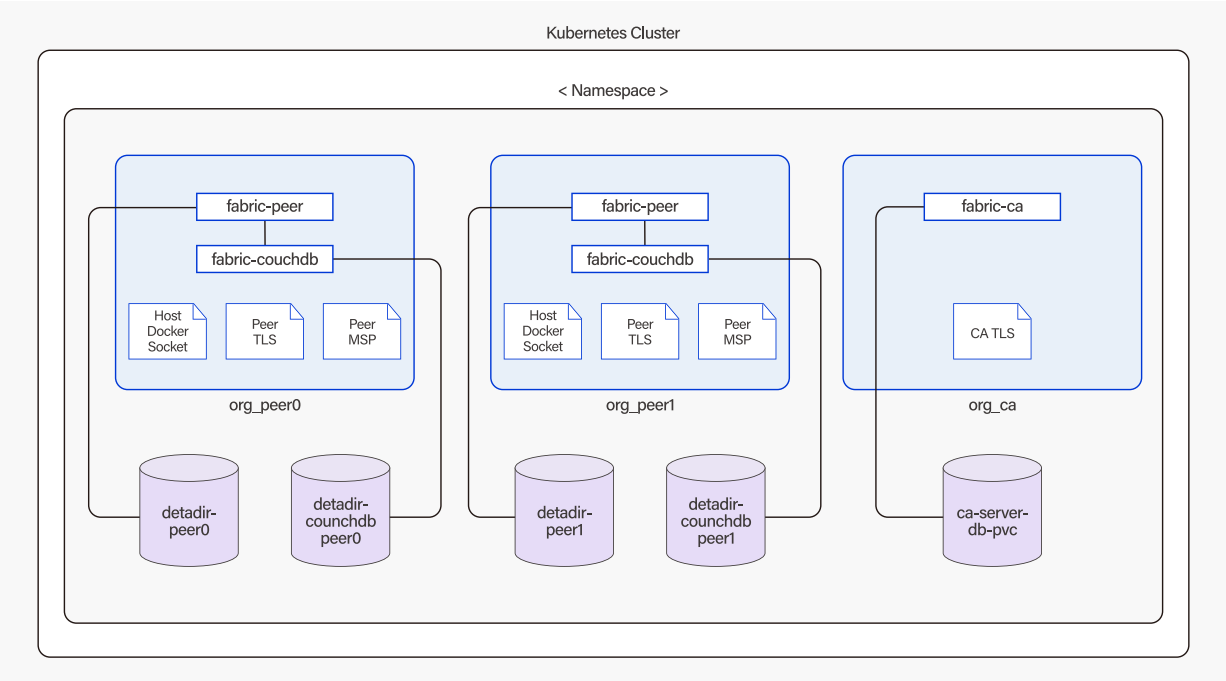
▣ 분산원장 네트워크 연결을 위한 Restful API를 분산원장 클라이언트 노드가 제공해야 하며, IoT 기기의 상태 정보에 대한 저장과 사용자의 Transaction 관련 상태를 변경하기 위한 인터페이스가 제공되어야 한다. 이는 Message Queue를 통해서 획득하게 되는 정보를 이용해서 각각의 인터페이스를 호출하는 방식으로 구현한다.

▣ 각각의 Restful API는 인증된 IoT 기기와 사용자를 대상으로 요청을 처리할 수 있는 방안을 제공한다. 인증되지 않은 요청의 경우에는 거부할 수 있으며, 거부 사유를 요청 측에 전달하도록 한다.

▣ Restful API는 새로운 계정의 생성과 계정 상태의 변경이 가능하도록 제공되며, 분산원장 특성상 계정의 삭제는 제공되지 않지만, 해당 계정의 사용은 막히게 된다. 즉, 삭제 요청이 있었다는 것을 인식하고 이를 분산원장의 상태(State)에 반영한다.

▣ 서비스 내부의 허가형 분산원장은 Reapchain Mainnet에 연결되며, 이를 위한 Agent역할을 정의하는 Adapter나 네트워크 노드가 존재한다. 서비스 내부에서 Reapchain Mainnet에 전달되어야 할 Transaction은 Agent를 통해서만 가능하다.

▣ 서비스 내부의 분산원장 네트워크는 최소 2,000 TPS이상을 제공한다. 이를 위해서 Hardware 및 네트워크 환경 구축에 대해서 적절하게 선택할 수 있다. 블록 생성 주기 및 블록 크기는 기본적으로 2초, 2 MB를 기준으로 하지만 설정에 의해서 변경될 수 있다.



