Blockchain Infrastructure

엔터프라이즈 시스템/네트워크 운영자 대상

(for IT Pros and System Administrators)

2018. 07.

안종석
james@jslab.kr

- 사용 유효기간을 2018년 12월 까지로 권합니다.
- 실습 교재는 별도 입니다.

JS Lab

목차

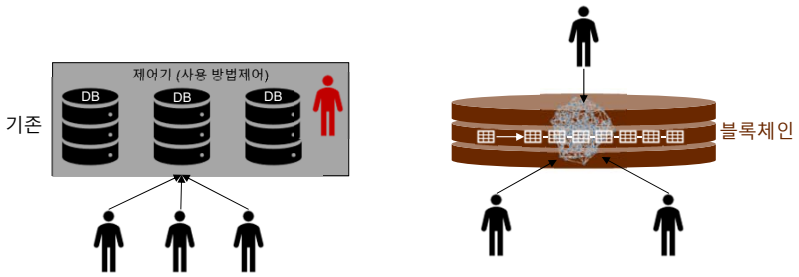
A. 블록체인 인프라
   I. 개요
   II. 퍼블릭 블록체인 (Public Blockchain)
   III. 스마트 계약(Smart Contract)
   IV. 프라이빗 블록체인 (Private Blockchain)
   V. 컨소시엄 블록체인 (Consotium Blockchain)
   VI. 하이퍼레져(Hyperledger)
   VII. 컨센서스 (Consensus)
B. 컨테이너
C. 실습 (별도 교재)

A.I. 개요

A.I. 개요

❖ 블록체인 기술의 기존 데이터베이스와 차이점

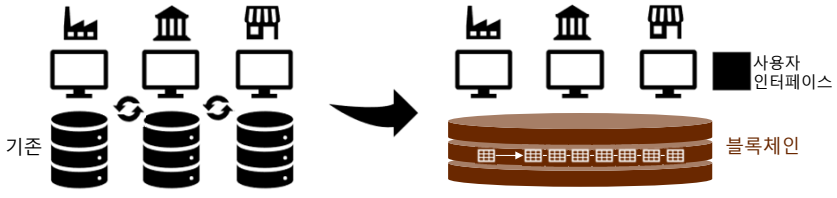
- 블록체인은 쓰기만 가능한 데이터 구조
- 새로운 기재(Entry)는 원장의 끝에 붙임
- 블록체인 내에서는 데이터의 편집이나 제거를 허락하는 관리자 권한이 없음
- 기존 장부에는 수표나 영수증 또는 약속어음의 교환내역이 기록되는 반면에, 블록체인은 그것 자체가 거래장부인 동시에 거래증서(수표, 영수증, 약속어음)



JS Lab

A.I. 개요

❖ **블록체인(Blockchain):** 데이터가 안전하게 저장되고 감사 가능한 공유 데이터 계층을 생성하여 사용자 인터페이스를 제공



사일로에 저장한 데이터로 호환에 제한적
(검증 받은 인터페이스 필요)

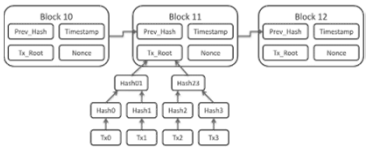
안전하게 저장되어 감사(Audit) 가능한
공유(Shared) 데이터

블록을 체인하여 구성

JS Lab

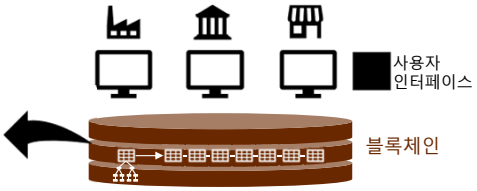
A.I. 개요

❖ **블록(Block):** 특정 시간 동안 발생한 거래(Transaction)들을 처리한 리스트
❖ **체인(Chain):** 각 블록에 생성시각, 거래내역 묶음 해시, 해시 값 생성한 계산 횟수, 전 블록의 블록 해시 등을 지정하고 암호화 알고리즘을 사용하여 시간 순으로 블록을 연결



Merkle Chains 또는 Merkle Trees
(효율적 저장)

해시 함수(hash function)는 임의의 길이의 데이터를 고정된 길이의 데이터로 매핑하는 함수이다. 해시 함수에 의해 얻어지는 값은 해시 값, 해시 코드, 해시 체크섬 또는 간단하게 해시라고 한다.

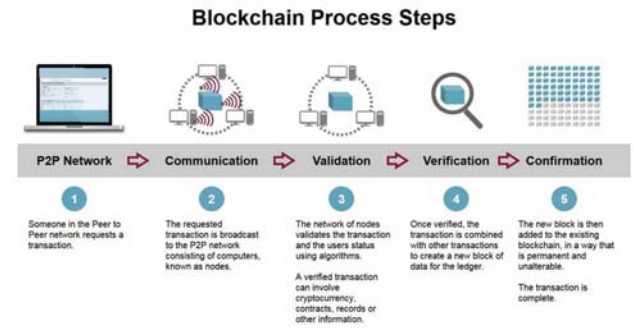


- **비트코인(Bitcoin)은 약 10분 단위로 전세계의 거래들을 모아 조건을 만족하는 해시(Hash) 값을 구해 새로운 블록을 생성(채굴)**
- **이더리움(Ethereum)은 12~15초마다 새로운 블록 생성**

A.I. 개요

❖ **블록체인 프로세싱 단계**

- **P2P 네트워크**
- **Communication**
- **Validation**
- **Verification**
- **Confirmation**



A.I. 개요

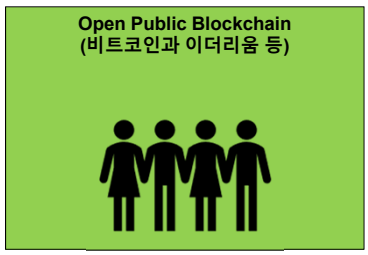
❖ 블록체인 앱 (Blockchain Applications)

- 블록체인은 디지털 인프라의 새로운 형태로 블록체인 앱은 블록체인 상에서 제공하는 게이트웨이를 사용하여 정보를 접속
- 클라이언트/사용자들은 앱을 통해 블록체인과 상호 작용
- 비트코인을 연결하는 단순 지갑(Wallet) 앱 부터 디지털 표식을 참조하거나 블록체인 상에서 금융거래를 수행하는 복잡한 앱을 포함한다. (e.g. UPort, KYC-Chain, Netki, etc.)

A.I. 개요

❖ 블록체인의 형태

- 블록체인은 참여 허락이 필요 없는 **permissionless**(Bitcoin 과 Ethereum)**와** 허락이 필요한 **permissioned** (Hyperledger blockchain frameworks) **블록체인이 있음**



A.I. 개요

❖ **블록체인의 종류:** 참여자 범위에 따라 퍼블릭, 프라이빗, 컨소시엄 3가지로 구분

- **퍼블릭 블록체인(Public Blockchain)**
- **프라이빗 블록체인(Private Blockchain)**
- **컨소시엄 블록체인(Consortium Blockchain)**

요소	퍼블릭 (Public)	프라이빗 (Private)	컨소시엄 (Consortium)
관리주체	모든 거래 참여자(탈중앙화)	한 중앙기관이 모든 권한 보유	컨소시엄에 소속된 참여자
거버넌스	한번 정해진 법칙을 바꾸기 매우 어려움	중앙기관의 의사결정에 따라 용이하게 법칙을 바꿀 수 있음	컨소시엄 참여자들의 합의에 따라 상대적으로 용이하게 법칙을 바꿀 수 있음
거래속도	네트워크 확장이 어렵고 거래 속도가 느림	네트워크 확장이 매우 쉽고 거래 속도가 빠름	네트워크 확장이 쉽고 거래 속도가 빠름
데이터 접근	누구나 접근 가능	허가받은 사용자만 접근가능	허가받은 사용자만 접근가능
식별성	익명성	식별 가능	식별 가능
거래증명	검증 알고리즘에 따라 거래 증명자가 결정되며 거래 증명자가 누구인지 사전에 알 수 없음	중앙기관에 의하여 거래증명이 이루어짐	거래 증명자가 인증을 거쳐 알려진 상태이며 사전 합의된 규칙에 따라 거래검증 및 블록생성이 이루어짐
활용사례	Bitcoin, Ethereum 등	나스닥 링크(Linq) 등	R3CEV, Tendermint 등

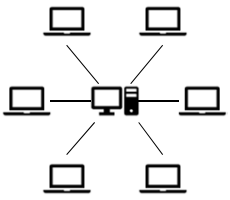
자료: 금융보안원, 한화투자증권

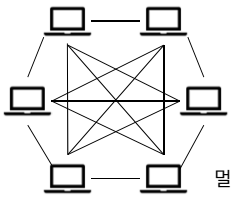
암호화폐가 불필요한 Private Blockchain을 단순한 분산형 데이터베이스로 보는 시각도 있음

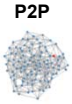
A.I. 개요

❖ **P2P(Peer-to-Peer) 네트워크 아키텍처**

- **피어들은 네트워크를 유지하기 위한 컴퓨팅 파워와 스토리지를 기부함**
- **P2P 네트워크는 일반적으로 중앙집중 네트워크보다 공격(single point of attack)에 대해 안전하게 여겨짐**
- **Permissionless P2P 시스템은 온라인을 위해 피어 수에 대한 제한이 필요하지 않으나 일반적으로 느림**
- **Permissioned P2P 네트워크는 통신 회선의 QoS와 가용성(Uptime)의 일정 수준이 필요함**







멀티캐스트 사용 가능

A.I. 개요

❖ 글로벌 동향:

- 뉴욕에서 비트코인 규제를 위한 BitLicense도입하여 관련 사업자는 이를 취득
- 중국은 정부가 적극적으로 인프라 구축등 기술 선도
- 영국은 정부에서 'Beyond Blockchain 전략' 제시
- 일본은 블록체인 기술은 물론 암호 화폐 거래에도 개방적
- 에스토니아 디지털 시민권 (e-Estonia, e-Residents)
- 스위스 크립토 밸리(Crypto Valley)로 규제의 샌드박스 제도는 ICO 증가

연도별, 국가별 블록체인 관련 특허 출원 현황 (단위: 건)

출원인 국적	'07년	'08년	'09년	'10년	'11년	'12년	'13년	'14년	'15년	'16년	'17년	'18년	합계
미국	4	4	5	2	10	12	18	62	136	186	58		497
중국		4	2	2		3	3	9	25	321	103		472
한국				1			2	11	33	41	10	1	99
일본		4	2		2	1	2	4	11	2	8		36
유럽				1		2	2	6	24	22	16		73
기타				2	1	3		6	29	22	8		71
총합계	4	12	9	8	13	21	27	98	258	594	203	1	1,248

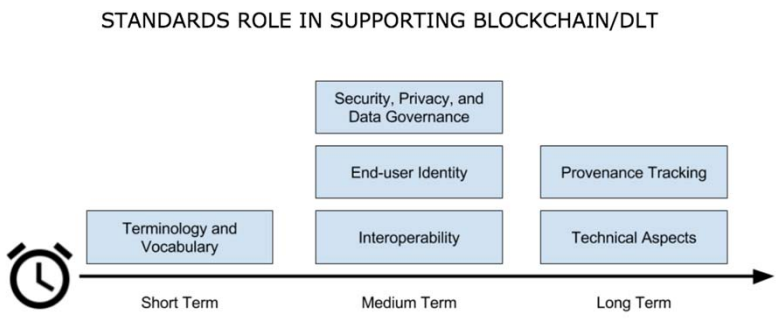
자료: 특허청, 한화투자증권

JS Lab

A.I. 개요

❖ 표준 (Standards)

- **ISO TC 307:** 블록체인과 분산 원장 기술을 위한 ISO 국제 표준기구가 2016년 설립되었고 미래의 표준을 위한 영역을 정의 중임(Clare Naden, 2017)



https://www.iso.org/news/Ref2188.htm

JS Lab

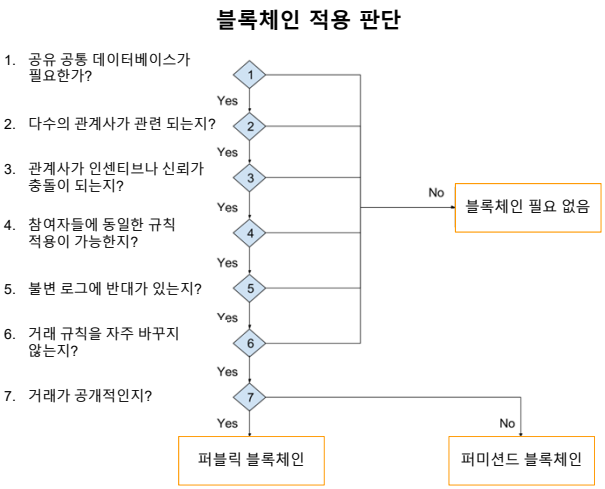
A.I. 개요

❖ 분산원장 기술 사용시 어려움

- 표준의 부족
- 규제
- 분산 원장 기술의 지식과 경험 부족
- 신 기술에 대한 저항

A.I. 개요

❖ 블록체인 기술의 비즈니스 적용 판단



https://imgur.com/r/CryptoCurrency/wl7YI

A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ 비트코인 Public Blockchain (Permissionless)의 기술 특징:

- **새로운 지불 시스템:** 합의된 결제 네트워크이자 디지털 신종 화폐
- **공개키 암호화 기술:** 익명성 기반의 암호화 제공 (Anonymous PKI-based)
- **P2P 분산 네트워크:** 자발적 참여로 거래 확증
- **번복 불가 시스템:** 가짜 지불 방지로 판매자 보호
- **타임 스템프:** 거래시각을 입증하여 이중지불 방지
- **통화공급량 제한:** 통화 공급의 안정성
- **오픈소스:** 시스템 운영 소스공개로 80%이상 동의로 누구나 수정 가능

❖ 이더리움 Public Blockchain (Permissionless)의 기술 특징:

- **블록체인 앱 플랫폼**(오픈소스로 공개한 Blockchain App Platform)
- **스마트 계약 가능:** 거래 내역과 함께 계약 등의 추가정보 기록 가능하여 입력한 조건이 만족하면 계약을 실행 하도록 코딩
- **EVM(Ethereum Virtual Machine):** 스마트계약 생성시 동작하여 스마트 계약의 검증과 실행을 블록체인에 기록

** **Public Blockchain(Permissionless)**은 주어진 네트워크 인프라 환경에서 기술 구현
** **Private Blockchain** 과 **Consortium Blockchain(Permissioned)**에서는 네트워크 인프라 구성 고려

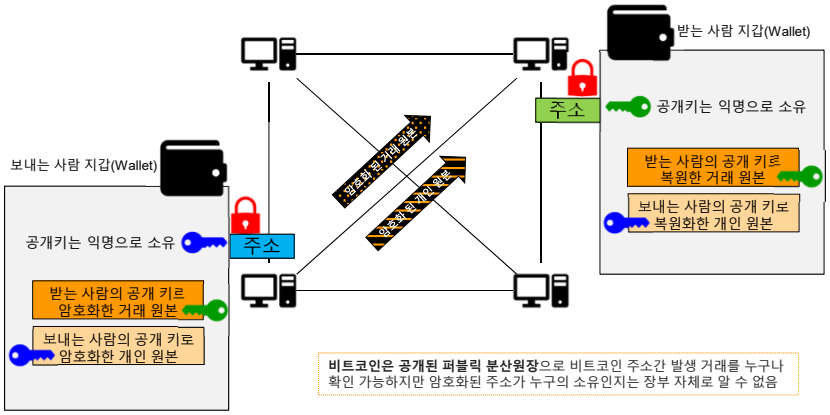
https://github.com/bitcoin/bitcoin

JS Lab

A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ 가상화폐 거래 (PKI 기반):

- **Wallet(지갑)**
- **개인키/공개키 생성** (공개키는 익명으로 소유)
- **주소생성**

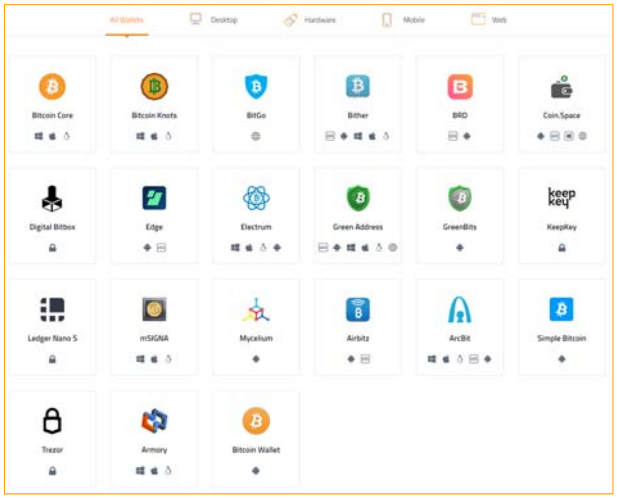


비트코인은 공개된 퍼블릭 분산원장으로 비트코인 주소간 발생 거래를 누구나 확인 가능하지만 암호화된 주소가 누구의 소유인지는 장부 자체로 알 수 없음

JS Lab

A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ **비트코인 지갑(Bitcoin Wallet):** 저장하고 보관하는 것은 비트코인이 아닌 접근 권한을 부여하는 개인 키 (bitcoin.org는 코드를 공개한 지갑만 표시)



https://bitcoin.org/en/choose-your-wallet

JS Lab

A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ **채굴(mining):**

- **입출금 내역을 검증하고 이를 포함하는 신규블록을 생성하는 작업**
- **채굴자에게는 보상으로 비트코인을 발행**
- **채굴자는 거래내역을 시스템 요구 해시함수 목표값을 찾기 위해 경쟁**

❖ **작업증명(Proof of Work):**

- **해싱을 통해 목표 값을 찾는 작업**
- **전(前) 블록의 해시값, 미승인 거래기록, 논스(nonse)에 임의 값을 입력하여 새 블록의 해시값을 계산해 목표 값과 비교하는 과정을 반복**

❖ **블록 생성:**

- **목표 값을 찾는데 성공한 참여자가 블록 발행, 이를 네트워크에 전파**
- **블록에는 전(前) 블록의 해시값, 현재 블록의 타임스탬프, 해시 목표값(난이도), 논스(Nonse)값, 현재 블록에 기록된 거래내역을 포함하는 해시 값을 기록**

- **채굴**은 해싱작업을 통해 임의로 생성한 값을 목표값과 대조해보는 작업이기 때문에 컴퓨팅 파워를 많이 투입 할 수록 유리하고 채굴 초기에는 일반적인 개인 컴퓨터에서도 가능 했으나 거래와 채굴노드가 증가 할 수록 채굴의 난이도가 높아져 전문 장비들을 활용
- 계산(채굴) 중에 다른 노드가 생성한 블록을 감지하면 다음 블록을 만드는 작업으로 전환

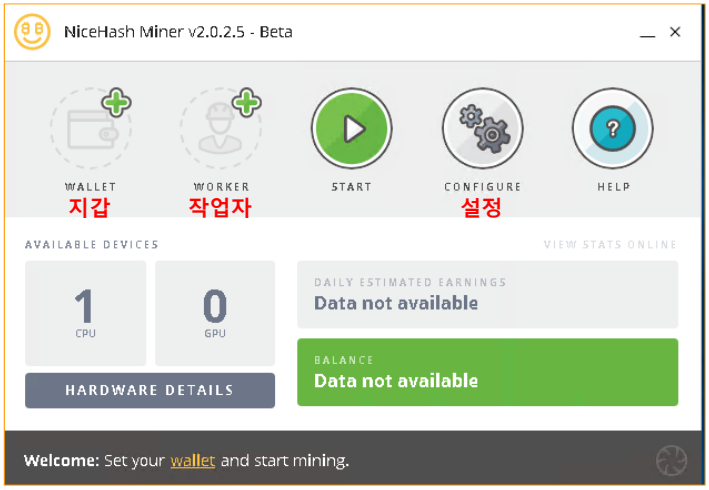
JS Lab

A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ 채굴 소프트웨어

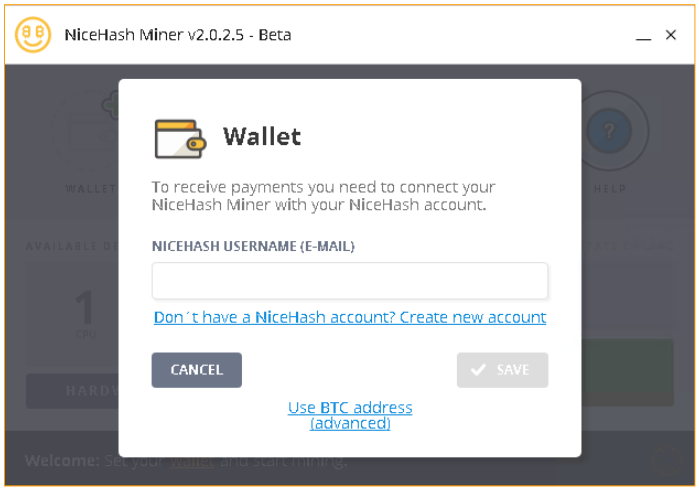
A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ 채굴 소프트웨어 시작 화면 (NiceHash Miner)



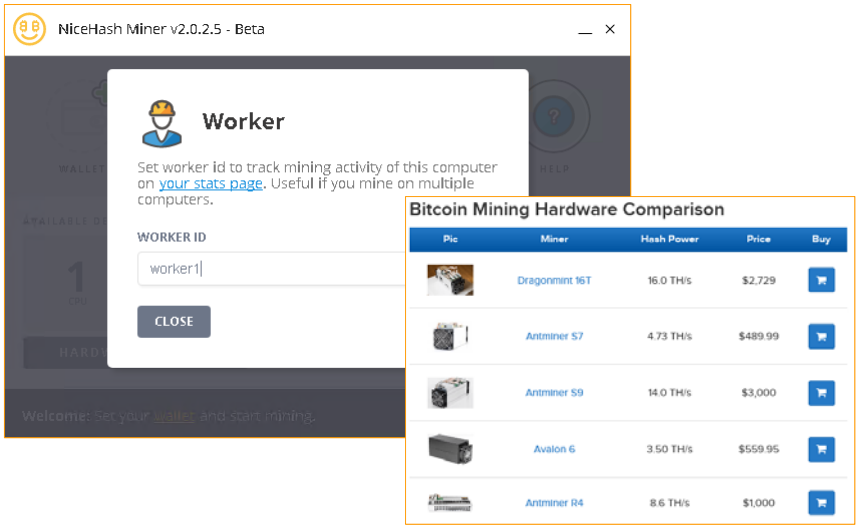
A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ 채굴자의 지갑



A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ **채굴자의 워커** (다수의 컴퓨터 사용시 편리)



A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ **비트코인 채굴(mining) 보상:**

- **비트코인은 총 공급량 2100만개로 제한**
- **블록당 채굴 비트코인은 21만번째 블록생성마다 절반으로 감소하도록 설계되어 약 4년 마다 절반으로 감소** (2009년 블록 발행 보상은 50 비트코인이었으며, 2013년에는 25비트코인, 2016년은 12.5 비트코인으로 감소)
- **현재 약 1700만개 이상 채굴 되었고 채굴 종료는 2140년으로 예상**

❖ **비트코인 분산원장 용량:**

- **2009년 최초 블록 생성 후 약 10분마다 1 MB의 새로운 블록이 생성**
- **블록당 약 2000건의 거래를 저장**
- **현재 약 170 GB 정도로 개인 PC에서도 저장 가능**

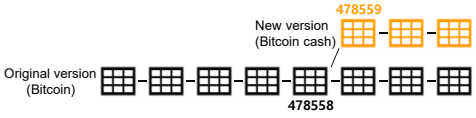
이더리움은 채굴 보상으로 3개의 이더 발행

https://www.reddit.com/r/Bitcoin/comments/3ame17/a_payment_network_for_planet_earth_visualizing/

A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ **소프트포크와 하드포크:**

- **블록체인은 동시에 새로운 블록이 발생가능 하며 긴 체인을 참으로 인정**
- **비트코인은 누구나 소프트웨어 수정안을 제시 가능하며 참여자들이 동의하면 새로운 버전의 소프트웨어를 다운로드 받아 실행**
- **이전 버전으로 생성한 블록과 호환하는 방법은 소프트포크(Soft fork), 호환되지 않는 변경을 하드포크(Hard fork)라 함**
- **2017년 8월 1일 비트코인에서 비트코인캐시(Bitcoin Cash)가 478558번 블록에서 분기되는 하드포크 발생** (블록 사이즈와 처리속도 개선 의견에 대한 합의가 되지 않아 갈라짐)



A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ **비트코인(Bitcoin)과 비트코인 캐시(Bitcoin Cash):**

- **2017년 7월 20일, 비트코인 채굴자들은 97%가 비트코인 개선 제안(Bitcoin Improvement Proposal, BIP) 91에 호응을 보였다. 비트메인 워런티의 엔지니어 제임스 힐라드가 낸 이 제안은 세그위트(Segregated Witness)를 활성화하기 위한 것이었다.**

- **8월 1일에 비트코인 캐시를 하드 포크로 구현할 것이라고 발표하였다. 해당 시점에서 비트코인 화폐의 거래 역사는 계승하지만 그 이후의 모든 거래는 분리되었다. 블록 478558은 마지막 공용 블록이었으로 최초의 비트코인 캐시 블록은 478559가 된다. 비트코인 캐시 암호화폐 지갑은 2017년 8월 1일 13:20 UTC 이후로 BTC 블록과 BTC 거래를 거부하기 시작했는데, 그 이유는 포크 시작을 위한 타이머가 사용되었기 때문이다. 블록 크기는 8 MB로 상승되도록 구현되어 있다. 비트코인 캐시는 7월 23일에 선물 거래를 0.5 BTC에서 시작했지만 7월 30일 0.1 BTC로까지 하락했다. 시가 총액은 2017년 8월 1일 23:15 UTC 이후로 등장하였다. 2017년 10월 29일 기준으로 1 비트코인 캐시는 대략 0.084 비트코인으로 거래되고 있었다.**

https://ko.wikipedia.org/wiki/%EB%B9%84%ED%8A%B8%EC%BD%94%EC%9D%B8

A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ **이더리움(Ethereum)과 이더리움 클래식:** 이더리움은 2015년 7월 30일 비탈릭 부테린(Vitalik Buterin)이 개발. 비탈릭 부테린은 가상화폐인 비트코인에 사용된 핵심 기술인 블록체인에 화폐 거래 기록뿐 아니라 계약서 등의 추가 정보를 기록할 수 있다는 점에 착안하여, 이 플랫폼을 이용하여 SNS, 이메일, 전자투표 등 다양한 정보를 기록하는 시스템을 창안.

