KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

YÜKSEK LİSANS TEZİ

NESNELERİN İNTERNETİ VERİLERİNİN GÜVENLİ BİR ŞEKİLDE BLOKZİNCİR AĞLARINA AKTARIMI

KAZIM ONUR TOKA

KOCAELİ 2021

KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

YÜKSEK LİSANS TEZİ

NESNELERİN İNTERNETİ VERİLERİNİN GÜVENLİ BİR ŞEKİLDE BLOKZİNCİR AĞLARINA AKTARIMI

KAZIM ONUR TOKA

Prof.Dr. Ahmet SAYAR
Danışman, Kocaeli Üniversitesi …….
Prof.Dr. Kerem KÜÇÜK
Jüri Üyesi, Kocaeli Üniversitesi …….
Doç.Dr. Mehmet Sıddık AKTAŞ
Jüri Üyesi, Yıldız Teknik Üniversitesi …….

Tezin Savunulduğu Tarih: 15.06.2021

ÖNSÖZ VE TEŞEKKÜR

Bu tez çalışmasında nesnelerin interneti cihazlarından üretilen verilerin, blokzincir ağlarına güvenilir bir şekilde iletilmesi için gerekli olan entegrasyonun nasıl sağlanabileceğine dair bir yaklaşımın sunulması amaçlanmıştır. Nesnelerin interneti gibi anlık, sürekli ve büyük veri üreten bir benzetim ortamı ile Hyperledger Fabric blokzincir ağı MQTT protokolü üzerinden haberleştirilerek, veri aktarımın güvenli bir şekilde gerçekleştirilmesine çalışılmıştır.

Tez çalışmam sırasında fikirleriyle çalışmalarıma yön veren, bana güvenen ve desteğini hiçbir zaman esirgemeyen danışmanım Prof.Dr. Ahmet SAYAR'a sonsuz teşekkürlerimi sunarım.

Hayatımın her anında yanımda durarak, desteklerini hiçbir zaman esirgemeyen, bana inanarak moral ve güç veren, her anımda sevinçlerimi ve üzüntülerimi paylaşan sevgili annem Nazife TOKA, babam Murat TOKA ve ablam Duygu ŞENER'e teşekkürlerimi sunarım.

Haziran - 2021 Kazım Onur TOKA

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

Kısaltmalar

AMQP : Advanced Message Queuing Protocol (Gelişmiş Mesaj Protokolü)
API : Application Programming Interface (Uygulama Programlama Ara yüzü)
BFT : Byzantine Fault Tolerance (Bizans Hata Toleransı)
CA : Certificate Authority (Sertifika Otoritesi)
CFT : Crash Fault Tolerance (Çökme Hata Toleransı)
CLI : Command Line Interface (Komut Satırı Ara yüzü)
COAP : Constrained Application Protocol (Kısıtlı Uygulama Protokolü)
DLT : Distributed Ledger (Dağıtık Defter Teknolojisi)
DTLS : Datagram Transport Layer Security (Datagram Aktarım Katmanı)
ECC : Elliptic Curve Cryptography (Eliptik Eğri Kriptolojisi)
GPS : Global Positioning System (Küresel Konumlama Sistemi)
HTTP : Hypertext Transfer Protocol (Hiper Metin Transferi Protokolü)
IBM : International Business Machines (Uluslararası İş Makineleri)
IOT : Internet of Things (Nesnelerin İnterneti)
IWF : IoT World Forum (Nesnelerin İnterneti Dünya Forumu)
MSP : Membership Service Providers (Üyelik Servis Sağlayıcıları)
MQTT : Message Queuing Telemetry Transport (Mesaj Telemetri Aktarımı)
M2M : Machine to Machine (Makineden Makineye)
PKI : Public Key Infrastructure (Açık Anahtar Altyapı)
POS : Proof of Stake (Hisse Kanıtı)
POW : Proof of Work (İşin Kanıtı)
P2P : Peer to Peer (Eşten eşe)
SSD : Solid State Disk (Katı Hal Sürücüsü)
SSL : Secure Sockets Layer (Güvenli Soket Katmanı)
TCP : Transmission Control Protocol (Geçiş Kontrol Protokolü)
TLS : Transport Layer Security (Taşıma Katmanı Güvenliği)
UDP : User Datagram Protocol (Kullanıcı Veribloğu İletişim Kuralları)
URI : Uniform Resource Identifier (Tekdüzen Kaynak Tanımlayıcı)

NESNELERİN İNTERNETİ VERİLERİNİN GÜVENLİ BİR ŞEKİLDE BLOKZİNCİR AĞLARINA AKTARIMI

ÖZET

IoT teknolojileri, endüstride, evlerde ve şehirlerde kelimenin tam anlamıyla günlük hayatımızın her alanında, her yerde yaygınlaşıyor. IoT tabanlı sistemlerin güvenliğini sağlamak, pil, işleme ve depolama gibi IoT cihazlarının doğasındaki eksiklikler nedeniyle zordur. Blokzincir teknolojisi, işlem kayıtlarını güvenli bir şekilde kaydetmek için kullanılan yeni ve popülerliği artan bir yaklaşımdır. Gizlilik, bütünlük, kullanılabilirlik, kimlik doğrulama, yetkilendirme ve hesap verebilirlik gibi bilgisayar ve internet güvenliği sorunlarına potansiyel çözümler sunar. Blokzincir, basitçe, birbirine bağlı bloklardan oluşan merkezi olmayan bir defterdir. Bu çalışmada kullanılan Hyperledger Fabric blokzinciri ağı, IoT cihazlarından elde edilen veriler için gizlilik, veri bütünlüğü, kimlik doğrulama ve veri güvenliği sağlar. IoT veri aktarımlarında yaygın olarak kullanılan MQTT protokolü önerilen yaklaşımın içindedir. Yaklaşım, simüle edilmiş IoT cihazlarına sahip Hyperledger Fabric ağında bir uygulamayla gerçekleştirilmiştir. Önerilen yaklaşım, ağ güvenliği boyutları açısından tartışılmıştır. Yapılan çalışma neticesinde, Hyperledger blokzincir ağının, IoT güvenlik eksikliklerini büyük ölçüde giderilebileceği değerlendirilmiştir.

Anahtar Kelimeler: Blokzincir, Güvenli Veri Transferi, Hyperledger Fabric, MQTT, Nesnelerin İnterneti.

SECURE TRANSFER OF INTERNET OF THINGS DATA TO BLOCKCHAIN NETWORKS

ABSTRACT

IoT technologies are spreading ubiquitously in literally every aspect of our daily life, in industry, homes and cities. Securing IoT-based systems is difficult due to the inherent shortcomings of IoT devices such as battery, processing, and storage. Blockchain technology is a new and increasingly popular approach to securely record transaction records. It offers potential solutions to computer and internet security problems such as privacy, integrity, usability, authentication, authorization and accountability. The blockchain is simply a decentralized ledger made up of interconnected blocks. The Hyperledger Fabric blockchain network used in this study provides privacy, data integrity, authentication and data security for data obtained from IoT devices. The MQTT protocol, which is widely used in IoT data transfers, is in the recommended approach. The approach is implemented with an application in Hyperledger Fabric network with simulated IoT devices. The proposed approach has been discussed in terms of network security aspects. As a result of the study, it was evaluated that the Hyperledger blockchain network can largely overcome the IoT security deficiencies.

Keywords: Blockchain, Secure Data Transfer, Hyperledger Fabric, MQTT, Internet of Things.

GİRİŞ

Dünya üzerinde akıllı cihazların kullanımı günden güne artmaktadır. Ve bu artış akıllı cihazların insan hayatı içeresinde daha etkin bir şekilde yer almasına yol açmaktadır. Kullandığımız akıllı cihazlar kolayca internete bağlanabilmekte ve bu bağlantı sonrası veri alış-verişini yine kolayca yapabilmektedirler. Juniper Research'e göre 2021'in sonuna kadar yaklaşık 46 milyar cihaz nesnelerin internetine bağlı olacağı öngörülmektedir [1].

IoT cihazları dediğimiz bu aygıtlar; akıllı evler, giyilebilir cihazlar, akıllı şehirler, sağlık hizmetleri, otomotiv, çevre, akıllı su, akıllı şebeke vb. gibi alanlarda üretimi optimize ederek ve endüstrileri bilgi teknolojilerine geçirerek kullanım alanını günden güne büyütmektedir. Ancak bu IoT cihazlarının çoğunun ele geçirilmesi ve aktardıkları verilerin değiştirilmesi kolaydır. Tipik olarak bir IoT cihazı bilgi işleme, depolama ve ağ kapasitesi bakımından sınırlı olduğu için saldırılara karşı daha savunmasızdır [2]. IoT sistemlerinde varlıklar arasındaki iletişimi sağlamak için kullanılan IoT ara yazılımı hizmetlerin sağlıklı bir şekilde sağlanması için güvenli bir yapıda olmalıdır. Gerekirse bu güvenli iletişimi sağlamak için farklı tasarımlar, farklı ara yüzler veya farklı ortamlar sisteme dâhil edilmelidir.

Günümüzde finansal kuruluşların ve hükümetin bilgilerine güveniyoruz, ancak diğer harici kuruluşlar tarafından sağlanan bilgilerin hiçbir şekilde tahrif edilmediğinden ve değiştirilmediğinden emin olabilir miyiz? Bu merkezi altyapı kullanan mimarilerde cevaplanması oldukça zor bir sorudur [3]. Güvenilmeyen etkenler kullandığımız verileri kendi çıkarlarına göre değiştirebilir ve bu nedenle sağladıkları bilgiler tamamen güvenilir olmayabilir. Bu noktada katılımcı tüm paydaşların bir güven ortamı içerisinde işlemlerini sürdürebileceği, bilgilerin hiç değişmediğinden emin olabileceği ve bütün bunları doğrulayabileceği bir sistemin ihtiyacı ortaya çıkmaktadır. Burada bahsettiğimiz altyapı, sistemde yer alan tüm paydaşların içerisinde bulundukları ekosistemdeki bütün verilere eşit şartlarda erişebildikleri, eriştikleri bu verilerin sisteme ilk tanımlandığı günden itibaren değiştirilmediğini sorgulayabilecekleri bir

mimariyi içermelidir. Tüm paydaşlara bu ekosistem içerisine bir şekilde dahil olmuş bütün verileri yaşam döngüleri boyunca ayrıntılı bir şekilde görüntüleyebilecekleri bir ortam sunmak gerekmektedir. Bahsettiğimiz gibi burada en önemli hususlar verilerin değiştirilemez ve ekosisteme dahil olan tüm paydaşlar tarafından izlenebilir olmalarıdır.

Somut olarak, Avrupa Birliği düzenlemeleri, gıda üreticilerinin, her birinin nihai varış noktasına ek olarak, gıda ürünlerinin detaylandırılmasında kullanılan tüm hammaddeleri izlemesini ve tanımlamasını gerektirir [4]. Örneğin, binlerce imalat tedarikçisi ve milyonlarca müşterisi olan büyük bir gıda şirketi söz konusu olduğunda, bilgilerin dijitalleştirilmesi ve yönetmeliğe uyması için işlenmesinin otomatikleştirilmesi gerekir. Örneğin üretimi gerçekleşen bir tarım ürününe ait verilerin Türkiye Cumhuriyeti Ticaret Bakanlığı Hal Kayıt Sistemi'ne işlenerek dijital ortamda kayıt altına alınması gerekmektedir [5]. Bu süreç birden çok adım ve birden çok katılımcı içermektedir. Üreticiler, tedarikçiler, müşteriler, toptancılar, perakendeciler, taşımacılar ve depolar bir tedarik zinciri sürecinin içerisinde yer alan ana aktörler olarak gösterilebilir. IoT teknolojileri birçok paydaşın benzer süreçlerde elde edilen verilerin gerçek zamanlı olarak kaydedilebilmesi ve sorgulanabilmesi için bize çok büyük fırsatlar sunmaktadır. IoT teknolojileri kullanılarak akıllı şehirler, tedarik zincirleri, sağlık sektörü ve ulaşım gibi birçok alanda süreçleri dijitalleştirmek artık mümkündür [6]. Ancak IoT teknolojilerinin bize sunduğu bu büyük fırsatlara rağmen bu teknolojilerden elde edilen verilerin güvenilirliğini ve doğruluğunu sağlamak henüz tam olarak aşılamamış bir sorun olarak önümüzde durmaktadır [7].

Bu noktada ilk olarak merkezi ve hiçbir otorite tarafından yönetilmeyen bir kripto para birimi olarak hayatımıza giren Bitcoin'in altında yatan teknoloji olan blokzincirin IoT cihazlarının yönetilmesinde, kontrol edilmesinde ve en önemlisi verinin güvenliğinin, doğruluğunun ve değişmezliğinin sağlanmasında önemli bir rol oynamaya hazır durumda görülmektedir.

Kısaca bahsedecek olursak, 2008 yılında "eşler arası elektronik nakit sistemi" tanımlamasıyla ortaya çıkan Bitcoin [8], herhangi bir merkezî otorite tarafından yönetilmeyen, blokzincir teknolojisi üzerine kurulu olan ve geleneksel para birimlerinden farklı çalışan bir dijital para birimidir. Kriptografik olarak güvence

altına alındığı için Bitcoin bir "kripto para birimi" olarak anılmaktadır. Bitcoin'in ortaya çıkışı ile hayatımıza giren blokzincir kavramı günümüzde diğer birçok alana uygulanmıştır [9]. Bu altyapı ile bir kripto para biriminin daha ötesine geçen uygulamalarda veri değişmezliği garanti edilmektedir.

İçinde bulunduğumuz yüzyılda hayatımızda bulunan dijital verilerin yoğunluğunu oldukça ivmeli bir şekilde artmaya başladı. Bu veriler insanoğlunun işlerini kolaylaştırmak, yaşam kalitesini iyileştirmek ve çevresinde olup bitenlerle alakalı daha aktif bilgi sahibi olmasını sağlamak için yoğun olarak kullanılmakta. Ve bu veriler aktif olarak internet ortamı üzerinden aktarıldığı için sürekli erişilebilir durumdadır. İnsanoğlu ve veriler arasındaki bu yoğun veri transferini sağlayan teknoloji IoT olarak adlandırılmaktadır. İnsanın hayatına değer ve bilgi kazandıracak bu veriler sensörler aracılığıyla toplanmaktadır. Bahse konu olan bu teknoloji insan hayatına çok önemli faydalar sağlasa da verinin güvenliği açısından hala akıllarda bir soru işareti olarak gözükmektedir. Verinin doğruluğu ve güvenliği sağlanamadığı takdirde IoT cihazlarından elde edilen veriler insanların aklında bir soru işareti olarak kalmaya devam edecektir [10]. Oluşturacağımız sistemde verinin güvenliğinin, gizliliğinin ve bütünlüğünün önemli olduğu bir alanda çalışıyorsak, verilerinin tahrif edilmediğinden emin olmak istiyorsak, sistemimizde kişisel mahremiyete yönelik verileri barındırıyorsak ve veri kayıtlarının geriye dönük olarak, tahrif edilmemiş ve kimliği kanıtlanabilir bir biçimde görüntülenecebileceği bir çözüme ihtiyaç duyuyorsak, blokzincir ve IoT entegrasyonu bizlere bu noktada önemli fırsatlar sunmaktadır.

Gerçekleştirdiğimiz çalışmada, blokzincir teknolojisini kullanarak IoT cihazları ve sistemler arasında gerçekleşen iletişimde güvenli, doğruluğu bütün paydaşlar tarafından kabul edilen, merkeziyetsiz ve denetlenebilir bir mimari tasarımının nasıl gerçekleştirilebileceği ortaya konulmuştur. Çalışmada izinli blokzincir alt yapılarından biri olan Hyperledger Fabric ağını, nesnelerin internetine entegre eden ve IoT verilerinin transferi sırasında iletişimin güvenilir bir şekilde gerçekleşmesini sağlayacak bir yaklaşım sunulmuştur.

1. GENEL BİLGİLER

1.1. Nesnelerin İnterneti Kavramı

Nesnelerin interneti kavramı, internet üzerinden herhangi bir cihaza veya sisteme bağlanmak ve bu bağlantı üzerinden veri alışverişi yapmak için sensörler ve yazılımlar aracılığıyla çalışan nesneler ağını tanımlar. IoT teknolojilerinin en büyük amacı, herhangi bir nesnenin herhangi bir yerde, zaman kavramı olmadan ideal olarak herhangi bir yolu/ağı ve herhangi bir hizmeti kullanan herkes tarafından bağlanmasına izin vermektir.

IoT kavramı ilk olarak 1999 yılında Kevin Ashton tarafından tanımlanmıştır. Asthon bu kavramı şu şekilde tanımlamıştır, "Nesnelerin İnterneti, tıpkı İnternet'in yaptığı gibi dünyayı değiştirme potansiyeline sahiptir. Belki daha da fazla [11]." Daha sonra IoT, 2005 yılında Uluslararası Telekomünikasyon Birliği tarafından resmi olarak sunuldu [12].

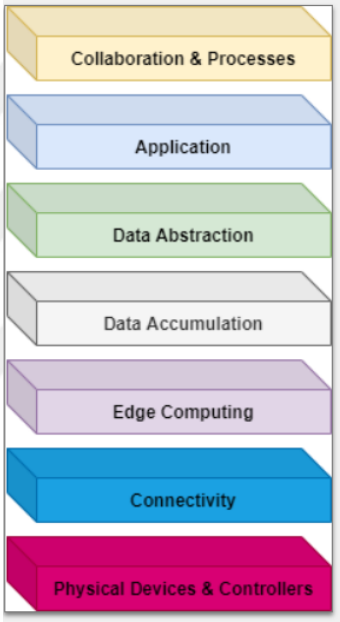
Birçok kuruluş ve kişi tarafından bugüne kadar onlarca kez tanımlanan IoT kavramları arasında en çok kabul gören tanımlama ise Uluslararası Telekomünikasyon Birliği tarafından 2012 yılında yapıldı. Uluslararası Telekomünikasyon Birliğine göre IoT, "Bilgi toplumu için küresel bir altyapı, var olan ve gelişen, birlikte çalışabilir bilgi ve iletişim teknolojilerine dayanan şeyleri (fiziksel ve sanal) birbirine bağlayarak gelişmiş hizmetleri mümkün kılan teknoloji" olarak tanımlanmıştır [13].

Günümüzde IoT teknolojileri büyük bir hızla büyümeye devam etmektedir. IoT sağladığı interaktif bilgi akışının yanında kurum ve kuruluşların milyarlarca dolar tasarruf etmelerini sağlamakta.

Küresel ekonomiye büyük bir katkı sağlamaya devam eden IoT teknolojileri artık hayatımızın her alanında yer almaktadır. İnsan hayatının en önemli alanlarına kadar girmiş olan IoT teknolojilerinin bu gelişimi, beraberinde veri güvenliği ve veri doğruluğunun sorgulanmasına neden olmaktadır.

1.1.1. Iot mimarisi

IWF mimarlık komitesi Ekim 2014'te bir IoT referans modeli yayınladı [14]. Tanımlanan bu model IoT'nin kullanım alanlarının hızla genişlemesi için ortak bir çerçeve sunmaktadır. Referans olarak sunulan bu modelin amacı, tüm paydaşları ortak bir noktada birleştirip işbirliğinin ve IoT teknolojisinin gelişiminin hızla devam etmesini sağlamaktı. Şekil 1.1'de IoT'nin sahip olduğu yedi katmanlı mimari gösterilmektedir. Yedi katmanlı olarak tanımlanan bu mimari belirli işlem türlerinin mimarinin farklı düzeylerinde ve nerede optimize edildiğini belirleyerek katılımcıların sistemlerle uyumlu IoT cihazları geliştirmelerine olanak sağlamaktadır.



Şekil 1.1. IoT Mimarisi

1.1.2. Iot veri güvenliği

Standart bir IoT cihazı/sistemi bir ağ üzerinden birbiri ile bağlantı kuran gömülü sensörlere sahip heterojen cihazlardır. IoT cihazları benzersiz şekilde tanımlanabilirler. Yani her IoT cihazı tekildir. Bu cihazlar genellikle küçük bellek, sınırlı bir işlem kapasitesi ve düşük güçle çalışmak üzere tasarlanır. Ağlar IoT cihazları ile kullanıcılar arasında bir köprü görevi görür. Uzaktan veri akışının sağlanması ve IoT cihazlarını gerçek dünyaya bağlamak için ağlar gereklidir.

IoT cihazları farklı kritiklik seviyelerinde güvenlik sorunları ile karşı karşıyadır. Ancak IoT cihazlarının doğası gereği küçük bellek, düşük güç ve sınırlı bir işlem

kapasitesi ile çalışmasından dolayı güvenlik konusunda önemli eksikleri bulunmaktadır. Ve bu darboğazlardan kaynaklı olarak IoT cihazlarında güvenlik meselesi çok yönlü bir şekilde ele alınmalıdır. IoT güvenliğinde, veri gizliliğinin ve bütünlüğünün korunması çok önemli bir noktadır. IoT cihazlarından alınan veriler nihai noktalarına varasıya kadar birçok katmanda ilerler [15]. Bu katmanlar arasındaki aktarımın güvenli bir şekilde yapıldığından emin olmak için uygun bir şifreleme yönetimi kullanılması gereklidir. Eğer gerekli önlemler alınmazsa saldırılara karşı oldukça korunmasız olan IoT cihazları kötü niyetleri kişilerin açık hedefi haline gelebilir. Bu durumda verilerin değişmezliği ve verilerin bütünlüğünün sağlanmasından söz edilemez.

İkinci önemli problem ise kimlik doğrulama, yetkilendirme ve muhasebedir. Günümüz internet ortamındaki en önemli sorunlardan biri kimlik doğrulama işlemidir. Güvenli ve kanıtlanabilir bir veri transferi için kimlik doğrulama büyük bir ihtiyaçtır. Bir noktadan çıkan verinin nereye gittiğinin bilinmesi iletişimin güvenliği açısından hayati bir önem taşır. Artık hayatının her alanından veri çıkartan IoT cihazları için de kimlik doğrulama en önemli ihtiyaçlardan biridir. Cihazlara ve verilere kimlerin erişebileceğinin tanımlanması verinin kötü niyetli kişilerce ele geçirilmesini engeller. Bu sebeplerle IoT'de iletişimi güvence altına almak için, birbirleriyle iletişim kuran iki taraf arasında kimlik doğrulama gereklidir. Aynı şekilde yetkilendirme mekanizmaları ile IoT sistemlerine veya IoT tarafından sağlanan verilere sadece yetkili kişilerin erişmesi sağlanmalıdır. Kimlik doğrulama ve yetkilendirme işlemleri sistem üzerinde doğru ve etkin bir şekilde uygulanırsa cihazlar/sistemler arasındaki iletişim için güvenli bir ortam sağlanmış olur. Bunların yanı sıra sistemin denetimi ile birlikte raporlama işlemleri de ağın güvenliğinin arttırılması için gereklidir. Son olarak, hizmetlerin konusu ve enerji verimliliği ele alınmalıdır.