< 그림 11. Kubernetes를 활용한 Hyperleger Fabric DLT Node의 구성 예 >

▣ 분산원장 네트워크 서비스의 내부 구성 노드에 대해서는 최소 3개 이상의 노드를 포함해야 한다. 그 외에 분산원장 네트워크를 구성하기 위한 노드들이 추가될 수 있다. 예를 들어, Messaging이나 인증(Certification)을 위한 노드들이 추가적으로 구성될 수 있다. 또한, 내부 서비스의 지속성에 영향을 줄 수 있기에 분산원장을 구성하는 노드의 수는 유동적으로 변경할 수 있도록 한다.

4. 외부 인터페이스 요구사항

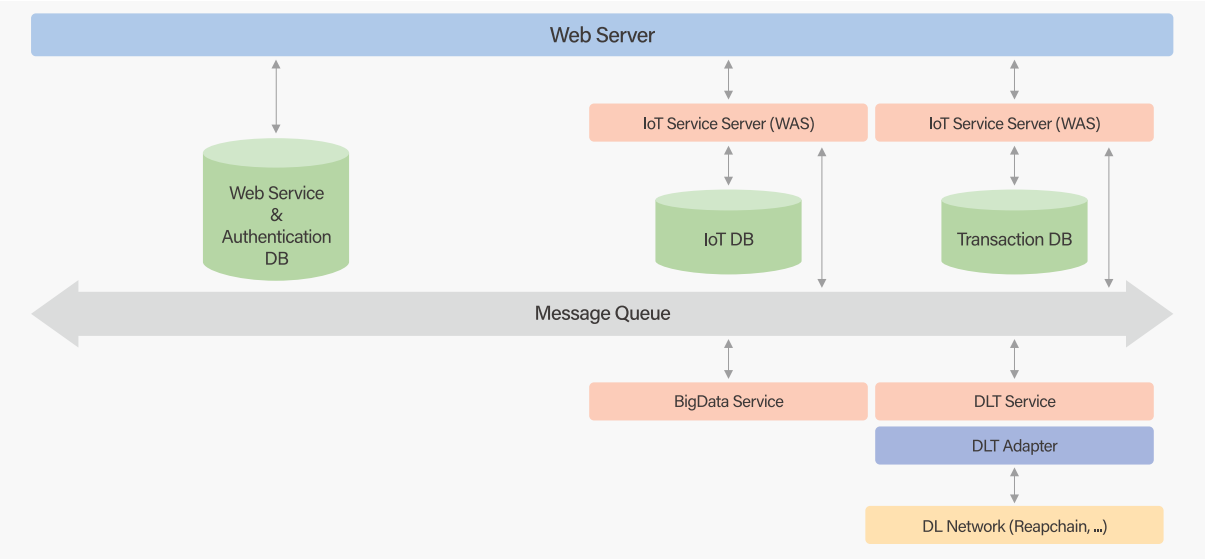
4.1 사용자 인터페이스

사용자는 별도의 앱을 사용하거나 Reapchain Middleware에서 구축한 웹 사이트에 접속해서 요청을 전달할 수 있다. 따라서, 웹 사용자 인터페이스와Restful API 두가지로 외부와 연결될 수 있으며, 추가적인 IoT 장치의 연결을 위해서 다양한 Protocol을 제공한다(예를 들어, MQTT).

▣ IoT 기기와의 연결은 기본적으로 Resful API를 제공하는 것으로 처리한다. 별도의 추가적인 외부 인터페이스는 정의하지 않는다.

▣ 웹 서비스의 사용자는 일반 사용자와 관리자로 나누어서 관리할 수 있기에, 각각 별도의 권한 설정에 따른 사용자 인터페이스를 제공해 줄 수 있어야 한다.

▣ IoT 기기와 직접적인 연결이 필요한 경우에는 해당하는 기기가 사용하는 Protocol을 추가 지원할 수 있다.