- **2016년 6월 The DAO에 대한 해킹 사건 발생후 기존 이더리움에 대한 하드포크(Hard Fork)를 진행하여 이더리움이 두 가지 버전으로 분리. 구 버전을 이더리움 클래식(Ethereum Classic, ETC)이라고 부르고, 신 버전을 이더리움이라고 함**

이더리움

Version	Code name	Release date
0	Olympic	May, 2015
1	Frontier	30 July 2015
2	Homestead	14 March 2016
3	Metropolis (vByzantium)	16 October 2017
3.5	Metropolis (vConstantinople)	TBA[26]
4	Serenity	TBA

Old version | Latest version | Future release

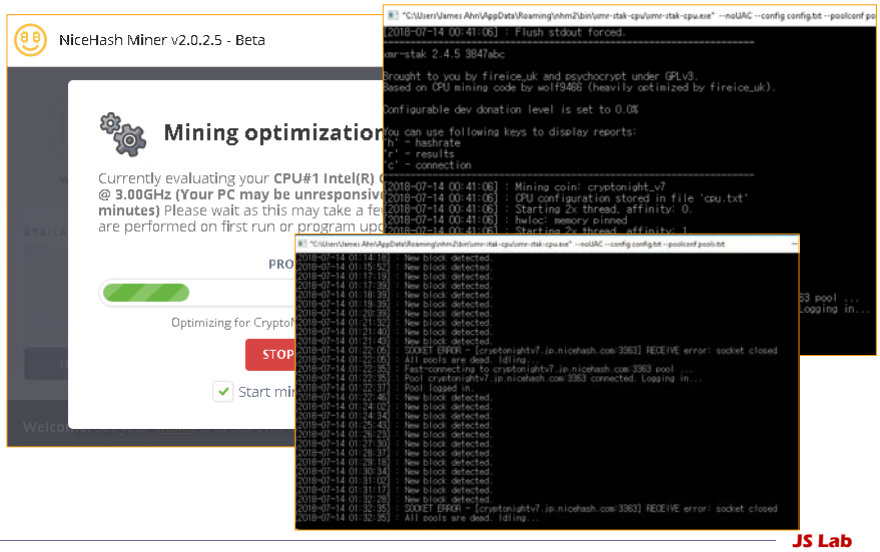
이더리움 클래식

Release Date	Code name	Hard Forks and Soft Forks
30 July 2015	Frontier	The release of the Ethereum Genesis block.
14 March 2016	Homestead	The 2nd major release of the Ethereum platform, which introduced EIP-2, EIP-7, and EIP-8.
25 October 2016	GasReprice	First fork after being renamed "Ethereum Classic". Repriced some operations to prevent DoS attacks affecting both Ethereum and Ethereum Classic networks. Introduced ECIP-1050.
14 January 2017	Die Hard	Delayed the difficulty bomb which was originally intended to force the network to move from proof-of-work to proof-of-stake and added replay protection to prevent transactions on the Ethereum network being accepted on the Ethereum Classic chain. Introduced ECIP-1010 and EIP-155.
11 December 2017	Monetary policy change	Change unlimited token emission to a fixed-cap monetary policy similar to bitcoin with a hard cap of around 210 Million.

https://en.wikipedia.org/wiki/Ethereum_Classic

A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ 채굴 정보 확인



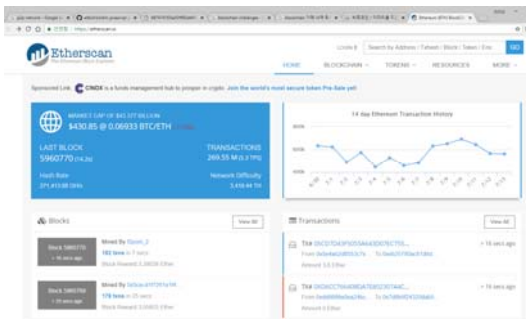
JS Lab

A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ 컨센서스 **PoW**(Proof of Works)

- **Hash based**
- **분산 p2p 기반 Consensus Protocol 사용하여 sybil node 감지**
- **긴 블록 선호 정책**
- **구성원에게 동기 부여 (채굴)**

❖ 체인(Chain) 공개:



https://etherscan.io/

JS Lab

A.II. 퍼블릭 블록체인 (Public Blockchain)

❖ Ethereum의 Dapps (Decentralized Applications)

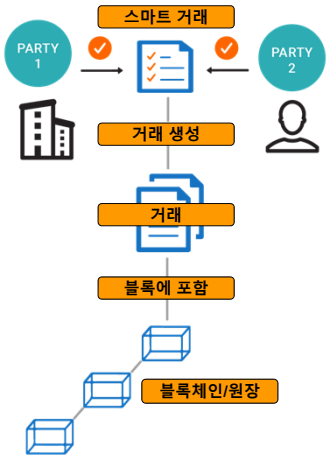
- "Ethereum은 누구나 블록체인 기술상에서 분산앱을 사용하게하는 공개 블록체인 플랫폼이다."
- 다양한 인터페이스를 통해 P2P 로 상호 접속 (social, financial, gaming, etc.)
- Ethereum 블록체인 플랫폼은 스마트 거래 등의 기능을 스크립트화 하여 네트워크 내의 노드들에서 구동
- 비트코인과 달리 거래의 추적은 물론 프로그램화 할 수 있음.
- 기술적으로 Ethereum은 암호화폐인 이더(ether)로 기존의 인지를 재구축하는 가상머신으로 표현
- 모든 앱사용자와 타협(compromise)한 앱 코드의 임의 변조는 불가능(이를 위한 노드들이 능동적으로 상호작용)

JS Lab

A.III. 스마트 계약 (Smart Contract)

❖ Smart Contracts

- 스마트 계약(Smart contract)은 거래가 지정한 상태를 만날 때 정한 일을 수행하는 컴퓨터 프로그램
- 스마트 계약은 원장을 갱신하는 거래의 언어를 제공
- 어떠한 가치를 보내거나 교환하는 것을 촉진 (주식, 돈, 컨텐츠, 자산)

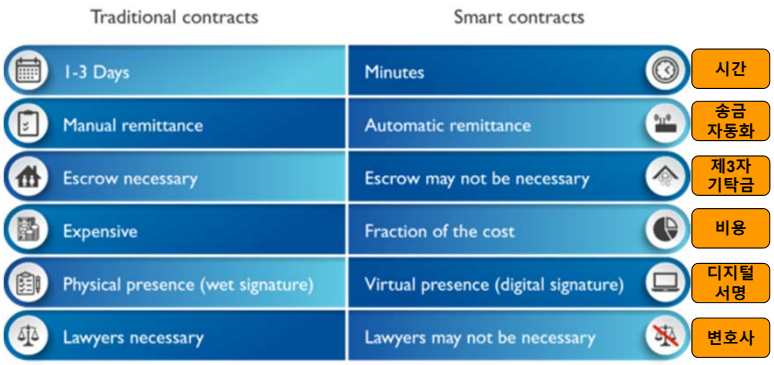


JS Lab

A.III. 스마트 계약 (Smart Contract)

❖ 스마트 계약의 기존과 다른점

- 블록체인에 거래를 기록시 쉽고 단순하게 하는 많은 솔루션들을 자동화 할 수 있음



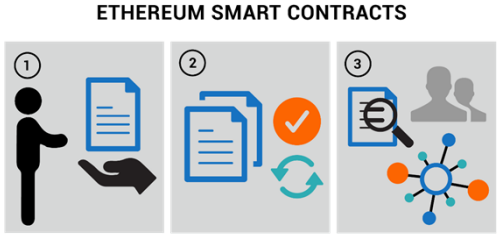
https://www.edureka.co/blog/smart-contracts/

JS Lab

A.III. 스마트 계약 (Smart Contract)

❖ 이더리움의 스마트 계약(Ethereum Smart Contracts)

- 스마트 계약은 투자금이 필요한 회사와 투자자 사이의 협약을 암호화 할 수 있음 ①
- 스마트 계약은 이더리움 퍼블릭 블록체인에 존재하며 Ethereum Virtual Machine (EVM) 상에서 동작함 ②
- 이벤트가 발생하면 유효 기간이 있는 것과 같이 코딩된 가격을 확인하고 자동으로 스마트 거래의 비즈니스 로직을 실행 ②
- 단속자(Regulator)가 시장의 활동을 감시 할 수 있으나 특정인의 ID를 손상하지 않는 장점이 있음 ③



JS Lab

A.III. 스마트 계약 (Smart Contract)

❖ 스마트 계약(Smart contract):

- 이더리움은 스마트 계약을 분산 어플리케이션(Decentralized application or DApp)이라 하며 1500 여개의 DApp이 만들어짐 (토큰 발행 가능)
- 이더리움은 비트코인과 같은 결제/송금 가능한 외부소유계정(Externally Owned Account)과 스마트 계약 가능 계약계정(Contract Account)으로 구분하며, 스마트계약 구동 시 컴퓨팅 파워 사용을 위한 가스(gas)필요
- 하이퍼레져패브릭(Hyperledger Fabric)은 Chaincode로 스마트계약 수행

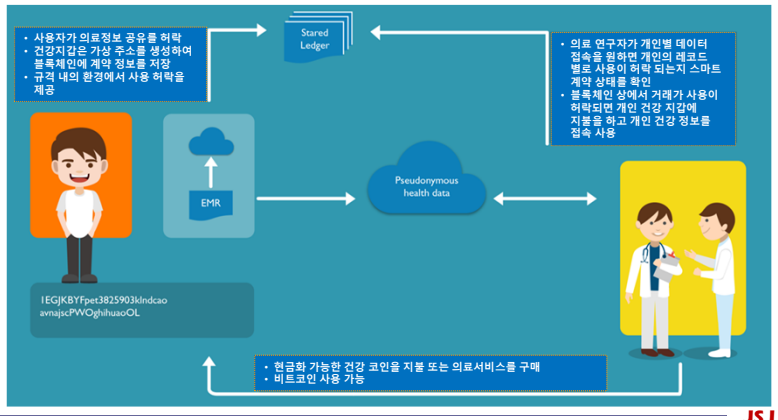
	이더리움	하이퍼레져 패브릭	R3 Corda
프로그램 언어	Solidity	Go, Java	Kotlin
거버넌스	참여자들의 분산 처리	체인 내에 리눅스 재단과 조직	R3와 관련 기관들
스마트 계약	법적 제한 없음	법적 제한 없음	법적 제한
컨센서스 알고리즘	PoW, PoS	PBFT	다수 컨센서스 알고리즘 동작 서기(Notary) 노드
확장성	확장성 이슈 있음	강조되지 않음	강조되지 않음
프라이버시	프라이버시 이슈 있음	강조되지 않음	강조되지 않음
통화	이더(Ether)	없음(Chaincode 사용 생성 가능)	없음

JS Lab

A.III. 스마트 계약 (Smart Contract)

❖ Use Case: Healthcare Industry

- 각 환자는 유일하기 때문에 개인에 맞는 진찰을 선택하기 위해 의료 정보를 접속
- 의료 커뮤니티에서는 정보 공유가 주요 어려움
- 블록체인 기반의 스마트 계약으로 해결



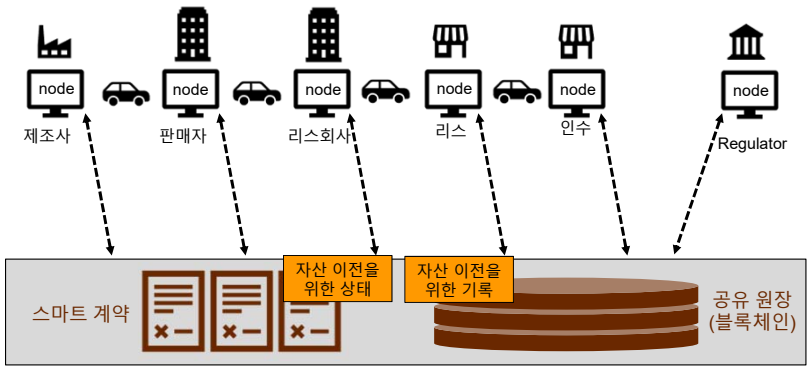
https://www.edureka.co/blog/smart-contracts/ https://medibloc.org/ko/

JS Lab

A.III. 스마트 계약 (Smart Contract)

❖ Use Case: 블록체인 기반 자동차 리스 비즈니스 네트워크

- 컨소시엄이 필요하고, 멀티클라우드 기반의 인프라가 요구됨
- 블록체인 상에서 개별 원장을 사용하는 복수의 채널을 가질 수 있음

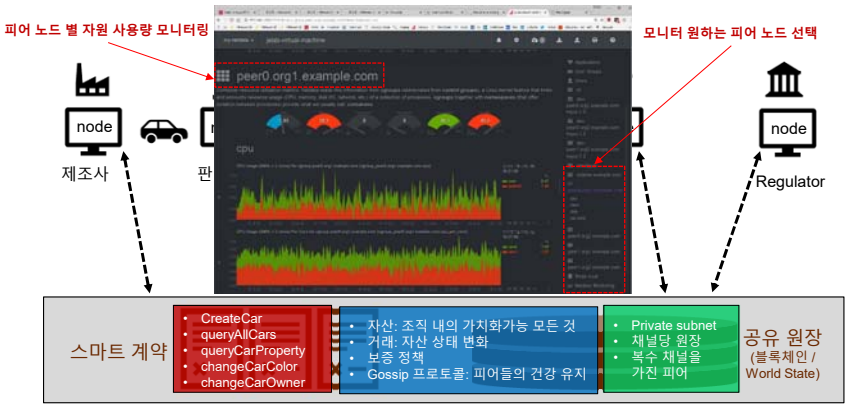


JS Lab

A.III. 스마트 계약 (Smart Contract)

❖ Use Case: 블록체인 기반 자동차 리스 비즈니스 네트워크

- 노드별 자원 사용 모니터링이 필요하며 확장 또는 이동이 필요 할 수 있다.

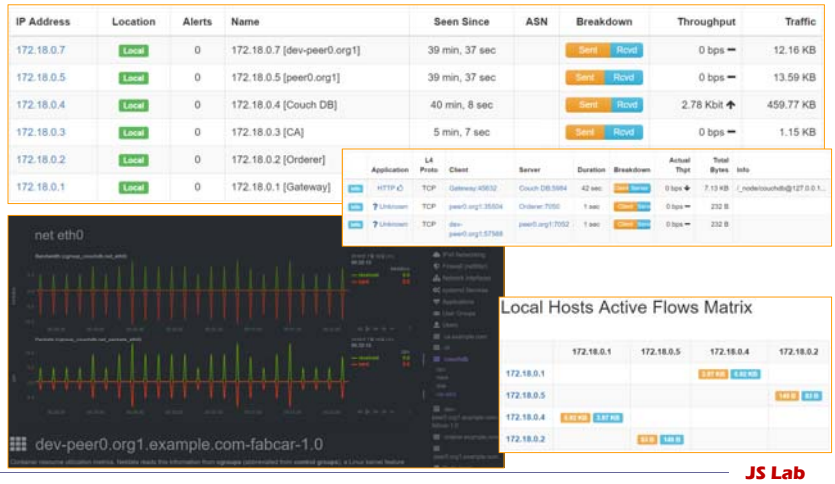


JS Lab

A.III. 스마트 계약 (Smart Contract)

❖ 불록체인 비즈니스 클러스터링 네트워크 모니터

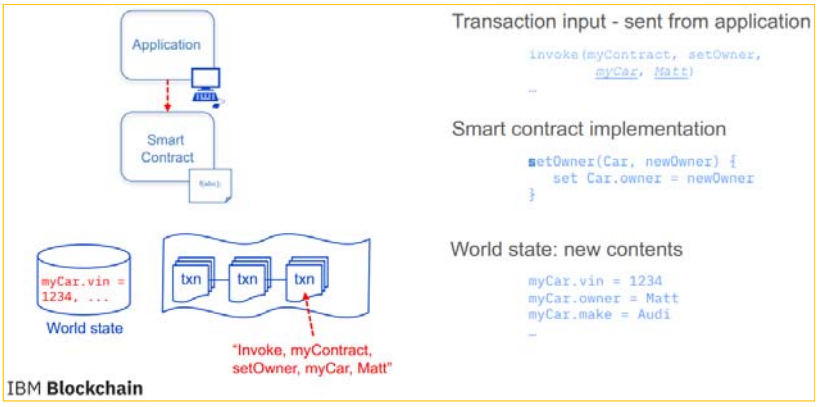
- 컨테이너 기반은 L4-L7 트래픽 플로우 모니터링 필요.



A.III. 스마트 계약 (Smart Contract)

❖ Use Case: 불록체인 기반 자동차 리스 비즈니스 네트워크

- 소유 변경 거래 (예)



A.III. 스마트 계약 (Smart Contract)

❖ 하이퍼레저(Hyperledger)의 스마트 계약(Smart contract):

- 하이퍼레저 패브릭(Hyperledger Fabric)은 Chaincode로 스마트계약 수행하며 Golang과 Javascript를 사용하고, Sawtooth는 파이손과 C등 더 다양한 언어를 사용 가능

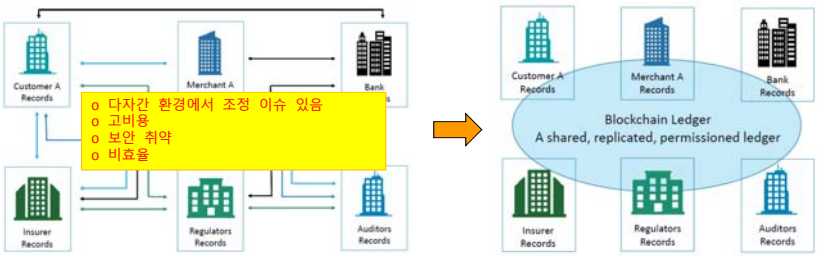
Smart Contract Implementations in Hyperledger Frameworks

Framework	Smart Contract Technology	Smart Contract Type	Language(s) for Writing Smart Contracts
Hyperledger Burrow	Smart contract application engine	On-Chain	Native language code
Hyperledger Fabric	Chaincode	Installed	Golang (> v1.0) or Javascript (> v1.1)
Hyperledger Indy	None	None	None
Hyperledger Iroha²	Chaincode	On-chain	Native language code
Hyperledger Sawtooth	Transaction families	On-Chain and Installed	C++, Go, Java, JavaScript, Python, Rust, or Solidity (through Seth)

A.IV. 프라이빗 블록체인

❖ **프라이빗 블록체인:** 중앙 통제 조직이나 대표자가 존재하고 허가된 참여자만 네트워크에 들어올 수 있어 참여자 간 식별이 가능하며 특화된 데이터 공유가 공유가 가능

- 거래 위임전에 참여자들 사이에 Consensus
- 거래 변조 확인하는 Immutability
- 자산 위치를 쉽게 추적하는 Provenance
- 진위 확인을 한곳에서 Finality



A.V. 컨소시엄 블록체인

❖ 블록체인 컨소시엄

- **Consortium Blockchain은 퍼블릭 블록체인과 프라이빗 블록체인의 중간 형태로 소유자가 모든 권한이 있는 프라이빗 블록체인과 달리 선정된 노드들이 권한을 가지는 블록체인**

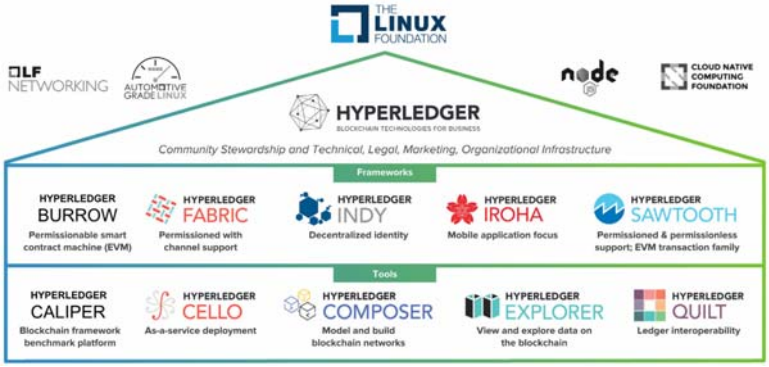
컨소시엄	참여기관	특징
R3 CEV	• 미국 IT 기업 R3사 설립 • 골드만삭스등 80여개 금융 기관 • 국내 은행(국민,신한,하나,우리)	• 분산 원장 플랫폼인 금융기관 계약 기록 관리 시스템(Corda) 개발
Hyperledger	• 리눅스재단 관리 • 100여개의 금융/비금융 기업 참여 • 국내기업(한국거래소,코스콤,SDS, 한국예탁결제원)	• 오픈소스 • 범 산업용 블록체인 플랫폼
SBI 핀테크 컨소시엄	• 일본 SBI 금융 그룹 주도 • 리플, 코인 플러그등 참여	• 오픈소스 • 범 산업용 블록체인 플랫폼
차이나레져	• 중국 완샹 블록체인 랩 주도 • 중국 11개 대형 금융기관 참여	• R3와 이더리움 재단 자문