IoT cihazları üzerine yapılan saldırı girişimleri genelde IoT cihazlarının hizmet vermesini engellemeye yönelik yapılmaktadır. Bu saldırılar IoT'nin farklı katmanları üzerinde gerçekleşebilmektedir.

Aynı zamanda doğası gereği sınırlı kaynaklara sahip olan IoT cihazlarına yapılan saldırılar cihazın kaynaklarını ve ağı gereksiz meşgul ederek yüksek seviyede enerji tüketimini neden olabilir. Bu durum sonucunda IoT cihazı kullanılamaz hale gelebilir.

1.1.3. Iot veri transferi protokolleri

1.1.3.1. Mqtt

MQTT, 1999'da tanıtılan en eski M2M iletişim protokollerinden biridir. IBM'den Andy Stanford-Clark ve Arcom Control Systems Ltd'den (Eurotech) Arlen Nipper tarafından geliştirilmiştir. Kısıtlı ağlarda hafif M2M iletişimleri için tasarlanmış bir yayınlama/abone olma mesajlaşma protokolüdür [16]. MQTT istemcisi, diğer istemciler tarafından abone olunan veya gelecekteki abonelik için saklanabilecek mesajları bir MQTT aracısına yayınlar. Her mesaj konu [17] olarak bilinen bir adrese yayınlanır. Müşteriler birden fazla konuya abone olabilir ve her konu için yayınlanan her mesajı alır. Taşıma protokolü olarak TCP ve güvenlik için TLS / SSL kullanır.

1.1.3.2. Coap

CoAP, IETF CoRE (Kısıtlı RESTful Ortamlar) çalışma grubundan hafif bir M2M protokolüdür. CoAP hem istek/yanıt hem de kaynak/gözlem (yayınlama/abone olmanın bir çeşidi) mimarisini destekler [16]. CoAP, temel olarak HTTP ve RESTful web ile basit proxy'ler aracılığıyla birlikte çalışmak üzere geliştirilmiştir. MQTT'den farklı olarak CoAP, konular [18] yerine Evrensel Kaynak Tanımlayıcı (URI) kullanır. Yayıncı, verileri URI'ye yayınlar ve abone, URI tarafından belirtilen belirli bir kaynağa abone olur. Bir yayıncı URI'ye yeni veri yayınladığında, tüm aboneler URI tarafından belirtildiği gibi yeni değer hakkında bilgilendirilir. CoAP, UDP'yi bir taşıma protokolü olarak ve DTLS'yi güvenlik için kullanır [19].

1.1.3.3. Amqp

AMQP, John O'Hara tarafından Londra, İngiltere'deki JPMorgan Chase'de 2003 yılında geliştirilen hafif bir M2M protokolüdür. Güvenilirlik, tedarik ve birlikte çalışabilirlik için tasarlanmış bir kurumsal mesajlaşma protokolüdür [20]. AMQP hem istek/yanıt hem de yayınlama/abone olma mimarisini destekler [21]. Güvenilir kuyruğa alma, konu tabanlı yayınlama ve abone olma, esnek yönlendirme ve işlemler [20] gibi mesajlaşmayla ilgili çok çeşitli özellikler sunar. AMQP, varsayılan aktarım protokolü olarak TCP'yi ve güvenlik için TLS/SSL kullanır [21].

1.1.4. Iot uygulamaları

IoT sistem/cihazları yaşantımızdaki neredeyse tüm fiziksel dünyayı, sanal dünya ile birbirine bağlayarak insan yaşamının kalitesini ve etkinliğini arttırmak için kullanılabilir. Bu bölümde IoT teknolojilerinin hayatımızda hangi alanlarda kullanıldığını açıklayacağız.

IoT uygulamaları, tedarik zinciri ve lojistik alanlarında ürün ömür devri süreçlerinin takibi için aktif olarak kullanılmaktadır [22]. Ürünler çeşitli sensörler kullanılarak üreticiden dağıtım konumuna kadar takip edilebilir. Özellikle günümüzde oldukça önem kazanan gıda kalitesinin bilinmesi konusunda IoT cihazları aktif olarak kullanılabilir. Örneğin üreticiden çıkıp dağıtıcıya doğru yol alan ve belirli bir sıcaklıkta nakledilmesi gereken gıda ürünlerinin bu yolculuk boyunca hangi ortam sıcaklığında taşındığı bir sıcaklık sensörü aracılığıyla kayıt altına alınabilir. Ve bu da müşteri de o ürüne karşı bir güven duygusu yaratır. Ayrıca tekil takibi gereken ürünler için de benzersiz tanımlamayı mümkün kılan IoT cihazları eşsizdir. Konum takibi gereken durumlarda da GPS gibi sensörler kullanılabilir. Kısaca IoT teknolojileri hem operasyonel verimlilikleri hem de gelir fırsatları ile tedarik zincirinin bir numaralı aktörü olmaya adaydır.

Sağlık hizmetleri alanında da IoT günümüzde aktif şekilde kullanılmaktadır [23]. IoT teknolojilerinin getirdiği anlık veri takibi ve anlık uyarı sistemleri gibi yenilikler sağlık sektörü için hayati bir önem taşımaktadır. IoT cihazları hastalarının sağlık durumlarının takibinde sağlık ekibi için en önemli yardımcılardır. Hastanın cihazlara bağlı geçirdiği tedavi süresi boyunca sağlık durumu anlım olarak takip edilir. Herhangi anormal bir durumda sağlık ekibi uyarılır.

IoT çözümlerinin en sık kullanıldığı alanlardan biri de 'akıllı şehirler' kavramıdır [24]. Burada bahsedilen 'akıllı' kavramı IoT teknolojileri ile can bulur. Akıllı şehirler kapsamında şehirle alakalı yaşamsal fonksiyonlara ait verileri izlemek, toplamak için sensörler kullanılır. En güzel örneklerden birisi çok kurak geçen 2020-2021 kış aylarında anlam bulmuştur. Barajların doluluk oranlarını ölçen sensörlerin ne kadar önemli olduğu bu kuraklık ortamında ön plana çıkmıştır. Ayrıca kamu hizmetlerinin ve şehir altyapısının her alanında gelişime katkı sağlamak için IoT teknolojileri

kullanılır. Bunlar arasında akıllı sokak aydınlatması, akıllı park yönetimi ve akıllı trafik yönetimi gibi şehri yaşanılabilir kılacak maddeler vardır.

Pazar payını hızla arttırmaya devam eden giyilebilir cihazlar teknolojisi de IoT'nin sıkça kullanıldığı alanlardan biridir [25]. Özellik Apple, Google ve Samsung gibi önde gelen şirketler giyilebilir teknolojiler konusunda birbirleri ile yarış halindeler.

Statista'ya göre, bağlı giyilebilir cihaz sayısının 2022 sonunda 1 milyara ulaşması bekleniyor.

Genel anlamda bir giyilebilir cihaz tamamen sensörlerle donatılmıştır. Örnek olarak bir akıllı kol saati içerisinde GPS sensörü, nabız ölçme sensörü ve kandaki oksijen basıncını ölçebilecek bir sensör yer alabilir. Ayrıca bu cihazlar kablosuz internet ve bluetooth ile başka cihazlara da bağlanabilirler. Günümüzde en yaygın giyilebilir cihazlar akıllı kol saatleri, akıllı fitness kol bandları ve akıllı gözlüklerdir.

1.2. Blokzincir

Blokzincir teknolojisi Bitcoin kripto para birimi ile ortaya çıktığı için gün geçtikçe popülaritesini artırarak gelişmeye devam ediyor. Blokzinciri temelde merkezi olmayan, dağıtılmış ve değiştirilemez bir defter olarak bilinir [26].

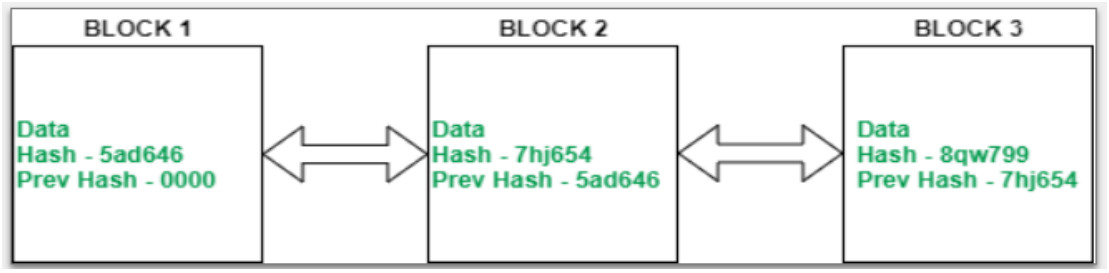
Bu dağıtılmış defter P2P varlıkların ve işlemlerin kaydını tutan ve paylaşılan bir defterdir. Blokzinciri veri doğrulama ve bütünlük için güçlü kriptografik kanıt sağlamak için ECC ve SHA-256 hash kullanır.

Blokzincir blokları temelde içerisinde gerçekleşen tüm işlemlerin listesini ve kendinden bir önce gelen bloğun bir özet değerini içermektedir.

Blokzincir ağda gerçekleşmiş olan tüm işlemlerin eksiksiz ve değiştirilmemiş olmasını garanti edebilecek şekilde bilgisine sahiptir.

Bir blokzincire eklenecek olan her yeni işlem ağda bulunan bütün paydaşların onayını alacak şekilde işleyen bir mutabakat mekanizmasına sahiptir.

Tipik bir blokzincir bloğu Şekil 1.2'deki gibidir.



Şekil 1.2. Blokzincir blok yapısı

Herkese açık bir blokzincir defteri ilk işlemin gerçekleştiği tarihten bu yana bütün işlemlerin sorgulanıp görüntülenebilmesine imkan sunar. Bu da şeffaf ve denetlenebilir bir yapıyı beraberinde getirmektedir.

1.2.1. Blokzincir temelleri

Bilişim teknolojisi sektörü, blokzincirin getirdiği özelliklerle veriler üzerinde şeffaflık, güvenlik ve denetlenebilirlik yetenekleri kazanmış görünüyor.

Geleneksel yani ilişkisel olarak tabir ettiğimiz veri tabanları birden çok tablodaki verileri birincil ve ikincil anahtarlarla birbirine bağlayarak sorgulama imkanı sunar. Tablolarda bulunan sütunlar bulunduğu alanı açıklar. Ve tablolarda bulunan satırlar veri tabanında saklanan bir kaydın tüm bilgilerini içerir.

Geleneksel ortamlar için bir veri tabanı sistemin ana unsuru olduğu gibi blokzincir sisteminin kalbinde de defter diye tabir ettiğimiz bir veri tabanı bulunmaktadır. Blokzincir defterlerinin geleneksel veri tabanlarından farkı standart olarak satırlar ve sütunlardan oluşmamasıdır. Blockhain defterleri blok diye tabir ettiğimiz ve zincir halinde birbiri ile bağlanan işlemlerin tümünü ifade eder. Bir defterde bulunan bütün bloklar birbirlerine bağlıdırlar. Bu bağlantı bir bloğun kendin önce gelen bloğun bilgilerini içeren bir hash değerini saklaması ile sağlanır. Blokzincir defterleri doğası gereği merkeziyetsiz, dağıtık ve değiştirilemez veri özelliklerini sunar.

Blokzincirin en önemli kavramlarından biri bloklardır. Blokzincirde bulunan her blok SHA256 şifreleme karma algoritması ile tanımlanmaktadır. Her blok kendinden önce gelen blok bilgilerini içeren şifrelenmiş bir değer taşır. Her blok bu şifrelenmiş değerler listesiyle birbirine bağlanır. Önceki blok hash'i yeni bloğun başlığında yer alır. Herhangi bir şekilde blokzincirde bulunan bir hash değeri değişirse ağda bulunan

bütün değerlerin değişmesi gerekir. Bu birbirini tetikleyen durum tüm blok değerlerinin yeniden hesaplamasını gerektirir. Bu yeniden hesaplama işlemi sistem üzerinde çok büyük boyutta bir yük getireceğinden gerçekleştirilmesi çok zordur. Bu da bir blokzincirin değiştirilemez olmasını sağlar.

Bloklar zincire eklendikten sonra açık bir blokzincir ağında herkes tarafından görülebilir olur. Blok içerisinde meta verileri ile birlikte gerçekleşen işlemlerin uzunca bir kaydının tutulduğu işlem listesi vardır.

Blokzincirde bulunan bir blok başlığı içerisinde üç parça üst veri bulunmaktadır. İlk parça mevcut bloğu bir önceki bloğa bağlayan şifrelenmiş bir özet değeri, ikinci parça zorluk diye adlandırılan ve mevcut blok için iş kanıtı algoritması zorluk hedefi anlamına gelen değer, zaman damgası ve nonce diye tanımlanan ve iş kanıtı algoritması için kullanılan bir sayaç anlamına gelen değerdir. Üçüncü parça ise merkle tree root dediğimiz ve mevcut bloğun işlemlerinin merkle ağacının kökünün bir karması olarak tutulan bir değerdir.

Bloklar hakkında bahsetmemiz gereken iki önemli nokta daha bulunmaktadır. Bunlardan ilki blok yüksekliğidir. Blok yüksekliği bir blokzincirin oluştuğu günden bugüne kadar zincire eklenmiş olan blok sayısını ifade eder. İkinci önemli nokta ise genesis bloktur. Genesis bloğu bir blokzincir oluştuğu anda gerçekleşen ilk işlemde zincire eklenen ilk bloktur. Ve doğal olarak zincirdeki tüm blokların doğuş noktasıdır. Bu da demek oluyor ki bir blokzincirdeki herhangi bir bloktan geriye doğru tarama yaptığınızda varacağınız nokta genesis bloğudur.

Blokzincir temel kavramlarından biri de hash fonksiyonlarıdır. Hash fonksiyonları blokzincirde yaygın olarak kullanılan kriptografik algoritmalardandır. Hash fonksiyonları blokzincirde veri bütünlüğünü korumak için tasarlanmıştır. Verilere ait güvenilir bir hash değeri verildiğinde, verilerin hash'ini hesaplamak ve iki değeri karşılaştırmak mümkündür. Karşılaştırılan iki veri eşleşirse veriler orjinal hash değeri oluştuğundan beri değiştirilmemiştir denilebilir.

Hash fonksiyonları, blokzincir defterlerinin değişmezliğini korumak için çok önemlidir. Bir blokzincir tarafından kullanılan hash fonksiyonu zarar görürse, saldırganlar çok önemli hash değerlerini tespit edebilirler. Bu, kötü niyetli kişilerin

blokzincir ağının geçmişini daha kolay değiştirmesine ve bir blokzincir sisteminin parçalanmasına neden olur.

İşlemler genellikle bir blok içerisinde genel kayıtlarda saklanan küçük bir görev birimidir. Kayıtlar genellikle blokzincir ağında katılımcı olarak yer alan kullanıcıların çoğunluğunun onayını aldıktan sonra blokzincire eklenir. Orada uygulanır ve saklanır. Bütün işlemler herkes tarafından her zaman görüntülenebilir. İşlemlerin boyutu madenciler için önemlidir; çünkü daha büyük boyuttaki işlemler blokta daha fazla alana ihtiyaç duyar ve bu işlemler daha fazla güç tüketir. Daha küçük işlemlerin doğrulanması daha kolaydır ve daha az güç tüketirler [27].

Blokzincir ağları, ağ içerisinde gerçekleşmiş olan bütün işlemlerin kaydını tutan dağıtılmış defterlerdir. Dünyanın herhangi bir yerinde ağa dahil olmuş kişiler tarafından saklandığı için bu şekilde adlandırılırlar. Dağıtık deftere işlenen veriler ağda bulunan tüm katılımcıların onayını alarak deftere kaydedilir. Buna mutabakat mekanizması denir.

Çeşitli blokzincir ağlarında kullanılan birden fazla mutabakat mekanizmaları vardır. Bunlardan en çok kullanılanları ise PoS ve PoW mekanizmalarıdır.

31 Ekim 2008'de Satoshi Nakamoto takma isimli biri veya birileri tarafından yayınlanan Bitcoin teknik dokümanında [8], karşımıza daha önceki kullanım alanlarından farklı bir şekliyle ortaya çıktı. İlk kriptopara Bitcoin'i ortaya çıkaran "Bitcoin: Eşler Arası Elektronik Nakit Sistemi" isimli doküman, Proof of Work protokolü sayesinde güvenilir bir ödeme sistemi ve kriptopara biriminin nasıl var olabileceğini ortaya koymuştur.

PoW protokolünün çalışma mantığı şu şekildedir; Bitcoin ve diğer pek çok kriptopara birimi, merkezsiz düğümlerin (node) bir araya gelerek oluşturduğu ağlar vasıtasıyla korunan blokzinciri sistemleridir. Bu sistemlerin temel görevi ağın sürdürülebilirliği ve sürekliliğini sağlamaktır. Ağda madenci ismi verilen ve blokzincirine yeni bloklar eklemekle görevli operatörler bulunmaktadır. Bu blokların eklenebilmesi ise bazı karmaşık matematik problemlerinin çözülmesiyle mümkün olabilmektedir. Söz konusu problemleri çözmek hayli zor olduğu için güçlü işlemcilere ihtiyaç duyulmaktadır.

Problemi çözen ve bloktaki işlemleri doğrulayan ilk madenci işlemi ağa yayınlayarak ağda belirlenen kriptopara ödülünü ve bloktaki işlemler için ödenen işlem ücretlerini almaya hak kazanır. Madenciler tarafından doğrulanan ağda yayınlanan işlemler blokzinciri olarak adlandırılan dağıtık defter sistemine kaydedilir.

Ağa bağlı tüm kullanıcılar, blokzincirinin bir kopyasını indirebilir ve aynı zamanda onaylanmış tüm blokların doğrulamasını gerçekleştirebilir.

Bitcoin ağında kullanılan PoW protokolü, işlemci gücünün çoğunu elinde tutan madencilerin ağda daha fazla söz hakkına sahip olduğu ve dolayısıyla daha fazla getiri elde ettiği bir sistemdir.

Bitcoin madenciliği, yüksek enerji tüketimine ihtiyaç duymaktadır ancak PoS protokolü ağ gücünü işlemci gücüne bakarak dağıtmaz. PoS'ta bir sonraki bloğun üretimi birkaç kombinasyonu aynı anda yerine getiren operatörler tarafından gerçekleştirilebilir. PoS protokolünün birden çok türü vardır.

PoS protokolünde, işlem doğrulayabilmek ve gelirden pay almak isteyen kullanıcılar, kriptopara varlıklarını, doğrulama için kullanılmak üzere kilitlemeleri gerekir. "Staking" (gelirden pay alma) olarak adlandırılan bu kilitleme işleminde, cüzdanda, bu işlem için kullanılmak istenen tutar, kilidi kaldırılana kadar cüzdandan çekilemez ve kullanıcının hissesi olarak ağda işaretlenir.

PoS protokolünü kullanan blokzincirlerinde, kullanıcılar, blok doğrulama ödüllerini ve diğer kullanıcıların ödediği transfer ücretlerini (madenci ücreti) hisseleriyle orantılı olarak paylaşır.

Bu işlemi, halka açık bir şirketin hisselerine sahip olmaya benzetebiliriz. Daha fazla hisse sahibi olan kişilerin, şirketin dağıttığı kardan daha yüksek pay alması gibi, "staking" için daha fazla kriptopara kaynağı ayıran kullanıcılar da gelirden daha yüksek pay alırlar.

1.2.2. Blokzincir anahtar özellikleri

Bu bölümde bir blokzincir ağının sahip olduğu altı temel özellikten bahsedeceğiz. Şekil 1.3'te verilen görsel bunun tam bir özetini sunmaktadır.

Şekil 1.3. Blokzincir altı temel özellik

Blokzinciri değerli hale getiren özelliklerden ilki ademi merkeziyetçi yapısıdır. Geleneksel olarak tüm insanlığın alışık olduğu ve hala daha en yaygın durum olan tüm bilgilerin tek bir merkezde saklanması günümüzde birçok probleme gebedir. Verilerin saklandığı cihaz olağan dışı bir şekilde çalışmaz ve kullanılamaz hale gelirse veri tabanında bulunan bilgilere gerçek zamanlı olarak erişemez duruma geliriz. Bu da birçok işin aksamasına sebebiyet verir. Bu durum kurumsal ortamda neden merkeziyetsiz yapılara ihtiyaç duyduğumuz açık bir şekilde ortaya koymaktadır. Blokzincir teknolojisi ise bu soruna bir çözüm getirmiş durumda. Bir blokzincir ağında veriler dağıtılmış bir defter veri tabanı olarak karşımıza çıkmaktadır. Bir blokzincir ağında bulunan veriler sadece tek bir ortamda saklanmaz ve kontrol edilmez. Ağda bulunan diğer tüm paydaşlarda da eş zamanlı olarak aynı veriler tutulmaktadır. Bu da geleneksel olarak tek bir merkezde tutulan verilerin karşı karşıya olduğu tehditten bizi kurtarmaktadır.

Geleneksel bir veri tabanının aksine herkese açık bir blokzincir ağında bütün bilgiler halka açık olarak sergilenir. Herkese açık bir blokzincir ağında tüm işlemler fikir birliği algoritmaları aracılığıyla onaylanır ve ağa eklenir. Bu da tek başına ağın bütününde bir değişiklik yapılamayacağı anlamına gelir. Gerçekleştirilen tüm işlemler ağda bulunan bütün düğümlerin defterine kaydedilir.

Tüm bu özellikleri ile blokzincir teknolojisi tüm paydaşlar tarafından izlenebilirliği en üst düzeyde sağlar.

Blokzincir teknolojisinin sağladığı en önemli katkılardan biri de iş süreçlerine kattığı şeffaflıktır.

Blokzincirin sahip olduğu en önemli özellik olan değişmez veri kavramı, sisteme eklenen herhangi bir verinin değiştirilemeyeceği veya kaldırılamayacağı anlamına gelir. Bu durum paydaşlar arasındaki güveni arttırarak, hesap verilebilirliği sağlar. Bütün paydaşların blokzincir ağını gerçek zamanlı olarak izleyebilmesini sağlar. Blokzincir özellikle sağlık ve tedarik zinciri sektörlerinde şeffaflık yönünden büyük katkı sunar.

Veri gizliliği ilkesi, izin verilen blokzincir ağlarının önemli bir özelliğidir. Özellikle izinli blokzincir ağlarında yalnızca düğümün yöneticisi tarafından izin verilen kullanıcılar verileri görüntüleyebilir. Bunların yanında organizasyonun onaylı kullanıcılara esneklik ve şeffaflık sunmasına yardımcı olur.

Blokzincir ağlarının verilerin değiştirilmesine yönelik koruma sağlayan açık anahtar yapısı güvenlik sorunlarını büyük oranda ortadan kaldırır. Bir blokzincir ağının katılımcıları ve fikir birliği mekanizması veri güvenliğini arttıran diğer etkenlerdir. Bütün bunlara ek olarak blokzincir ağında işlemler birden fazla paydaşın onayıyla ilerlediği için tek hata noktası ortadan kalkmış olur.