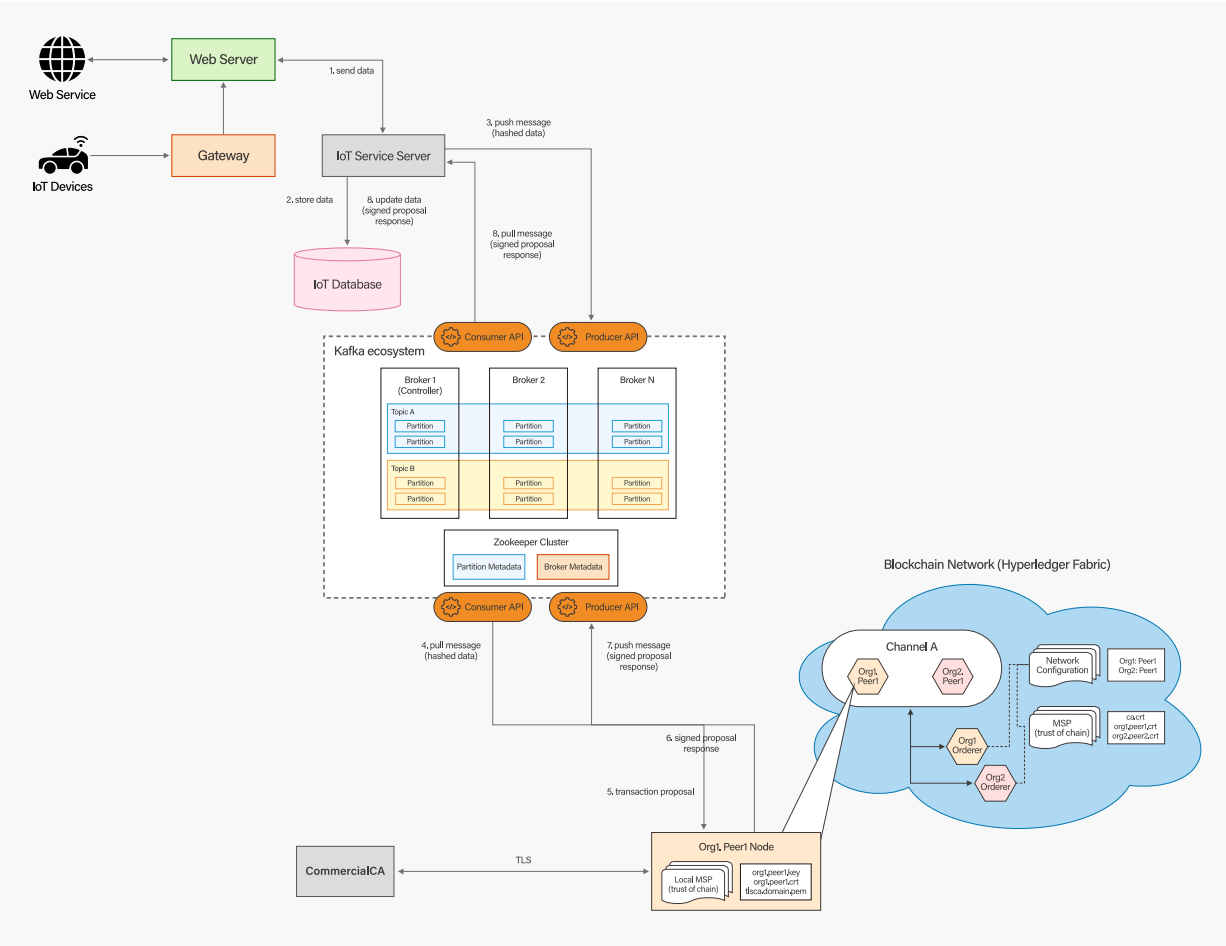
< 그림 12. Reapchain Middleware의 서비스 구성도 >

4.2 소프트웨어 인터페이스

Reapchain Middlware는 웹 및 Restful API를 사용한 IoT와 거래(Transaction)를 위한 서비스를 구축하는 것을 목적으로 한다. 따라서, 기본적으로 소프트웨어 인터페이스는 Restful API로 한정해서 구현될 것이다. IoT 기기 연결과 같은 추가적인 요구에는 해당하는 Protocol 구현을 포함할 수 있다.

▣ 트랜잭션 서비스의 경우는 웹 및 Restful API로 구현될 수 있으며, 기본적으로 Restful API를 제공한다. 만약 앱 구현이 Hybrid 형태인 경우, 이를 위해서 서버 측에서 구현해야 할 부분은 담당하도록 한다. 이는 구현에 의존적인 부분이라 PoC 단계 이후에 변경될 수 있다.