JS Lab

A.V. 컨소시엄 블록체인

❖ Hyperledger Umbrella Strategy

- **Infrastructure:** Hyperledger에서 생태계를 위한 기술, 규제, 마케팅, 조직
- **Framework:** Fabric, Burrow, Indy, Iroha, Sawtooth
- **Tools:** Caliper, Cello, Composer, Explorer, Quilt



JS Lab

A.VI. 하이퍼레져(Hyperledger)

❖ Hyperledger

- 일반적으로 permissioned blockchain이며 네트워크 내에 허가된 참여자만 접속 가능
- 엔터프라이즈 급의 오픈소스로 분산원장프레임워크를 생성하는 것이 목표이고 비즈니스 Use Case를 지원하는 코드 기반
- 오픈소스 기반의 노력으로 향상된 산업간 블록체인 기술 생산에 노력
- 리눅스재단이 관리하며 다양한 사업과 기관들이 멤버
- 엔터프라이즈를 위해 준비된 솔루션을 알림
- 블록체인 프레임워크와 플랫폼을 만드는 소프트웨어 개발자를 위한 커뮤니티

A.VI. 하이퍼레져(Hyperledger)

❖ Permissioned Blockchain 장점

- Permissionless blockchains은 누구나 접속 가능하여 감염 의심자가 접속 할 수 있으나 Permissioned Blockchain는 이를 방지하여 보안을 강화 가능
- Hyperledger는 참여자가 원하는 부분의 거래만 공개 가능
- Hyperledger 아키텍처는 블록체인의 모든 기능을 제공
  - ✓ data privacy
  - ✓ information sharing
  - ✓ immutability, with a full stack of security protocols
  - ✓ all for the enterprise

A.VI. 하이퍼레져(Hyperledger)

❖ 주요 Hyperledger Projects

- **Hyperledger Fabric: supply-chain 네트워크에 사용**
- **Hyperledger Sawtooth: 인텔 기부, 어류 이동 추적 기반 등의 서비스체인(예)**
- **Hyperledger Burrow: 하이퍼레져 네트워크 내에서 Ethereum 스마트 계약 사용**
- **Hyperledger Iroha: 블록체인의 모바일 앱에 사용**
- **Hyperledger Indy: 비즈니스를 위한 ID 데이터베이스의 분산화**

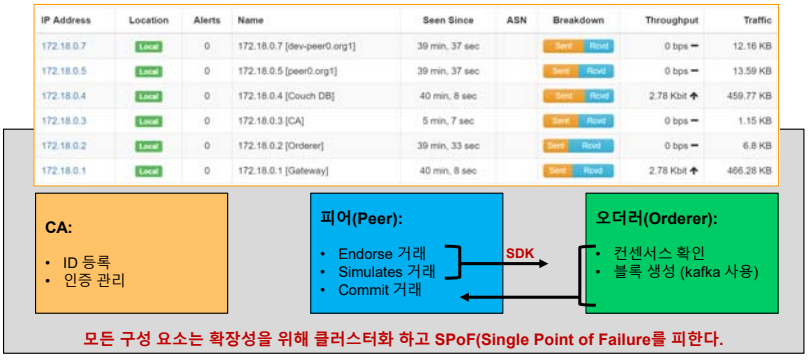
https://www.edureka.co/blog/what-is-hyperledger/

JS Lab

A.VI. 하이퍼레져(Hyperledger)

❖ Hyperledger Fabric Architecture

- **CA:** ID 등록 및 인증 관리
- **피어(Peer):** Endorse 거래, Simulates 거래, Commit 거래
- **오더러(Orderer):** 컨센서스 확인, 블록 생성 (kafka 사용)



JS Lab

A.VI. 하이퍼레져(Hyperledger)

❖ **Sawtooth 1.0**

- **On-chain governance –** Utilizing smart contracts to vote on blockchain configuration settings such as the allowed participants and smart contracts.
- **Advanced transaction execution engine –** Processing transactions in parallel to accelerate block creation and validation.
- **Support for Ethereum –** Running Solidity smart contracts and integrating with Ethereum tooling.
- **Dynamic consensus –** Upgrading or swapping the blockchain consensus protocol on the fly as networks grow, enabling the integration of more scalable algorithms as they are available.

- 인텔이 만들고 있는 프라이빗 블록체인 플랫폼인 Sawtooth LAKE의 경우 인텔 CPU에서만 생성할 수 있는 데이터를 가지고 합의하는 확률적 알고리즘인 PoET 합의 알고리즘을 사용

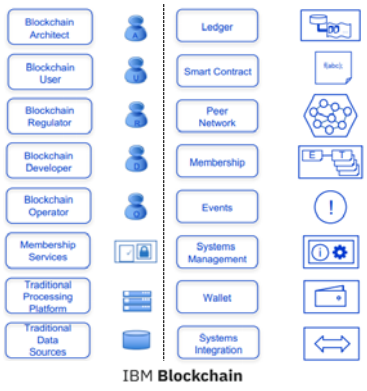
https://bitcoinmagazine.com/articles/hyperledger-releases-sawtooth-10/

A.VI. 하이퍼레져(Hyperledger)

❖ **솔루션 구성 요소 (IBM)**

- Actors
- Components

Actors



Components

A.VI. 하이퍼레져(Hyperledger)

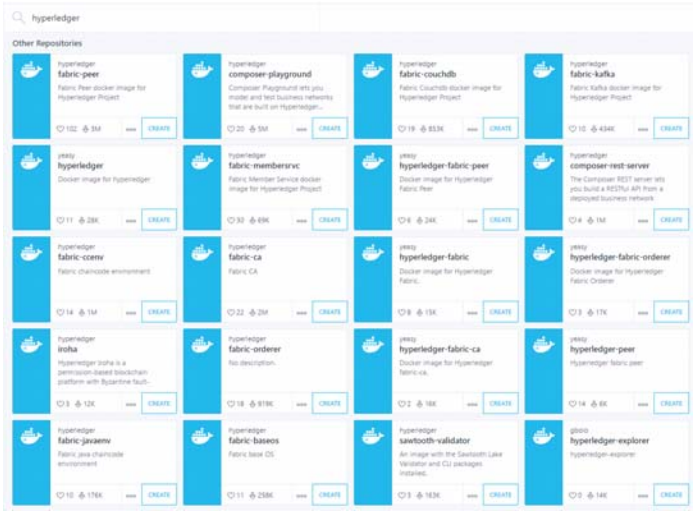
❖ 요약

- 비교 (Hyperledger : Bitcoin : Ethereum)

	Bitcoin	Ethereum	Hyperledger Frameworks
암호 화폐 기반	Yes	Yes	No
허가 (Permissioned)	No	No	Yes (in general)*
익명 사용	Yes	No	No
감사 가능	Yes	Yes	Yes
불변의 원장	Yes	Yes	Yes
모듈화	No	No	Yes
스마트 계약	No	Yes	Yes
컨센서스 프로토콜	PoW	PoW	Various**

A.VI. 하이퍼레져(Hyperledger)

❖ Docker Hub



A.VII. 컨센서스 (Consensus)

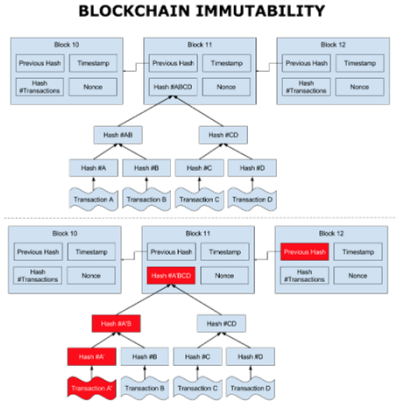
http://www.plays-in-business.com/consensus-decisioning-how-to-find-minimal-viable-decisions/

JS Lab

A.VII. 컨센서스 (Consensus)

❖ 컨센서스 알고리즘 (Consensus Protocol/ Consensus Mechanism)

- 클러스터링된 인프라의 오류 발생 환경에서 서비스 지속 가능한 알고리즘
- 정상 작동 노드가 과반수 이상이면 분산원장의 변경을 서비스를 지속 가능



Merkle Tree

- 바이너리 해시 트리
- 각 데이터의 해시를 저장하는데 사용하는 데이터 구조

거래별 관련 해시

- 전(前) 트랜젝션 해시
- 동일 블록 내의 거래 모음 해시
- 자신의 거래 해시

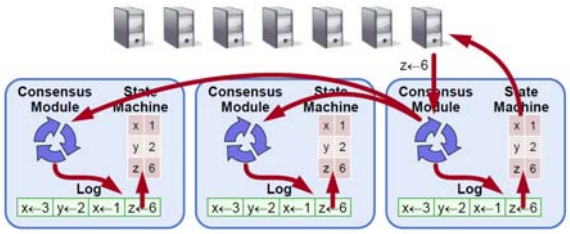
JS Lab

A.VII. 컨센서스 (Consensus)

❖ **Consensus Algorithms**

- 분산 시스템에 참여하고 있는 모든 노드가 같은 결과 값을 결정해야 함
- 합의 문제는 분산 시스템의 신뢰도를 보장하기 위해 나온 개념으로 블록체인이 나오기 전부터 존재
- 결정된 데이터는 특정 Process에 의해 제안된 것이어야 함
- 모든 시스템의 상태는 0이나 1로 결정 (모두 1인지 0인지 판단 할 수 있어야 함)
- 악의적인 노드가 존재하더라도 신뢰도 있는 시스템 제공 보장

Raft 알고리즘

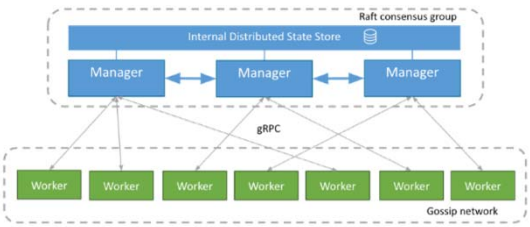


- Leader Election
- Log Replication
- Safety

A.VII. 컨센서스 (Consensus)

❖ 스웜모드 아키텍처 토폴로지
❖ 스웜(Swarm) Manager 는 3개 or 5개 or 7개등 홀수 권장

스웜 매니저간들의 통신은 항상 쿼럼(Quorum) 가능



- /var/lib/docker/swarm/docker.state.json

{"LocalAddr":"","RemoteAddr":"192.168.99.118:2377","ListenAddr":"0.0.0.0:2377","AdvertiseAddr":""}

- /var/lib/docker/swarm/state.json

[{"node_id":"9c5eqant0s2w7arlfk47tkxm0","addr":"192.168.99.118:2377"},{"node_id":"bexym9a2cxbd60ow40xibycw5","addr":"192.168.99.115:2377"},{"node_id":"cviejn6myjln6s4ysw4wg59rn","addr":"192.168.99.119:2377"}]

'docker info'의 swarm 정보 표시(예)

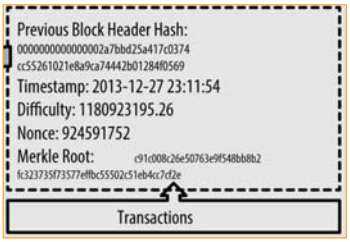
```
Swarm: active
 NodeID: kj4l7bnzmr8yphsmln33zs7a8
 Is Manager: true
 ClusterID: tzzk07zwhg8axv0jayf2u58yd
 Managers: 3
 Nodes: 8
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 3
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
  External CAs:
    cfssl: https://192.168.99.131:12381/api/v1/cfssl/sign
    cfssl: https://192.168.99.129:12381/api/v1/cfssl/sign
    cfssl: https://192.168.99.130:12381/api/v1/cfssl/sign
 Node Address: 192.168.99.129
 Manager Addresses:
  192.168.99.129:2377
  192.168.99.130:2377
  192.168.99.131:2377
```

A.VII. 컨센서스 (Consensus)

❖ Proof of Work (PoW)

- "작업 증명"(Proof of Work, POW)이라는 용어는 1999년 Markus Jakobsson와 Ari Juels의 논문에서 의해 처음 만들어져 공식화
- 작업을 요구함으로써 서비스 거부(DoS) 공격과 기타 서비스 악용(예: 네트워크 상의 스팸)을 단념하게 만들기 위한 경제적인 수단
- 마이닝(mining) 프로세스로 알려져 있으며 증명(Proof)은 어렵게 생성 되지만 검증(Verify) 은 쉽게 함
- 마이닝 성공을 위한 인센티브를제공
- PoW consensus algorithm의 에너지 과다 사용에 대한 비판 시각이 있음.

비트코인 블록 구조
(출처 : mastering bitcoin)



5,000,000 TH/s(1 TH/s = 초당1,000,000,000,000번의 해시연산) 이상의 해시 파워가 필요

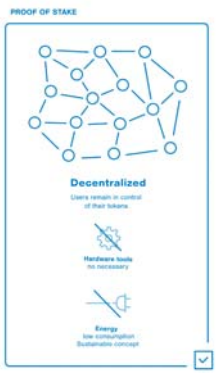
A.VII. 컨센서스 (Consensus)

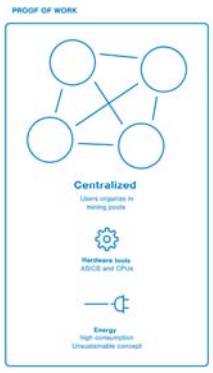
❖ Proof of Stake (PoS)

- 컴퓨팅 파워 낭비가 아닌 자신이 가진 지분(stake)을 통해 블록을 생성
- 자신이 가지고 있는 지분(Stake)과 지분이 생성된 날짜에 의해 결정되며, 한번 블록 생성을 위해 사용된 지분의 날짜는 초기화
- 합의 노드가 변조등 악의적인 행동을 했을 경우 프로토콜에 의해 몰수당하거나 '슬래싱(slashing)' 될 수 있음
- PoW에서는 합의 노드를 채굴자(miner)라고 하지만 PoS에서는 합의 노드를 검증인(validator)이라고 함
- 검증인의 투표권(voting power) 내지 가중치(weight)는 보유 지분 또는 위임된 지분의 총량과 비례
- PoS algorithm은 컴퓨팅 자원을 절약

A.VII. 컨센서스 (Consensus)

❖ **Proof of Work(PoW)와 Proof of Stake (PoS)**





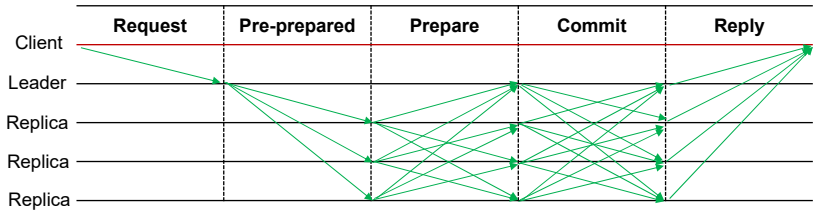
https://hackernoon.com/consensus-mechanisms-explained-pow-vs-pos-89951c66ae10

JS Lab

A.VII. 컨센서스 (Consensus)

❖ **Practical Byzantine Fault-tolerance (PBFT)**

- 분산시스템이 약속된 행동을 하지 않는 비잔틴 노드가 존재할 수 있는 비동기 시스템일 때 해당 분산시스템에 참여한 모든 노드가 성공적으로 합의를 이룰 수 있도록 개발된 합의 알고리즘
  1) 리더가 클라이언트들의 요청을 수집/정렬하고 실행 결과와 함께 다른 노드들에 전파
  2) 메시지를 받은 노드들은 다른 노드들에서 받은 메시지를 다시 나머지 노드들에게 전파
  3) 모든 노드는 자신이 다른 노드에서 가장 많이 받은 같은 메시지(정족수 이상의)가 무엇인지 다른 노드들에게 전파
  4) 1) 2) 3)의 과정이 끝나면 모든 노드들은 정족수 이상이 동의한, 즉 합의를 이룬 같은 데이터를 가짐



https://theintelligenceofinformation.wordpress.com/2017/02/15/practical-byzantine-fault-tolerance-consensus-talk-by-miguel-castro/

JS Lab

A.VII. 컨센서스 (Consensus)

❖ 비교

- Consensus 는 여러가지가 있으며 네트워크의 요구에 따라 목표 기능을 구현

Types of Blockchain	Example of Blockchain	Consensus Algorithm	Nature
Public Blockchains	Bitcoin, Ethereum, Litcoin etc	PoW, PoS, DPoS	Open and decentralized
Federated Blockchains	R3, B3I, EWF	No	Controlled and Restricted
Private Blockchains	Company Internal	PBFT, RAFT	Restricted
Permisioned Blockchain	Hyperledger, Ripple	PBFT	Closed and restrcited

Characteristics	PoW	PoS	DPos	PBFT	RAFT
Byzantine Fault Tolerance	50%	50%	50%	33%	N/A
Crash Fault Tolerance	50%	50%	50%	33%	50%
Verification Speed	>100s	<100s	<100s	<10s	<10s
Throughput(TPS)	<100	<1000	<1000	<2000	>10k
Scalability	strong	strong	strong	weak	weak

https://medium.com/coinmonks/know-which-blockchain-or-dlt-platform-works-well-within-your-usecase-comparison-of-different-a8dc34782af3

Community for KOREN AI Network Lab
james@jslab.kr
JS Lab

A. 블록체인 인프라

B. 컨테이너

- **I. 개요**
- **II. 시장**
- **III. 기술 동향**
- **IV. 기술**
- **V. 도커 스웜(Swarm)**
- **VI. 도커 보안(Security)**
- **VII. Docker Datacenter**
- **VIII. 응용**
- **IX. 도커 네트워킹(Networking)**

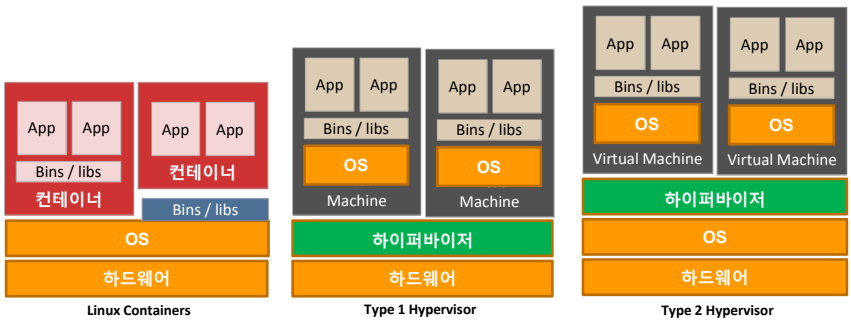
C. 실습 (별도 교재)

I. 개요
II. 시장
III. 기술 동향
IV. 기술
V. 도커 스웜(Swarm)
VI. 도커 보안(Security)
VII. Docker Datacenter
VIII. 응용
IX. 도커 네트워킹(Networking)
부록: Kubernetes (K8s)

I. 개요

❖ **컨테이너:** 동시에 복수개의 루트파일 시스템을 실행하는 호스트 리눅스커널을 사용하며, 각각의 루트파일 시스템을 컨테이너(Container)라고 함

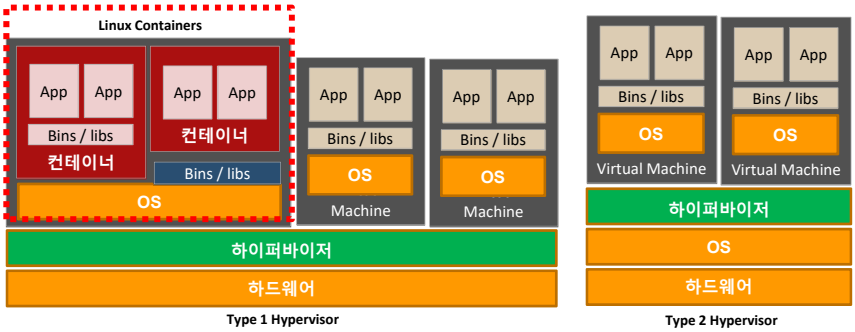
- 가상머신보다 적은 자원 사용
- 스케일링을 가능하게 해주는 관리 도구 포함



JS Lab

I. 개요

❖ 실습 구성은 VM을 컨테이너의 호스트로 사용



JS Lab

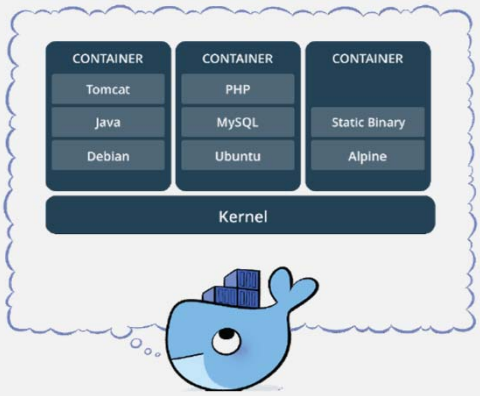
I. 개요

❖ 컨테이너 플랫폼

Docker	rkt
docker pull	rkt fetch
docker push	N/A
docker commit	Acbuild
docker log	N/A
docker run	rkt run
docker start/stop	rkt stop

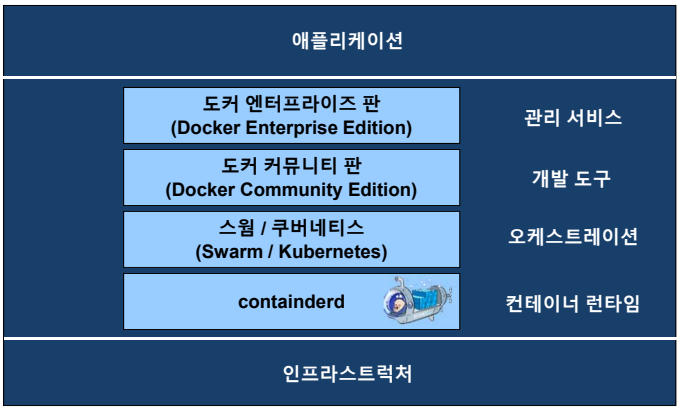
I. 개요

❖ **도커(Docker) 컨테이너:** 도커가 컨테이너를 실현하기 위해 리눅스 컨테이너 (LXC)를 사용하였지만, 도커 0.9 버전 부터 직접 만든 자체 컨테이너를 사용



I. 개요

❖ Containerd (2016)는 gRPC를 사용하여 컨테이너 기능 외부 노출
❖ 도커는 이미지(Image), 볼륨(Volume), 네트워크(Network), 빌드(Build) 외부 연결