Akıllı sözleşmeler blokzincir ağlarında bulunan paydaşların arasındaki iş süreçlerini tanımlayan birer kod parçalarıdır.

Paydaşlar arasındaki süreçleri otomatikleştirmeye yarayan akıllı sözleşmeler, sözleşme içerisindeki bir işlevin şartlarının yerine getirildiği zaman sıralı eylemleri icra eder. Akıllı sözleşmelerin sürece olan bu katkısı ile işlemler maliyet etkin, zaman açısından verimli ve güvenli bir şekilde gerçekleşir.

1.2.3. İzinli ve izinsiz blokzincir ağları

İzinsiz bir blokzinciri ağına hemen hemen herkes katılabilir ve her katılımcı anonim durumdadır. Öte yandan, izinli blokzinciri ağlarında belirli bir derecede güven sağlayan bir ağ modeli altında çalışan bilinen, tanımlanmış ve sıklıkla incelenen katılımcılar arasında bir blokzinciri çalıştırılır [28]. İzinli bir blokzinciri ağları ortak bir amacı olan ancak birbirine tamamen güvenmeyen bir katılımcı kümesi arasındaki

etkileşimleri güvence altına almak için bir çözüm sunar. Katılımcıların kimliklerini ispat ederek, zaman ve işlem gücü maliyetli bir madencilik işlemi yerine geleneksel bir yöntem olan BFT konsensüs protokolü kullanabilir. Katılımcılar birbirleri tarafından bilinir ve ister uygulama işlemlerini göndermek, ister ağ yapılandırmasını değiştirmek veya akıllı bir sözleşme uygulamak olsun, tüm eylemler, ağ ve ilgili işlem türü için oluşturulan bir onay politikasının ardından blokzincirine kaydedilir.

1.3. Hyperledger Fabric Blokzincir Platformu

Hyperledger Fabric kurumsal bağlamlarda kullanılmak üzere tasarlanmış, açık kaynaklı, kurumsal düzeyde izinli bir DLT platformudur [29].

Fabric, rakip blokzincirlerinin aksine akıllı sözleşmenin kodlanmasında birden çok programlama dilini desteklemektedir. Bu programlama dilleri bugün için Java, Go ve Node.js olarak sunulmaktadır [30].

Fabric platformunda izinsiz ağın aksine, katılımcılar anonim olmak yerine birbirlerini tanırlar. Bu, katılımcılar birbirlerine tam olarak güvenmeseler bile bir ağın katılımcısı olmaları durumunda güvene dayalı bir yönetişim modeli altında çalışabilecekleri anlamına gelir.

Hyperledger Fabric platformunu farklı kılan önemli özelliklerinden biri de platformun belirli kullanım durumlarına göre özelleştirilmesini sağlayan tak-çıkar dediğimiz türde fikir birliği protokollerini desteklemesidir [31]. Örneğin, Fabric ağı güvenilir bir otorite tarafından çalıştırıldığında, CFT fikir birliği protokolü fazlasıyla yeterli olabilirken, çok taraflı, merkezi olmayan bir kullanım durumunda, daha geleneksel bir BFT uzlaşma protokolü gerekebilir.

Tüm bu modüler özellikleri ile Fabric, hem işlem işleme hem de işlem onay gecikmesi açısından iyi performans gösteren platformlardan biridir [32].

1.3.1. Hyperledger fabric anahtar kavramlar

1.3.1.1. Kimlik yönetimi

Fabric ağında eşler, sipariş verenler ve istemci uygulamaları gibi birden fazla aktör bulunmaktadır. Bu aktörlerin her biri kendisini tanımlayan X.509 dijital sertifikasında

saklanmış bir dijital kimliğe sahiptir. Bu kimlikler aktörlerin ağdaki izinlerini ve blokzinciri ağında sahip olduğu bilgilere erişimi belirlerler.

Fabric ağında bir dijital kimliğin doğrulanabilir olması için güvenilir bir otoriteden gelmesi gerekir. MSP, Fabric'te güvenilen otorite olarak çalışmaktadır. MSP kavramı ağ içerisindeki bir organizasyonun geçerli kimliklerini yöneten ve kurallarını tanımlayan bir bileşendir. MSP uygulaması, X.509 sertifikalarını kimlik olarak kullanır ve geleneksel bir PKI modelini benimser.

1.3.1.2. Sertifika otoritesi

Organizasyonların ve aktörlerinin doğrulanabilir bir dijital kimliğe sahip olmasının temelini oluşturan bileşen "Sertifika Otoritesi"dir. Fabric ağında, ağ ile etkileşimde bulunmak isteyen her aktörün bir kimliğe ihtiyacı vardır.

Dijital kimlikler, X.509 standardıyla uyumlu ve bir CA tarafından verilen, kriptografik olarak doğrulanmış dijital sertifikalar biçimindedir.

"Sertifika Otoritesi"nin çok önemli olmasından dolayı Fabric, oluşturulan blokzinciri ağlarında "Sertifika Otoritesi" oluşturulmasına olanak tanıyan yerleşik bir "Sertifika Otoritesi" bileşeni sağlar. Fabric CA olarak bilinen bu bileşen, X.509 sertifikaları biçimine sahip Fabric katılımcılarının dijital kimliklerini yönetebilen özel bir kök CA sağlayıcısıdır.

1.3.1.3. Üyelik servis sağlayıcısı

Üyelik servis sağlayıcısı, ağın yapılandırmasına eklenen ve bir organizasyonu hem içeriden hem de dışarıdan tanımlamak için kullanılan bir dizi klasördür. Sertifika Otoritesi tarafından kimliklendirilen organizasyonların ve aktörlerin kimliklerinin bir listesini içerir. Üyelik servis sağlayıcısı, üyelerinin kimliklerini listelerken, hangi CA'ların üyeleri için geçerli kimlikler vermeye yetkili olduğunu belirtir.

Bir kullanıcı Fabric CA'ya kaydolduğunda, yönetici, eş, kullanıcı, sipariş veren veya üye rolünün kullanıcıyla ilişkilendirilmesi gerekmektedir. İşte yeni bir aktörün bir düğümde veya kanalda sahip olduğu belirli ayrıcalıkları tanımlayarak kimliği bir role dönüştüren "Üyelik Servis Sağlayıcısı"dır.

1.3.1.4. Politikalar

Politikalar Fabric ağının yönetim mekanizmasıdır. Politikalar organizasyonların veya aktörlerin ağ, kanal ve akıllı sözleşmedeki değişiklikleri kabul etme veya reddetme konusunda nasıl mutabakata varacaklarını belirler. Politikalar bir kanal oluşturulduğunda kanal katılımcıları tarafından kabul edilir, ancak zaman içerisinde değiştirilebilirler. Örneğin, bir kanalda blokların nasıl oluşturulduğunu veya akıllı bir sözleşmeyi onaylamak için gereken organizasyon sayısını belirtirler. Özetle Fabric ağında yapılmak istenen bir şeyin nasıl yapılacağı politikalar tarafından belirlenir.

1.3.1.5. Eşler

Bir Fabric blokzinciri ağı, eşlerden oluşur. Eşler, ağın temel unsurlarıdır çünkü defterleri ve akıllı sözleşmeleri barındırırlar. Akıllı sözleşmeler ağdaki iş yapış şekillerini belirlerken, defterler yapılan işlemler sonucunda verileri tutan mekanizmalardır. Yani akıllı sözleşmeler defterlere erişebilen kod parçalarıdır. Bir eş bağlantı aracılığıyla, uygulamalar bir defteri sorgulamak veya güncellemek için akıllı sözleşmeleri çalıştırabilir. Tek bir eş üzerinde, kayıtlı olduğu kanal sayısına göre birden çok defter ve akıllı sözleşme olabilir. Bir eş, defterler ve akıllı sözleşmelere erişim için bağlantı noktası olduğu için uygulamalar bu noktalara erişmek istiyorlarsa, bir eşle etkileşime girmelidirler. Bu nedenle eşler, Fabric ağının en temel yapı taşları olarak kabul edilmektedir.

1.3.1.6. Defterler

Hyperledger Fabric'te defter olarak adlandırılan veri tabanı dünya durumu ve blokzincir olarak iki farklı parçadan oluşur. Dünya durumu, fabric ağındaki tüm işlem günlüğünü tutmak yerine, bir varlığın mevcut değerine doğrudan erişilmesini kolaylaştırır. Dünya durumunda varlıklar anahtar-değer çiftleri olarak ifade edilir. Yani bir varlığın anahtar değeri bilindiğinde dünya durumu üzerinden güncel hali sorgulanabilir. İkinci olarak, varlıkların mevcut dünya durumuna neden olan tüm değişikliklerini kaydeden ve bir işlem günlüğü olan blokzinciri vardır. İşlemler, blokzincirine eklenen bloklar içinde toplanır ve varlıklar üzerinde yapılmış olan değişikliklerin geçmişini anlamanıza yarar. Blokzinciri veri yapısı içerisinde yapılan bir işlem bir kez yazıldığında değiştirilemezdir.

1.3.1.7. Sipariş hizmeti

Fabric blokzinciri ağ içerisinde birden fazla sipariş düğümü oluşturmaya izin veren yapısı ile birlikte bir sipariş hizmeti oluşturan ve sipariş veren adı verilen bir düğüme sahiptir. Fabric tasarımı gereği fikir birliği algoritmalarına dayandığından, yapılan bir işlem sonucunda üretilen bir bloğun doğru olduğu garanti edilir. Bu durum Bitcoin gibi izinsiz ağlarda karşılaştığımız blok çatallanması sorununu ortadan kaldırır. Sipariş verenler eşlerde yürütülen akıllı sözleşme simülasyonlarından gelen onayları toplar. Ve onay sonuçlarına göre yeni bir blok yaratma işlemini başlatır. Yeni bir blok yaratıldıktan sonra eşlere dağıtılarak yayımlanır.

Fabric ağında sipariş verenler Raft konsensüs algoritmasını kullanır. CFT olan bu algoritmada lider ve takipçi modeli kullanılmaktadır. Örneğin ağı tasarlarken üç tane sipariş veren hizmeti yaratıldığı düşünüldüğünde birinin çökmesi veya erişilemez olması durumunda dahi kalan iki sipariş verenin hizmet vermeye devam etmesi beklenir. Raft konsensüs algoritmasının bu özelliği yüksek seviyede erişilebilirlik ve kullanılabilirlik sunmaktadır.

1.3.1.8. Akıllı sözleşmeler

Akıllı sözleşmeler ağda gerçekleştirilecek olan iş mantığının otomatik olarak yürütülmesine olanak sağlayan kod parçalarıdır. Fabric ağlarında akıllı sözleşmeler Java, Go ve Node.js programlama dilleri ile kodlanabilmektedir. Örneğin, akıllı bir sözleşme, belirli bir zaman çerçevesinde yeni bir araba teslimatının yapılmasını otomatik olarak sağlayabilir. Akıllı sözleşmenin uygulanması, manuel süreçlerin otomatik olarak işletilmesine olanak sağladığı için oldukça verimlidir.

1.3.1.9. Kanal yapısı

Bir Hyperledger Fabric kanalı temel olarak paydaşlar arasındaki özel ve gizli işlemlerin gerçekleştirilmesi amacıyla kurulan bir alt ağdır. Bir kanal, katılımcı üyeler, eşler, defter, akıllı sözleşmeler ve sipariş verenler tarafından oluşur.

Ağda gerçekleştirilen her işlem, ağ içerisinde kimliği kanıtlanmış ve yetkilendirilmiş kullanıcılar tarafından yürütülür.

1.3.2. Fabric veritabanı

Hyperledger Fabric, iki tür eş durum veri tabanını desteklemektedir. Bunlardan biri olan LevelDB, eş düğümde gömülü olan varsayılan durum veri tabanıdır. LevelDB, zincir kodu verilerini basit anahtar-değer çiftleri olarak depolar ve yalnızca anahtar, anahtar aralığı ve bileşik anahtar sorgularını destekler. CouchDB, defterdeki verilerin JSON olarak modellenmesine ve anahtarlar yerine veri değerlerine karşı zengin sorgulara olanak tanıyan isteğe bağlı, alternatif bir durum veri tabanıdır. CouchDB desteği, sorguları daha verimli hale getirmek ve büyük veri kümelerin sorgulanmasına olanak tanır.

Hyperledger Fabric ağını kurmadan önce hangi veri tabanının kullanıp kullanmayacağına karar verilmesi gerekmektedir. Veri uyumluluk sorunları nedeniyle bir eşin LevelDB'den CouchDB'ye geçişi desteklenmemektedir. Ağdaki tüm eşler aynı veri tabanı türünü kullanmalıdır.

1.4. Yapılan Entegrasyon Çalışmaları

Bu bölümde IoT ve blokzincir entegrasyonu ile alakalı literatürde yapılmış çalışmaların bir özeti ve bu çalışmaların kapsamına dair bir değerlendirme sunulmuştur.

1.4.1. Blockchain as a service for iot

Mayra Samaniego ve Ralph Deters [33], blokzincir'in dağıtılmış, merkezi olmayan ve değiştirilmeye karşı dirençli olan yapısını IoT nin doğası gereği karşı karşıya olduğu bazı sorunları çözmek için kullanılabileceğinden bahsetmektedirler. Blokzincir birçok alanda kullanıldığı gibi IoT cihazları ile entegre edilerek de kullanılmaya çalışılmaktadır. IoT cihazlarının sınırla kaynak kapasitesine sahip olmasından dolayı, IoT cihazları içerisinde bir blokzincir barındırmak neredeyse imkansızdır.

Yazarlar yaptıkları çalışmada bu sorunu çözmek için bulut ve sis tabanlı bir çözüm sunmaktadırlar. Yapılan çalışmada IoT cihazı olarak Intel Edison Arduino kartı kullanılmakta olup blokzincir ise sis ve bulut üzerinde ayrı ayrı çalışmaktadır. Yapılan deneylerde IoT cihazı bir Python sunucusu aracılığıyla blokzincir'e veri yazmaktadır.

Sis üzerinde çalışan blokzincir uygulamasının ağ gecikmeleri sebebiyle bulut üzerinde çalışan blokzincir uygulamasından daha iyi performans gösterdiği gözlemlenmiştir.

1.4.2. Managing iot devices using blockchain platform

Seyoung Huh, Sangrae Cho ve Soohyung Kim yaptıkları çalışmada [34], blokzincir kullanarak IoT cihazlarını kontrol edip, yapılandırmayı hedeflemektedirler. Ethereum blokzincir üzerinde tasarladıkları sistemde akıllı sözleşmeler kullanarak IoT cihazlarının ayarlarını yönetmeyi düşünmektedirler.

Önerilen sistemde bir akıllı telefon ve üç adet raspberry pi bulunmaktadır. Sistemde bulunan üç adet raspberry pi sırasıyla sayaç, klima ve ampul gibi davranmaktadırlar. Kullanıcı akıllı telefon aracılığıyla sistemdeki politikayı ayarlayabilmektedir. Örneğin; sistemdeki elektrik sarfiyatı 150 kw'a ulaştığında elektrik tasarrufu modunda çalışacak şekilde sistem güncellenebilmektedir. Akıllı cihaz üzerinden yapılan konfigürasyon değişiklikleri Ethereum ağına kaydedilmektedir. Ethereum ağına kaydedilen bu veriler periyodik olarak ampul ve klima tarafından okunmaktadır. Ayrıca sayaç olarak kullanılan cihaz sürekli olarak elektrik kullanımını izleyerek Ethereum ağına verileri eklemektedir.

Tasarlanan sistemde üç adet akıllı sözleşme fonksiyonu bulunmaktadır. Bunlardan ilki sayaç değerlerini takip etmektedir. Klima ve ampülün konfigürasyonlarını kaydetmek için bir akıllı sözleşme bulunmaktadır. Ayrıca hesabın kimliğini doğrulamak için de bir akıllı sözleşme sistemde yer almaktadır.

Yapılan çalışma sonucunda yaklaşık 12 saniyelik bir veri işleme hızı yakalansa da bazı sistemler için bu hızın yeterli olmadığından bahsedilmiştir. Zaman kavramının önemli olduğu sistemlerde bu işlem hızının yeterli olamayabileceği değerlendirilmektedir.

1.4.3. Blockchain everywhere - a use-case of blockchain

Thomas Bocek, Bruno B. Rodrigues, Tim Strasser ve Burkhard Stiller [35], IoT cihazlarından alınan verilerin, akıllı sözleşme kullanan bir blokzincir ağına kaydedilmesindeki en önemli faydanın, bu verilerin otomatik olarak değerlendirilerek yine otomatik olarak gönderene veya alıcıya bildirilebilmesi olarak açıklamışlardır.

Blokzincir doğası gereği izinsiz veri değişimine karşı dayanıklı olmasından dolayı tıbbi ürün verilerinin sağlıklı ve güvenilir bir şekilde tutabileceği ve otoriteler tarafından yapılacak denetimlerde bu verilerin güvenilir bir şekilde kullanılabileceğinden bahsetmektedirler.

Ethereum tabanlı merkezi olmayan bir sistem öneren yazarlar, tedarik zincirindeki birden fazla paydaşı bir araya getirerek süreçleri otomatikleştirmek ve böylece blokzincirin sağladığı güven ortamını tesis ederek maliyetlerden tasarruf etmeyi amaçlamışlardır.

Yazarların sunduğu yaklaşımda Ethereum Blokzincir ağı, akıllı sözleşme, veri tabanı, sunucu, mobil cihazlar ve sensörler ana bileşenler olarak gösterilmiştir. Sensörlerden alınan sıcaklık veri akıllı sözleşmeler aracılığıyla Ethereum ağına kaydedilmektedir. Uygulama üzerinde çalışan sunucu, Ethereum ağına yeni katılan ve akıllı sözleşmelerdeki güncellemeleri takip eden, yeni akıllı sözleşmeler yaratabilen veya akıllı sözleşme fonksiyonlarını çağırabilen bir Ethereum düğümünü barındırmaktadır. Bu düğüm HTTP üzerinden JSON ile iletişim kurmaktadır. Blokzincir ağında saklanamayacak veriler burada PostgreSQL2 veri tabanında saklanmaktadır.

Sistemde çalışan akıllı sözleşme, sensörlerden gelen sıcak değerlerini doğrulayarak blokzincire kaydeder. Mobil istemciler ise REST API kullanarak sunucuyla iletişime geçmektedir. Müşteriler bu mobil istemciler aracılığıyla sistemdeki verileri kontrol edebilmektedirler.

1.4.4. Management and monitoring of iot devices using blockchain

Kristián Košt'ál, Pavol Helebrandt, Matej Belluš, Michal Ries ve Ivan Kotuliak, çalışmalarında blokzinciri tabanlı bir ağ izleme ve yönetim mimarisi önerdiler [36]. Ağdaki yöneticiler, ağ cihazlarının konfigürasyonlarındaki güncellemeleri kontrol ettikleri blokzinciri üzerindeki cihaz konfigürasyonundaki değişiklikleri kaydederek ağ cihazlarını dolaylı olarak kontrol ederler.

Önerilen tasarımda, yetkili ağ yöneticileri dijital kimliklerini kullanarak bir blokzincirine kayıtlı cihaz veya cihaz grubunun konfigürasyon bilgilerini değiştirebilmektedirler. Yeni oluşturulan konfigürasyonun hata oranını en aza

indirgemek için bir sözdizimi doğrulaması ile kontrol edilmektedir. Yöneticinin sertifika bilgileri aynı zamanda yeni konfigürasyonu tanımlamak amacıyla da kullanılmaktadır. Yapılan değişiklik işlemi bir zaman damgası, yönetici kimliği, cihaz kimliği ve şifreli cihaz yapılandırmasıyla bir blok olarak oluşturulur ve blokzincirine kaydedilir. İşlem daha sonra blokzincirindeki diğer eşlere dağıtılır. Ağda bulunan bütün cihazlar bu değişimden haberdar olduklarında kendi özel anahtarlarını kullanarak bu yapılandırmayı kendilerinde uygularlar. Tüm güncelleme geçmişi blokzincirine kaydedilmektedir.

1.4.5. Distributed logistics platform based on blockchain and iot

Nejc Rozman, Rok Vrabic, Marko Corn, Tomaz Pozrl ve Janez Diaci'nin yapmış oldukları çalışmada, IoT ve blokzincir teknolojilerinin tedarik zinciri süreçlerine entegre edilmesine yönelik bir yaklaşım sunmaktadırlar [37].

Çalışmada Blokzincir tabanlı dağıtılmış bir lojistik platform önerilmektedir. Önerilen platformda bir tedarik zincirindeki aktörlerin bir düğüm olarak sisteme eklenmesini içermektedir.

Genesis düğüm adından da anlaşıldığı gibi önerilen platformdaki ilk düğümdür. Yeni kullanıcıların platforma katılabilmeleri için köprü görevi görür ve platforma dair tüm bilgileri barındırır. Yeni oluşacak düğümlerin nasıl ekleneceği platformdaki tüm düğümlerin bir listesi tutulmaktadır. Aynı zamanda genesis düğümü bir akıllı sözleşme olarak da uygulanmaktadır.

Hizmet düğümü tedarik zincirine hizmet sağlayan paydaşları göstermektedir. Bunlar arasında nakliye şirketleri ve depolar örnek olarak gösterilmektedir.

Kullanıcı düğümü platformdaki diğer tüm düğümlerle iletişim kurmak veya iş yapmak isteyen bütün kullanıcıları göstermektedir. Örnek olarak bir mal üreten şirket gösterilebilmektedir.

Ara yüz düğümü aracılığıyla kullanıcılar sistemden haberdar olmaktadırlar. Genesis düğümü tarafından platform hakkında bilgi sunmak için kullanılmaktadır.

Önerilen platformda taşınan bir malın sanal kopyası bulunmaktadır. Bu sanal kopya IoT cihazları ile oluşturulmaktadır. Taşınan malın konumu, sıcaklığı ve nem gibi diğer verilerin izlenmesi sanal kopya aracılığıyla sağlanmaktadır.

1.5. Değerlendirme

Literatürde bulunan entegrasyon çalışmaları genel olarak sadece IoT ve blokzincir entegrasyonu üzerine kurgulanmıştır. Ancak IoT sistemlerinin en büyük dezavantajlarından biri olan veri aktarımındaki güvenliğin nasıl sağlanacağı konusuna bir açıklık getirilmemektedir. Tezimizde önerdiğimiz yöntemde bu entegrasyona tüm yönleriyle bir çözüm sunmaktayız. Bir blokzincir ağı ile IoT cihazlarını entegre edecek yöntem bu alanın henüz çok yeni olmasından dolayı net olarak açıklanmamış bir konudur. Özellikle bu entegrasyonu sağlarken kullanılabilecek bir mesaj transfer protokolünün nasıl çalıştırılabileceği ile ilgili ayrıntılı bir çalışma neredeyse yoktur.