▣ 서비스 내부의 분산원장 네트워크에 대한 인터페이스는 Restful API로 한정한다. 특정 분산원장 사용 여부를 외부에서는 판단할 수 없으므로, 기본적으로 IoT 및 Transaction에 대한 두가지 인터페이스를 제공하는 것으로 한정한다.

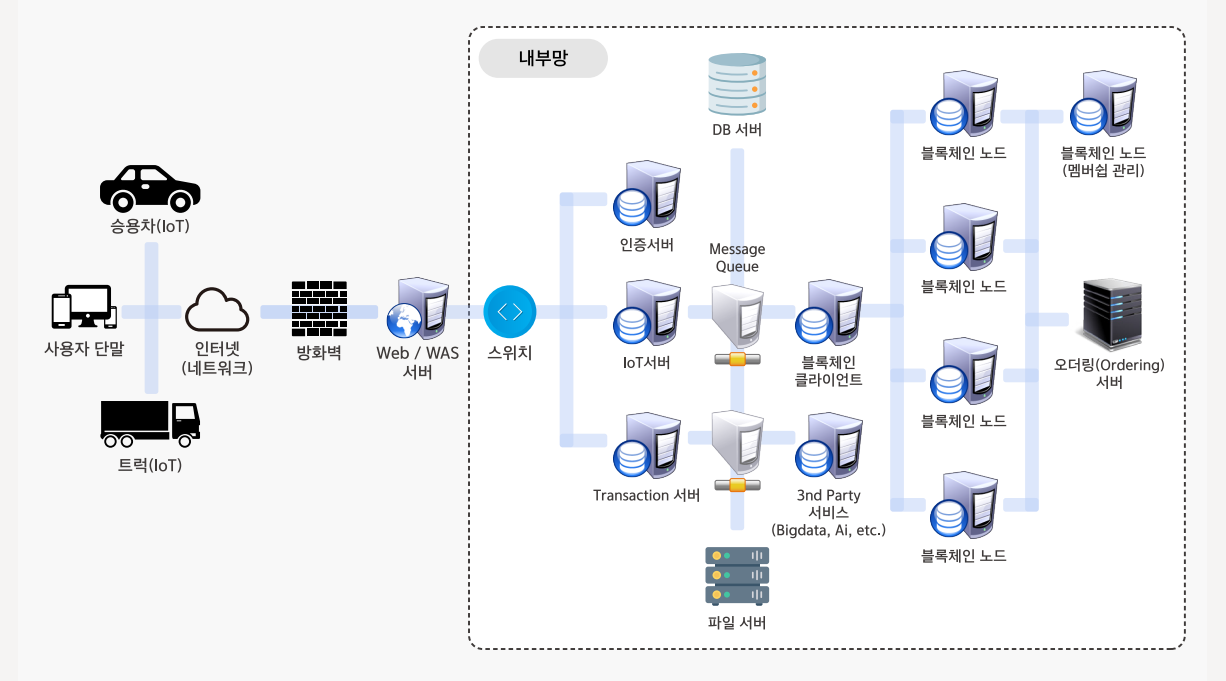


< 그림 13. 분산원장 기반 IoT(자동차 운행정보) 기록 시스템의 전체 구성 >

Reapchain Middleware에서도 다양한 외부 서비스와 결합할 수 있는 기본적인 API를 제공하고 있으며, 추가적인 서비스를 확장할 수 있는 내부 구조도 정의하고 있다. Web 및 IoT Gateway를 통한 인터페이스를 제공하고 있으며, 단지 IoT 서비스에 한정하지 않고 추가적인 API 확장도 가능하다. 내부 서비스의 확장을 위해서도 Message Queue를 사용함으로써 서비스 간 의존성을 최소화하고 있으며, 각각의 노드 간에 필요한 통신도 최소화하고 있다. Message Queue 자체는 CFT를 가지고 있어 데이터 손실 문제에 대응할 수 있다. 또한, 내부 허가형 분산원장 사용을 통해 원본 데이터의 조작 여부에 대한 문제도 충분히 대응할 수 있다.

4.3 Web/WAS 구성

Web/WAS는 외부 시스템과 인터페이스 및 사용자 인터페이스를 담당한다. 사용자 등록 및 시스템의 관리 및 모니터링 기능도 가지고 있다. 외부 시스템 연계는 REST API를 통해서 주로 담당하게 되지만, 실시간성 데이터의 연계를 위해서는 추가적인 프로토콜을 제공할 수 있다. 예를 들어, MQTT(Message Queuing Telemetry Transport)를 지원할 수 있다.