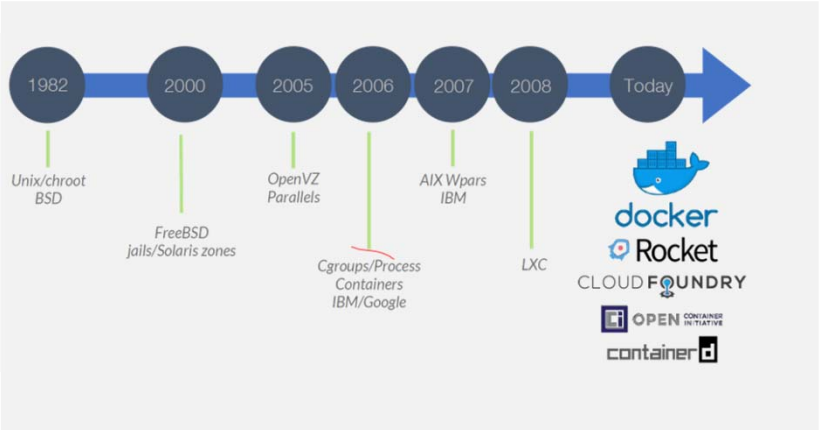


RunC (2015)

도커(Docker)와 런타임 사이의 추상화로 다른 벤더에서 독자적 컨테이너 버전을 가질 수 있음

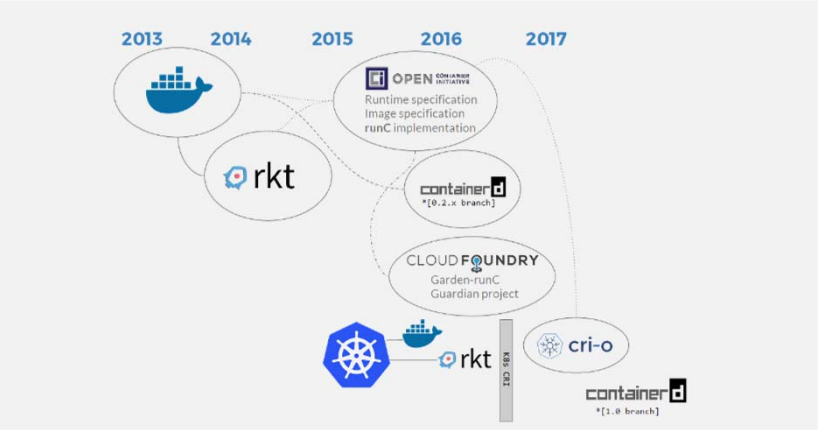
I. 개요

❖ 도커(Docker) 이전



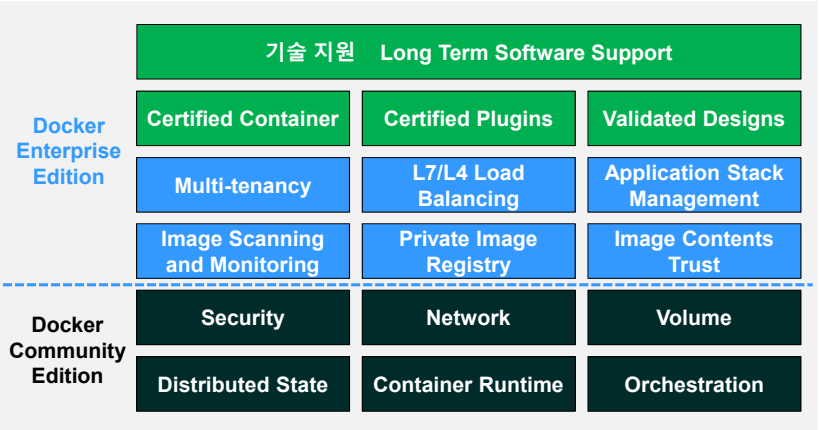
I. 개요

❖ 도커(Docker) 이후



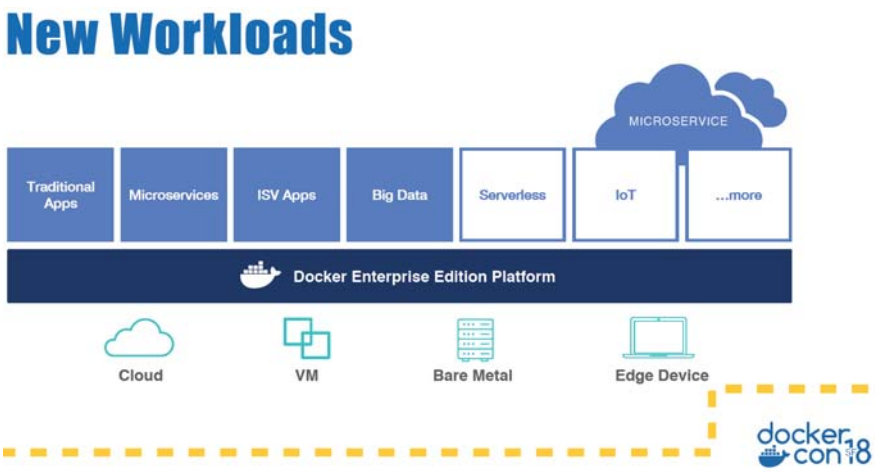
I. 개요

❖ **도커(Docker)의 발전:** 2013 부터 공개를 시작해서 2017년 엔터프라이즈 용을 별도 판매



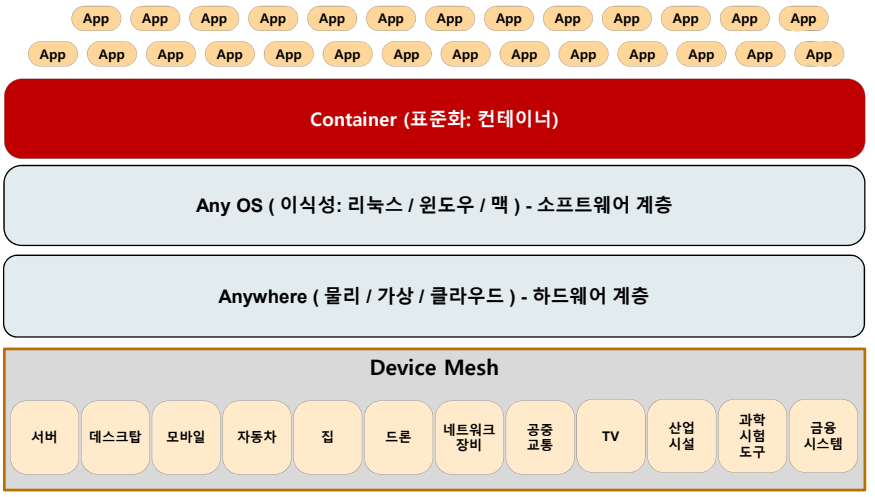
I. 개요

❖ Workload



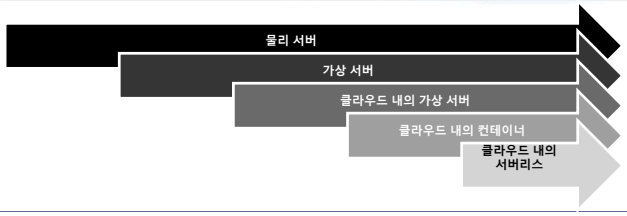
I. 개요

❖ 요약



II. 시장

❖ 클라우드 환경의 컨테이너



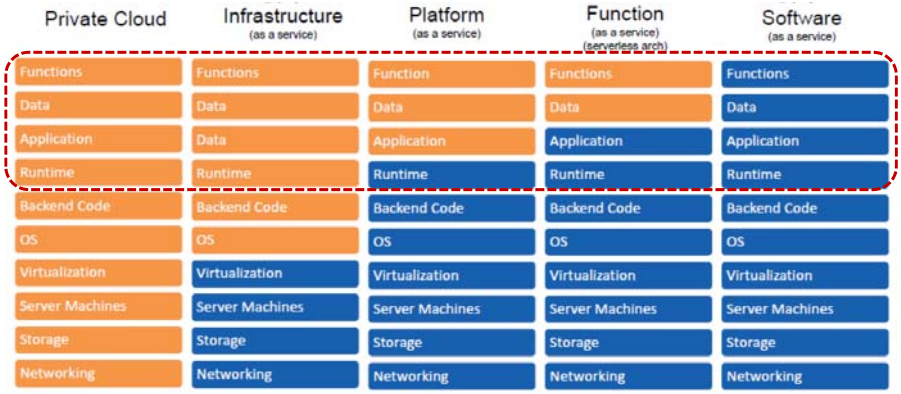
II. 시장

❖ 클라우드 서비스 비교

① **Private Cloud**

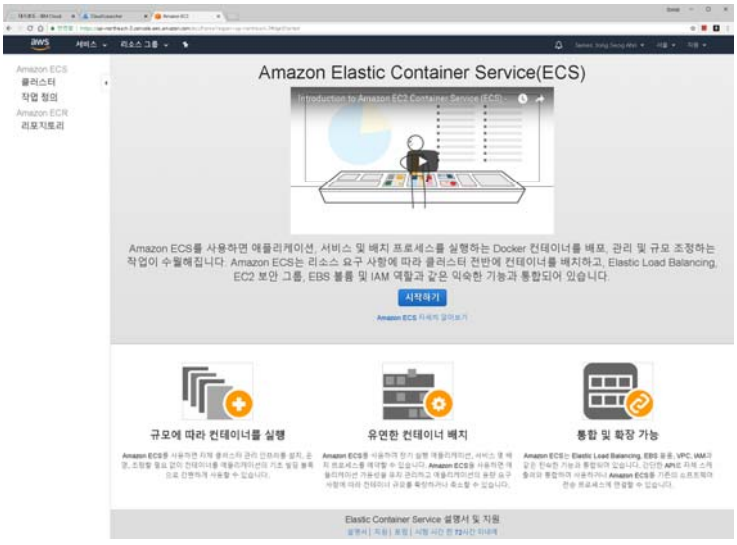
② **Public Cloud:** IaaS, PaaS, FaaS(Serverless), SaaS

③ **Hybrid Cloud, Multi Cloud**



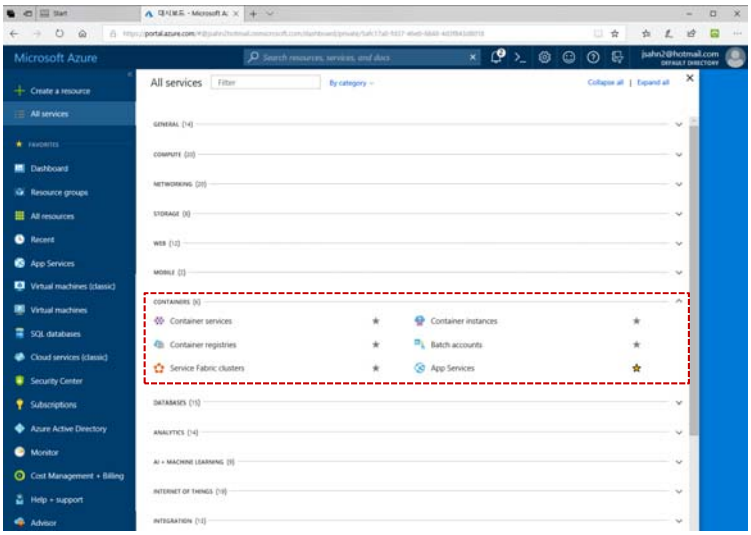
II. 시장

❖ 아마존 AWS



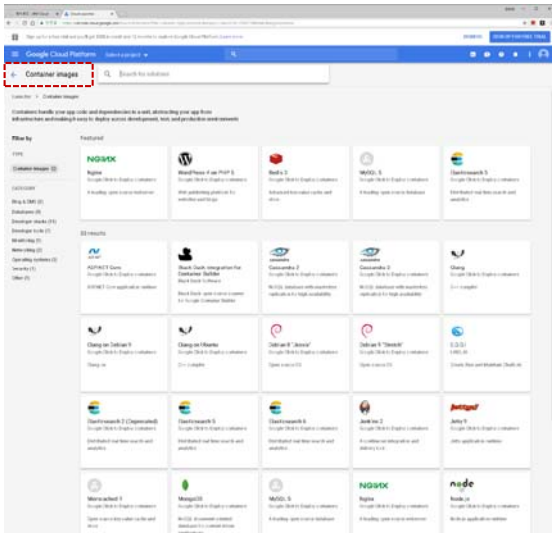
II. 시장

❖ 마이크로소프트 Azure



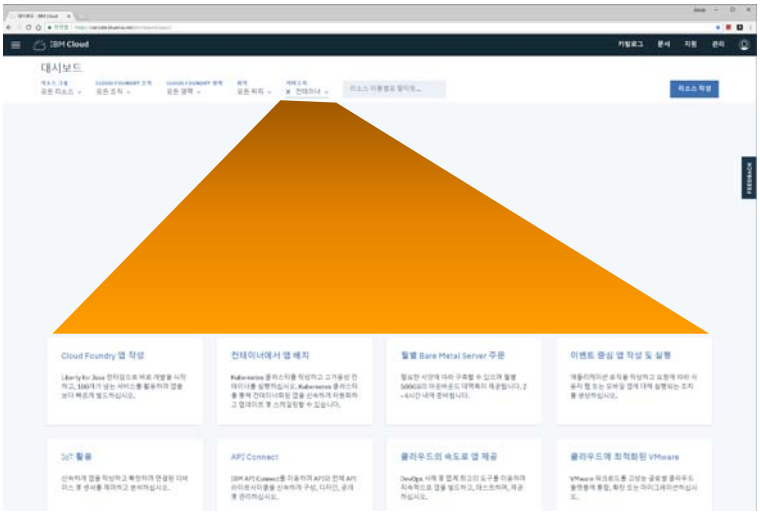
II. 시장

❖ Google Cloud Platform



II. 시장

❖ IBM Cloud



Community for KOREN AI Network Lab
jamesj@jslab.kr

JS Lab

III. 기술 동향

❖ OCI (Open Container Initiative)

- 오픈 컨테이너 프로젝트(OCP)-> 오픈 컨테이너 이니셔티브(OCI)

OPEN CONTAINER INITIATIVE

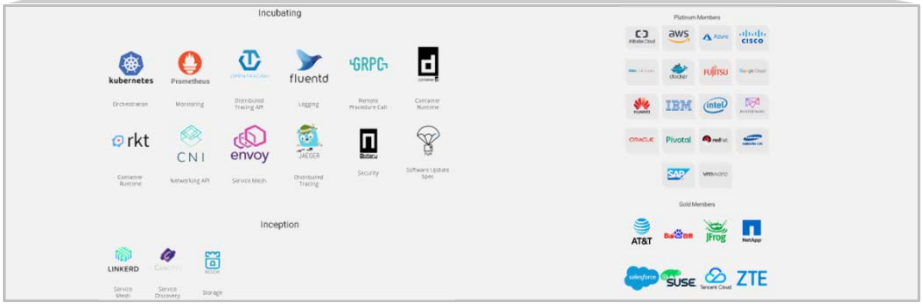
JS Lab

III. 기술 동향

❖ 리눅스 이상의 리눅스 재단(LINUX Foundation)

Security Networking Cloud Automotive Blockchain Web

❖ CNCF (Cloud Native Computing Foundation)



JS Lab

https://www.slideshare.net/JamesAhn/

III. 기술 동향

❖ 클라우드 지원 관리

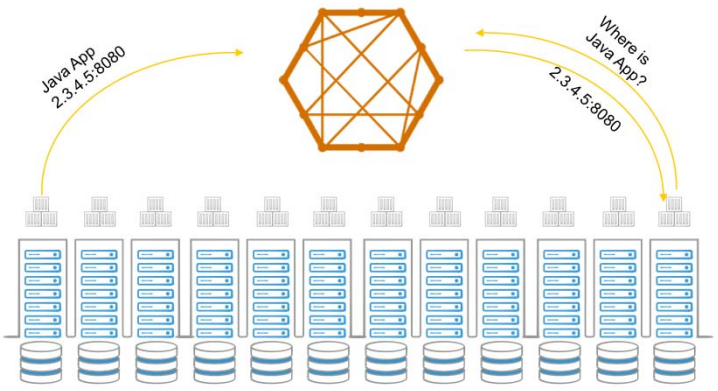
	클라우드 관리 플랫폼 (Infrastructure First)	애플리케이션 플랫폼 (Application First Approach)	자동화/오케스트레이션 플랫폼 (Automation First)
장점	• Single Pane of Glass • 비용 분석과 제어	• 쿠버네티스(Kubernetes or K8s)와 컨테이너는 주요 클라우드에서 지원 • 애플리케이션과 마이크로서비스 중심	• 퍼블릭/프라이빗 클라우드등 다양하고 폭넓은 지원 • 모든 클라우드나 애플리케이션 지원 구조 • 커스터마이징 가능
단점	• 제한적인 애플리케이션 인식 • DevOps프로세스에 부적합 • 제한적인 클라우드 서비스	• 대부분 그린필드에 적합 • 다른 영역에 호환성 부족	• 애플리케이션과 클라우드 단위로 커스터마이징 필요
	RED HAT CLOUDFORMS, RIGHTSCALE CLOUD MANAGEMENT, IBM, vmware, MORPHEUS, SCALR	OPENSHIFT, RANCHER, CLOUD FOUNDRY, kubernetes	TERRAFORM, CLOUDIFY, ARIA, OASIS TOSCA

JS Lab

III. 기술 동향

❖ 서비스 디스커버리 (Service Discovery)

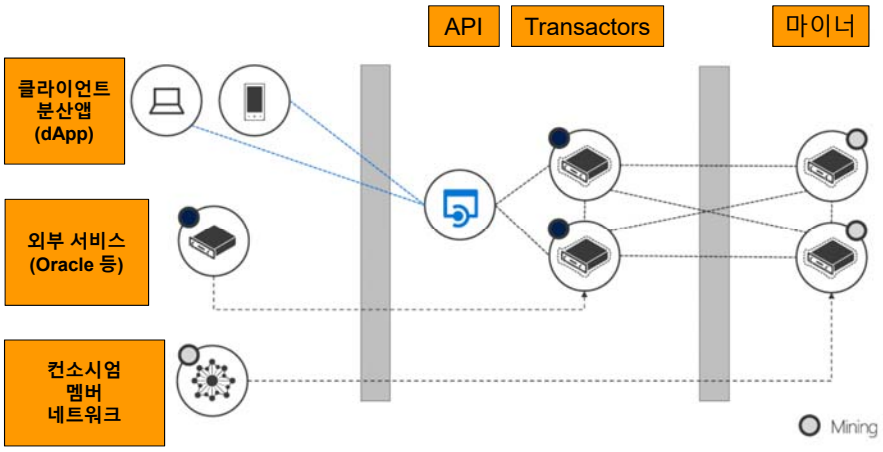
- 앱들과 인프라가 자동으로 서로의 필요 부분을 찾아 연결
- 애플리케이션 서비스 트래킹을 지속하여 연결하여 사용



JS Lab

III. 기술 동향

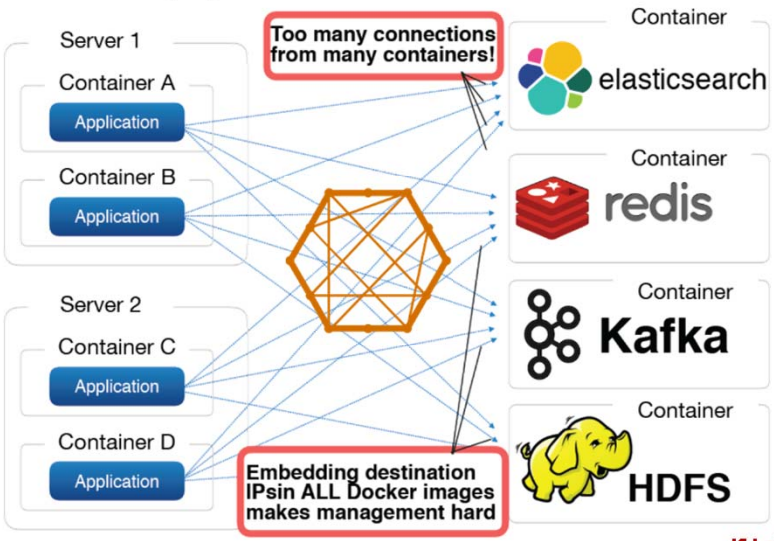
❖ 블록체인 토폴로지



Microsoft JS Lab

III. 기술 동향

❖ 컨테이너 기반 블록체인 인프라 적용 (예)

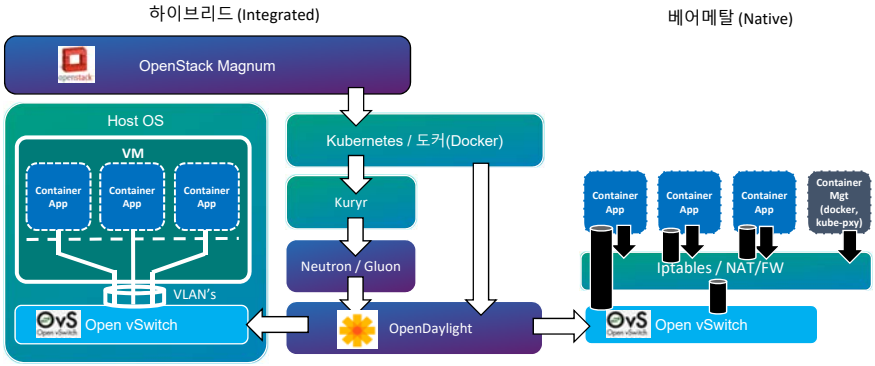


JS Lab

III. 기술 동향

❖ ODL의 컨테이너 네트워킹을 위한 2가지 적용 방법

- 하이브리드
- 베어메탈

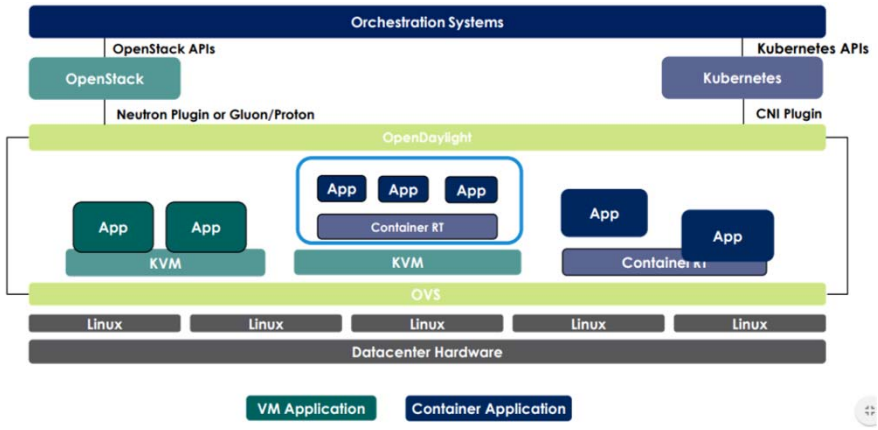


JS Lab

III. 기술 동향

❖ VNF(Virtual Network Function) 적용 시나리오

- 가상머신 오케스트레이션은 오픈스택(OpenStack) 이용
- 컨테이너 오케스트레이션은 Google의 쿠버네티스(K8S) 이용

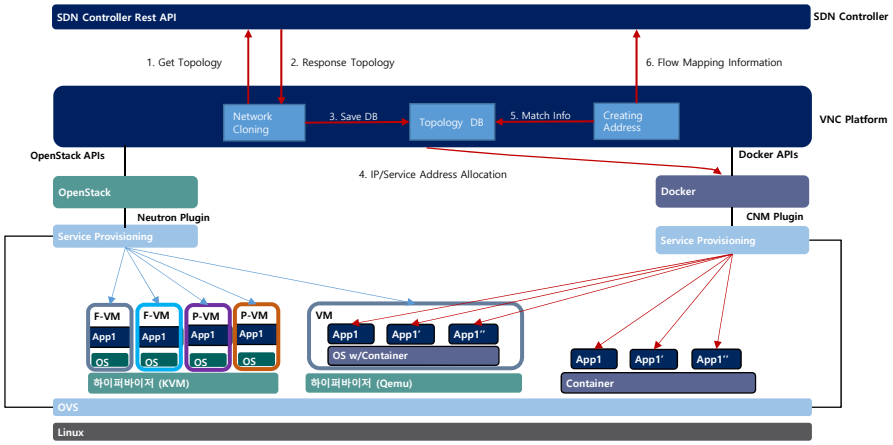


JS Lab

III. 기술 동향

❖ 가상 네트워크 클로닝 (VNC)