Benzer çalışmalar incelendiğinde genel olarak IoT ve blokzinciri entegrasyonundan bahsedilmiştir. Ancak bu entegrasyon sırasında bir mesaj transfer protokolünün kullanılmasına dair bir çalışma yapılmamıştır. Gerçekleştirdiğimiz tez çalışmasında bu entegrasyon sırasında MQTT protokolünün Hyperledger Fabric ağına nasıl entegre olabileceğine dair bir yöntem sunmaktayız.

Aynı zamanda benzer çalışmalarda herkese açık bir ağ olan Ethereum ağının sıkça kullanıldığını görmekteyiz. Ethereum ağının herkese açık olmasından dolayı veri güvenliği ve gizliliği gibi konularda dezavantajlar getireceği bir gerçektir. Tez çalışmamızda kullanmış olduğumuz Hyperledger Fabric blokzinciri ağının izinli bir yapıya sahip olması, sadece yetkili organizasyonların yetkilendirilmiş kullanıcılarının kendi güvenlik sertifikaları ve anahtarları ile ağ içerisinde işlem yapmasına izin veriyor olması, aynı zamanda sadece bu yetkilendirilmiş kullanıcıların verilere erişebiliyor olması veri gizliliği ve güvenliği konusunda bizlere somut çözümler getirmektedir.

İlgili çalışmalar uçtan uca bu entegrasyonun nasıl yapılabileceği konusunda kapsamlı bir yönteme yer vermemektedir. Gerçekleştirdiğimiz tez çalışmasında;

Hyperledger Fabric'in tüm gereksinimleri ile birlikte sorunsuz kurulmasını sağlayacak yöntemi, ayrı bulut sunucularda Hyperledger Fabric ağının kurulumunu, karşılaştığımız zorlukları ve bu zorluklar için ideal çözümleri, Hyperledger Fabric'in sunmuş olduğu esnek akıllı sözleşme çözümlerini kullanmayı, MQTT ile Hyperledger Fabric ağının nasıl entegre olacağı ve bir Hyperledger Fabric blokzincir ağının dış dünya ile nasıl haberleşebileceğini, ayrıntılı bir şekilde tezimizde sunmaktayız.

Gerçekleştirmiş olduğumuz entegrasyon çalışması kapsamında, Hyperledger Fabric blokzincir ağının esnek bir yapıya sahip olması, bu çözümün tedarik zinciri başta olmak üzere, IoT ve blokzinciri entegrasyonuna için ihtiyaç duyulan diğer kurumsal alanlarda uygulanabileceği değerlendirilmektedir.

2. PROBLEM VE ZORLUKLAR

Günümüzde IoT cihazlarından sağlanan verilerin hiçbir şekilde tahrif edilmediğinden ve değiştirilmediğinden emin olamamaktayız. Bu durum özellikle merkezi mimari kullanan sistemlerde cevap verilmesi en güç sorulardan biridir.

IoT cihazlarından sağlanan verilerin değiştirilmemesini sağlayacak tek yöntem, cihazlardan alınan verilerin sisteme güvenli bir şekilde aktarılması ve verilerin sisteme aktarıldıktan sonra değiştirilmediğini garanti eden ve birden çok paydaşın onayı ile sisteme veri eklemeye olanak sağlayan dağıtık bir yapıdır. Bu yapıda tüm paydaşların IoT cihazlardan gelen verilerin tamamına erişimi olacaktır. Ve bu sayede verilerin sisteme ilk tanıtıldığı andan son anına kadar değiştirilmeden, güvenli bir şekilde sistemde tutulduğu garanti edilecektir.

Bir blokzinciri en temel ifadeyle birbirine bağlantılı kayıtları içeren, merkezi olmayan ve dağıtılmış bir defter teknolojisi olarak tanımlanabilir. Blokzinciri defterine eklenmek istenen bir kayıt ağda bulunan bütün eşlerin çoğunluğunun onayı ile eklenebilmektedir. Bir blokzinciri doğası gereği birbirine bağlantılı bloklar halinde oluştuğundan, verilerin değiştirilmeye karşı korumalı olduğunu garanti eder. Yüksek düzeyde güvenlik ve gizlilik sunan blokzincirin bu ademi merkeziyetçi yapısı günümüzde birçok alana entegre edilmeye çalışılmaktadır. Bu alanlardan en önemlisi IoT teknolojileri olarak karşımıza çıkmaktadır. IoT cihazlarının sınırlı kaynak kapasitesi olmasından dolayı verilerine kaydetmek için ekstra bir sisteme ihtiyaç duyar. Bu noktada kaydedilecek veri için yüksek düzeyde güvenlik, gizlilik, kimlik doğrulama ve cihaz yetkilendirme olanağı sunan blokzincir teknolojileri ön plana çıkmaktadır.

2.1. IoT Verilerini Blokzincir'de Saklamanın Zorlukları

IoT verilerinin blokzincirlerde depolanması birçok fırsatı beraberinde getiriyor olsa da bazı noktalarda hala çözülmesi gereken sorunlar vardır. Bu sorunlardan başlıcaları aşağıda olduğu gibidir;

2.1.1. Gecikme

IoT cihazlarından alınan veriler bir blokzincirine işlem isteği olarak gönderildiğinde ağda bulunan bütün paydaşların onayı ile sisteme eklenmektedir. Blokzincirine işlem isteği olarak gelen bir veri etkileşime girdiği eş üzerinden simüle edilir. Daha sonra bu işlemin diğer tüm akıllı sözleşme çalıştıran eşler tarafından simüle edilip sonuçlarının sipariş verene aktarılması gerekir. Eğer kurduğumuz blokzincir ağında çok fazla sayıda eş bulunuyorsa ve gönderilen verinin boyutu büyükse bu işlemlerin sonuçlanması gecikmelere yol açabilmektedir.

2.1.2. Ölçeklenebilirlik

IoT cihazlarının sayısının günden güne katlanarak artması blokzincir çerçevesinde bizleri ölçeklenebilirlik sorunu ile karşı karşıya getirmektedir. Özellikle herkese açık blokzincir ağlarının bir blok oluşturma süreleri çok uzundur. Bu darboğaz da saniyede milyonlarca veri üreten IoT verilerinin bir blokzincirinde saklanabilmesi için ölçeklenebilirlik sorunlarına yol açmaktadır.

2.1.3. Depolama

Depolama konusunda da bazı kısıtlarla karşı karşıyayız. IoT cihaz sayısının günden güne katlanarak artması üretilen verinin de hızlı bir şekilde artmasına sebep olmaktadır. Bir blokzincir ağını düşündüğümüzde ise ağda bulunan eş sayısı kadar bu veri büyüklüğünü çarpmamız gerekmektedir. Çünkü bir blokzincirinde doğası gereği her eş ağda tutulan bütün verilerin kopyasını defter dediğimiz bir veri tabanında tutmaktadır. Bu durumda ağın depolama kapasitesi açısından verimsiz bir durum olarak gözükmektedir.

2.2. IoT-Blokzincir Entegrasyonu ile Çözülen Problemler

IoT ve Hyperledger Fabric blokzinciri entegrasyonu gecikme, ölçeklenebilirlik ve depolama zorluklarının yanında, henüz çözümlenmemiş birçok noktada ideal çözümler sunmaktadır. Bu entegrasyon özellikle güvenlik konusunda sunmuş olduğu çözümler ile bugün hemen hemen herkesin ilgisini çekmektedir. Bu tez çalışması ile elde etmiş olduğumuz potansiyel çözümler aşağıda olduğu gibidir;

2.2.1. Gizlilik

Hyperledger Fabric ağının sunduğu gizlilik kontrolü ve veri güvenliği ile ilgili çözümlerle gizlilik sağlanmaktadır. Hyperledger Fabric, paydaşlar arasında yetkili bir kanal oluşturarak tanımlı kullanıcılar üzerinden işlem yapılmasına izin veren bir yapıya sahiptir. Hyperledger Fabric bünyesinde oluşturulan organizasyonlar üzerinden kullanıcı oluşturarak işlem yapılmasına imkan veren bu yapı ile yetkisiz kişilerin ağ içerisinde işlem yapması engellenmektedir.

2.2.2. Veri bütünlüğü

Ağın verileri defter denilen dağıtık veri tabanlarında tutularak verilerin bütünlüğü sağlanır. Blokzincir defterleri, blok adı verilen ve bir zincirde birbirine bağlanan tüm işlemleri ifade eder. Bu bağlantı, önceki bloğun kendisinin bilgilerini içeren bir sağlama değerini depolayan bir blokla elde edilir. Bir eş, Hyperledger Fabric durum veri tabanına doğrudan müdahale etmek isterse, diğer eşler durumun böyle olduğuna ikna olmayacağından veriler değiştirilemez. Tüm paydaşların bu dağıtılmış veri tabanlarına sahip olması veri bütünlüğünü sağlar.

2.2.3. Kullanılabilirlik

Hyperledger Fabric blokzincir ağı, dağıtılmış yapısı ile arızalara karşı direnç sağlar ve veriler yetkili kullanıcılar için sürekli erişilebilir durumdadır. Defter veri tabanlarını taşıyan birden fazla eşte, bir eş erişilemez olsa bile veriler diğer sağlıklı eşler üzerinden erişilebilir olacaktır. Hyperledger ağı, daha ucuz emtia donanımı kullanarak birim zamanda yüksek bir işlem sayısı da sağlar.

2.2.4. Kimliklendirme, doğrulama ve yetkilendirme

Hyperledger Fabric API sunucusunda çalışan MQTT yayıncısı, Hyperledger Fabric organizasyonları tarafından tanımlanan yetkili bir kullanıcıyla işlem yapmalıdır. Aynı zamanda, yetkilendirme için MQTT tarafından sağlanan kullanıcı adı ve şifre alanları kullanılarak çift taraflı kimlik doğrulama sağlanır. Hyperledger Fabric kullanıcı kimliği ve MQTT protokolü tarafından sağlanan bu tanımlama ile, protokol aracılığıyla gönderilen verilerin kimliği doğrulanacak ve verilere yetkisiz erişim engellenecektir. Ek olarak, isteğe bağlı MQTT TLS desteği, kimlik doğrulama

protokollerini kullanarak mesajları şifrelemeyi ve istemcilerin kimliklerini doğrulamayı kolaylaştırır.

2.2.5. Hesap verebilirlik

Hesap verebilirlik, önerilen mimaride çalışacak tüm unsurlar bu işlemleri belirli bir kimlikle gerçekleştirebilecek ve bu işlemler, işlemin hangi IoT cihazından, ne zaman ve hangi Hyperledger Fabric kullanıcısından yapıldığı gibi sorulara cevap vererek hesap verebilir bir sistem oluşturulmuş olacaktır.

3. YÖNTEM

Tezimizin bu bölümünde literatürdeki boşluğu doldurmaya aday bir yöntem öneriyoruz. Literatürdeki çalışmaların çoğunun sadece IoT ve blokzincir entegrasyonuna odaklanması, veri aktarımındaki güvenlik, gizlilik, veri bütünlüğü, kimliklendirme ve doğrulama gibi diğer kritik noktalarla ilgili bir öneri sunmaması bu alanın hala tartışmaya açık bir alan olduğunu ortaya koyuyor.

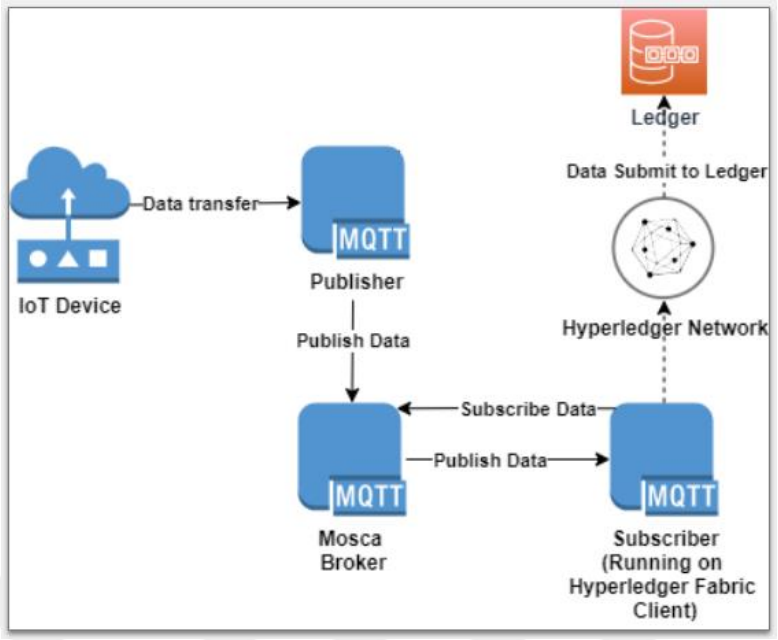
Önerdiğimiz çözümü anlatırken, mimarinin nasıl kurulduğu, karşılaşılan zorluklar, Hyperledger Fabric ağının kurulması, akıllı sözleşme kodlarının yazılması ve MQTT altyapısının kurulması adım adım açıklanacaktır.

3.1. Önerilen Sistem Mimarisi

Bu bölümde önerilen yaklaşımda blokzincir ağı olarak Hyperledger Fabric ve IoT verilerinin aktarımı için MQTT protokolü kullanılmaktadır.

MQTT mesajlarının iletiminde 'Quality of Service (QoS)' (Hizmet Kalitesi) yayıncı ve abone arasında bir mesajın iletilmesine ilişkin bir anlaşmadır. Üç seviyeye sahip olan QoS, seviye sıfırda veri aktarım hacmi açısından en düşük maliyetlidir. Bu, yayıncı ve aracı arasında güvenilir bir bağlantınız olduğunda uygundur. Her mesajın iletildiğinden emin olunması gerekiyorsa seviye bir iyi bir seçimdir, ancak bu durumda IoT uygulamamızın bir mesajı birden fazla kez iletmesini tolere etmeliyiz. Seviye iki ise mesaj iletiminin tam olarak bir kez gerçekleştiğini garanti eder, ancak veri aktarımı açısından nispeten yüksek bir maliyeti vardır. Çalışmamızda MQTT entegrasyonu mesajı yayınla ve unut prensibi olan sıfırıncı seviye ile sağlanmıştır.

Şekil 3.1'de önerilen mimaride, bir IoT cihaz simülatörü (seri veri gönderme işleminde Postman kullanılmıştır) aracılığıyla aktarılacak olan verileri yakalar ve MQTT mesaj yayıncısına bu mesajı yayınlar. Mesaj aracısı alınan mesajları belirlenen konu üzerinden kayıtlı abonelere dağıtır. Uygulamamızda Mosca MQTT mesaj aracısı kullanılmıştır. Fabric client olarak çalışmakta olan bir MQTT abonesi belirlenen konu üzerinden MQTT mesaj aracısına abone olarak simülatörden gelen verileri elde eder.



Şekil 3.1. Önerilen Mimari

MQTT mesaj abonesi, Hyperledger Fabric client uygulaması üzerinde çalışırken MQTT aracısı tarafından aldığı mesajı ağ içerisinde kimliği doğrulanmış bir kullanıcı aracılığıyla blokzinciri ağı içindeki ilgili akıllı sözleşmelere gönderir. Bu işlem gerçekleştiğinde Hyperledger Fabric ağı üzerinde gerekli işlem isteği oluşturur.

Bu yaklaşım, basit bir Hyperledger Fabric ağ kurulumunda simüle edilmiştir. Çalışmamızda Hyperledger Fabric ağı içerisinde üç farklı organizasyon oluşturulmuştur. Bu organizasyonlardan ikisi ağın dış dünya ile etkileşime girmesine yarayacak, akıllı sözleşmeler aracılığıyla işlem isteği yürütebilen ve Hyperledger Fabric durum veri tabanını ve işlem günlüklerini barındıran organizasyonlardır. Üçüncü organizasyon ise sipariş veren olarak tanımlanmıştır. Sipariş veren organizasyonu ağ içerinde onaylanan işlemlerin eşlere dağıtımından ve blok üretiminden sorumlu mekanizmadır.

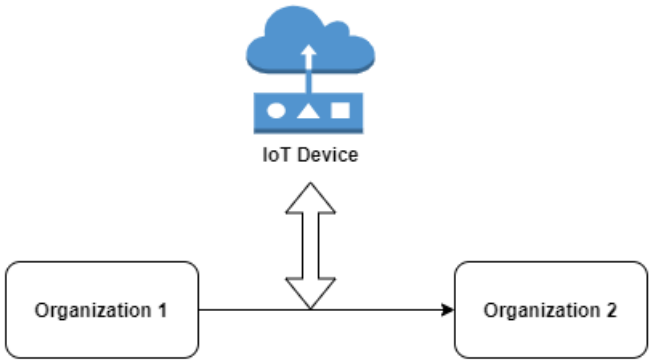
Uygulanan yöntemde üç organizasyon DigitalOcean servis sağlayıcı üzerinde tanımlanmış üç ayrı bulut sunucu olarak, dağıtık bir yapıda çalışmaktadır.

Organizasyonlar, Docker Swarm aracılığıyla birbirleriyle iletişim kurar. Docker Swarm aracılığıyla iletişime geçmiş organizasyonlar Docker Network çatısı altında toplanarak aynı ağ içerisinde çalışır duruma getirilmektedirler. Ağımızda bulunan üç organizasyon, ağdaki süreçleri, katılımcı olarak katıldıkları bir kanal politikası ile

yürütür. Ağdaki organizasyonlar kendileri için tanımlı bir kullanıcı yaratır ve tanımlı süreçleri akıllı sözleşmelerle çalıştırır.

Organizasyonlar arasındaki Şekil 3.2'de simüle edilmiş IoT cihazları, sensörlerden toplanan veriler aracılığıyla durum değişikliklerinde ağı bilgilendiren olayları tetikler. Durum değişiklikleri, bir malzeme üretimi olarak tedarik zinciri süreçlerinde malların üretimi, transferi, sıcaklık ve nem değerlerinin ölçümü gibi birçok ilgi alanına uyarlanabilir.

Hyperledger Fabric'in sunmuş olduğu bu esnek yapı kurumsal ölçekte blokzincir çözümlerine ihtiyaç duyulan her alanda özelleştirilmiş bir şekilde kolayca uygulanabilmektedir.



Şekil 3.2. Blokzincir veri akışı

Önermiş olduğumuz bu yaklaşımla IoT cihazından alınan verilerin, IoT cihazı için tanımlanan Hyperledger Fabric kullanıcısı üzerinden ağa güvenli bir aktarım katmanı üzerinden aktarılması, verilere yetkisiz erişim ve değişikliklerin engellenmesini sağlayacak bir yaklaşımın nasıl olabileceği açıklanmıştır.

3.1.1. Hyperledger fabric ağının kurulması

Hyperledger Fabric ağının eksiksiz bir şekilde kurulabilmesi için bazı ön koşullar vardır [38]. Tezimizde Hyperledger Fabric ağını Ubuntu işletim sistemi içerisinde kurmaktayız.

Hyperledger Fabric ağının kurulumu için gerekli olan imajları indirmeden önce Şekil 3.3'teki gereksinimler kurulmalıdır;

Şekil 3.3. Proje gereksinimleri

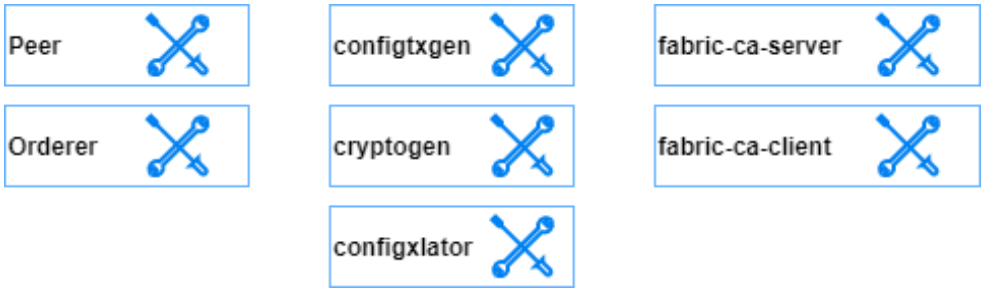
Tezimizde tüm kurulumlar için Şekil 3.4'te gösterilen script dosyası kullanılmıştır.

```
sudo apt-get install -Y curl
sudo apt-get install git
sudo apt-get install -y nodejs
sudo apt-get install -y npm
sudo apt-get install python
sudo apt-get update
sudo curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
docker run hello-world
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.1/docker-compose-$(uname -s)-$(uname -m)" -
o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo wget https://dl.google.com/go/go1.16.3.linux-amd64.tar.gz
sudo tar -xzvf go1.16.3.linux-amd64.tar.gz
sudo mv go/ /usr/local

echo 'export GOPATH=/usr/local/go' >> ~/.bashrc
echo 'export PATH=$PATH:$GOPATH/bin:/root/fabric-samples/bin' >> ~/.bashrc

source ~/.bashrc

sudo curl -sL https://deb.nodesource.com/setup_10.x | sudo bash -
sudo apt-get install -y nodejs
sudo curl -sSL https://bit.ly/2ysbOFE | bash -s
source ~/.bashrc
```

Şekil 3.4. Kurulum dosyası

Bu komutla beraber hem Hyperledger Fabric imajları ve bir ağı kurmak için ihtiyacımız olan ikili dosyaları hem de bütün gereksinimleri sistemimize kurmuş oluyoruz.

Kurulumla beraber gelen imajlar ve ikili dosyalar Şekil 3.5'te sunulmuştur. Bu komponentleri daha sonra ağımızı oluştururken kullanacağız.



Şekil 3.5. Hyperledger Fabric komponentleri

3.1.1.1. Karşılaşılan zorluklar ve öneriler

Tezimizde önerdiğimiz blokzincir altyapısını kurarken karşılaştığımız en büyük sorun Docker Swarm mod ile makinaları aynı ağ altına katılmasını sağlamak olmuştur. Önerdiğimiz çözümde bulut sunucuları kullanmaktayız. Ancak bulut sunucuları kullanmadan önce önerilen mimariyi farklı fiziksel makinalarda kurarak haberleştirmeyi hedeflemiştik. Ancak burada makinaların Docker Swarm aracılığıyla birbirleriyle haberleşmeleri için bazı port yönlendirme ayarlarının yapılması gerekmektedir. Örneğin Docker Swarm modun düzgün bir şekilde çalışması için aşağıdaki portların her makinada başarılı bir şekilde yönlendirilmiş [39] olması gerekmektedir;

Küme yönetimi iletişimi için 2377 numaralı TCP bağlantı noktası, düğümler arası iletişim için 7946 numaralı TCP ve UDP bağlantı noktası, yer paylaşımlı ağ trafiği için 4789 numaralı UDP bağlantı noktası.

Ancak sadece port yönlendirmelerin başarılı bir şekilde yapılması sorunun çözümü için yeterli olmamaktadır. Makinaların birbirleri ile haberleşebilmesi için “Statik Ip” hizmetine ihtiyaç vardır.