< 그림 14. IoT 및 사용자 Transaction을 위한 Reapchain의 네트워크 구성 예 >

제공되는 데이터 트래픽의 종류는 크게 IoT 기기에서 발생하는 데이터와 사용자 응용프로그램에서 발생하는 데이터가 있을 수 있다. 각각의 데이터는 종류가 다르기에 분리해서 다루어야 한다. 처리되는 위치는 IoT 서비스를 담당하는 서비스 노드와 사용자 Transaction을 처리하는 서비스 노드로 분리되며, 각각은 서비스 내부의 분산원장에 저장되는 형태로 다르게 관리된다. 예를 들어, DLT의 경우 채널(Channel)을 통해서 각각을 분리해서 분산원장에 저장할 수 있도록 지원하고 있다.

4.4 Message Queue 구성

메시지 큐는 다양한 서비스 연동을 위한 데이터 버스(Bus)의 역할을 한다. 향후 서비스 확장을 위해서 공용으로 데이터를 주고받을 수 있는 통로가 필요하며, 현재는 RabbitMQ와 Apache Kafka등을 고려할 수 있다. 보내는 측에서는 "Publisher/Subscriber"의 형태로 데이터의 전달이 완료되었음을 확인할 필요가 없으며, 받는 측에서는 데이터가 도착하지 않았을 때는 다른 내부적인 작업을 진행할 수 있도록 비동기 방식으로 운영될 수 있다.

메시지 큐 자체가 CFT(Crash Fault Tolerance)를 제공하기에 데이터 손실에 대해서 내구성을 가지고 있다고 볼 수 있으며, 새로운 내부 처리 과정을 추가하거나 관리하기에 쉽다. 물론 데이터 처리상에 "Route"가 길어지는 문제점을 가질 수 있지만, 이는 내부 처리 과정을 최적화하고 Transaction의 일관성을 보장하는 방법으로 해결할 수 있는 문제라고 생각한다.

4.5 DLT Client

메시지 큐에서 서비스 내부 허가형 분산원장에 저장할 데이터가 도착했을 때 이를 처리하는 부분이다. 분산원장 네트워크에 멤버쉽(Membership)을 가진 노드(Node)로 등록되며, 서비스 내부의 허가형 분산원장으로 트랜잭션을 전달하기 위해서 분산원장에서 제공되는 SDK를 사용해야 한다. 이를 위해서 필요한 SDK를 구현 언어별로 제공받을 수 있어야 한다. 이때 분산원장에서 실행되어야 할 "Smart Contract"를 제공하는 역할을 수행할 수 있다.

분산원장의 Smart Contract는 비즈니스 로직을 구성하는 프로그램화 된 계약을 실행하는 부분이기에, 각각의 제공하는 서비스 마다 달라질 수 있다. 따라서, 구현하고자 하는 서비스에 특화된 Smart Contract를 작성하는 것은 개별 서비스의 몫으로 남겨둔다. 예를 들어, "Voting, IoT, Authentication & Identification(DID), NFT(Non-Fungible Token)"등을 구축할 수 있을 것으로 전망한다.

4.6 DLT

분산원장 네트워크를 구성해 비가역성 데이터의 저장과 검색/수정 등을 처리하기 위해서 필요한 부분이다. 현재는 허가형(Permissioned) DLT를 사용한다. 서비스 채널 별 멤버쉽(Membership) 권한 부여와 관리 기능 노드와 DLT에서 사용하는 Smart Contract 를 실행/검증/블록 쌓기를 하기 위한 기능 노드, Transaction의 순서와 블록 생성을 위한 기능 및 관리를 위한 노드, 생성된 블록을 검증하고 체인형태로 저장하기 위한 노드(Node)들로 구성된다.

기본적으로 DLT를 구성하는 모든 노드들은 “Docker” 이미지 형태로 만들어지며, 필요시 추가적으로 인스턴스(Instance)를 생성해서 서비스를 제공할 수 있도록 한다. 또한, 네트워크 자원의 효율적인 관리를 위한 “Kubernetes”와 모니터링을 위한 “Prometeus”와 “Graphana”, “Blockchain Explorer” 등도 구축해서 서비스 확장과 운영에 대한 부분을 책임지고 있다. 이와 더불어 분산원장 자체의 운영과 관리를 위한 도구들도 분산원장 운영에 필요하다고 생각되며, 지속적으로 구축해 나갈 수 있도록 할 것이다.