- **VNC 속도 개선을 위한 구조** (주소 매핑 테이블 생성/관리)
- **자원 절약과 클로닝 확장성 제공**

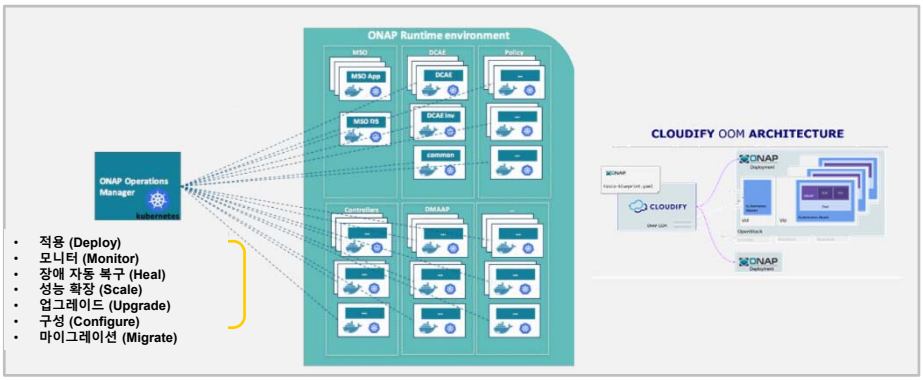


JS Lab

III. 기술 동향

❖ ONAP Operation Manager (OOM)

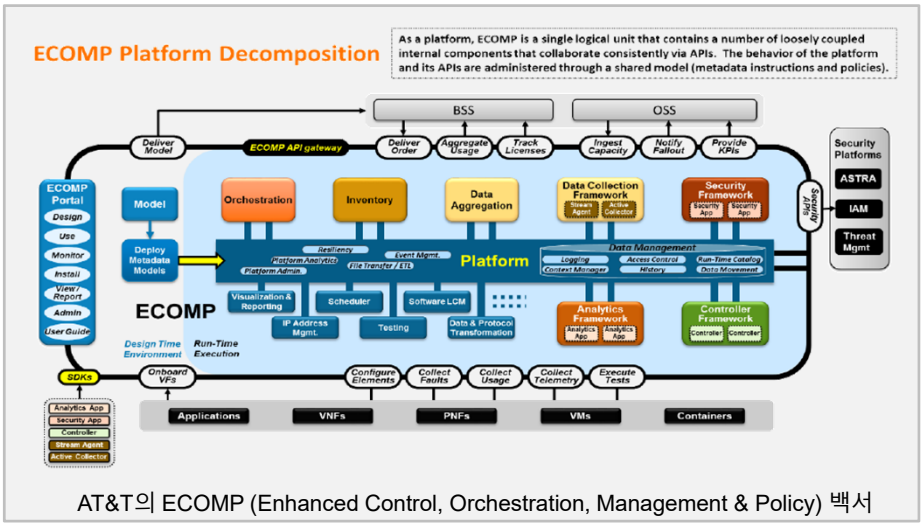
1. **리눅스재단 프로젝트:** AT&T에서 제안한 ECOMP (Enhanced Control, Orchestration, Management & Policy)
2. **도커(Docker)와 쿠버네티스(Kubernetes) 사용 수십개의 마이크로서비스 구성**
3. **멀티스택 / 멀티클라우드**



JS Lab

III. 기술 동향

❖ ONAP Operation Manager (OOM)을 위한 ECOMP 프레임워크

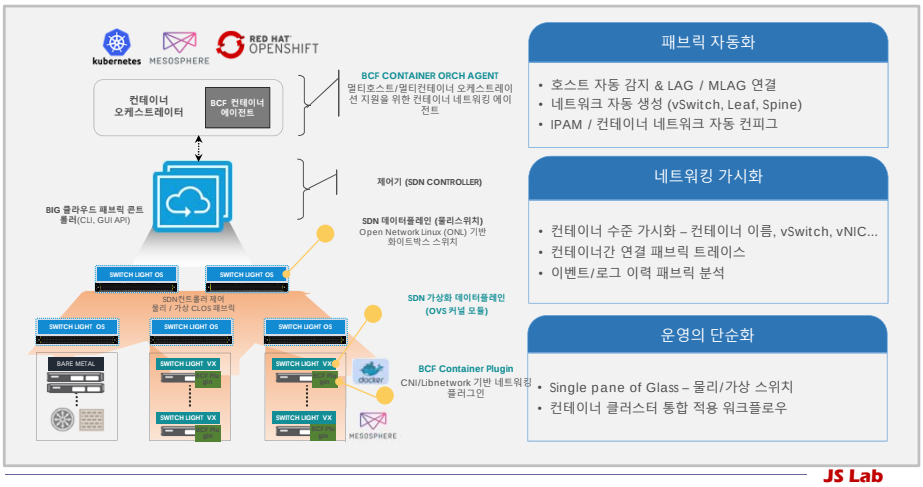


AT&T의 ECOMP (Enhanced Control, Orchestration, Management & Policy) 백서

JS Lab

III. 기술 동향

❖ BigSwitch/오픈스택(OpenStack)의 뉴트론(Neutron)연동 (예)
❖ 분산라우팅, Heat, LBaaS, 방화벽, VM-to-VM 경로/정책 가시화
❖ 프라이빗 클라우드를 위한 하드웨어 플랫폼



JS Lab

III. 기술 동향

❖ **요약** (엔터프라이즈 운영)

- **SLA 유지**
- **온프레미스(On-premise)와 클라우드 간의 통합관리**
- **상호 운영 가능한 클라우드간 경유 적용**
- **비용 절감**
- **실시간 프로비젼(Provision)과 스케일(Scale)**

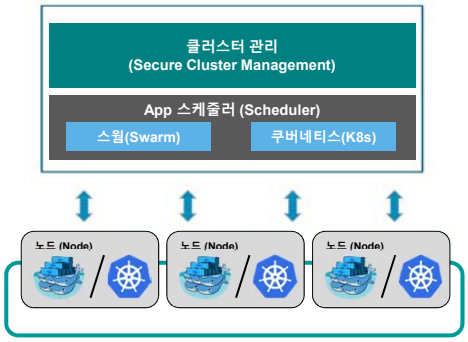
JS Lab

Community for KOREN AI Network Lab
jamesjslab.kr

IV. 기술

❖ **Docker**

- 개발자는 오케스트레이터를 선택 할 필요 없음
- 필요시 오케스트레이터를 변경 가능함
- EE Manager 노드(Node)는 스웜(Swarm)과 쿠버네티스(K8s) 동시 활성화 가능
- 모든 Worker노드(Node)는 Kubernetes API와 Swarm API 동시 Ready 가능

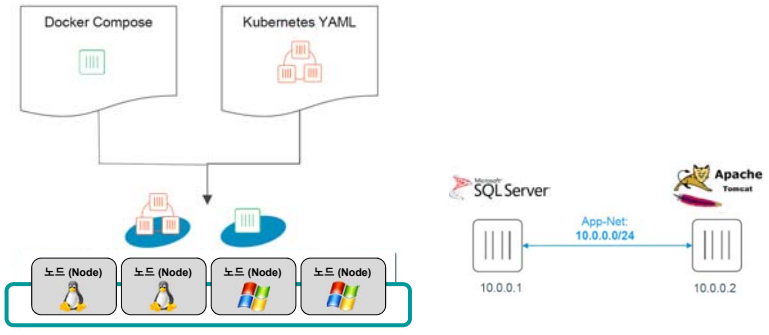


도커(Docker) EE 클러스터

IV. 기술

❖ 도커 컨테이너 플랫폼은 리눅스와 윈도우의 호환성을 제공

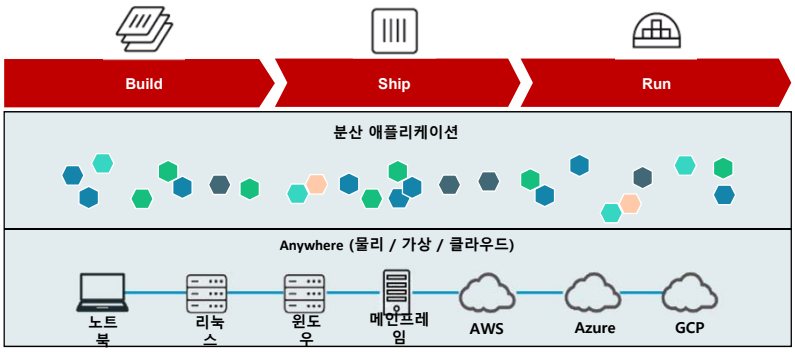
- 윈도우와 리눅스의 기술 보존 수단
- 동일한 클러스터와 오버레이 네트워크에서 컨테이너 간 연결
- 지능적인 위치 지정과 스케줄링을 위한 Label과 Constraints를 사용



IV. 기술

❖ **Docker Mission**

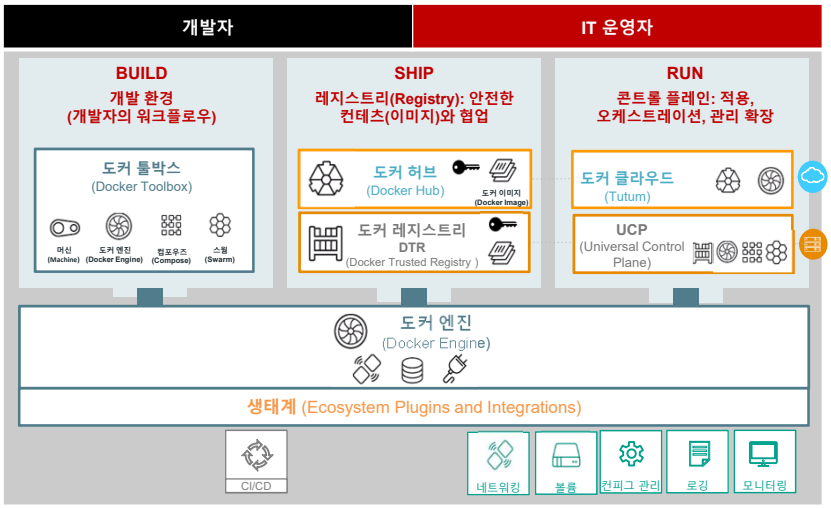
- **Build :** 도커 개발 환경에서 이미지를 만들고,
- **Ship :** DTR 중심으로 협업하며 이미지들을 탑재하여,
- **Run :** 오케스트레이션 등을 통한 실행과 이의 관리



JS Lab

IV. 기술

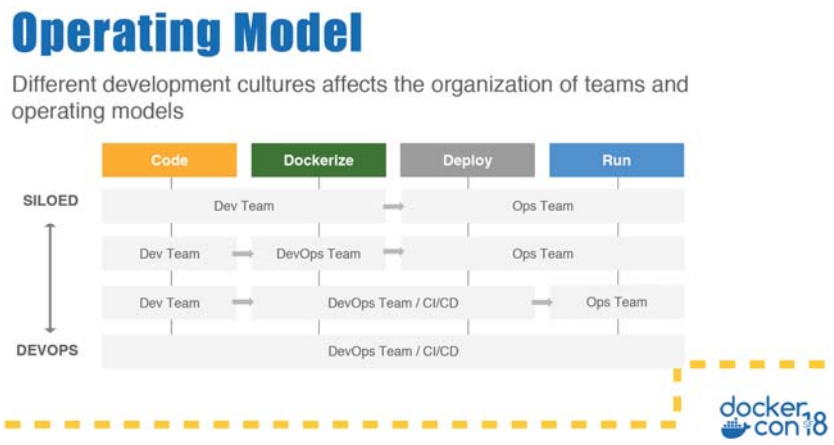
❖ **워크 플로우(Workflow)**



JS Lab

IV. 기술

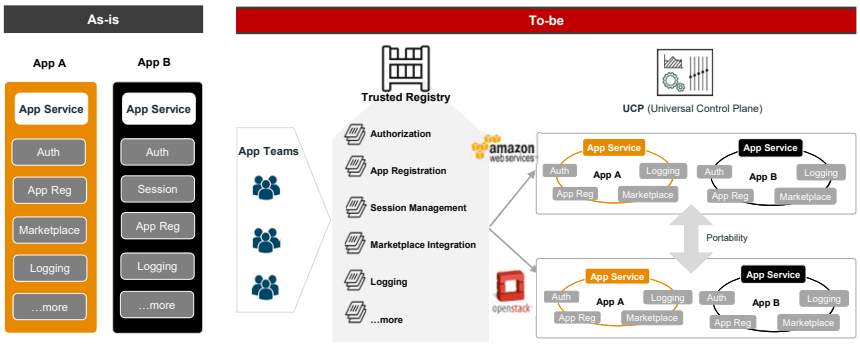
❖ **Operating Model**



IV. 기술

❖ **도커의 As-is To-be**

- 하이브리드 오케스트레이션 / 통일된 소프트웨어 Supply Chain / 인프라 독립적
- 운영자는 운영에 집중 (클라우드 기반 적용 vs. 온프레미스 기반 적용)



IV. 기술

❖ **Docker Editions:** Open Source (LinuxKit, Moby Project), CE, EE

JS Lab

IV. 기술

❖ **엔터프라이즈용 Docker EE 비용** (Linux)

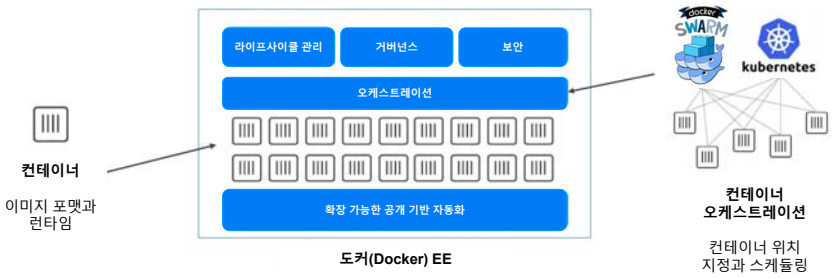
	EE BASIC SUPPORTED CONTAINER RUNTIME	EE STANDARD ENTERPRISE MANAGEMENT FEATURES FOR SINGLE TEAM USE	EE ADVANCED ADDITIONAL CAPABILITIES FOR MULTI-TEAM USE, POLICY-BASED AUTOMATION
Secure container engine with networking, security, and storage	✓	✓	✓
Docker Certified infrastructure, plugins and containers	✓	✓	✓
Private image registry with caching		✓	✓
Integrated app and cluster management across Swarm and Kubernetes		✓	✓
Enhanced RBAC, LDAP / AD support		✓	✓
Integrated secrets management, image signing policy		✓	✓
Secure multi-tenancy with node-based isolation			✓
Policy-based, automated image promotions			✓
Image mirroring across registries			✓
Image security scanning and continuous vulnerability scanning			✓

https://www.docker.com/pricing

JS Lab

IV. 기술

❖ **Docker EE는 컨테이너 오케스트레이션 제공**
❖ **라이프싸이클 관리, 거버넌스, 보안, 자동화, 지원**



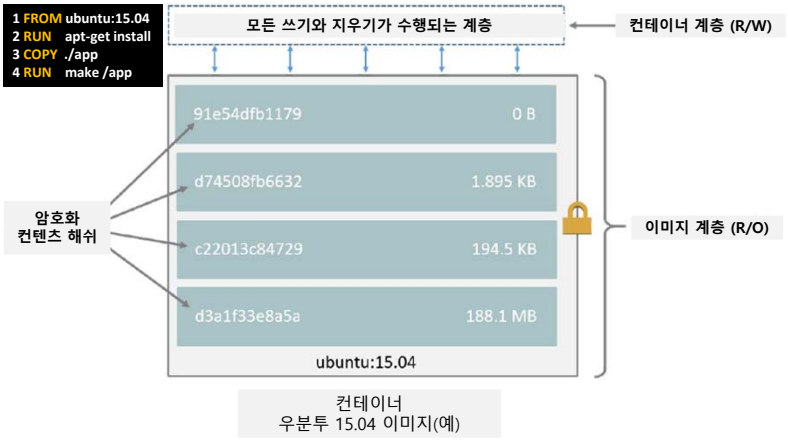
IV. 기술

❖ **'containerd'의 분리 공개 후 생태계** (CNCF내의 TOC 프로젝트로 소스 기부)

- 분리한 containerd는 버전 1.0을 2017년 Q2 발표 예정
- containerd는 로드맵에 CNI를 네트워킹에 포함
  https://github.com/docker/containerd/issues/362
- Cloud Native Computing Foundation (CNCF) 내의 Technical Oversight Committee (TOC)의 프로젝트로 'containerd' 소스 기부
- 향후 'containerd' 지원 오케스트레이션을 위한 벤더 선택 가능 예상

IV. 기술

❖ **도커 이미지 생성:** Build한 이미지(Image)는 컨테이너 실행 시 읽기 전용으로 사용하며 컨테이너에서 생성한 계층에서 쓰기와 읽기를 실행

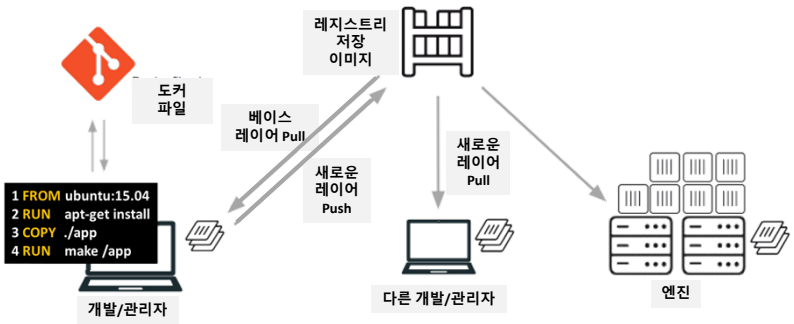


컨테이너
우분투 15.04 이미지(예)

JS Lab

IV. 기술

❖ **도커 이미지 탑재:** Build한 이미지(Image)는 컨테이너 실행 시 읽기 전용으로 사용하며 컨테이너에서 생성한 계층에서 쓰기와 읽기를 실행




```
$ docker build -t reg.corp.com/appteam/corp:1.4
$ docker push -t reg.corp.com/appteam/corp:1.4
```

JS Lab

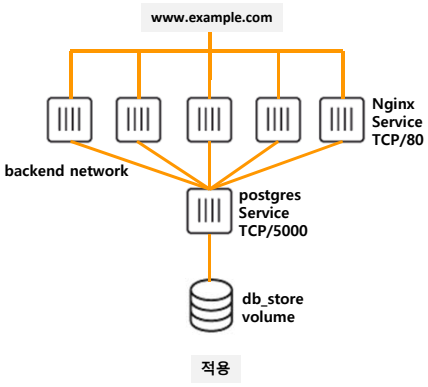
IV. 기술

❖ **도커 서비스:** Compose / Stack 파일로 서비스 정의

```
version: '3'

services:
 web:
  image: nginx
  ports:
        - 80:80
  network:
        - frontend
  deploy:
   replicas: 5
database:
  image: postgres
  port:
        - 5000
  network:
        - backend
  volume:
        - db_store
```

stack.yml

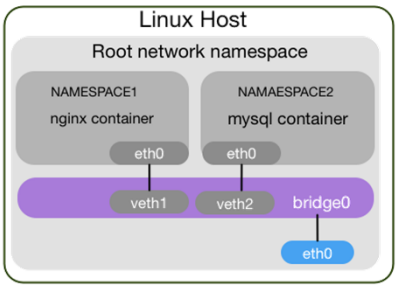


적용

IV. 기술

❖ **컨테이너 네트워킹(Container Networking) 종류**

- **Container Network Model (CNM):** Docker libnetwork에서 사용하며 Cisco Contiv / Kuryr, OVN, Project Calico / VMware / Vwave에서 사용
- **Container Network Interface (CNI):** CoreOS에서 사용하며 kuberntes / Kurma / rkt / Apache Mesos / Cloud Foundry / Cisco Contiv / Project Calico / Weave



CNI 의 주요 플러그인은 Main과 IPAM;

1) **Main 플러그인:** Network 네임스페이스로 'veth pair'를 생성하여 내부에서 컨테이너와 브릿지등을 연결

2) **IPAM(ip management) 플러그인:** IP세팅과 할당

- https://github.com/containernetworking/cni/blob/master/scripts/docker-run.sh#L8-L20
- http://murat1985.github.io/kubernetes/cni/2016/05/14/netns-and-cni.html

IV. 기술

❖ 컨테이너 네트워킹(Container Networking) 종류

- None: 호스트간 연결 없음
- 브릿지(Bridge): L2 브릿지를 사용
- 오버레이(Overlay): 터널링 사용 오버레이로 호스트 간 네트워크 연결
- 언더레이(Underlay): 컨테이너를 물리적 인터페이스에 직접 연결

	오버레이	브릿지 / 포트맵핑	언더레이
멀티 호스트 연결	Yes	No (native support)	No (native support)
서비스 발견 (Service Discovery)	클러스터 간의 글로벌 SD	호스트 네트워크 상의 로컬 SD	호스트 네트워크 상의 로컬 SD
로드밸런싱	-내부 글로벌 VIP 기반 -내부 글로벌 DNS 기반 -외부 라우팅 메쉬	내부 로컬 DNS 기반	내부 로컬 DNS 기반
IP Addressing	-컨테이너 당 내부 주소체계 -오버레이당 글로벌 범위	컨테이너 당 내부 주소체계 브릿지당 로컬 범위	물리 네트워크 상의 컨테이너당 외부 주소 체계
암호화	Yes, 선택	No	No
요구사항	엔진 1.12 이상 클러스터 스웜 (Swarm)모드	엔진 1.7 이상	호스트 인터페이스에 Promiscuous mode 필요

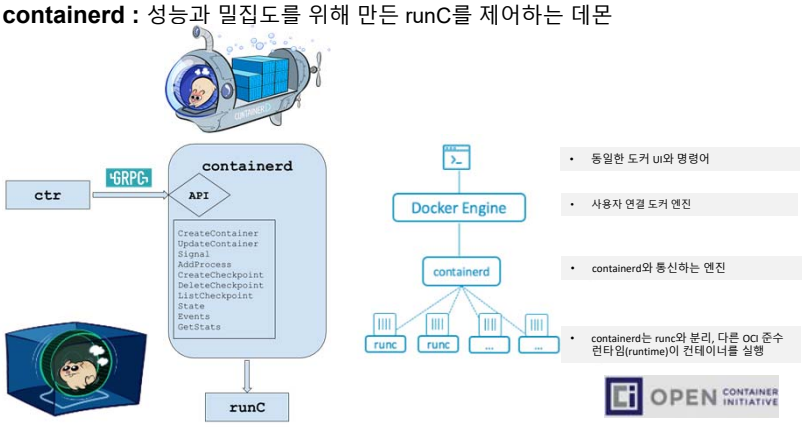
IV. 기술

❖ **도커 엔진 분리:** Docker 1.11에서 3개의 구성 분리(containerd, container-shim, runc)