Statik ip hizmetine geçildikten sonra ise yine bir sorunla karşılaşılmıştır. Bu noktada makinalardan biri swarm modu başlattığında diğer makina ağa “worker” olarak katılabiliyorken “manager“ olarak katılımı sağlayamamaktadır. Önerdiğimiz mimaride ise bütün organizasyonlar swarm ağına “manager” olarak katılmalıdır. Tespit edilemeyen ağ sorunları nedeniyle farklı fiziksel makinalarda Docker Swarm ağına katılımlarda problemler yaşanabilmektedir.

Tüm bu sebeplerden dolayı makinaların aralarında herhangi bir kısıta takılmadan swarm mod üzerinden haberleşebilmeleri için bulut sunucular üzerinde çalışma kararı alınmıştır. Ve bulut sunucular üzerinden başarılı bir şekilde swarm moda erişim sağlanmıştır.

3.1.1.2. Bulut sunucu özellikleri

Bulut sunucu üzerinde çalışma kararı alındıktan sonra en önemli nokta sunucunun özelliklerini belirlemektir.

Tezimizde IoT ve blokzincir arasındaki iletişim için güvenli bir mimari önerdiğimizden dolayı sunucu seçiminde orta seviyede özellikleri olan sunucular seçilmiştir.

Ancak veri işleme hızının çok süratli olacağı değerlendirilen gerçek hayattaki uygulamalarda sunucu özellikleri daha da arttırılmalıdır.

Proje kapsamında kullanılan sunucuların özellikleri; 8 GB RAM, 4 CPUs, 160 GB SSD Disk ve 5 TB Transfer kapasitesidir.

3.1.1.3. Organizasyonların kripto materyallerinin oluşturulması

Bu bölümde oluşturacağımız Hyperledger Fabric organizasyonları için sertifika otoritesini kullanarak kripto materyallerinin nasıl oluşturulduğu anlatılacaktır.

Fabric ağlarında kripto materyallerin yani açık anahtar, gizli/özel anahtar ve TLS sertifikaları gibi yetkilendirme, kimliklendirme içeren materyallerin oluşturulması için sertifika otoritelerine ihtiyaç duyulmaktadır.

Bu sertifika otoritesi Hyperledger Fabric kurulumunda gelen “hyperledger/fabric-ca” imajı aracılığıyla oluşturulur.

Bir organizasyon için sertifika otoritesi oluşturmak istediğimizde bir docker-compose file yapılandırma dosyası oluşturmamız gerekir.

Proje kapsamında oluşturulmuş olan “docker-compose-ca.yaml” yapılandırma dosyası Şekil 3.6’daki gibidir.

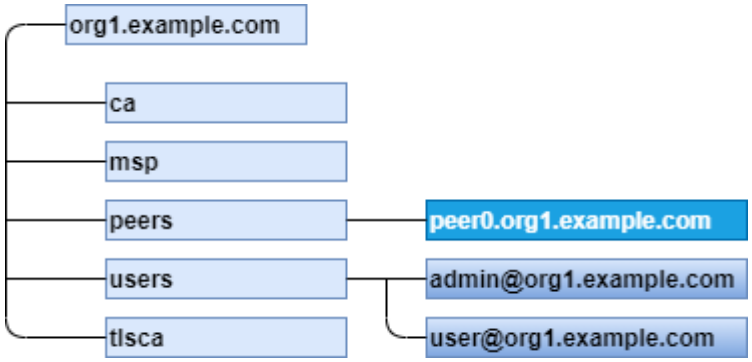
```
services:
  ca_org1:
    image: hyperledger/fabric-ca
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca.org1.example.com
      - FABRIC_CA_SERVER_TLS_ENABLED=true
      - FABRIC_CA_SERVER_PORT=7054

    ports:
      - "7054:7054"
    command: sh -c 'fabric-ca-server start -b admin:adminpw -d'
    volumes:
      - ./fabric-ca/org1:/etc/hyperledger/fabric-ca-server
    container_name: ca.org1.example.com
    hostname: ca.org1.example.com
    networks:
      - test
```

Şekil 3.6. Hyperledger Fabric ağı sertifika otoritesi Docker konteyneri

Burada 7054 portunda çalışacak bir "fabric-ca" imajı bir Docker Konteyner olarak oluşturulmaktadır.

Daha sonra bu docker-compose yapılandırma dosyası aracılığıyla oluşturulacak olan materyallerin depolanacağı klasör yapısı hazırlanmalıdır.

Örnek bir klasör yapısı Şekil 3.7'de sunulduğu gibidir.



Şekil 3.7. MSP klasör yapısı

Hyperledger Fabric ağlarında sertifika oluşturma süreci aşağıdaki iki aşamadan oluşmaktadır;

Enrollment ve Register.

Enrollment, bir kullanıcının belirli bir sertifika otoritesinden dijital bir sertifika istediği ve elde ettiği süreçtir. Register süreci ise, genellikle bir kayıt yetkilisi tarafından yapılır ve bir sertifika otoritesine dijital sertifikayı vermesini söyler.

Öncelikle sertifika otoritesine bir yönetici kaydolur. Ardından yönetici için imzalama anahtarını ve sertifikasını alır.

Yönetici daha sonra yaratılacak olan bir kullanıcıyı uygun bilgilerle sertifika otoritesine kaydeder. Sertifika otoritesi de yeni kullanıcı için bir imzalama anahtarı ve sertifikası oluşturarak kaydı tamamlar. Bu işlemi yapan kod parçası Şekil 3.8'deki gibidir;

```
fabric-ca-client enroll -u https://admin:adminpw@localhost:7054 --caname ca.org1.example.com --
tls.certfiles ${PWD}/fabric-ca/org1/tls-cert.pem
```

Şekil 3.8. Yöneticinin sertifika otoritesine kaydolması

Bu komut ile yönetici kaydedilir ve MSP materyalleri alınır.

```
echo 'NodeOUs:
  Enable: true
  ClientOUIdentifier:
    Certificate: cacerts/localhost-7054-ca-org1-example-com.pem
    OrganizationalUnitIdentifier: client
  PeerOUIdentifier:
    Certificate: cacerts/localhost-7054-ca-org1-example-com.pem
    OrganizationalUnitIdentifier: peer
  AdminOUIdentifier:
    Certificate: cacerts/localhost-7054-ca-org1-example-com.pem
    OrganizationalUnitIdentifier: admin
  OrdererOUIdentifier:
    Certificate: cacerts/localhost-7054-ca-org1-example-com.pem
    OrganizationalUnitIdentifier: orderer' >${PWD}/../crypto-config/peerOrganizations/org1.example.com/msp/config.yaml
```

Şekil 3.9. Kimlik sınıflandırma işlemleri

Şekil 3.9'da sunulan komutlarda "NodeOUs: Enable: true" olarak girildiğinde kimlik sınıflandırma işlemi aktif olacaktır.

Organizational Unit Identifier kavramı bir organizasyonun geçerli üyelerinin X.509 sertifikalarına dahil edilmesi için config.yaml dosyasına belirtmesi gereken parametreleri içerir.

Certificate alanı client, admin, peer ve orderer kimliklerinin doğrulaması gereken sertifika otoritesinin yoluna ayarlanmalıdır.

Hyperledger Fabric MSP entegrasyonu client, peer, admin ve orderer olmak üzere birçok kimlik için kimlik sınıflandırmasına olanak tanır. Tezimizde gerçekleştirilen tasarımda client, peer, admin ve orderer olmak üzere dört farklı kimlik belirlenmiştir.

Geçerli bir kimlik ağ üzerinde bir işlem yapacaksa bu "client" olarak sınıflandırılmalıdır.

Yine geçerli bir kimlik ağ içerisinde bir kanala eş tanımlama veya bir kanal yapılandırma güncelleme işlemi gibi yönetim görevlerini yerine getiriyorsa, "admin" olarak sınıflandırılmalıdır.

Geçerli bir kimlik ağ içerisindeki işlemleri onaylayan ve dünya durumu dediğimiz veri tabanını tutuyorsa, "peer" olarak sınıflandırılmalıdır.

Eğer bir kimlik sipariş veren organizasyonuna ait ise "orderer" olarak sınıflandırılmalıdır.

Client rolünde bir varlığın oluşturulması için çalıştırılması gereken örnek komutlar Şekil 3.10'daki gibidir.

Benzer şekilde peer, admin ve orderer varlıkları da oluşturulmaktadır.

```
fabric-ca-client register --caname ca.org1.example.com --id.name user1 --id.secret user1pw --id.type client --
tls.certfiles ${PWD}/fabric-ca/org1/tls-cert.pem
fabric-ca-client enroll -u https://user1:user1pw@localhost:7054 --caname ca.org1.example.com -M ${PWD}/../crypto-
config/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp --tls.certfiles ${PWD}/fabric-ca/org1/tls-
cert.pem
```

Şekil 3.10. Client rolünde bir varlığın oluşturulması

Yukarıdaki kodlarda gördüğümüz "fabric-ca-client" Hyperledger Fabric ağında kripto materyallerini oluşturmaya yarayan bir yazılımdır.

Organizasyonun sahip olduğu sertifika otoritesi aracılığıyla ağda yaratılacak olan bir aktörün kripto materyallerini oluşturmaya yarar.

3.1.1.4. Docker swarm ve docker network modu

Docker Swarm, docker konteynerlerini kümelemek ve planlamak için kullanılan bir araçtır [40]. Basit bir deyişle, Swarm, bir küme içinde bir araya getirilen bir docker motoru üzerinde çalışan makineler koleksiyonudur. Swarm, yüksek kullanılabilirlik, ölçeklendirme, yük dengeleme, merkezi olmayan yönetim ve felaket kurtarma gibi modern bir uygulamaya birçok ek özellik getirmektedir.

Docker Swarm getirdiği bu özellikleri, Hyperledger Fabric ağlarında farklı organizasyonlarda Fabric Peers ve Orderers barındırmak için kullanıyoruz. Swarm aracı, Hyperledger Fabric ağını düzenlemek için bir dizi komut satırı yardımcı programı sağlar. Raft sipariş hizmetini destekleyen swarm, ağı dağıtmak, test etmek ve temizlemek için basit kabuk komut dosyaları kullanılır.

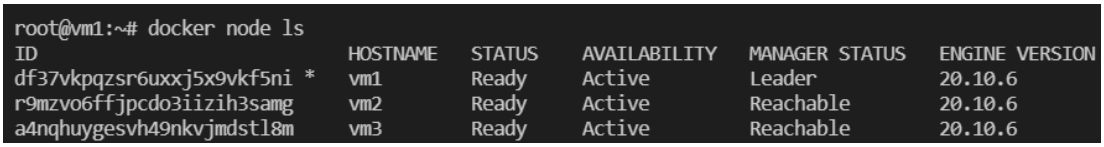
Docker Swarm ağında düğümler, manager ve worker düğümler olarak ikiye ayrılırlar. Manager düğümler, ağ üzerinde kontrole sahip olanlardır ve felaket durumunda yük dengeleme ve konteyner kurtarmadan sorumludurlar. Worker düğümleri, kapsayıcıları çalıştırmaya yardımcı olacak düğümlerdir.

Tezimizdeki kullanım durumumuzda üç organizasyon bulunmaktadır. Ve bu üç organizasyon da lider olarak Docker Swarm ağına üç ayrı bulut sunucu üzerinden katılacaklardır.

Çalışmamızda kullanılacak olan iki organizasyon ve bir orderer organizasyonun kripto materyallerini oluşturduktan sonra Docker Swarm aracılığıyla bu organizasyonları aynı ağın altında manager olarak çalışır hale getirmemiz gerekir. Bunun için belirlenen bir organizasyon tarafından bir Docker Swarm ağı oluşturulur.

Şekil 3.11. Docker Swarm ağının kurulması

Şekil 3.11'de yer alan komut ile bir Docker Swarm ağı kurulmuş olur. Şekil 3.12'de yer alan komut ile ağın oluşturulmuş olduğunu ve ağı kuran düğümün "leader" olarak ağa dahil olduğunu görebiliriz.



Şekil 3.12. Docker Swarm ağı katılımcıları

Bu komutun çalıştırılması ile bize Docker tarafından bir komut dönecektir. Aşağıda yer alan bu komut oluşturulmuş olan bir Docker Swarm ağında worker olarak eklenilmesi istenen bir düğümün çalıştıracağı join komutudur. Ancak biz ağımıza bütün organizasyonları manager olarak eklemek istemekteyiz. Bu sebeple "docker swarm join-token manager" komutu çalıştırılarak bir katılımcıyı manager olarak ekleyecek token'i elde etmiş oluruz. Dönen değer komut satırında girilerek katılımcı ağa manager olarak katılmış olacaktır.

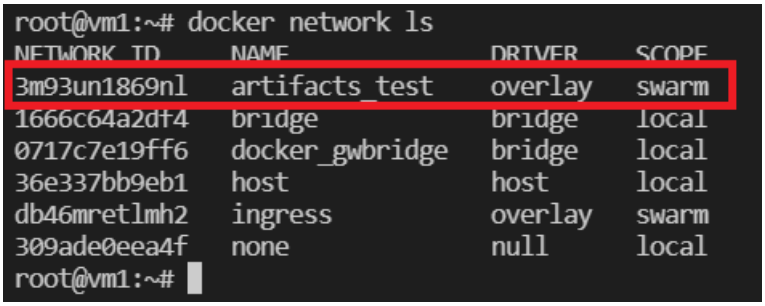
Docker Swarm aracılığıyla aynı küme altına topladığımız organizasyonlarımızı Docker Network aracılığıyla haberleştireceğiz [41]. Bunun için bir "overlay" network yaratılması gerekmektedir.

Overlay Networks (Yer Paylaşımlı Ağlar), kısaca birden çok Docker arka plan programını birbirine bağlayan ve swarm hizmetlerinin birbirleriyle iletişim kurmasını sağlayan altyapıdır [42].

Swarm ağını yarattıktan ve organizasyonların bu ağa katılmasından sonra, şimdi swarm ağı içinde iletişim kurmak için kullanılacak bir overlay ağı oluşturmalıyız.

docker network create --driver overlay --attachable artifacts_test

Yukarıdaki komutla Docker Network kurulmuş olur ve aşağıdaki resimde yer alan komutla "artifacts_test" ağına katılmış olunduğu görülür.



Şekil 3.13. Docker Network ağı

3.1.1.5. Docker konteynerlerin sunucularda çalıştırılması

Docker Swarm ve Docker Network alt yapısı kurulduktan sonra her organizasyonda ayrı ayrı çalışacak olan CouchDB veri tabanı, organizasyon eşleri, eşler içerisinde çalışacak olan CouchDB ve akıllı sözleşme kodlarının portları bir docker-compose dosyası olarak hazırlanarak çalıştırılmalıdır.

```
couchdb0:
container_name: couchdb0
image: hyperledger/fabric-couchdb
environment:
- COUCHDB_USER=
- COUCHDB_PASSWORD=
ports:
- 5984:5984
networks:
- test
```

Şekil 3.14. CouchDB Docker Konteyneri

Şekil 3.14'te Organizasyon1 içerisinde 5984 portunda çalıştırılacak olan CouchDB veri tabanı gereksinimleri yaratılmaktadır.

```
peer0.org1.example.com:
  container_name: peer0.org1.example.com
  extends:
    file: base.yaml
    service: peer-base
  environment:
    - FABRIC_LOGGING_SPEC=DEBUG
    - ORDERER_GENERAL_LOGLEVEL=DEBUG
    - CORE_PEER_LOCALMSPID=Org1MSP
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=artifacts_test
    - CORE_PEER_ID=peer0.org1.example.com
    - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
    - CORE_PEER_LISTENADDRESS=0.0.0.0:7051
    - CORE_PEER_CHAINCODEADDRESS=peer0.org1.example.com:7052
    - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
```

Şekil 3.15. Organizasyon1'e ait eş

Şekil 3.15'te Organizasyon1 içerisinde çalıştırılacak olan bir eşin (peer0) yaratılması için gerekli olan parametre ayarları yapılmaktadır. Burada daha önceki bölümde oluşturmuş olduğumuz Docker Network ağının adı verilir, çalışacak olan eşin 7051

portunda çalışacağı belirtilir. Eş içerisinde çalışacak olan akıllı sözleşmenin ise 7052 portunda çalışacağı belirtilmiştir.

3.1.1.6. Kanal oluşturulması ve organizasyonların kanala katılımı

Hyperledger Fabric ağlarında kanallar, ağın genel omurgasını oluştururlar. Ağ içerisinde birbirleriyle iş yapmak isteyen organizasyonların aynı kanala dahil olmaları gereklidir. Kanallar dinamik olarak oluşturulabilir ve yeniden yapılandırılabilir. Ve aynı zamanda organizasyon eşleri de anında kanala dahil olabilir. Kanala dahil olan eşler, defter veri tabanının bir kopyasını ve organizasyonların aralarındaki iş süreçlerini yönetmek için kullandıkları akıllı sözleşmeleri bünyelerinde barındırırlar.

Kanal oluşturma işlemleri iki adımda gerçekleşir. Bunlardan ilkinde Hyperledger Fabric kurulumunda gelen ikili dosyalar "configtxgen" komutu kullanılmaktadır.

Configtxgen komutu, kullanıcıların kanal yapılandırmasıyla ilgili ayarları oluşturmasına ve incelemesine olanak tanır.

```
# System channel
SYS_CHANNEL="sys-channel"
# channel name defaults to "mychannel"
CHANNEL_NAME="mychannel"
# Generate System Genesis block
configtxgen -profile OrdererGenesis -configPath . -channelID $SYS_CHANNEL  -outputBlock ./genesis.block
# Generate channel configuration block
configtxgen -profile BasicChannel -configPath . -outputCreateChannelTx ./mychannel.tx -
channelID $CHANNEL_NAME
echo "#######    Generating anchor peer update for Org1MSP  ##########"
configtxgen -profile BasicChannel -configPath . -outputAnchorPeersUpdate ./Org1MSPanchors.tx -
channelID $CHANNEL_NAME -asOrg Org1MSP
echo "#######    Generating anchor peer update for Org2MSP  ##########"
configtxgen -profile BasicChannel -configPath . -outputAnchorPeersUpdate ./Org2MSPanchors.tx -
channelID $CHANNEL_NAME -asOrg Org2MSP
```

Şekil 3.16. Kanal kripto materyallerinin yaratılması

Şekil 3.16'daki gibi bir kanala ait materyaller yaratılır. Hyperledger Fabric ağlarında her kanal konfigürasyonu bir genesis blokunun oluşturulması ile başlar. 17. Satırda görülen komut ile "configtxgen" komutu yardımıyla genesis bloku (genesis.blok) oluşturulmuş olur. Bu komut içerisinde bulunan "-profile OrdererGenesis" komutu önemlidir. OrdererGenesis, kanal oluşturmak için kullanılacak olan profilin

configtx.yaml'deki adıdır. İçerisinde bir yapılandırma bilgisi içerir. Bu yapılandırma bilgisi Şekil 3.17'deki gibidir.

```
OrdererGenesis:
    <<: *ChannelDefaults
    Capabilities:
      <<: *ChannelCapabilities
    Orderer:
      <<: *OrdererDefaults

OrdererType: etcdraft
      EtcdRaft:

Consenters:
        - Host: orderer.example.com
          Port: 7050
          ClientTLSCert: ../../setup1/vm4/crypto-
config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt

ServerTLSCert: ../../setup1/vm4/crypto-
config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt
        - Host: orderer2.example.com
          Port: 8050
          ClientTLSCert: ../../setup1/vm4/crypto-
config/ordererOrganizations/example.com/orderers/orderer2.example.com/tls/server.crt
          ServerTLSCert: ../../setup1/vm4/crypto-
config/ordererOrganizations/example.com/orderers/orderer2.example.com/tls/server.crt
      Addresses:
        - orderer.example.com:7050
        - orderer2.example.com:8050

      Organizations:
      - *OrdererOrg
      Capabilities:
        <<: *OrdererCapabilities
    Consortiums:
      SampleConsortium:
        Organizations:
        - *Org1
        - *Org2
```

Şekil 3.17. OrdererGenesis yapılandırma ayarları

Burada dikkat edilmesi gereken noktalar bulunmaktadır.

"ChannelDefaults" komutu oluşturulan kanaldaki politikaların belirlendiği kısımdır.

```
Channel: &ChannelDefaults
  Policies:
    Readers:
      Type: ImplicitMeta
      Rule: "ANY Readers"
    Writers:
      Type: ImplicitMeta
      Rule: "ANY Writers"
    Admins:
      Type: ImplicitMeta
      Rule: "MAJORITY Admins"

  Capabilities:
    <<: *ChannelCapabilities
```

Şekil 3.18. ChannelDefaults ayarları

Şekil 3.18'de görüldüğü gibi kanal politikaları belirlenmektedir. Burada kanal içerisinde üç farklı rol bulunmaktadır. Bunlar; "Readers", "Writers" ve "Admins" tir. "Readers" ve "Writers" yapılandırmaları altında yer alan "Rule" ayarı bu kanal için hazırlanmış herhangi bir API Server'ın herhangi bir "Readers" ya da "Writers" ayrıcalığını taşıyan kullanıcı tarafından çağrılabileceğini belirtir. "Admins" alanı içerisinde yer alan "Rule" komutu ise kanal yapılandırma ayarlarını sadece "Admin" rolündeki kullanıcıların değiştirebileceğini belirtir. Tip olarak kullanılan "ImplicitMeta" politikasının önemli bir avantajı, kanala yeni bir yönetici organizasyonu eklendiğinde kanal politikasını güncellenmesini gerektirmemesidir. Bu nedenle, "ImplicitMeta" politikalarının, konsorsiyum üyeleri değiştikçe daha esnek olduğu değerlendirilmektedir.

```
Orderer: &OrdererDefaults
  OrdererType: etcdraft
  EtcdRaft:
    Consenters:
    - Host: orderer.example.com
      Port: 7050

  BatchTimeout: 2s
  BatchSize:
    MaxMessageCount: 10
    AbsoluteMaxBytes: 99 MB
    PreferredMaxBytes: 512 KB
```

Şekil 3.19. OrdererDefaults ayarları

“OrdererDefaults, sipariş verenle ilgili parametreler için bir yapılandırma ayarı içerir. Burada “OrdererType” parametresi, işlemleri, konfigürasyon güncellemelerini, sipariş veren hizmet düğümleri arasındaki işlemlerin sıralanması konusunda fikir birliğine varmak için kullanılacak olan protokolü belirler. Hyperledger Fabric, Raft (etcdraft) protokolünün uygulanmasına dayanan bir CFT sipariş hizmetini kullanır. “Addresses” alanı eşlerin ve kullanıcıların sipariş verene ulaşabilecekleri adresi göstermektedir. Sırasıyla kalan parametreler aşağıdaki gibi açıklanabilir;

BatchTimeout, bir toplu iş oluşturmadan önce beklenecek süre. BatchSize, bir blok halinde gruplanan mesajların sayısını kontrol eder.

MaxMessageCount, bir toplu işte izin verilecek maksimum ileti sayısı. Hiçbir blok bu mesaj sayısından fazlasını içermez.

AbsoluteMaxBytes, bir toplu işteki mesajlar için izin verilen mutlak maksimum bayt sayısı. Bu değerin üzerindeki herhangi bir işlem reddedilecektir.

PreferredMaxBytes, bir toplu işteki mesajlar için tercih edilen maksimum bayt sayısı. Gruba yeni bir mesaj eklemek, grubun tercih edilen maksimum baytı aşmasına neden oluyorsa, mevcut grup kapatılır ve bir bloğa yazılır ve yeni mesajı içeren yeni bir grup oluşturulur.

Şekil 3.16’da görüldüğü gibi yine “configtxgen” komutu kullanılarak bu sefer kanalın işlem defterleri oluşturulur. 21. satırda (mychannel.tx) görüldüğü gibi bu işlem için “BasicChannel” profili kullanılır. Şekil 3.16’da 24 ve 27. satırlarda (Org1MSPanchors.tx ve Org2MSPanchors.tx) da benzer komutlar görülmektedir. Bu komutlar yine aynı şekilde bu sefer organizasyonlar için işlem defterlerini oluşturur. “BasicChannel” profili Şekil 3.20’de gösterilen yapılandırmaları içermektedir.

```
BasicChannel:
    Consortium: SampleConsortium
    <<: *ChannelDefaults Application:
      <<: *ApplicationDefaults
      Organizations: - *Org1- *Org2
      Capabilities:  <<: *ApplicationCapabilities
```

Şekil 3.20. BasicChannel profil ayarları

Burada sadece iki organizasyonun yer aldığını görüyoruz. Sipariş veren (orderer) organizasyonun yer almamasının sebebi, ağ içindeki görevinin işlem sonuçlarını toplayıp yeni blok oluşturma görevi olmasıdır. "ChannelDefaults" politikaları daha önce orderer için anlatılanla eş değerdir. Burada ordererdan farklı olan politika ayarı "ApplicationDefaults" olarak göze çarpmaktadır.

```
Application: &ApplicationDefaults

  Organizations:

  Policies:
    Readers:
      Type: ImplicitMeta
      Rule: "ANY Readers"
    Writers:
      Type: ImplicitMeta
      Rule: "ANY Writers"
    Admins:
      Type: ImplicitMeta
      Rule: "MAJORITY Admins"
    LifecycleEndorsement:
      Type: ImplicitMeta
      Rule: "MAJORITY Endorsement"
    Endorsement:
      Type: ImplicitMeta
      Rule: "MAJORITY Endorsement"

  Capabilities:
    <<: *ApplicationCapabilities
```

Şekil 3.21. ApplicationDefaults ayarları

"ApplicationDefaults" politikaları kanalda bulunan organizasyonların sahip olduğu üyelerin uygulama seviyesinde hangi politikalara göre işlem yapabildiğini gösterir. Tip, "ImplicitMeta" daha önce anlatıldığı şekilde burada da geçerlidir. "Rule" politikası ise; örneğin kimin kanal defterini sorgulayabileceği, bir zincir kodu çağırabileceği, bir kanal yapılandırmasını güncelleyebileceğini veya işlemin çoğunluk onayına göre onaylanabileceğini belirler. Bu politikalar, her organizasyon için tanımlanan alt politikalara işaret eder. Şekil 3.22'de tanımlanan "Org1" organizasyonu, uygulama bölümündeki "Reader", "Writer" ve "Admin" "ImplicitMeta" ilkeleri tarafından değerlendirilen "Reader", "Writer" ve "Admin" alt politikalarını içerir. Aşağıdaki resimde sistem kanalındaki "Org1" organizasyonunun

politika tanımları gösterilmektedir; burada politika Türü “Signature” ve “Endorsement” kuralı "OR('Org1MSP.peer')" olarak tanımlanır.

Bu politika, onay işlemini “Org1MSP” üyesi olan bir eşin imzalaması gerektiğini belirtir.

```
&Org1
        Name: Org1MSP
        ID: Org1MSP

        MSPDir: ../../setup1/vm1/crypto-config/peerOrganizations/org1.example.com/msp


          Policies:
           Readers:
              Type: Signature
              Rule: "OR('Org1MSP.admin', 'Org1MSP.peer', 'Org1MSP.client')"

          Writers:
              Type: Signature
              Rule: "OR('Org1MSP.admin', 'Org1MSP.client')"
           Admins:
              Type: Signature
              Rule: "OR('Org1MSP.admin')"

          Endorsement:
              Type: Signature
              Rule: "OR('Org1MSP.peer')"

        AnchorPeers:
          - Host: peer0.org1.example.com
            Port: 7051
```

Şekil 3.22. Organizasyon1 politikaları

Tüm yapılan bu yapılandırma süreçleri neticesinde;

Kanal konfigürasyonları ve kanal politikaları oluşturulmuştur.

Sırada bir organizasyon tarafından kanal oluşturma işlemleri ve oluşturulan bu kanal diğer organizasyonun katılımının yapılması gerekmektedir.

Çalışmamızda “Organizasyon1” üzerinden bir kanal kurulmuş ve “Organizasyon2” bu kanala dahil edilmiştir. Her iki organizasyon da kanal içerisinde eşit haklara sahiptir.

```
export CORE_PEER_TLS_ENABLED=true
export ORDERER_CA=${PWD}/../vm4/crypto-
config/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
export PEER0_ORG1_CA=${PWD}/crypto-
config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export FABRIC_CFG_PATH=${PWD}/../../artifacts/channel/config/
export CHANNEL_NAME=mychannel
setGlobalsForPeer0Org1(){
   export CORE_PEER_LOCALMSPID="Org1MSP"
   export CORE_PEER_TLS_ROOTCERT_FILE=$PEER0_ORG1_CA
   export CORE_PEER_MSPCONFIGPATH=${PWD}/crypto-
config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
   export CORE_PEER_ADDRESS=localhost:7051
}
setGlobalsForPeer1Org1(){
   export CORE_PEER_LOCALMSPID="Org1MSP"
   export CORE_PEER_TLS_ROOTCERT_FILE=$PEER0_ORG1_CA
   export CORE_PEER_MSPCONFIGPATH=${PWD}/crypto-
config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
   export CORE_PEER_ADDRESS=localhost:8051
}
```

Şekil 3.23. Kanal parametreleri

Şekil 3.23'te bir kanalın yaratılması ile ilgili gerekli olan parametreler görülmektedir. Burada kullanılacak olan sipariş verenin (orderer) sertifika otorite dosyaları, Organizasyon1 için kullanılacak olan MSP dosyaları ve gerekli diğer sertifika, adres gibi parametreler tanımlanmıştır.

```
createChannel(){setGlobalsForPeer0Org1
   peer channel create -o 142.93.110.47:7050 -c $CHANNEL_NAME \
   --ordererTLSHostnameOverride orderer.example.com \
   -f ./../../artifacts/channel/${CHANNEL_NAME}.tx --outputBlock ./channel-artifacts/${CHANNEL_NAME}.block \
   --tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA}
joinChannel(){
   setGlobalsForPeer0Org1
   peer channel join -b ./channel-artifacts/$CHANNEL_NAME.block
   setGlobalsForPeer1Org1
   peer channel join -b ./channel-artifacts/$CHANNEL_NAME.block}
updateAnchorPeers(){
   setGlobalsForPeer0Org1
   peer channel update -o 142.93.110.47:7050 --ordererTLSHostnameOverride orderer.example.com -
c $CHANNEL_NAME -f ./../../artifacts/channel/${CORE_PEER_LOCALMSPID}anchors.tx --
tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA}
```

Şekil 3.24. Kanal oluşturma komutları

Kullanılacak olan parametreler belirlendikten sonra kanal oluşturma işlemleri başlar. Şekil 3.24'te bulunan komutların işlevleri aşağıdaki gibidir;

createChannel, kanal oluşturma komutları içerisinde yer alan -f bayrağı kullanarak kanal oluşturma işlem dosyasının yolu sağlanır. Kanal adını belirtmek için ise -c bayrağı kullanılır. -o bayrağı, kanalı oluşturmak için kullanılacak sipariş vereni seçmek için kullanılır. --cafile, sipariş verenin TLS sertifikasına giden yoldur. joinChannel, kanal oluşturulduktan sonra organizasyonda bulunan herhangi bir eşi kanala katılabiliriz. Kanalın üyesi olan organizasyonlar, "peer channel join" komutunu kullanarak eşleri kanala ekleyebilir ve kanalda daha önceden oluşturulmuş olan blokları sipariş veren üzerinden alabilir. Herhangi bir eş kanala katıldığında sipariş veren üzerinden blokları alarak blokzinciri defterini oluşturacaktır. updateAnchorPeers, organizasyon eşleri kanala katıldıktan sonra içlerinden birisi "Anchor Peer" olarak seçilmelidir. Anchor peer, farklı organizasyonlardaki akranların birbirlerini tanımalarını sağlamak için kullanılır. Ağ içerisinde güncelleme içeren bir blok yaratıldığında, diğer eşler "Anchor" eşlerine ulaşır ve yeni bir blok yaratıldı bilgisini onlardan alır.

Yukarıdaki anlatılan komutlarda bir organizasyonun bir kanal yaratma işlemleri ve yaratılan kanala katılma işlemleri anlatılmıştır. Ancak kanala katılacak diğer bir organizasyonun farklı olarak uygulayacağı bir komut bulunmaktadır. Kanala yeni katılacak bir organizasyon "createChannel" komutunu kullanamaz. Çünkü kanal zaten yaratılmış durumdadır. Bunun yerine aşağıda gösterilen "fetchChannelBlock" komutu kullanılacaktır.

```
fetchChannelBlock() {
   rm -rf ./channel-artifacts/*
   setGlobalsForPeer0Org2
   peer channel fetch 0 ./channel-artifacts/$CHANNEL_NAME.block -o 142.93.110.47:7050 \
      --ordererTLSHostnameOverride orderer.example.com \
      -c $CHANNEL_NAME --tls --cafile $ORDERER_CA
}
```

Şekil 3.25. Fetch komutu

Şekil 3.25'te gösterilen komutta yer alan "peer channel fetch" komutu sipariş verene giderek kanal içerisindeki güncel blok yapısını almasını sağlar. Daha sonra "peer

channel join" ve "peer channel update" komutları tekrarlanarak kanala katılım sağlanmış olur.

Bu bölümde Hyperledger Fabric ağı içerisinde bir kanalın nasıl oluşturulacağı ve organizasyon eşlerinin oluşturulan bu kanala nasıl dahil olacakları detaylıca anlatılmıştır.

3.1.1.7. Akıllı sözleşmelerin yazılması

Akıllı sözleşmeler ya da Hyperledger Fabric ağındaki adıyla zincir kodları [43], uygulamamızın temel taşlarından birini oluşturmaktadır. Hyperledger ağının izin verdiği zincir kodları ile ağ içerisinde işletilecek olan iş mantığı detaylıca kodlanabilir.

Hyperledger Fabric ağının zincir kodu yazımında üç farklı programlama diline destek verdiğini söylemiştik. Tezimizde kaynak açısından en zengin olan Go programlama dili ile zincir kodlarımızı yazdık. Zincir kodlarının ağ içerisinde işleme alınması iki aşamadan oluşmaktadır.

Önce bağımsız bir ortamda zincir kodlarımızı yazdıktan sonra bu zincir kodlarını organizasyonlarımızda bulunan eşimize yükledik. Yaptığımız uygulamada defter veri tabanları her iki eş üzerinde de yer alırken zincir kodlarımız sadece bir eş üzerinden erişilebilir şekilde uygulanmıştır.

Tezimizde bir IoT benzetimi gibi kullandığımız Postman aracı üzerinden sıcaklık, nem ve basınç değerlerini üreterek, MQTT aracılığıyla ağımıza verileri gönderiyoruz.

Akıllı sözleşmemizi yazarken ilk olarak ağa yüklenecek olan veri yapısını Şekil 3.26'daki gibi tanımladık.

```
type Sensor struct {
   IotName  string `json:"iotname"`
   IotSensorId  string `json:"iotsensorid"`
   Temperature  string `json:"temperature"`
   Humidity string `json:"humidity"`
   Pressure  string `json:"pressure"`
   TimeStamp  string `json:"timestamp"`
}
```

Şekil 3.26. Akıllı sözleşme veri yapısı

Burada görüldüğü gibi bir IoT benzetimi üzerinde çalışacağımız için sıcaklık, nem ve basınç değerlerinin kaydedilmesini sağlayan bir yapı oluşturduk. Burada "IotName" olarak görünen parametre cihazın adı olarak verilebilmektedir. "IotSensorId" ise ağ üzerinde işlem yaparken kullandığımız sertifikalı kullanıcının kimliğidir. Bu sayede hangi kimlik üzerinden işlem yapıldığı kolayca tespit edilebilecektir. "TimeStamp" olarak gözüken parametre ise işlemin yapıldığı anın tarihini içeren bir zaman damgasıdır.

```
func (s *SmartContract) createSensorData(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

    if len(args) != 5 {
        return shim.Error("Incorrect number of arguments. Expecting 5")
    }

    dt:= time.Now()

    var sensor = Sensor{IotName: args[0], Temperature: args[1], Humidity: args[2], Pressure: args[3]}

    sensor.IotSensorId=args[4]
    sensor.TimeStamp=dt.Format("01-02-2006 15:04:05")

    sensorAsBytes, _ := json.Marshal(sensor)
    APIstub.PutState(args[0], sensorAsBytes)

    indexName := "status~key"
    colorNameIndexKey, err := APIstub.CreateCompositeKey(indexName, []string{sensor.IotSensorId, args[0]})
    if err != nil {
        return shim.Error(err.Error())
    }
    value := []byte{0x00}
    APIstub.PutState(colorNameIndexKey, value)

    return shim.Success(sensorAsBytes)
}
```

Şekil 3.27. Sensör verisi oluşturmak için kullanılan akıllı sözleşme fonksiyonu

Akıllı sözleşmedeki ilk fonksiyonumuz Şekil 3.27'de görüldüğü gibi "createSensorData" fonksiyonudur. Burada Postman üzerinden seri olarak aldığımız IoT sensör benzetim değerlerini ağ içerisine kaydedilmesi işlemleri yapılmaktadır. En az beş parametre kabul eden fonksiyonumuz, zaman damgası değerini time.Now() fonksiyonu üzerinden alarak verileri ağa kaydetmektedir.

```
func (s *SmartContract) changeSensorData(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {
    if len(args) != 4 {
        return shim.Error("Incorrect number of arguments. Expecting 4")
    }
    dt := time.Now()
    sensorAsBytes, _ := APIstub.GetState(args[0])
    sensor := Sensor{}

    json.Unmarshal(sensorAsBytes, &sensor)

    sensor.Temperature= args[1]
    sensor.Humidity=args[2]
    sensor.Pressure=args[3]
    sensor.TimeStamp=dt.Format("01-02-2006 15:04:05")

    sensorAsBytes, _ = json.Marshal(sensor)
    APIstub.PutState(args[0], sensorAsBytes)
    return shim.Success(sensorAsBytes)
}
```

Şekil 3.28. Sensör verisini güncelleme fonksiyonu

İkinci akıllı sözleşmemiz ise “updateSensorData” fonksiyonudur. Bu fonksion aracılığıyla bir IoT cihazı tarafından yaratılan verinin güncellenmesi simüle edilmiştir. Kısaca özetleyecek olursak sıcaklık, nem ve basınç değerlerinin zamana bağlı kayıtları blokzincirde değiştirilemez şekilde kayıt edilmektedir.

Akıllı sözleşmelerimizde kullandığımız son iki fonksiyon ise “”querySensorData” ve “getHistoryForAsset” fonksiyonlarıdır.

```
func (s *SmartContract) querySensorData(APIstub shim.ChaincodeStubInterface, args []string) sc.Response {

    if len(args) != 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1")
    }

    sensorAsBytes, _ := APIstub.GetState(args[0])
    return shim.Success(sensorAsBytes)
}
```

Şekil 3.29. Bir kaydın güncel durumunu sorgulama

Şekil 3.29’da görülen “querySensorData” fonksiyonu aracılığıya bir IoT cihazından üretilmiş verinin en güncel hali sorgulanır. Burada Hyperledger Fabric’te bulunan iki farklı veri tabanı türünden world state (dünya durumu) sorgulanır. World state kavramı

Hyperledger Fabric ağına kaydedilmiş bir verinin güncel halinin kolaylıkla sorgulanabilmesi için CouchDB üzerinde oluşturulmuş bir veri tabanıdır.

```
func (t *SmartContract) getHistoryForAsset(stub shim.ChaincodeStubInterface, args []string) sc.Response {
    if len(args) < 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1")
    }
    resultsIterator, err := stub.GetHistoryForKey(productName)
    if err != nil {
        return shim.Error(err.Error())
    }
    defer resultsIterator.Close()
    var buffer bytes.Buffer
    buffer.WriteString("[")
    bArrayMemberAlreadyWritten := false

    for resultsIterator.HasNext() {
        response, err := resultsIterator.Next()
        if err != nil {
            return shim.Error(err.Error())
        }
        if bArrayMemberAlreadyWritten == true {
            buffer.WriteString(",")
        }
        buffer.WriteString("{\"TxId\":")
        buffer.WriteString("\"")
        buffer.WriteString(response.TxId)
        buffer.WriteString("\"")
        buffer.WriteString(", \"Value\":")
        if response.IsDelete {
            buffer.WriteString("null")
        } else {
            buffer.WriteString(string(response.Value))
        }
        buffer.WriteString(", \"Timestamp\":")
        buffer.WriteString(time.Unix(response.Timestamp.Seconds, int64(response.Timestamp.Nanos)).String())
        buffer.WriteString(", \"IsDelete\":")
        buffer.WriteString(strconv.FormatBool(response.IsDelete))
        buffer.WriteString("}")
        bArrayMemberAlreadyWritten = true
    }
    buffer.WriteString("]")
    return shim.Success(buffer.Bytes())
}
```

Şekil 3.30. Bir kaydın tüm değişiklerini blokzincirde sorgulama

Şekil 3.30'da görülen "getHistoryForAsset" fonksiyonu ise Hyperledger Fabric ağı içerisinde yaratılmış olan bir kaydın bütün revizyonları ile birlikte, blokzincirinden sorgulandığı fonksiyondur.

Bu fonksiyon belirli zamanlarda değiştirilmiş olan bir kaydın bütün revizyonlarını bize bir zaman damgası ile sunar. Bu da bizlere verinin şeffaflığı, doğruluğu ve bütünlüğü açısından bir bakış açısı sunar.

Go programlama dile yazmış olduğumuz akıllı sözleşmelerimizi farklı bulut sunucuda çalışan her iki organizasyona da aktarmaktayız.

```
presetup() {
   echo Vendoring Go dependencies ...
   pushd ./../../artifacts/src/github.com/fabcar/go
   GO111MODULE=on go mod vendor
   popd
   echo Finished vendoring Go dependencies
}
presetup
CHANNEL_NAME="mychannel"
CC_RUNTIME_LANGUAGE="golang"
VERSION="4"
CC_SRC_PATH="./../../artifacts/src/github.com/fabcar/go"
CC_NAME="example"

packageChaincode() {
   rm -rf ${CC_NAME}.tar.gz
   setGlobalsForPeer0Org1
   peer lifecycle chaincode package ${CC_NAME}.tar.gz \
      --path ${CC_SRC_PATH} --lang ${CC_RUNTIME_LANGUAGE} \
      --label ${CC_NAME}_${VERSION}
   echo "===================== Chaincode is packaged on peer0.org1 ===================== "
}

packageChaincode

installChaincode() {
   setGlobalsForPeer0Org1
   peer lifecycle chaincode install ${CC_NAME}.tar.gz
   echo "===================== Chaincode is installed on peer0.org1 ===================== "
}
installChaincode
```

Şekil 3.31. Akıllı sözleşmenin eşlere yüklenmesi

```
queryInstalled() {
   setGlobalsForPeer0Org1
   peer lifecycle chaincode queryinstalled >&log.txt
   cat log.txt
   PACKAGE_ID=$(sed -n "/${CC_NAME}_${VERSION}/{s/^Package ID: //; s/, Label:.*$//; p;}" log.txt)
   echo PackageID is ${PACKAGE_ID}
   echo "===================== Query installed successful on peer0.org1 on channel ===================== "
}

approveForMyOrg1() {
   setGlobalsForPeer0Org1
   peer lifecycle chaincode approveformyorg -o 142.93.110.47:7050 \
      --ordererTLSHostnameOverride orderer.example.com --tls \
      --cafile $ORDERER_CA --channelID $CHANNEL_NAME --name ${CC_NAME} --version ${VERSION} \
      --init-required --package-id ${PACKAGE_ID} \
      --sequence ${VERSION}
   echo "===================== chaincode approved from org 1 ===================== "
}
queryInstalled
approveForMyOrg1

checkCommitReadyness()
{
   setGlobalsForPeer0Org1
   peer lifecycle chaincode checkcommitreadiness \
      --channelID $CHANNEL_NAME --name ${CC_NAME} --version ${VERSION} \
      --sequence ${VERSION} --output json --init-required
   echo "===================== checking commit readyness from org 1 ===================== "
}
checkCommitReadyness
```

Şekil 3.32. Akıllı sözleşmenin eşlere yüklenmesi

Şekil 3.31. ve Şekil 3.32'de sözleşmeleri organizasyonlara yükleyen bir komut satırı bulunmaktadır. "presetup()" komutu içerisinde Go programlama dili ile ilgili bağımlılıkları almaktayız.

packageChaincode, "peer lifecycle chaincode package" komutu bir zincir kodunun paketlenip, tar.gz formatında bir dosya olarak oluşturulmasını sağlamaktadır.

installChaincode, "peer lifecycle chaincode install" bir önceki komutta paketlediğimiz zincir kodunu açarak belirlenen eşe yükler. queryInstalled, "peer lifecycle chaincode queryinstalled" komutu bir eş üzerine yüklenen zincir kodunun yüklenip yüklenmediğini sorgular.

approveForMyOrg1, “peer lifecycle chaincode approveformyorg” komutu yüklendiği doğrulanan zincir kodunu organizasyon için onaylar. checkCommitReadyness, “peer lifecycle chaincode checkcommitreadiness” komutu ise onaylanmış zincir kodunun kanala yüklenmeye hazır olup olmadığını kontrol eder. Hazır durumda “true” sonucu dönecektir.