❖ **도커 데몬 분리:** Docker 1.12에서 dockerd를 분리(libnetwork, VolumeAPI, AuthZ)

❖ **RunC :** 유니버설 컨테이너 런타임 (The universal container runtime)

❖ **containerd :** 성능과 밀집도를 위해 만든 runC를 제어하는 데몬



IV. 기술

❖ 요약

- **Container :** 소프트웨어를 위해 코어 운영체제의 방해 없는 환경을 생성하는 개념
- **도커(Docker):** 소프트웨어 컨테이너 플랫폼
- **도커 기본 기능:** 도커 이미지, 도커 컨테이너, 도커엔진, 도커 레지스트리
- **도커 오케스트레이션:** 도커 머신, 도커 스웜, 도커 컴포우즈
- **엔터프라이즈용 별도 제공:** 서비스 라이선스 비용 필요한 기능 (GUI 기반등)

기본 기능	오케스트레이션
도커 이미지 (Docker Image)	**머신 (Machine) :** 도커 설치한 인프라 제공
도커 컨테이너 (Docker Container)	**스웜 (Swarm) :** 호스트 클러스터링 제공 (Docker 1.12 이후 내장)
도커 엔진 (Docker Engine)	**컴포우즈 (Compose) :** 복수 컨테이너 적용 (Docker 1.13 이후 내장)
도커 레지스트리 (Docker Trusted Registry)	

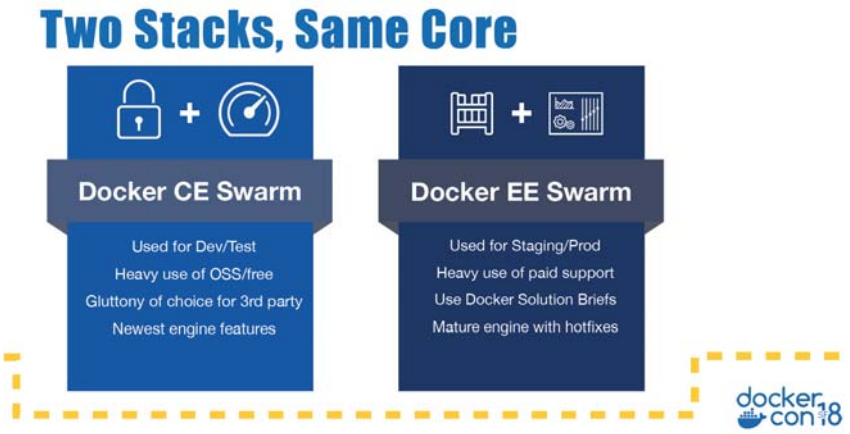
JS Lab

Community for KOREN AI Network Lab
jamest@jslab.kr

JS Lab

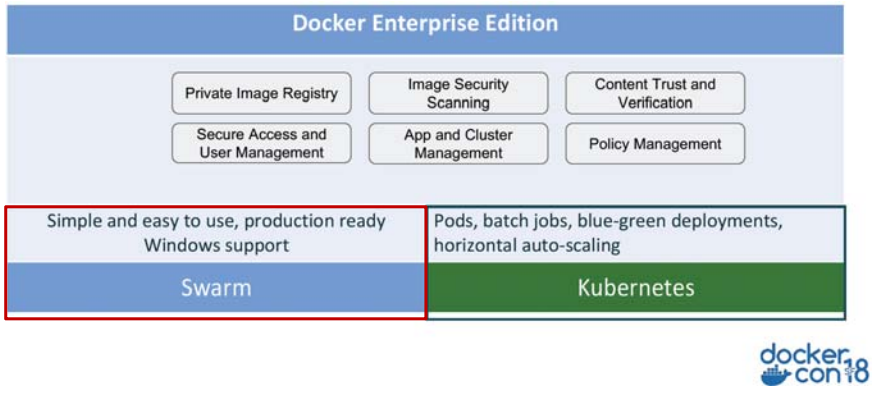
V. 도커 스웜(Swarm)

- CE vs EE
- CE는 개발/테스트를 위해 사용하며 최신의 엔진 기능을 사용 가능
- EE는 상용으로 안정된 엔진 사용



V. 도커 스웜(Swarm)

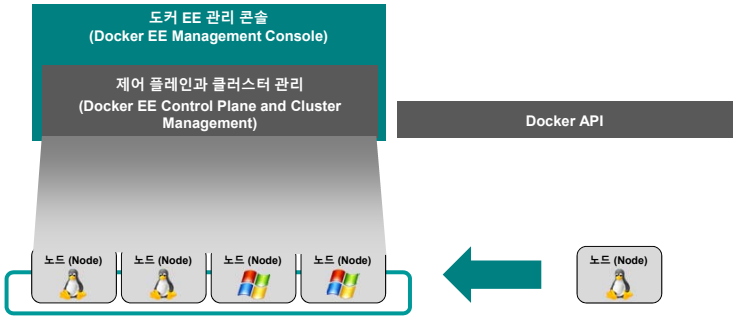
- Swarm and Kubernetes



V. 도커 스웜(Swarm)

❖ 확장

- 한번의 명령어로 새로운 Swarm/K8s** 노드를 클러스터에 추가
- 새로 추가한 노드는 기존 제어와 정책에 자동으로 통합
- 필요한 서비스는 자동으로 설치



V. 도커 스웜(Swarm)

❖ 스웜(Swarm) 종류

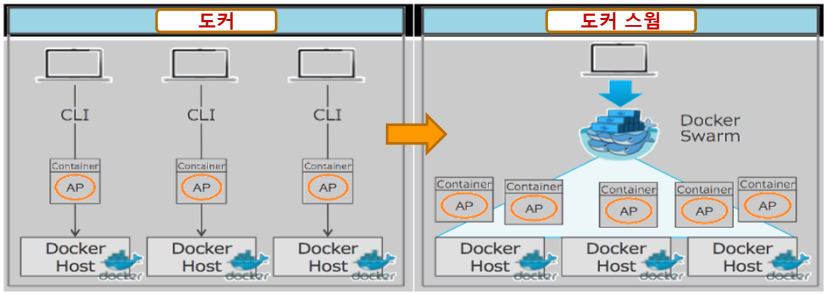
- **Docker Swarm:** 본래의 Docker Swarm으로 도커 에서 Call
- **Docker SwarmKit:** Github 사용 별도의 Open Source Project
- **Docker 내장 Swarm:** Docker 1.12 이후 내장 Docker Swarm Mode로 Routing Mesh, Load Balance, Service Discovery 지원

	Docker Swarm	Swarm Kit	내장 Swarm
별도 KV DB 저장	필요(progrium/consul)	내장	내장
보안	None	내장	내장
추가 설치	불필요	추가 설치 필요	불필요
Extra Service	None	None	Routing Mesh, Load Balance, Service Discovery
Docker-Compose	지원	지원	지원

V. 도커 스웜(Swarm)

❖ 스웜(Swarm) @ 도커(Docker) 1.12 이상

- **장애 자동복구(Fault-tolerant) 애플리케이션 적용 플랫폼**
  - ✓ 스케듈링 (Scheduling)
  - ✓ 장애시 재스케듈링 (Rescheduling on failure)
  - ✓ 고가용 HA(High Availability) 복수의 매스터 사용(Multiple Masters)
  - ✓ 스케듈링 방법 제어 (Labels, affinities, constraints를 사용하는 스케듈링 결정 제어)
  - ✓ DNS기반 서비스 발견 (DNS-based service discovery)
- **스케일('scale') 사용 성능 제어**
- **보안 강화 네트워킹**

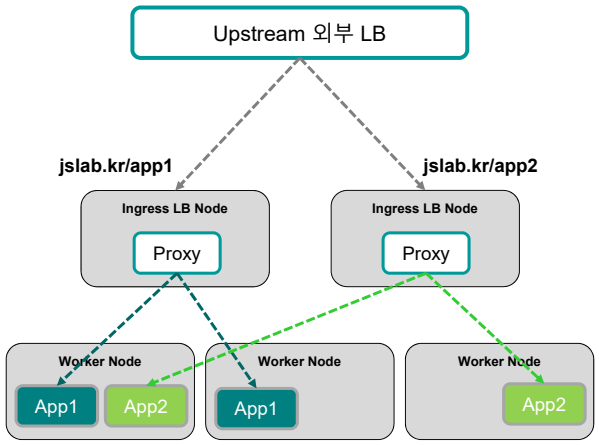


JS Lab

V. 도커 스웜(Swarm)

❖ 스웜(Swarm) @ 도커(Docker) 1.12 이상

- **장애 자동복구(Fault-tolerant) 애플리케이션 적용 플랫폼**



JS Lab

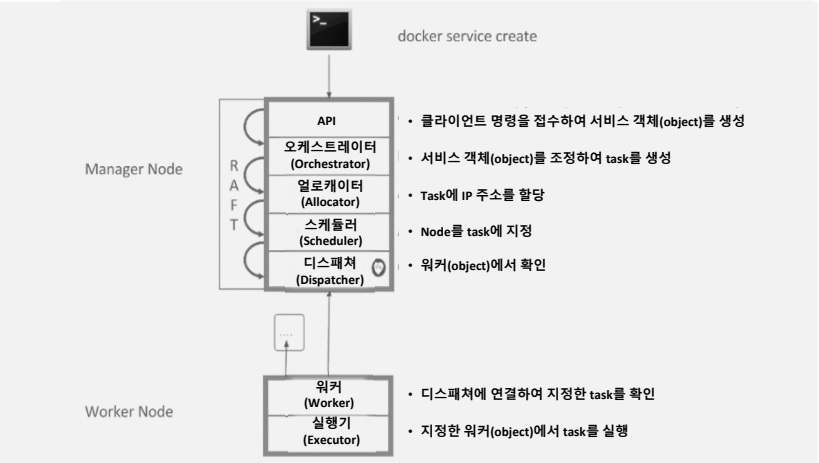
V. 도커 스웜(Swarm)

❖ **도커 스웜(Docker Swarm) 기능:**

- **하이퍼바이저 또는 오픈스택(OpenStack) 상의 스웜(Swarm) 클러스터**
- **기본 구성: 매니저(manager) 1 개, 워커(worker) 1 개**
- **인프라 구성 수준 지정 확인 유지**
  - ✓ 장애 자동 회복 (Autohealing)
  - ✓ 자동 컨테이너 상향(Auto-scale up)
  - ✓ 자동 컨테이너 하향(Auto-scale down)
- **도커 스웜 노드 작업 'docker node'**
  - ✓ 'demote' : 스웜 내에서 매니저(Manager) 노드를 워커(Worker) 노드로 강등(Demote)
  - ✓ 'promote' : 스웜 내에서 워커(Worker) 노드를 매니저(Manager) 노드로 승격(Promote)
  - ✓ 'rm' : 스웜 내에서 노드를제거
  - ✓ 'update' : 노드의 갱신
  - ✓ 기타 스웜 노드 관련 정보 확인: 'inspect' , 'ls' , 'ps'

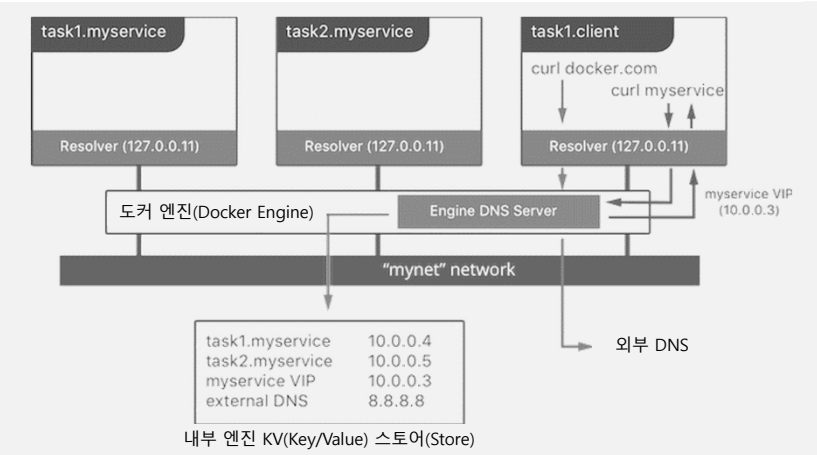
V. 도커 스웜(Swarm)

❖ **매니저/워커 네트워킹 프로파일:** VxLAN 기반 데이터 경로 / 외부 키값 저장 필요 없음 / 중앙 집중 자원 할당 / 개선된 처리능력 / 확장성



V. 도커 스웜(Swarm)

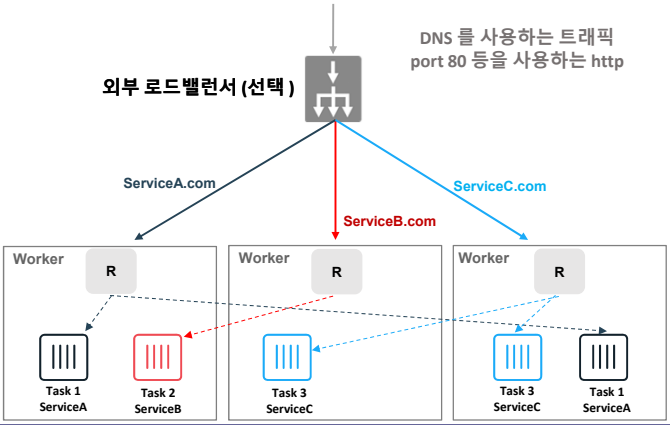
❖ 서비스 디스커버리
- 발견(Discovery): IP와 포트 주소
- Health check: 건강한 서비스만 트래픽 처리 참여
- 로드밸런싱: 해당하는 모든 인스턴스에서 로드밸런싱



JS Lab

V. 도커 스웜(Swarm)

❖ **라우팅 메쉬(Routing mesh):** 에지 라우팅을 위한 내장 라우팅 메쉬(routing Mesh)에서 모든 워커 노드(Worker Node)가 인그레스 라우팅 메쉬(Ingress Routing Mesh)에 참여하여 공개된 포트(Published Port)의 접속 요청을 수용하고 포트 변환은 워커노드에서 수행한다. 내부 로드밸런싱 매커니즘은 외부 요청에 동일하게 사용 (http/https 포트번호 임의 지정 가능)



JS Lab

V. 도커 스웜(swarm)

❖ **Rolling Update**

- **스웜 기반 서비스의 업그레이시에 사용**
- **Redis 3.0.6 컨테이너 이미지를 Redis 3.0.7 컨테이너 이미지 업그레이드(예)**

```
$ docker service create \                    # swarm manage에서 실행 '10초간 갱신 지연 지정
  --replicas 3 \
  --name redis \
  --update-delay 10s \                       # 10초 지연시간은 갱신과 서비스 Task 간격
  redis:3.0.6

0u6a4s31ybk7yw2wyvtikmu50

$ docker service update --image redis:3.0.7 redis  # Rolling Update 실행
Redis
```

스케듈러의 rolling updates 초기 설정

1. Stop the first task.
2. Schedule update for the stopped task.
3. Start the container for the updated task.
4. If the update to a task returns RUNNING, wait for the specified delay period then start the next task.
5. If, at any time during the update, a task returns FAILED, pause the update.

https://docs.docker.com/engine/swarm/swarm-tutorial/rolling-update/

JS Lab

V. 도커 스웜(swarm)

❖ **Docker stack: 17.03 or 1.13 스웜 모드에서 실행**

❖ **'docker stack COMMAND' 관리 명령어로 도커에 내장**

- **deploy: Deploy a new stack or update an existing stack**
- **ls: List stacks**
- **ps: List the tasks in the stack**
- **rm: Remove the stack**
- **services: List the services in the stack**

```
version: '3'
services:
  micro:
    image: gianarb/micro:1.2.0
    deploy:
      mode: replicated
      replicas: 2
      resources:
        limits:
          cpus: '0.25'
          memory: 512M
        reservations:
          cpus: '0.25'
          memory: 256M
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
```

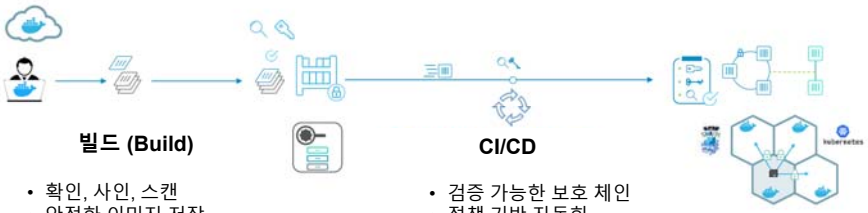
도커 스택 실행 명령어 (예)

```
docker stack deploy --compose-file=docker-compose.yml my_stack
```

JS Lab

VI. 도커 보안

❖ 종단간 소프트웨어 제공 체인 (Docker EE secures the End-to-End Software Supply Chain



빌드 (Build)

- 확인, 사인, 스캔
- 안전한 이미지 저장
- 안전한 민감 데이터

CI/CD

- 검증 가능한 보호 체인
- 정책 기반 자동화

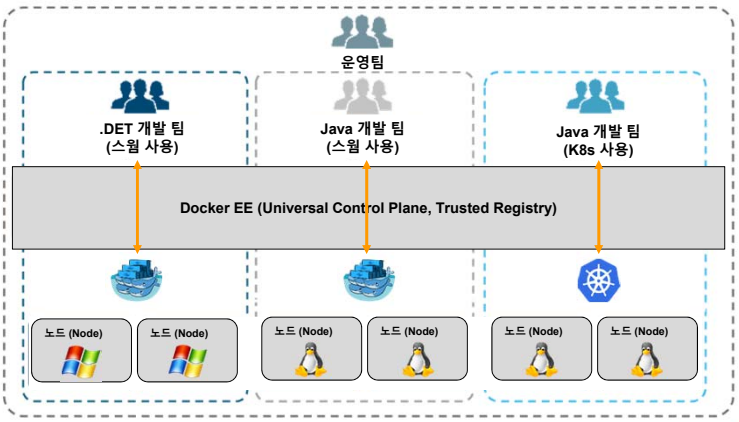
Run

- 운영보안

VI. 도커 보안

❖ 클러스터 보안 환경 영역을 지정

- LDAP/AD 연동
- Node 기반의 Namespace 분리
- Docker Community Edition은 CLI기반 수동 관리



JS Lab

VI. 도커 보안

❖ **Build:** 최소한의 공식적인 이미지를 신뢰하는 저장소에 저장

- 최소한의 이미지 사용 (Dockerfile에 암호등 보안 관련내용을 써넣지 않음)
- 공식적인 이미지 사용 (이미지 버전 pinning 등으로 지정 사용하고, 'latest'나 갱신(update) 명령어 등을 사용하지 않음)
- 신뢰하는 컨텐츠의 이미지 가져오기 (Pull) – 신뢰하는 저장소 구축 사용

❖ **Ship:** 신뢰하는 이미지 저장 장소 구축 사용

- 신뢰하는 컨텐츠를 Registry에 Push
- 주기적 도커 보안 스캐닝 결과 확인 (예: EE Advanced 사용)

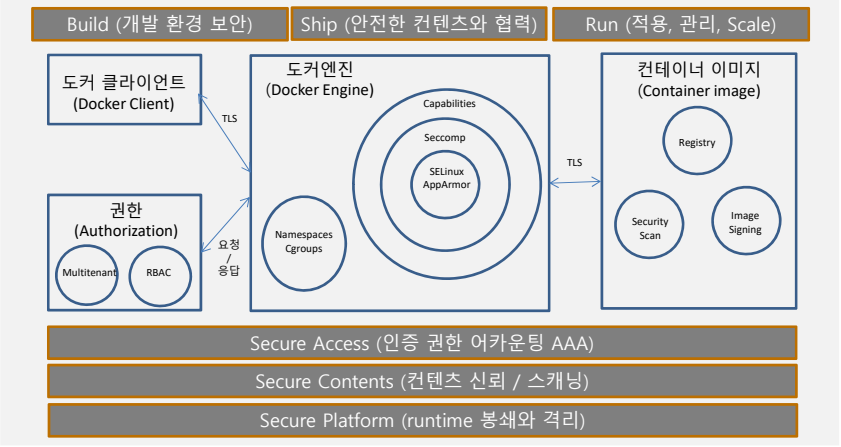
❖ **Run:** 실행 중인 컨테이너가 SPF(single Point of Failure)인 리눅스 커널의 CPU나 메모리 자원 소진을 방지하고 커널 침투 경로 차단을 위한 격리

- Client와 Engine 간에 상호 TLS 사용 인증
- 볼륨과 컨테이너는 읽기 전용
- Daemon 내의 user namespace
- Cgroups의 자원 제한
- 기본 제공 apparmor/seccomp/capabilities, 또는 테스트 된 프로파일
- 불필요 패키지, 'setid', 'setgid' 허용하지 않고, --privileged 사용하지 않음

JS Lab

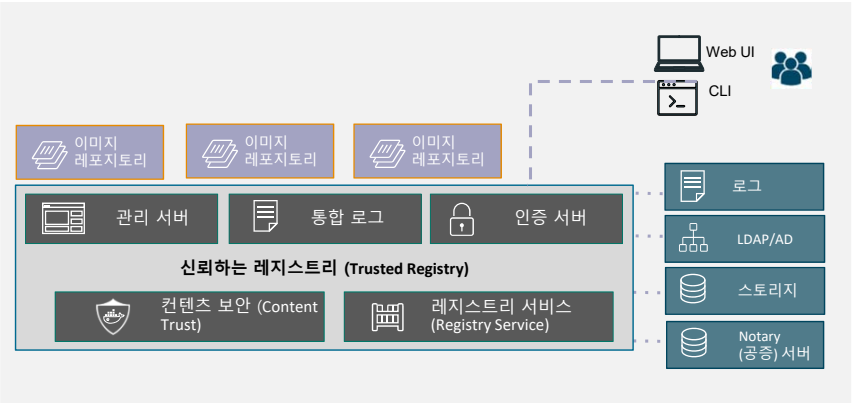
VI. 도커 보안

❖ **도커 보안 서비스 구성:** 구성 요소간 TLS 기반 연결, 인증 서버 사용, 리눅스 제공 보안 기능 사용하며, AAA기반 안전한 접속(Access)/신뢰하는 이미지 등의 컨텐츠(Contents) 제공/방화벽 IDS등의 보안 강화 플랫폼(Platform) 제공



VI. 도커 보안

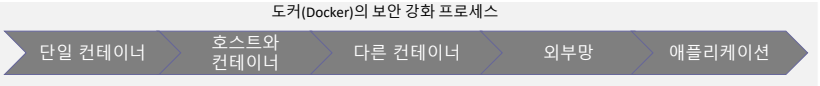
❖ **이미지의 보안을 위한 엔터프라이즈 구성(예):** 신뢰 할 수 있는 레지스트리 구성을 위해 외부 관리서버, 통합로그, 인증서버 등을 사용하여 컨텐츠 보안과 서비스를 제공 할 수 있음