Yukarıda anlatılan tüm işlemler her iki organizasyonda da uygulandıktan sonra sıradaki işlem “peer lifecycle chaincode commit” işlemidir. Zincir kodu ile ilgili yukarıdaki bütün işlemler tamamlandıktan sonra iki organizasyonların onayıyla yükleme işleminin tamamlanması gerekir. Bu işlemi uygulamamız için CLI konteynere ihtiyaç duyarız. CLI konteynerin ağ içerisindeki amacı, kanal ve ağ ile ilgili tüm işlemleri yapmaktır. Kısaca Fabric ağı ile etkileşimimizi sağlar. CLI konteyner tüm bu işlemleri yerine getirmek için, Fabric CA istemcisi, configtxgen ve peer gibi gerekli araçları içerir. Çalışmamızda Organizasyon1 içerisinde kurduğumuz CLI konteyner aracılığıyla aşağıdaki komutları çalıştırırız. Ve artık akıllı sözleşmeler her iki organizasyon tarafından da kullanılabilir duruma gelecektir.

CLI Konteyner’a giriş:

“docker exec –it cli bash”

Açılan komut satırına Şekil 3.33’teki komutlar yazılarak çalıştırılır.

```
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/channel/crypto-
config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/channel/crypto-
config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=peer0.org1.example.com:7051 export CHANNEL_NAME="mychannel"
export CC_NAME="example" export ORDERER_CA=/etc/hyperledger/channel/crypto-
config/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
export VERSION="4" peer lifecycle chaincode commit -o orderer.example.com:7050 --
ordererTLSHostnameOverride orderer.example.com \--tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA \--
channelID $CHANNEL_NAME –name ${CC_NAME} \--peerAddresses peer0.org1.example.com:7051 --
tlsRootCertFiles /etc/hyperledger/channel/crypto-
config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt \
--peerAddresses peer0.org2.example.com:9051 --tlsRootCertFiles /etc/hyperledger/channel/crypto-
config/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt \
--version ${VERSION} --sequence ${VERSION} --init-required
```

Şekil 3.33. CLI Konteyner komutları

İşlem sonucunda akıllı sözleşme yükleme işlemleri başarılı bir şekilde tamamlanacaktır.

Hyperledger Fabric ağlarında ihtiyaca göre yaşayan bir ağın içerisinde akıllı sözleşmeler güncellenerek tekrar yüklenebilmektedir.

Bu işlem çok basit bir şekilde yapılabilmektedir. Güncellenen akıllı sözleşme bir önceki bölümde anlatılan şekilde yükleme işlemlerine tabi tutulur. Tek değişiklik yeni akıllı sözleşmenin versiyonunu sayısal olarak bir arttırmaktır.

```
CHANNEL_NAME="mychannel"
CC_RUNTIME_LANGUAGE="golang"
VERSION="4"
CC_SRC_PATH="./../../artifacts/src/github.com/fabcar/go"
CC_NAME="example"
```

Şekil 3.34. Akıllı sözleşme sürüm güncelleme

Şekil 3.34'te kırmızı ile yazılan "VERSION" parametresi örnekte olduğu gibi bir arttırılarak "5" yapılır. Ve aynı yükleme işlemleri gerçekleştirilir. İşlem sonucunda akıllı sözleşmeler tüm organizasyon eşlerinde güncellenmiş olur.

3.1.1.8. Api server ve mqtt yapısının kurulması

Tezimizde kurmuş olduğumuz API Server ağımızın en önemli yapı taşlarından birisidir.

API Server içerisinde veri üretimi için simülatör olarak kullandığımız Postman'den verilerin alınması, MQTT yayıncısının kurulumu, Hyperledger Fabric ağı ile etkileşime geçebilmemiz için gerekli olan bir fabric-client uygulaması (ağ içerisindeki kimlik doğrulama işlemleri, akıllı sözleşmelere erişim, MQTT abonesi burada çalışmaktadır) gibi işlemleri yaparak kullanıcının Fabric ağı ile etkileşime geçmesini sağlar.

Mosca MQTT mesaj aracısı ise sunucu üzerinde belirlenen bir portta bağımsız olarak çalışmaktadır.

API Server bir Docker Konteyner olarak 4000 numaralı portta çalışmaktadır. API Server kurulumuna yapan Docker komponentleri Şekil 3.35'te olduğu gibidir.

```
version: "2.1"
networks:
 test:
services:
 api:
   image: api:1.0
   build: .
   networks:
    - test
   ports:
    - 4000:4000
```

Şekil 3.35. API server kurulumu

API Server konteyneri içerisinde bulunan dosyalar Şekil 3.36'daki gibidir.

```
root@vm1:~/FabricMultiHostDeployment/setup1/vm1/api-2.0# docker exec -it artifacts_api_1 sh
/usr/src/app # ls
app                broker.js          node_modules       package-lock.json  postman-collection
app.js             config             org1-wallet        package.json
/usr/src/app #
```

Şekil 3.36. API Server konteyneri

"app.js" Postman üzerinden Fabric ağına ulaşmamızı sağlayan ilk katmandır. Burada hem MQTT yayıncısı çalışmakta hem de "GET" ve "POST" metotları ile Fabric ağına işlem isteği gönderilebilmektedir. Şekil 3.37'deki kod parçalarında Postman ara yüzünden alınan veriler, "POST" metodu ile "invoke.js" de bulunan "invokeTransaction()" metodunu tetiklemektedir. İşlem başarı ile gerçekleştiğinde Fabric ağına bir kayıt eklenmiş olur.

```
app.post('/channels/:channelName/chaincodes/:chaincodeName', async function (req, res) {
   try {
      var topic = 'fabric'
      if (fcn === "createSensorData" || fcn =="changeSensorData") {
         var sensor ='{ "fcn":"'+fcn+'", "chaincodeName":"'+chaincodeName+'", "channelName":"'+channelName+'", "args":
["'+args[0]+'", "'+args[1]+'", "'+args[2]+'", "'+args[3]+'"],"orgname":"'+req.orgname+'","username":"'+req.username+'"}';
         var mesaj= JSON.parse(sensor)
         client.publish(topic, JSON.stringify(mesaj))
      }
      let message = await invoke.invokeTransaction();
```

Şekil 3.37. API Server post metodu

"app" klasörünün içinde ise Şekil 3.38'de gösterildiği gibi "app.js" tarafından kullanılan bazı Javascript kodları bulunmaktadır.

Şekil 3.38. API Server konteyner app dizini

```
var { Gateway, Wallets } = require('fabric-network');
const path = require('path');
const FabricCAServices = require('fabric-ca-client');
const fs = require('fs');
```

Şekil 3.39. Fabric ağıyla iletişime geçmek için kullanılan gereksinimler

Şekil 3.39'daki gibi "fabric-network", bir Fabric ağına bağlanmak, işlemleri göndermek ve defterde sorgular gerçekleştirmek için API'ler sağlamaktır. "fabric-ca-client" ise, Fabric ile etkileşim kurmak için node.js uygulamaları yazabilmemize olanak sağlayan bir yazılım geliştirme kitidir. Bu paket, kullanıcı kaydetme, yenileme ve iptal etme gibi kullanıcı sertifikalarının yaşam döngüsünü yönetmek için Fabric CA ile etkileşimde bulunmak üzere API'leri içerir.

Şekil 3.38'de yer alan "helper.js" kodu içerisinde Postman üzerinden Fabric ağına bir işlem isteği gönderen kullanıcının yetkileri, kimlik bilgileri ve bağlantı bilgileri gibi bilgiler kontrol edilmektedir.

Şekil 3.40'da "helper.js" içerisinde yer alan ve kullanıcı kimlik bilgilerinin bulunduğu dosya dizinini bulan "getWalletPath" fonksiyonu gösterilmektedir.

```
const getWalletPath = async (org) => {
   let walletPath;
   if (org == "Org1") {
      walletPath = path.join(process.cwd(), 'org1-wallet');
   } else if (org == "Org2") {
      walletPath = path.join(process.cwd(), 'org2-wallet');
   }else
      return null
   return walletPath
}
```

Şekil 3.40. Organizasyonların kimlik dizinlerini bulmak

"invoke.js" kodu ise Fabric ağı içerisine gönderilen "POST" etiketli metotların akıllı sözleşmelerle etkileşim kurup, işlem isteğinin gönderildiği yerdir.

```
const invokeTransaction = async () => {
  try {
    logger.debug(util.format('\n============ invoke transaction on channel %s ============\n', channelName));

    const ccp = await helper.getCCP(org_name)
    const walletPath = await helper.getWalletPath(org_name)
    const wallet = await Wallets.newFileSystemWallet(walletPath);
    console.log(`Wallet path: ${walletPath}`);
    let identity = await wallet.get(username);
    if (!identity) {
      console.log(` ${username} adlı kullanıcının kaydı bulunamadı, lütfen kayıt edin.`);
      await helper.getRegisteredUser(username, org_name, true)
      identity = await wallet.get(username);
      return;
    }
    const connectOptions = {
      wallet, identity: username, discovery: { enabled: true, asLocalhost: false },
      eventHandlerOptions: {
        commitTimeout: 100,
        strategy: DefaultEventHandlerStrategies.NETWORK_SCOPE_ALLFORTX
      }
    }
    const gateway = new Gateway();
    await gateway.connect(ccp, connectOptions);
    const network = await gateway.getNetwork(channelName);
    const contract = network.getContract(chaincodeName);
    let result
    let message;
    if (fcn === "createSensorData") {
      args4=username
      result = await contract.submitTransaction(fcn, args, args1, args2, args3, args4);
      message = `Successfully added the sensor asset with key ${args}`

    }else if (fcn === "changeSensorData") {
      result = await contract.submitTransaction(fcn, args, args1, args2, args3);
      message = `Successfully updated sensor values with key ${args}`

    }
    else {
      return `Invocation require either createProduct or changeProductStatus as function but got ${fcn}`
    }
    await gateway.disconnect();
```

Şekil 3.41. invokeTransaction metodu

Şekil 3.41'de gösterilen kod parçasında "app.js" tarafında tetiklemiş olduğumuz "invokeTransaction" metodu aracılığıyla Hyperledger Fabric ağına işlem isteği

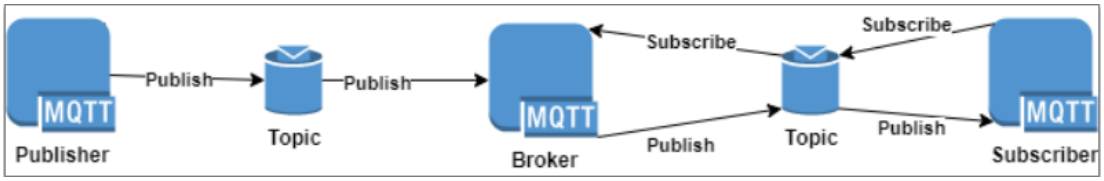
gönderilmiş olur. Postman ara yüzünden gelen bilgiler ile burada kullanıcının organizasyon bilgisi, kimlik bilgileri gibi bilgiler doğrulanarak ağ üzerinde bir bağlantı açılır ve akıllı sözleşmelere erişilir. İşlem tamamlandıktan sonra "gateway.disconnect()" metodu ile bağlantı sonlandırılır.

"org1-wallet" klasörü içerisinde ise Şekil 3.42'de gösterildiği gibi Organizasyon1 için yaratılmış yetkili kullanıcıların kimlik bilgileri bulunmaktadır.

```
/usr/src/app # cd org1-wallet/
/usr/src/app/org1-wallet # ls
NewIoT.id   NewIoT1.id  NewIoT2.id  admin.id
/usr/src/app/org1-wallet #
```

Şekil 3.42. Organizasyon1 kimlik kayıtları

Tezimizde kullandığımız Postman IoT benzetimi ile Hyperledger Fabric blokzincir ağı arasında gerçekleştirilecek olan haberleşmenin güvenli bir şekilde işletilmesi için kurulan "MQTT" yapısının uçtan uca nasıl çalıştığı Şekil 3.43'te gösterilmiştir. Burada "MQTT Publisher" projemizde "app.js" içerisinde çalışmaktadır. Postman üzerinden gelen veri burada "POST" metodu içerisinde "publish" edilir. "1234" numaralı portta çalışmakta olan "Mosca" mesaj yayıncısı bu veriyi alır. "invoke.js" içerisinde çalışmakta olan bir "MQTT Subscriber" ise aynı konu üzerinden "Mosca Broker"a abone olarak gelen mesajları almaya başlar.



Şekil 3.43. MQTT çalışma prensibi

"Mosca" mesaj yayıncısına ait kod Şekil 3.44'teki gibidir.

```
var mosca = require('mosca') var settings = {port: 1234}
var broker = new mosca.Server(settings)
broker.on('ready', ()=>{
   console.log('Broker is ready!')
})
```

Şekil 3.44. MQTT Mosca mesaj yayıncısının çalışması

Burada görüldüğü üzere “Mosca” mesaj yayınıcı, localhostta “1234” numaralı portta çalışmaktadır.

“MQTT” yayıncısı ise “app.js” üzerinde çalışmaktadır.

```
const mqtt = require('mqtt');
const client = mqtt.connect('mqtt://206.81.21.240:1234');
client.on('connect', ()=>{


})
```

Şekil 3.45. MQTT yayıncı ve abone bağlantısı

Şekil 3.45’te görüldüğü gibi “MQTT” protokolü “206.81.21.240:1234” adresinde çalışmakta olan “Mosca” mesaj yayıncısına bir konu üzerinden bağlanır.

```
var topic = 'fabric'
      if (fcn === "createSensorData" || fcn =="changeSensorData") {
         var sensor ='{ "fcn":"'+fcn+'", "chaincodeName":"'+chaincodeName+'", "channelName":"'+channelName+'", "args":
["'+args[0]+'", "'+args[1]+'", "'+args[2]+'", "'+args[3]+'"],"orgname":"'+req.orgname+'","username":"'+req.username+'"}';
         var mesaj= JSON.parse(sensor)
         client.publish(topic, JSON.stringify(mesaj))
      }
```

Şekil 3.46. MQTT yayıncısının mesajı yayımlanması

Şekil 3.46’da “topic=’fabric’” konusu üzerinden Postman’den alınan veriler JSON formatına dönüştürülerek “client.publish()” metodu ile “Mosca” mesaj yayıncısına yayımlanır. Hyperledger Fabric client uygulaması olarak çalışan “invoke.js” içerinde bir “MQTT” abonesi çalışmaktadır. Şekil 3.45’te görüldüğü “MQTT” abonesi de aynı şekilde “Mosca” mesaj yayıncısına bağlanır.

```
var topic = 'fabric'
client.on('connect', () => {
   client.subscribe(topic)})
client.on('message', (topic, message) => {var data = JSON.parse(message)
   org_name = data.orgname; username = data.username;
   chaincodeName = data.chaincodeName; channelName = data.channelName;
   fcn = data.fcn; args = data.args[0];
   args1 = data.args[1];
   args2= data.args[2];
   args3 = data.args[3];
```

Şekil 3.47. MQTT abonesinin mesajı alması

Şekil 3.47'de "MQTT" abonesinin belirli bir konu üzerinden "Mosca" mesaj yayıncısına bağlandığı görülmektedir. "topic=fabric" konusu üzerinden "client.subscribe(topic)" metodu ile abonemiz mesaj yayıncısına abone olarak artık abone olduğu konu üzerinden gelen her türlü veriye erişebilir olmaktadır. "MQTT" abonesi daha sonra "client.on()" metodu ile "MQTT" yayıncısından gelen veriyi açar ve "JSON.parse" metodunu kullanarak veriyi kullanılabilir hale getirir.

4. UYGULAMA VE BULGULAR

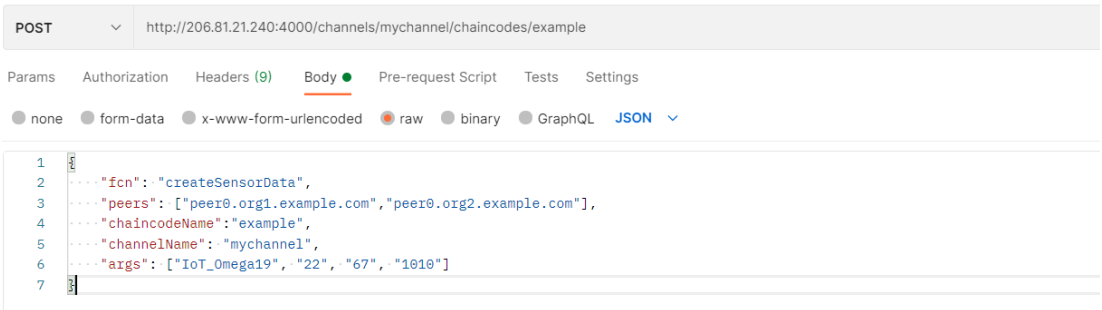
Çalışmamızda kullanmış olduğumuz API'leri işlem isteği göndermemize olanak sağlayan Postman uygulamasını bir IoT veri üreticisi olarak kullanmaktayız. Postman üzerinden tek bir veri gönderilebileceği gibi belirli bir veri seti üzerinden sıralı veriler de gönderilebilmektedir. İşlem isteklerinin başarılı bir şekilde ağa iletilebilmesi için öncelikle "Mosca" mesaj yayıncısı başlatılmalıdır. "node broker.js" komutu çalıştırılarak mesaj yayıncısı aktif hale getirilir.

```
root@vm1:~/FabricMultiHostDeployment/setup1/vm1/api-2.0# node broker.js
Broker is ready!
```

Şekil 3.48. MQTT mesaj yayıncısının aktif hale getirilmesi

Eğer "broker.js" çalışır durumda olmazsa "MQTT" yayıncısının mesajlarını yayınlayacak bir aracı bulunamaz. Aynı şekilde "MQTT" abonesi de ortada herhangi bir konu olmadığı için mesajlara abone olamaz. Bu durumda işlemin gerçekleşmemesine sebebiyet verir.

Şekil 3.49'da Postman üzerinden tek bir verinin nasıl gönderildiği gösterilmiştir.



Şekil 3.49. Postman IoT benzetiminden veri gönderilmesi

Burada gönderilen parametre bilgilerinden fcn, işlem isteği sırasında kullanılacak olan akıllı sözleşme fonksiyonumuz.

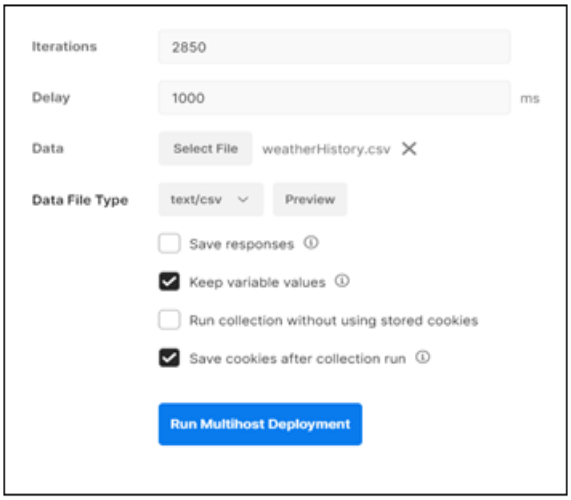
Peers, Fabric ağı içerisinde işlemi onaylayacak olan eşler.

ChaincodeName, fabric ağı içerisinde çalışan, kullanacağımız akıllı sözleşmenin adı. ChannelName, fabric ağı içerisinde, organizasyonlarımızın çalıştığı kanal adı. Args, "createSensorData" fonksiyonumuzun işlemi gerçekleştirebilmek için aldığı parametreler.

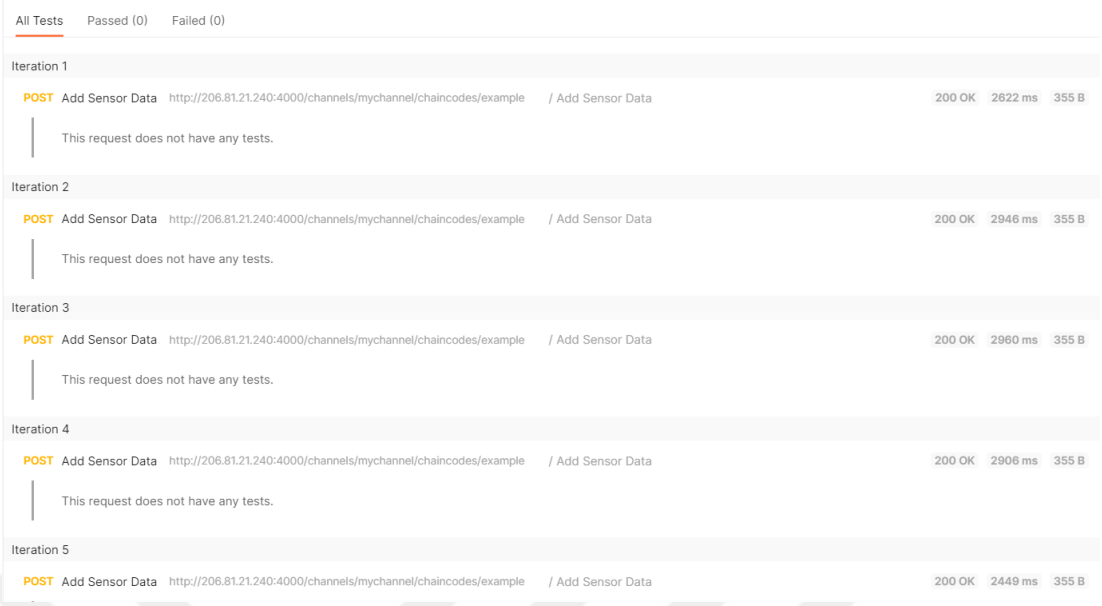
Gerekli parametreler girilip sisteme gönderildikten sonra işlem gerçekleştirilir. İşlem başarı ile gerçekleştiyse Şekil 3.50'deki gibi bir sonuç dönecektir.

Şekil 3.50. Veri ekleme işleminin başarı ile gerçekleştirilmesi

IoT benzetimini gerçekleştirmek için Postman'in bir veri seti üzerinden sıralı işlem gönderme modülünü kullanıyoruz. Veri setimizde sıcaklık, nem ve basınç bilgilerini içeren 2840 adet veri bulunmaktadır. Postman'in "Runner" özelliği kullanılarak veri seti sıralı bir şekilde Fabric ağına işlem isteği olarak gönderilmektedir.

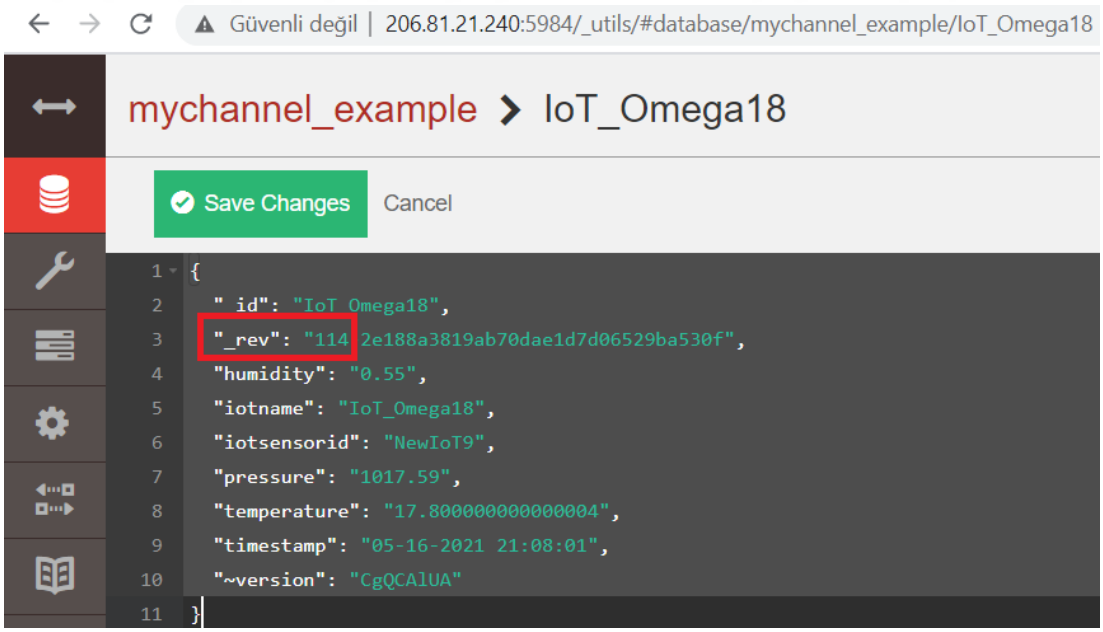


Şekil 3.51. Veri kümesinin eklenmesi



Şekil 3.52. Veri kümesinin Fabric ağına sıralı bir şekilde gönderilmesi

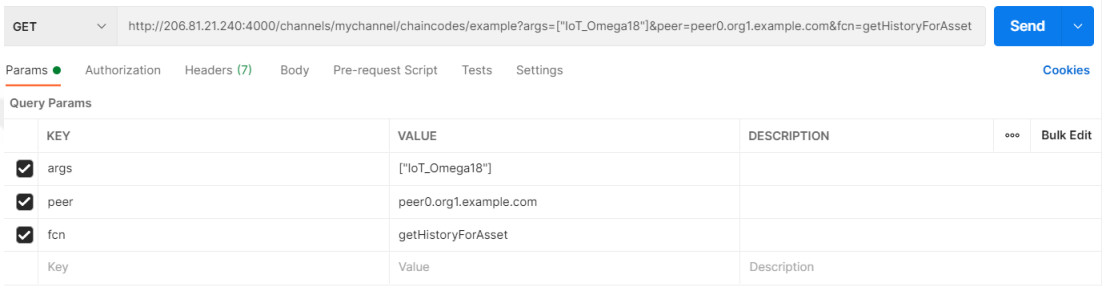
Şekil 3.52'deki gibi işlem istekleri başarılı bir şekilde gönderilip, bir sensör benzetimine ait verilerin anlık güncellenmesiyle bir işlem zinciri oluşmaktadır. Her işlem yaklaşık üç saniye sürede gerçekleşmektedir. Blokzincirdeki bir kaydın son durumuna "CouchDB" üzerinden Şekil 3.53'teki gibi ulaşılabilmektedir.



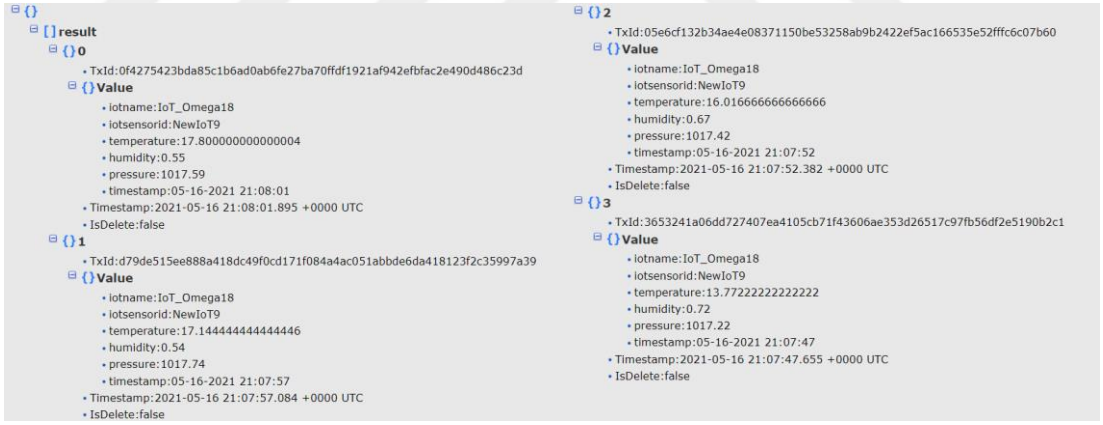
Şekil 3.53. CouchDB dünya durumu veri tabanı

"IoT_Omega18" adlı cihazın ürettiği verilerin son halini Şekil 3.53'teki gibi görüntüleyebiliyoruz. Ancak bahse konu cihaz o ana kadar kırmızı dikdörtgen içerisinde gösterildiği gibi 114 kez revizyona uğramış durumda. IoT sensörleri doğası

gereği, özellikle sıcaklık, nem ve basınç ölçen cihazlar, anlık olarak veri üretirler. Ve bu anlık üretilen büyük verinin değiştirilemez ve birbirine bağlı bir şekilde tutulması gerekir. Hangi anda nasıl bir veri ürettiği, hangi kimlik ile bu verileri ürettiği ya da ağ içerisinde izni olmayan bir cihazın veri üretmesinin engellenmesi IoT, blokzincir ve MQTT entegrasyonu ile sağlanmaktadır. Bu noktada aşağıdaki gibi "getHisoryForAsset" sorgulaması yapılarak cihazın ürettiği verinin değiştirilemez bir şekilde tutulduğu, hangi onaylı kimlikle işlem yapıldığının gösterildiği blokzincirine erişilir.



Şekil 3.54. "IoT_Omega18" adlı cihaza ait blokzincirin sorgulanması



Şekil 3.55. "IoT_Omega18" adlı cihaza ait blokzincir kayıtları

Sorgu sonucu dönen blokzincirdeki 114 revizyon içerisindeki dört değer örnek olarak Şekil 3.55'te gösterilmiştir.

5. SONUÇLAR VE ÖNERİLER

IoT cihazları ve blokzincirler arasındaki entegrasyon günümüzdeki en popüler konulardan biridir. Bu entegrasyon IoT alanındaki birçok sorunu çözebilecek niteliktedir. IoT cihazlarının doğası gereği düşük kapasiteli ve güvenliksiz cihazlar olmasından dolayı, bu cihazlardan elde edilen verilerin depolanması ve güvenlik kriterleri en büyük problemler olarak görünmektedir. Ancak blokzincir teknolojilerinin sunmuş olduğu yüksek düzeydeki çözümler bu entegrasyonu oldukça popüler kılmaktadır. Özellikle tezimizde kullanmış olduğumuz Hyperledger Fabric blokzincir platformu sunmuş olduğu birçok avantajlı özellikle IoT cihazlarına büyük fırsatlar sunmaktadır.

IoT cihazlarından alınan verilerin güvenilir bir şekilde Hyperledger Fabric blokzincir ağına aktarılabilmesi bu entegrasyonun birçok alanda da kullanılabilir olduğunu ortaya koyar. Özellikle tedarik zincirlerindeki belirsizlikler için bu entegrasyon ideal bir çözümdür. Marketlerde satın aldığımız ürünlerin geçmişini bilmek bizler için çok önemlidir. Ve bu bilgilerin güvenilir bir şekilde sisteme işlendiğini biliyorsak satın aldığımız ürüne karşı güvenimiz artacaktır. Üretildiği andan itibaren IoT cihazları ile güvenilir bir şekilde Hyperledger Fabric blokzinciri ağına kaydedilecek olan veriler müşteriler için bir güven unsuru oluşturacaktır. Belirli sıcaklık aralıklarında saklanması gereken ürünlerin gerçekten gerekli koşullar altında saklandığı ve bunun şeffaf bir şekilde müşterilere sunulduğu bir yapı her zaman güven sağlayacaktır.

IoT blokzincir entegrasyonu daha birçok alanda uygulanabilir görünmektedir. Sağlık sektörü, akıllı ev sistemleri, akıllı şehirler ve araç takip sistemleri gibi birçok alanda kullanılmakta olan IoT cihazları, kullanıldığı bu alanlarda anlık olarak büyük miktarda veri üretmektedirler. Bu verilerin büyük bir kısmı kişisel mahremiyeti olan ya da veri transferindeki güvenliğinin kesinlikle sağlanması gereken verilerdir.

Blokzincir ağlarının fikir birliği algoritmaları ile çalışması yanlış ve güvenilmez olan işlemlerin tespit edilerek engellenmesini sağlar. Akıllı sözleşmelerle yürütülen iş mantığı dışında gerçekleşen olaylarda yine işlem istekleri kabul edilmez. Tezimizde

kullanmış olduğumuz Hyperledger Fabric blokzincir ağının izinli bir yapıda çalışması kurumsal alanlarda uygulanabilirliğini arttırır. Sadece izin verilen kullanıcıların ağ içerisinde işlem yapabilmesi güvenilirliği ve hesap verebilirliği sağlar.

IoT cihazlarının blokzincir ağları ile güvenilir ve doğrulanmış bir şekilde haberleşmesine olanak sağlayacak olan uygulamamız, IoT verilerini güvenli bir şekilde blokzincir ağlarına aktarabilecek bir şekilde çalışır. Hyperledger Fabric platformunun IoT verilerini yönetmek için kullanılabileceği, kimliklendirme, doğrulama ve yetkilendirme ile verilerin güvenliğini, gizliliğini, geçerliliğini ve bütünlüğünü sağlayabileceği sonucuna vardık. Hyperledger Fabric ağındaki tüm işlemler TLS sertifikalı olup, dahili ağ iletişimi sırasında verilerin tehlikeye atılmamasını sağlar.

Hyperledger Fabric ağının sunmuş olduğu diğer bir önemli avantaj ise birçok blokzincir platformunun aksine birçok programlama dili ile akıllı sözleşmeler yaratılabilmesidir. Özellikle paydaşlar arasında işletilen süreçlerin belirli bir iş mantığı ile ilerlemesi gerekiyorsa akıllı sözleşmeler bunu düzenlemenin en iyi yoludur. Hyperledger Fabric ağının akıllı sözleşemelerde birden fazla programalama dilini destekliyor olması, bu alandaki esnekliği de arttırmaktadır. Örneğin bir sıcaklık verisi üreten sensörden alınan değerler belirli bir aralığın dışına çıktığında, akıllı sözleşmelerde yazmış olduğumuz kontrol kodları bunu yakalayacak ve ilgili birimlere uyarı verecek şekilde tasarlanabilir. Akıllı sözleşmeler tedarik zinciri süreçlerinde ise paydaşlar arasındaki sözleşmelerin otomatik olarak işletilmesini sağlayabilmektedir.

Hyperledger Fabric aracının açık kaynak kod olması, yeni bir teknoloji olması, shell scriptler kullanarak komut satırından çalışmayı gerektirmesi ve kaynak sayısının kısıtlı olması bir dizi zorluğu beraberinde getirmektedir. Gerçekleştirdiğimiz tez çalışması sonucunda literatürde bulunan bir dizi boşluğun giderildiği değerlendirilmektedir.

Hyperledger Fabric'in tüm gereksinimleri ile birlikte kurulumu kurulumu ile ilgili adım adım anlatımlar literatürde yer alsa da yaptığımız tezde buna tek bir çözüm sunduk. Hazırladığımız bir script ile Fabric'in tüm gereksinimleri ile birlikte kurulumu kolayca ve sağlıklı bir şekilde yapılabilmektedir.

Birden fazla organizasyonlu Fabric ağlarının kurulumu ile bilgiler literatürde yer almaktadır. Ancak Docker Swarm kümeleme aracının getirmiş olduğu port kısıtları bulunmaktadır. Başta üç farklı fiziksel makine kullanılarak gerçekleştirmeye çalıştığımız Fabric ağımızda port yönlendirmeler ile ilgili sorunlar yaşanmıştır. Birçok yöntem denendikten sonra bulut sunucu kullanmanın en sağlıklı yöntem olduğuna karar verilmiştir. Çoklu Fabric ağlarında, fiziksel makine mi kullanmalıyız yoksa bulut sunucu mu kullanmalıyız sorusuna literatürde net bir cevap bulunmamaktadır. Edindiğimiz tecrübeler neticesinde tezimizde bulut sunucu ile birlikte Linux kullanımını önermekteyiz.

Bir Fabric ağının kurulumu ile ilgili literatürde bulunan bilgiler dağınık bir şekilde yer almaktadır. Aynı zamanda Fabric'in son versiyonu ile ilgili net bir kurulum anlatımı bulunmamaktadır. Bütün bir ağı adım adım nasıl kuracağımız modüler bir yapı sağlayan kurulum scriptlerimiz aracılığıyla sağlanmıştır. Tezimizin bu alana önemli bir katkı sağladığı değerlendirilmektedir.

Literatürde IoT ve bir blokzincir ağını entegre etme ile ilgili yöntemler bulunmaktadır. Ancak Hyperledger Fabric ağının MQTT ile hangi seviyelerde entegre edilebileceği ile ilgili bir anlatım bulunmamaktadır. Gerçekleştirdiğimiz tez çalışması ile MQTT'nin üç aktörü olan 'publisher', 'broker' ve 'subscriber' ın bir Fabric ağı ile nasıl entegre olacağı açıklanmıştır. Gerçekleştirdiğimiz çalışmada 'publisher' aktörünün ağın tamamen dışarısında API sunucu içerisinde çalışması gerektiği saptanmış, 'broker' aktörünün tamamen bağımsız bir şekilde çalışacağı değerlendirilmiş ve 'subscriber' aktörünün ise bir Hyperledger Fabric Client'ı üzerinde çalışması gerektiğine karar verilmiştir. Yapılan çalışmalar sonucunda kurduğumuz yöntemde veri aktarımı sorunsuz bir şekilde gerçekleşmektedir. Bu entegrasyonun sağlanmasının tezimizin en önemli katkılarından biri olduğu değerlendirilmektedir.

Tezimizin diğer önemli katkılarından biri ise Fabric ağının bir API sunucu aracılığıyla dış dünyadan işlem yapılabilir hale getirilmesidir. Literatürde genelde bir Command Line Interface (CLI) konteyner aracılığıyla komut satırı üzerinden Fabric ağlarıyla iletişime geçilmesi anlatılmıştır. Tezimizde yaptığımız çalışma ile buna bir yöntem sunmaktayız. Hazırlamış olduğumuz API sunucu Fabric ağının tamamen dışında bir

Docker konteyner olarak çalışmaktadır. Bu API sunucuya POSTMAN aracılığıyla dış dünyadan veri aktarımı sağlanmıştır. Veri aktarımı esnasında MQTT entegrasyonunun sağlıklı bir şekilde çalıştığı gözlemlenmiştir. Aynı zamanda kayıtlı bir kullanıcının 'token' i kullanılarak yetki kontrolü de gerçekleştirilmiş ve ağ içerisinde sadece yetkili kullanıcıların işlem yapabildiği görülmüştür.

Hyperledger Fabric blokzincir ağının sahip olduğu modüler yapı, yüksek düzeydeki güvenlik ve akıllı sözleşmelerin uygulanarak iş mantığını işletebilmesi gibi özellikleri, IoT cihazları ile entegrasyonu ve IoT cihazları ile bütünleşik çalışan diğer tüm kurumsal sektörler için önemli çözümler sunmaktadır.

KAYNAKLAR

[1] juniperresearch.com/press/press-releases/‘internet-of-things’-connected-devices-triple-2021, (Ziyaret tarihi: 20 Mayıs 2021).

[2] Oh S.R., Kim Y.G., Security Requirements Analysis for the IoT, 2017, 1-6.

[3] Panarello A., Tapas N., Merlino G., Longo F., Puliafito A., Blockchain and IoT Integration: A Systematic Survey, *Sensors*, 2018, 2575.

[4] https://ec.europa.eu/food/safety/general_food_law/general_requirements_en, (Ziyaret tarihi: 20 Mayıs 2021).

[5] http://www.hal.gov.tr/Sayfalar/Kanun.aspx, (Ziyaret tarihi: 20 Mayıs 2021).

[6] Shahid N., Aneja S., Internet of Things: Vision, application areas and research challenges, *International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2017, 583-587.

[7] Singh S., Singh N., Internet of Things (IoT): Security challenges, business opportunities reference architecture for E-commerce, *International Conference on Green Computing and Internet of Things (ICGCIoT)*, 2015, 1577-1581.

[8] https://bitcoin.org/bitcoin.pdf, (Ziyaret tarihi: 20 Mayıs 2021).

[9] Zheng Z., Xie S., Dai H.-N., Chen X., Wang H., Blockchain challenges and opportunities: a survey, *International Journal of Web and Grid Services*, 2018, 352-375.

[10] Deogirikar J., Vidhate A., Security attacks in IoT: A survey, *International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2017, 32-37.

[11] Atlam H., Walters R., Wills G., Internet of Things: State-of-the-art, Challenges, Applications, and Open Issues, *International Journal of Intelligent Computing Research*, 2018, 928-938.

[12] http://www.itrco.jp/libraries/RFIDjournal-That%20Internet%20of%20Things%20Thing.pdf, (Ziyaret tarihi: 20 Mayıs 2021).

[13] https://www.itu.int:443/en/ITU-T/gsi/iot/Pages/default.aspx, (Ziyaret tarihi: 20 Mayıs 2021).

[14] itu.int/ITU-T/recommendations/rec.aspx?rec=y.2060, (Ziyaret tarihi: 20 Mayıs 2021).

[15] Zhong C.-L., Zhu Z., Huang R.-G., Study on the IOT Architecture and Gateway Technology, *14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, 2015, 196-199.

[16] Bandyopadhyay S., Bhattacharyya A., Lightweight Internet protocols for web enablement of sensors using constrained gateway devices, *International Conference on Computing, Networking and Communications (ICNC)*, 2013, 334-340.

[17] eclipse.org/community/eclipse_newsletter/2014/february/article2.php, (Ziyaret tarihi: 20 Mayıs 2021).

[18] Thangavel D., Ma X., Valera A., Tan H.-X., Tan C. K.-Y., Performance evaluation of MQTT and CoAP via a common middleware, *IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2014, 1-6.

[19] Ludovici A., Moreno P., Calveras A., TinyCoAP: A Novel Constrained Application Protocol (CoAP) Implementation for Embedding RESTful Web Services in Wireless Sensor Networks Based on TinyOS, *Journal of Sensor and Actuator Networks*, 2013, 288-315.

[20] https://theinternetofthings.report/whitepapers/messaging-technologies-for-the-industrial-internet-and-the-internet-of-things/3149, (Ziyaret tarihi: 20 Mayıs 2021).

[21] Han N., Semantic service provisioning for 6LoWPAN : powering internet of things applications on Web, 2015.

[22] Abdel-Basset M., Manogaran G., Mohamed M., Internet of Things (IoT) and its impact on supply chain: A framework for building smart, secure and efficient systems, *Future Generation Computer Systems*, 2018, 614-628.

[23] Tyagi S., Agarwal A., Maheshwari P., A conceptual framework for IoT-based healthcare system using cloud computing, *6th International Conference - Cloud System and Big Data Engineering (Confluence)*, 2016, 503-507.

[24] Kim T., Ramos C., Mohammed S., Smart City and IoT, *Future Generation Computer Systems*, 2017, 159-162.

[25] Arias O., Wurm J., Hoang K., Jin Y., Privacy and Security in Internet of Things and Wearable Devices, *IEEE Transactions on Multi-Scale Computing Systems*, 2015, 99-109.

[26] Nofer M., Gomber P., Hinz O., Schiereck D., Blockchain, *Bus Inf Syst Eng*, 2017, 183-187.

[27] Peters G., Panayi E., Understanding Modern Banking Ledgers Through Blockchain Technologies: Future of Transaction Processing and Smart Contracts on the Internet of Money, *Social Science Research Network*, Rochester, NY, 2015.

[28] Lai R., LEE K. C. D., Chapter 7 - Blockchain – From Public to Private, Handbook of Blockchain, Digital Finance, and Inclusion, Volume 2, *Ed. Academic Press*, 2018, 145-177.

[29] https://www.hyperledger.org/use/fabric, (Ziyaret tarihi: 20 Mayıs 2021).

[30] https://developer.ibm.com/recipes/tutorials/writing-hyperledger-fabric-chaincode-using-go-programming-language/, (Ziyaret tarihi: 20 Mayıs 2021).

[31] https://learn.nextdecentrum.com/hyperledger-architecture-volume-1, (Ziyaret tarihi: 20 Mayıs 2021).

[32] hyperledger.org/learn/publications/blockchain-performance-metrics, (Ziyaret tarihi: 20 Mayıs 2021).

[33] Samaniego M., Deters R., Blockchain as a Service for IoT, 2016, 433-436.

[34] Huh S., Cho S., Kim S., Managing IoT Devices using Blockchain Platform, 2017, 4.

[35] Bocek T., Rodrigues B. B., Strasser T., Stiller B., Blockchains everywhere - a use-case of blockchains in the pharma supply-chain, *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbon, Portugal, 2017, 772-777.

[36] Košťál K., Helebrandt P., Belluš M., Ries M., Kotuliak I., Management and Monitoring of IoT Devices Using Blockchain, *Sensors*, Basel, 2019.

[37] Rožman N., Vrabič R., Corn M., Požrl T., Diaci J., Distributed logistics platform based on Blockchain and IoT, *Procedia CIRP*, 2019, 826-831.

[38] https://hyperledger-fabric.readthedocs.io/en/release-2.2/prereqs.html, (Ziyaret tarihi: 20 Mayıs 2021).

[39] https://docs.docker.com/engine/swarm/swarm-tutorial/, (Ziyaret tarihi: 20 Mayıs 2021).

[40] https://docs.docker.com/engine/swarm/, (Ziyaret tarihi: 20 Mayıs 2021).

[41] https://docs.docker.com/network/, (Ziyaret tarihi: 20 Mayıs 2021).

[42] https://docs.docker.com/network/network-tutorial-overlay/, (Ziyaret tarihi: 20 Mayıs 2021).

[43] hyperledger-fabric.readthedocs.io/en/release-2.2/smartcontract/smartcontract.html, (Ziyaret tarihi: 20 Mayıs 2021).

KİŞİSEL YAYIN VE ESERLER

[1] Dikilitaş Y., **Toka K.O.**, Sayar A., Current Research Areas in Blockchain, *International Congress on Human Computer Interaction, Optimization and Robotic Applications*, Ankara, 11-13 Haziran 2021.

ÖZGEÇMİŞ

Lise öğrenimini Milas Anadolu Lisesi'nde tamamladı. 2011 yılında girdiği Trakya Üniversitesi Bilgisayar Mühendisliği Bölümü'nden 2015 yılında mezun oldu. 2019 yılında Kocaeli Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı'nda yüksek lisans eğitimine başladı. Yüksek lisans eğitiminde blokzincir teknolojileri konusunda çalışmaktadır.