VI. 도커 보안

❖ 컨테이너 보안

- **단일 컨테이너:** 컨테이너 보안의 기초로 리눅스 커널에 의존하여 격리와 자원 제어를 실행으로 위협을 완화하며, 코드와 컨테이너 환경이 안전 하다는 것을 전제
- **호스트와 컨테이너:** CPU/메모리/디스크 자원을 소진하는 DOS공격이나 접속을 히여 커널 변형이나 정보 유출 등의 위협이 있고, 리눅스가 권하는 것을 적용(예: Kernel 4.3, seccomp 1.10, TLS 인증 사용, --privileged' 모드 사용 금지 등등)
- **다른 컨테이너:** 호스트의 경우와 유사한 방법으로 CPU/메모리/디스크 자원을 소진하는 DOS공격이나 접속을 하여 커널 변형이나 정보 유출 등의 위협 등을 완화
- **외부망:** DDoS 공격과 외부의 접속 시도 그리고 OS의 패치 미적용 등에 관한 공격이 가능하며 리눅스가 권하는 것을 적용 하고 외부의 취약점 발견 스캐닝을 할 수 있음 (IBM Bluemix, Docker Data Center, CoreOS Clair, Red Hat "SmartState" CloudForms (w/Black Duck)
- **애플리케이션:** 특정한 유형의 공격을 정할 수는 없으나 VM 베어메탈 클라우드 등의 경우와 같고, 적절한 DevOps 과정 불필요 서비스 비노출 보안 코딩이나 디자인등으로 공격을 완화



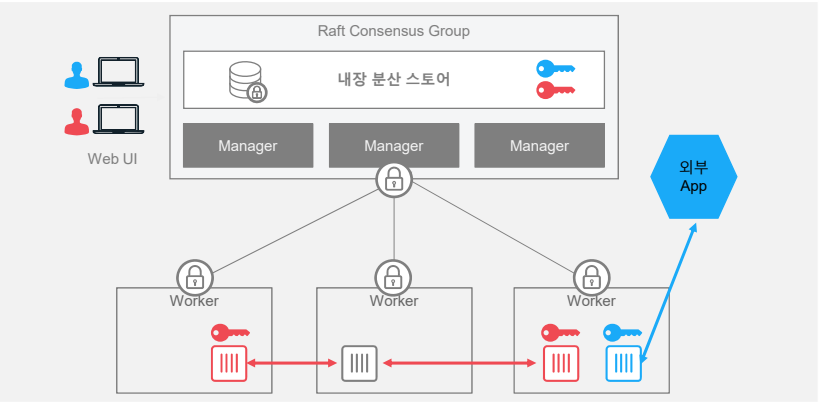
https://docs.docker.com/engine/security/non-events/

JS Lab

VI. 도커 보안

❖ 스웜 보안

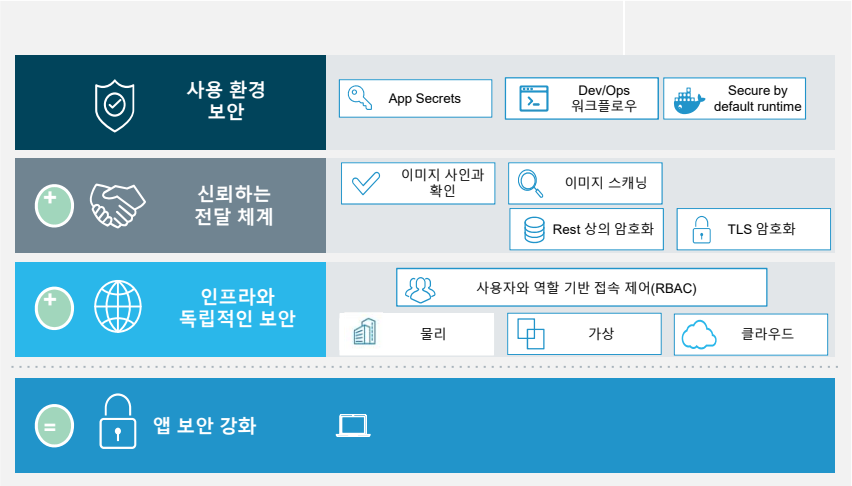
- **관리자가 클러스터 내의 보안키를 추가/삭제/갱신**
- **Exposed to a container via a "/secrets" tmpfs volume**
- **관리자는 사용자와 팀에 RBAC 권한 제공**
- **GUI 사용 모든 컨테이너들에 보안키 갱신**
- **클러스터 내의 사용자 접속 Auditing**



JS Lab

VI. 도커 보안

❖ Docker EE 보안은 도커 컨테이너 생태계에서 보안 제공
❖ Docker EE Platform의 추가 보안기능: 이미지 스캐닝 등의 서비스를 제공



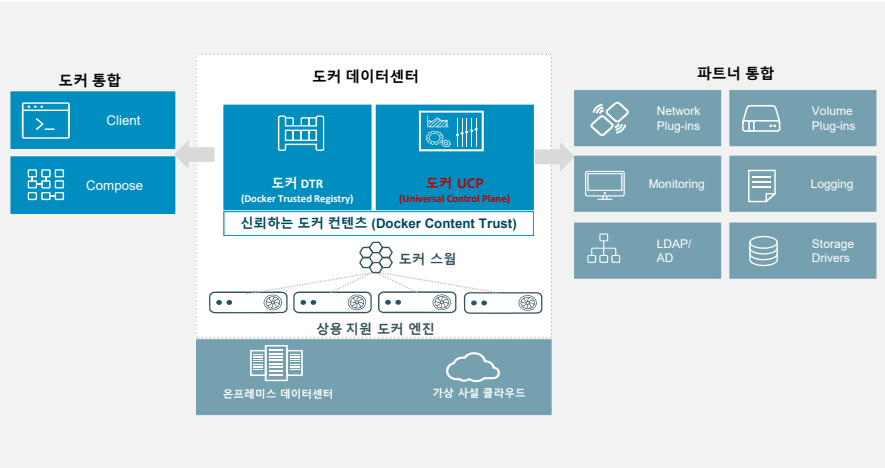
JS Lab

JS Lab

Community for KOREN AI Network Lab
jamest@jslab.kr

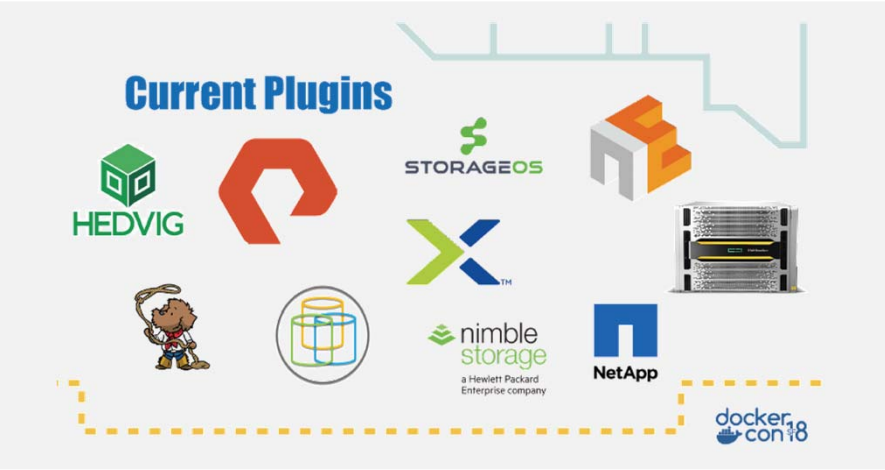
VII. Docker Datacenter(도커 데이터센터)

❖ Docker Datacenter: Docker DTR and UCP



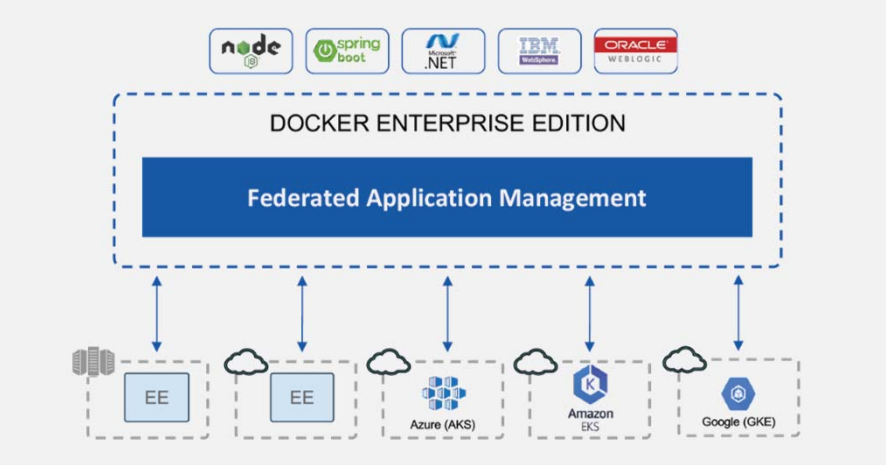
VII. Docker Datacenter(도커 데이터센터)

❖ Docker Volume Plugins



VII. Docker Datacenter(도커 데이터센터)

❖ Federated Application Management (Coming Soon)

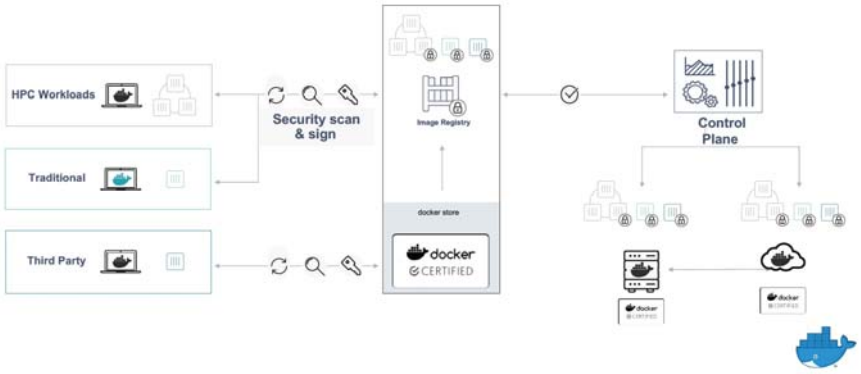


Community for KOREN AI Network Lab
jamesj@jsl.kr

JS Lab

VIII. 응용

❖ Leveraging HPC in the Enterprise



VIII. 응용

❖ Space

All factors drive:

- Cost
- Reliability
- Low Memory (~16MB)
- No virtual memory
- 32 bit CPU (~100MHz)
- Process
- Testing. And more testing

There are no space mechanics (yet) and turning it off and on again is NOT cool

VIII. 응용

- ❖ **Fanless 하드웨어** (IoT Gateway)
- ❖ **오픈소스 기반**
- ❖ **SDN 기반 컨테이너 네트워킹 사용** (인증키 생성/서비스 배포/암호화 터널링 내장)

IoT Gateway 오픈 하드웨어 플랫폼 비교

	Items	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6
IoT Gateway Hardware (Fanless)	CPU Type	Celeron J1900	ARM v7	ARM v7	ARM v6	Atom	Quark
	# of CPU Cores	4	4	4	1	2	1
	CPU Clock Speed	2 GHz	1.2 GHz	900 MHz	700MHz	500 MHz	400 MHz
	RAM	4 GB	1 GB	1 GB	512 MB	1 GB	256 MB
항목	Docker 설치	OK	OK	OK	OK	No	No
	Docker Show	OK	OK	OK	No	No	No
	Container 2 MB 구동	OK	OK	OK	No	No	No
	Container 784 MB 구동	OK	No	No	No	No	No
	Container Cluster Manager	OK	OK (동일H/W)	No	No	No	No
	Agent for UCP	Yes	No	No	No	No	No
	Alarm	Yes	Yes	Yes	Yes	Yes	N/A
	Sensor / Actuator	N/A	OK	OK	OK	OK	OK
	Local Logger	OK	OK	OK	OK	N/A	N/A
	OVS	OK	OK	OK	OK	No	No
	PoE 지원 (변환기 사용)	No	Yes(변환기)	Yes(변환기)	Yes(변환기)	No	Yes (전용 모듈)
	Wi-Fi 지원	Yes	Yes	Yes	Yes	Yes	Yes
	Bluetooth	Yes(동글)	Yes(내장)	Yes(내장)	Yes(동글)	Yes(내장)	N/A
	Flow Agent	Yes	N/A	N/A	N/A	N/A	N/A
	IDS	Yes	No	No	No	No	No
	보안 생태계 연동	Yes	Yes(저성능)	Yes(저성능)	N/A	N/A	N/A

JS Lab

VIII. 응용

- ❖ **A small IPC enabled with Docker**
- ❖ **Allow many Application Stack to run**
- ❖ **Managed Connectivity to the Cloud**
- ❖ **Local Management Interface**

Siemens H"Edge" Gateway

- Remotely Managed IoT Focused Gateway
- Linux based x86 Industrial
- Supports Linux Container Workloads
- Remote Re-Configurable based on the use case
- Supports Connectivity, Both limited and dedicated to the Cloud for non-connected devices
- Facilitates Legacy Application Interfaces
- Allows Consolidation of data before it's pushed/pulled to the cloud

SIEMENS

JS Lab

VIII. 응용

❖ Node Sizing

Node Sizing

Manager Nodes	Worker Nodes
• **CPU:** 4 vCPU • **Memory:** 16GB • **Disk:** SSD for /var/lib/docker • Support 100s of worker nodes • 3 or 5 managers is preferred	• Depends on application workloads • If migrating there will be less overhead from OS • Leave headroom for rescheduling events • Run under load and test

VIII. 응용

❖ Building Docker Stack

Open Source Stack

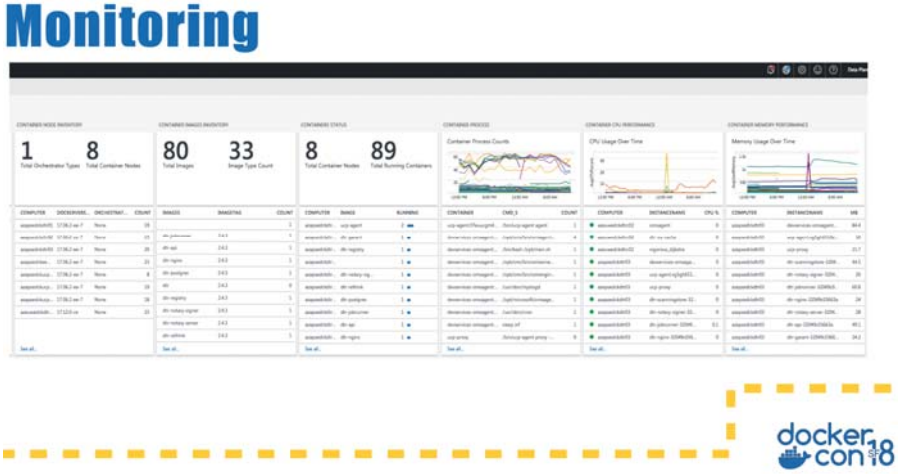
Swarm GUI	Portainer
Central Monitoring	Prometheus + Grafana
Central Logging	Elastic ELK
Layer 7 Proxy	Traefik + Let's Encrypt
Storage	REX-Ray + Digital Ocean Volumes
Networking	Docker Swarm Overlay
Orchestration	Docker Swarm
Runtime	Docker CE
HW / OS	Docker Machine + Digital Ocean

Docker EE on AWS Stack

Swarm GUI	Docker EE UCP
Central Monitoring	AWS Cloudwatch + Telegraph
Central Logging	AWS Cloudwatch Logs
Registry	Docker EE DTR
Layer 7 Proxy	HTTP Routing Mesh (Interlock+Nginx)
Storage	Docker Cloudstor EBS/EFS
Networking	Docker Swarm Overlay
Orchestration	Docker Swarm
Runtime	Docker EE
HW / OS	Terraform + Ansible + AWS

VIII. 응용

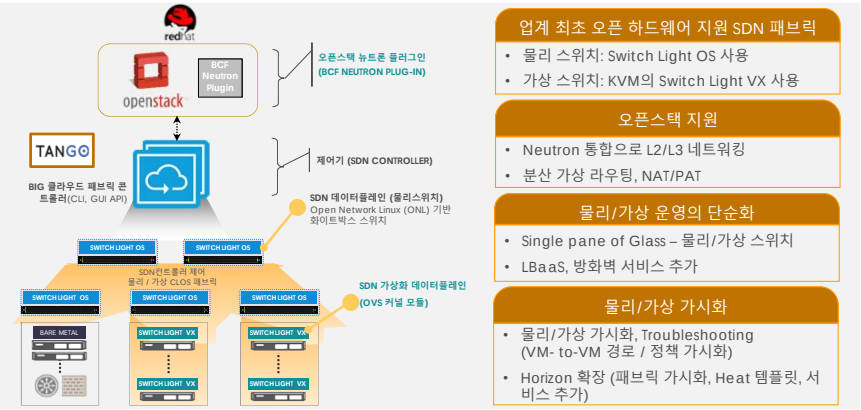
❖ Monitoring



VIII. 응용

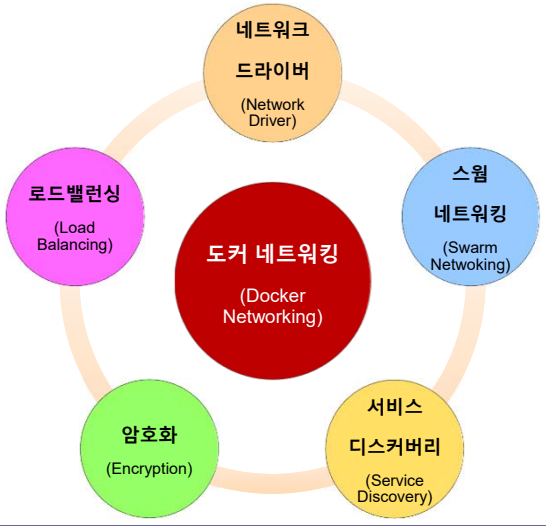
❖ 오픈스택(OpenStack)의 뉴트론(Neutron)연동

❖ 분산라우팅, Heat, LBaaS, 방화벽, VM-to-VM 경로/정책 가시화



IX. 도커 네트워킹

❖ 도커 네트워킹



Community for KOREN AI Network Lab
jamest@jslab.kr

JS Lab

IX. 도커 네트워킹

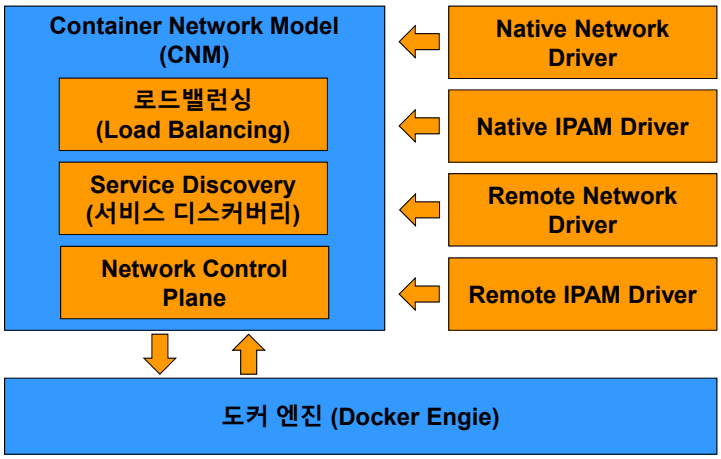
❖ 도커 네트워크 드라이버 종류

드라이버/ 기능	브릿지 (Bridge)	User Defined Bridge	호스트 (Host)	오버레이 (Overlay)	Macvlan/ ipvlan
연결	동일 호스트	동일 호스트	동일 호스트	멀티 호스트	멀티 호스트
서비스 디스커버리 / DNS	'links' 사용, DNS 사용 /etc/hosts	도커엔진에서 DNS 서버 사용	도커엔진에서 DNS 서버 사용	도커엔진에서 DNS 서버 사용	도커엔진에서 DNS 서버 사용
외부 접속	NAT	NAT	호스트 게이트웨이 사용	외부 접속 없음	언더레이 게이트웨이 사용
Namespace	분리	분리	동일 호스트	분리	분리
스웜 모드	미지원	미지원	미지원	지원	미지원
캡슐화	이중 캡슐화 없음	이중 캡슐화 없음	이중 캡슐화 없음	VxLAN 사용 더블 캡슐화	이중 캡슐화 없음
애플리케이션	노스(North), 사우스(South) 외부 접속	노스(North), 사우스(South) 외부 접속	모든 네트워킹 제어, 분리 필요 없음	호스트간 컨테이너 연결	컨테이너는 언더레이 네트워킹 직접 연결 필요

JS Lab

IX. 도커 네트워킹

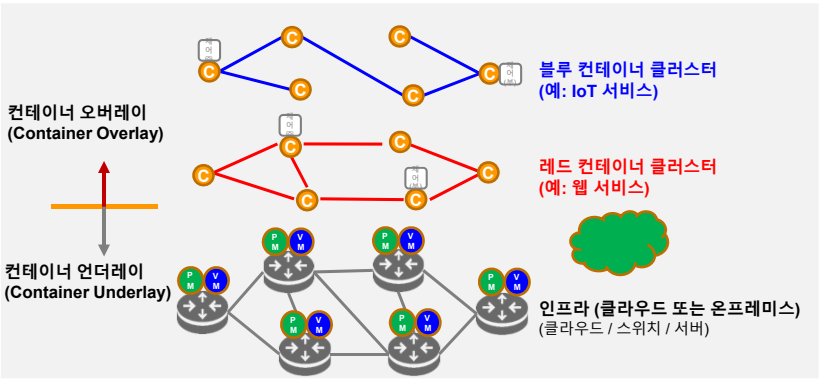
❖ Container Network Model (CNM)



JS Lab

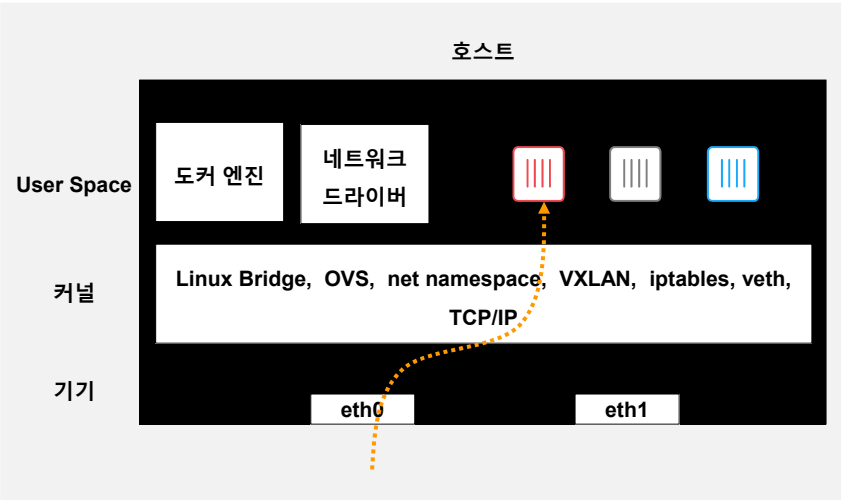
IX. 도커 네트워킹

- ❖ 컨테이너 네트워킹의 오버레이와 언더레이
- ❖ 도커 스웜의 클러스터 네트워킹: 제어 관리자(Manager)와 Worker로 오버레이 클러스터 구성하며, Manager에서도 Worker 기능 동시 제공
- ❖ 도커 스웜 Manager HA: 3개 또는 5개 등 홀수를 권장하며 Leader는 1개로 동작



IX. 도커 네트워킹

- ❖ 도커 네트워킹은 리눅스 네트워킹

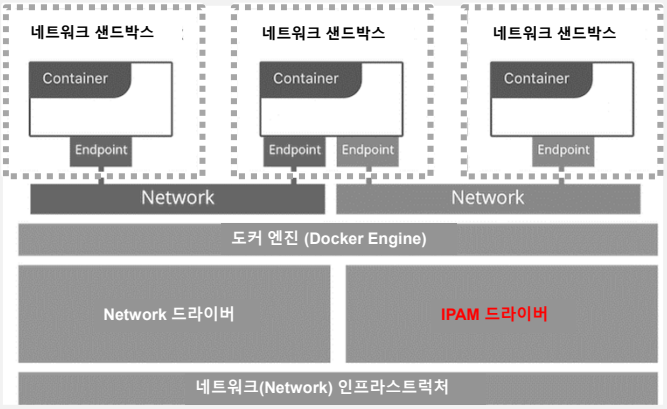


IPAM: IP Address Management

IX. 도커 네트워킹

❖ 도커 레퍼런스 아키텍처: 확장 기능 설계, 포터블 도커 컨테이너 네트워크

컨테이너 네트워킹 모델



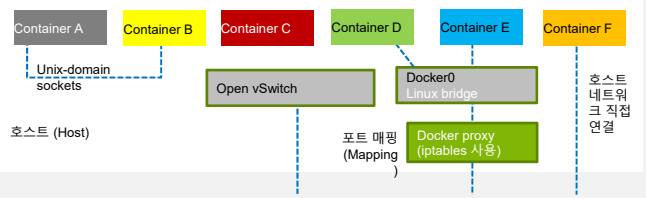
IPAM: IP Address Management

JS Lab

IX. 도커 네트워킹

❖ 하이레벨 (High-level) 기능

Namespace	/proc에서 프로세스 수준 관리의 컨테이너 네트워킹
Linux Bridge	커널에서 포워딩에 사용하는 L2/MAC을 인식하는 스위치
Open vSwitch	프로그램 가능하고 터널링을 지원하는 개선한 브릿지 (SDN 스위치)
NAT	네트워크 주소 변환 IP address + Ports (Types: SNAT, DNAT)
iptables	커널 내의 정책 엔진으로 패킷전송, 방화벽, NAT를 관리함
Unix domain sockets	단일 호스트 내 통신 기반의 File descriptor, FIFO 파이프로 동작
User-space vs Kernel-space	자원과 성능을 정상화 제어하는 애플리케이션도메인 • 컨테이너(Container) 애플리케이션(applications)은 user-space 에서 실행 • 네트워크 전송은 kernel space에서 실행

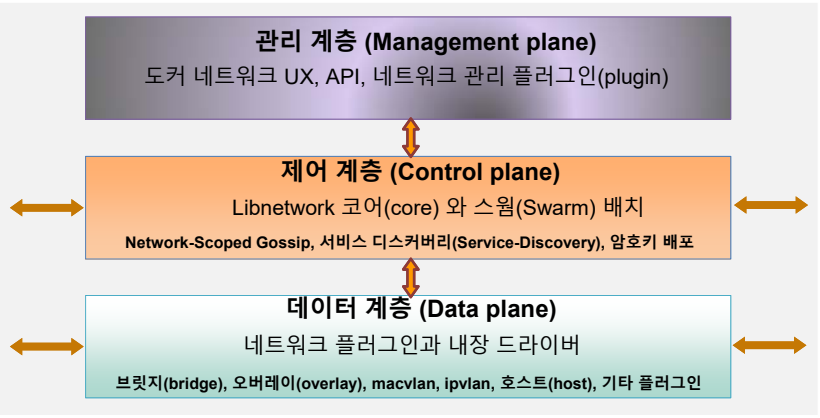


JS Lab

IX. 도커 네트워킹

❖ 도커 네트워크 플레인 구성

- 관리 계층 (Management Plane)
- 제어 계층 (Control Plane)
- 데이터 계층 (Data plane)

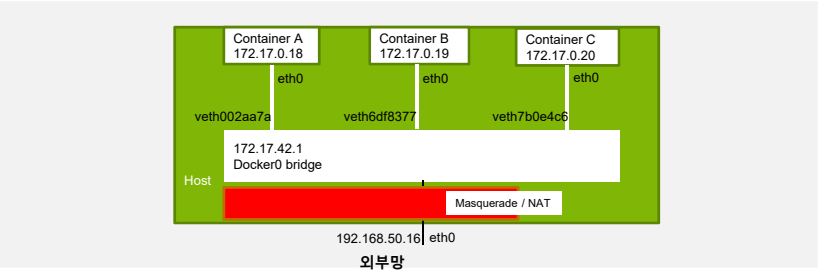


JS Lab

IX. 도커 네트워킹

❖ Docker0 브릿지

- 네트워크 선택 지정이 없는 경우 자동으로 생성 (no additional options "--net")
- 각 컨테이너는 도커(Docker)에 의해 고정 IP 주소를 할당
- KVM or VirtualBox와 유사한 기본 설정
- 호스트는 브릿지에 할당한 IP 주소를 통해 연결
- 외부 트래픽은 컨테이너에 접속 불가

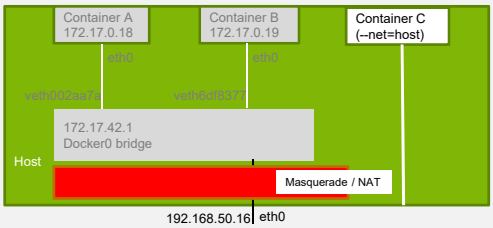


JS Lab

IX. 도커 네트워킹

❖ 호스트(Host) “--net=host”

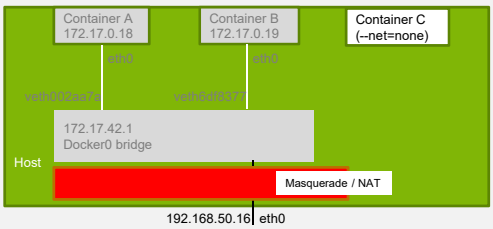
- 네트워크 선택을 host로 지정하는 경우 생성 (options “--net=host“)
- 컨테이너는 호스트 네트워크 자원을 사용
- 호스트는 컨테이너가 생성한 포트 주소를 통해 연결
- 외부 트래픽은 호스트 네트워크 자원을 통해 컨테이너에 접속



IX. 도커 네트워킹

❖ None “--net=none”

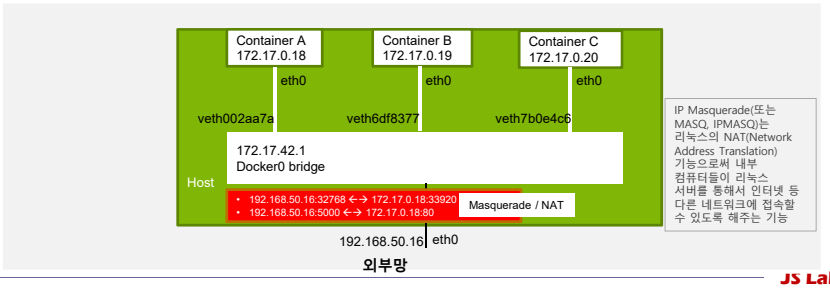
- 네트워크 선택을 none으로 지정하는 경우 생성 (options “--net=none“)
- 컨테이너는 네트워크 연결 단절
- 호스트는 컨테이너가 생성한 포트 주소를 통해 연결
- 호스트와 외부 트래픽은 컨테이너 접속 불가
- 수동으로 링크를 OVS등에 연결하여 통신 가능



IX. 도커 네트워킹

❖ 브릿지 / 포트맵핑 (Port Mapping)

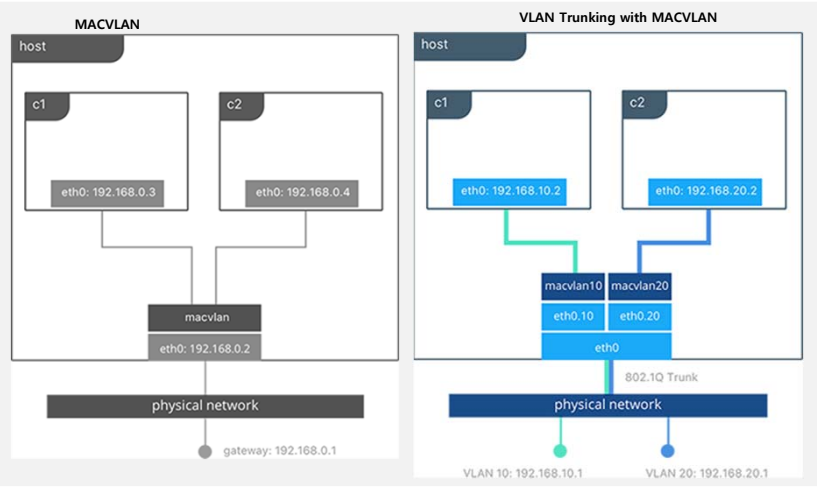
- 외부 연결 가능
- docker run -d --name onos -p 3999:8181 -p 4999:6653 onosproject/onos
- docker run -d -p 9200:9200 -p 9300:9300 elasticsearch



IX. 도커 네트워킹

❖ MACVLAN 드라이버를 사용 “docker network create -d macvlan”

❖ VLAN 트렁크 연결 가능

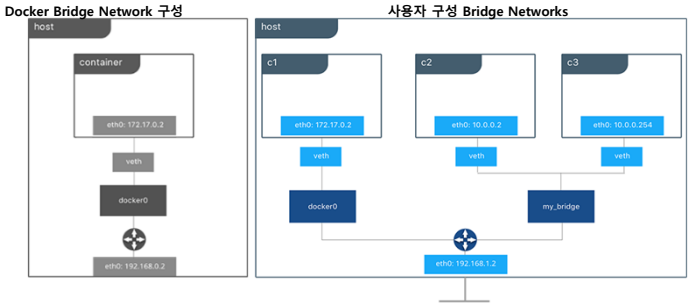


IX. 도커 네트워킹

❖ 사용자 구성 브릿지 네트워크

❖ Open vSwitch (OVS) 예

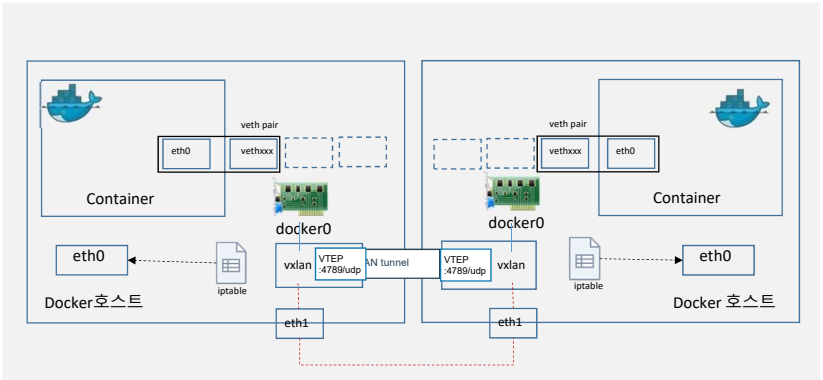
- ✓ OVSDB / OpenFlow 프로토콜을 이용한 프로그래밍
- ✓ VxLAN, GRE, VLAN 기반 캡슐화 / L2 포워딩(forwarding)
- ✓ 캡슐화(Encapsulation)는 컨테이너가 원하는 MAC/IP 주소에 전달 가능
- ✓ ARP 프록시, L3 라우팅, 로드밸런싱 가능
- ✓ 접속 제어, 트래픽 제한, 우선순위 등급화 가능
- ✓ 10G/s 이상 처리 가능
- ✓ 선택적 DPDK 가속화로 1) kernel, 또는 2) userspace



IX. 도커 네트워킹

❖ Docker Network Mode - Overlay

- 호스트간 연결시 사용
- Overlay network은 스웜(Swarm)모드의 Manager에서 생성 가능



IX. 도커 네트워킹

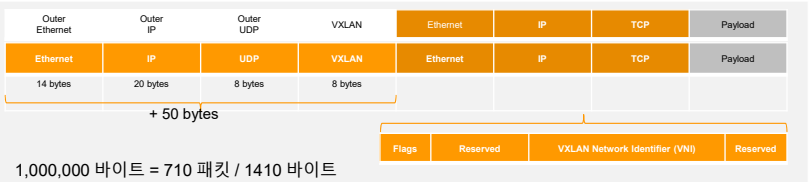
- ❖ 컨테이너 오버레이 드라이버를 사용 “docker network create -d overlay”
- ❖ VLAN 트렁크 연결 가능
- ❖ 도커 1.12 버전 이상 지원 기능

  - 도커 엔진에 스웜(Swarm) 통합하며 오버레이 보안 강화
  - 스웜은 매니저(Manager)와 워커(Worker)기반의 제어와 수행을 분리
  - 스웜 클러스터기반의 오버레이에 로드밸런싱/서비스 디스커버리 내장
  - 스웜모드에서는 외부 KV store 불필요
  - 스웜모드 콘트롤 플레인 보안
  - 스웜모드 VXLAN 암호화 가능
  - 로드 밸런싱은 가상 IP와 DNS RR 모두 지원
  - 서비스 디스커버리(Service-Discovery) 내장 (임의 포트 지정 가능)
  - 라우팅 메쉬(Routing Mesh) 지원

JS Lab

IX. 도커 네트워킹

- ❖ 1 MB 데이터 전송시 TCP/IP 의 VXLAN 오버헤드

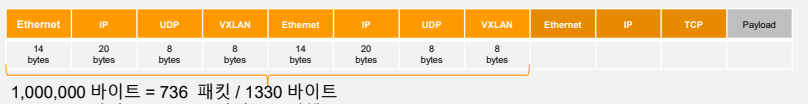


1,000,000 바이트 = 710 패킷 / 1410 바이트
710 x 50 바이트 = 35,500 바이트 오버헤드

1,035,500 바이트 전송

3.55 %

- ❖ 1 MB 전송시 TCP/IP 의 더블 VXLAN 오버헤드



1,000,000 바이트 = 736 패킷 / 1330 바이트
736 x 100 바이트 = 73,600 바이트 오버헤드

1,073,600 바이트 전송

7.36 %

JS Lab

IX. 도커 네트워킹

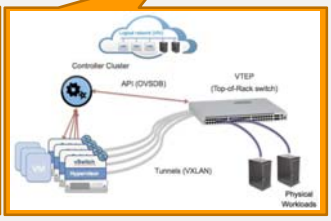
❖ **오버레이 네트워킹의 성능 한계**

- 오버레이 모드는 CPU 사용율이 매우 높음
- CPU 과부하 해결을 위해 ToR 스위치나 NIC카드로 오버레이 기능을 Offload
- 하드웨어 오버레이 게이트웨이 사용

오버레이 모드는 CPU 사용율이 매우 높음

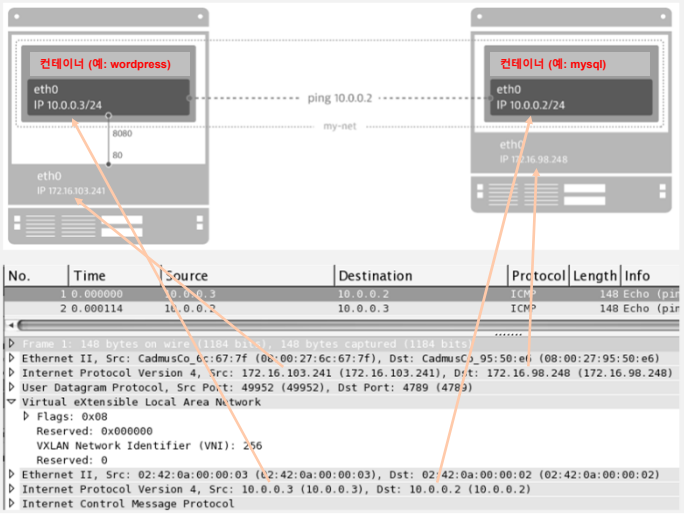
CPU 과부하 해결을 위해 ToR 스위치나 NIC카드로 오버레이 기능을 Offload

	Throughput	Recv side cpu	Send side cpu
Linux Bridge:	9.3 Gbps	85%	75%
OVS Bridge:	9.4 Gbps	82%	70%
OVS-STT:	9.5 Gbps	70%	70%
OVS-GRE:	2.3 Gbps	75%	97%



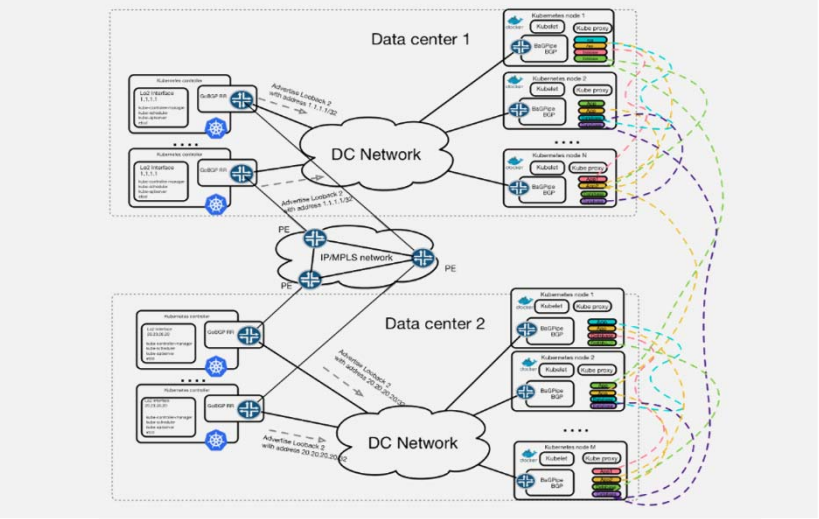
IX. 도커 네트워킹

❖ **VXLAN 데이터플레인을 사용하는 오버레이 아키텍처 데이터 구성 분석**



IX. 도커 네트워킹

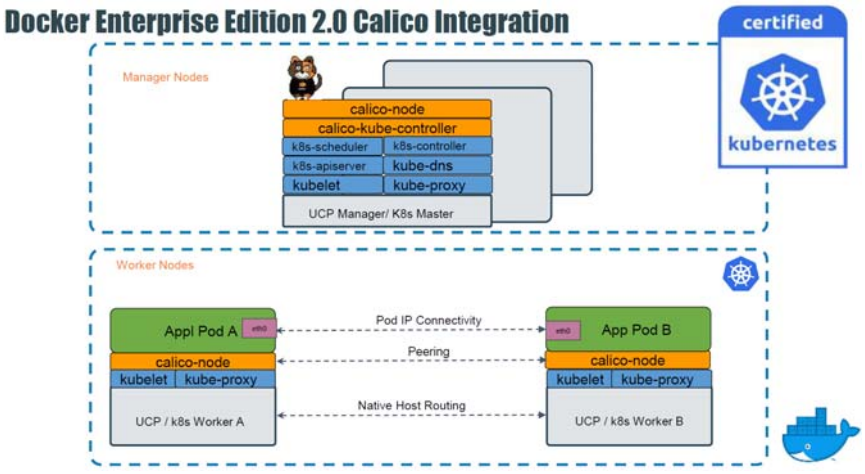
❖ 컨테이너 오버레이: 사이트간 연결 응용 (예)



http://murat1985.github.io/kubernetes/cni/2016/05/15/kubernetes.html

IX. 도커 네트워킹

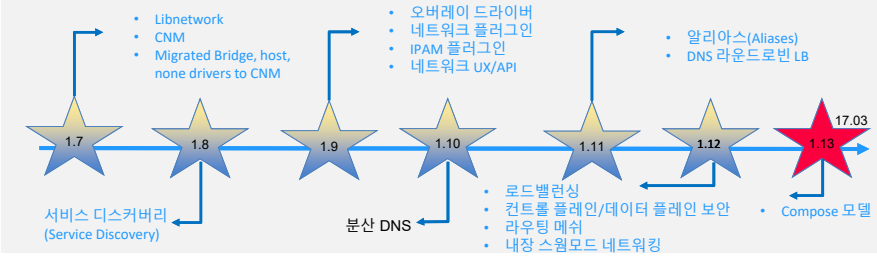
❖ 도커 EE 2.0 Calico 통합



IX. 도커 네트워킹

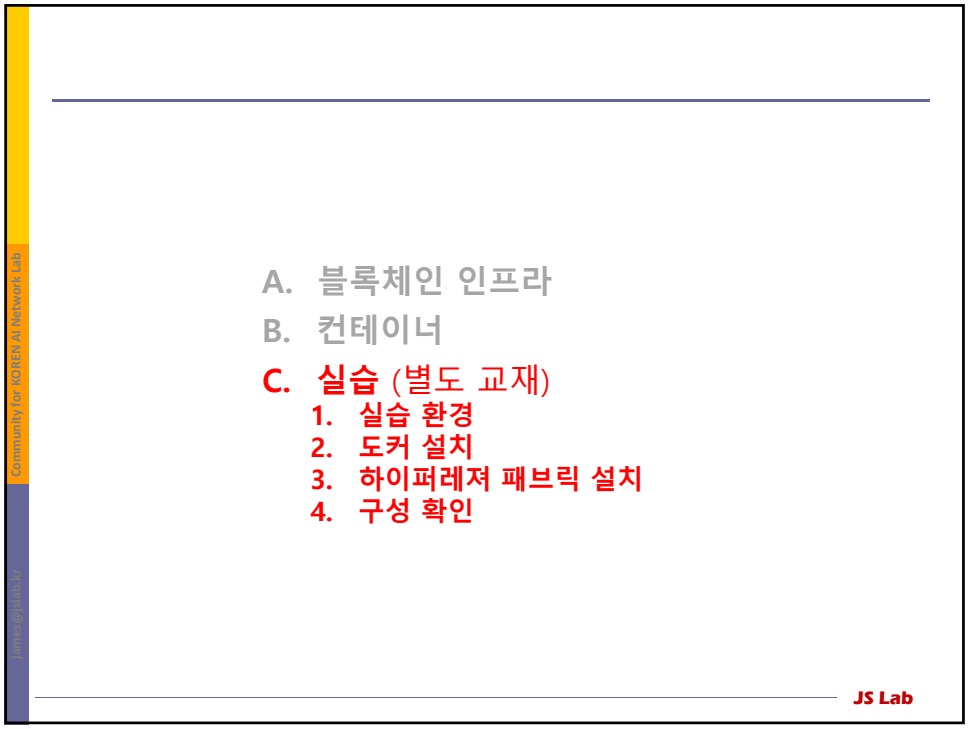
❖ 요약

- 도커 엔진에 스웜(Swarm) 통합
- Manager 와 Worker
- 자동 복구, 수동 확장
- 고정 IP 사용 LB 지원
- 오버레이 네트워크와 DNS
- Manager의 HA(High Availability)
- 네트워크 보안 (TLS) with CA
- 노드 선정 + Affinity 와 anti-affinity
- Compose 사용 적용 모델링



5. Q&A

Community for KOREN AI Network Lab
jamest@jslab.kr


Community for KOREN AI Network Lab
jamesjslab.kr
A. 블록체인 인프라
B. 컨테이너
C. 실습 (별도 교재)
1. 실습 환경
2. 도커 설치
3. 하이퍼레져 패브릭 설치
4. 구성 확인
JS Lab