On the Applications of Consortium Blockchain Technology as a Dedicated Security Network Deployment

Master's Degree Project

Robert J. Kretschmar III
armersuender@csu.fullerton.edu

California State University, Fullerton

Contents

1 Abstract

I am the Abstract.

2 Introduction

2.1 Background

Blockchain technology has been a disruptive force in this world. Originally, that disruption was felt in the financial sector with Bitcoin. This relegation to the financial sector existed because the Bitcoin scripting language Script was intentionally non-Turing complete; perfect for its Use Case as a cryptocurrency, but otherwise significantly under-powered for general applications. However, it wasn't long until the next level came in the form of Ethereum which introduced a Turing-complete scripting language along side its cryptocurrency offering. This opened up the world to the possibility of decentralized applications (dApps). While those platforms were necessarily public and open, others still thought about other Use Cases that would benefit from the use of this type of distributed technology – ledgers in the form of cryptographically connected blocks, decentralized node consensus, scripting to build applications on these platforms, etc. – in a private and permissioned space. One of those offerings was a system, itself defined and built by a consortium of organizations (from Intel to IBM), maintained by The Linux Foundation called The Hyperledger Project.

FabSec (short for Fabric Security) is an exploration in the potential of using Hyperledger Fabric's Distributed Ledger Technology[1] as a dedicated security network. Hyperledger is an ecosystem of different tools, libraries and frameworks for creating different types of Blockchains: from private and permissioned to public and permissionless. Fabric is one of those Blockchain frameworks. It allows multiple actors to share a blockchain between themselves for any Use Case which to they could think to apply it.

2.2 Project Motivation

This project attempts to explore use of this technology in the security space, namely having a dedicated security network between actors. These actors can be two

[1]A note about terminology: You'll see a few terms being used here such as Consortium Blockchain and Distributed Ledger. For the purposes of this project, these are interchangeable. This holds as a Blockchain system at its base is nothing more than a Distributed Ledger of transactions. It's just as the public Blockchain sphere increases, that term generally refers to the Bitcoin, Ethereum, et al menagerie of systems. Another example seen later of this is Chaincode vs Smart Contracts.

(or more) organizations looking to join forces and pool security resources. If fact, how this idea got started was thinking about having a dedicated security network as an overlay to something like a Wide-Area Network (WAN) specifically that of a Metropolitan-Area Network (MAN). You could have multiple organizations within a city each helping to strengthen the security mission of their networks without having that security centralized as a city is often its own ecosystem of businesses, departments, and other stakeholders. An extended hope is that this could one day be applied to non-permissioned and/or public blockchains in future work.

The choice to start with using Fabric for this idea was the ability to have full control of the blockchain in question while the structure was being planned out and the scripts and chaincode were being developed. A public blockchain such as Etheruem sounded like too many unknown variables right out-of-the-gate. That being said, and as mentioned above, it is a hope that once the plans are solidified translating this work to a public blockchain won't be too difficult. However, something thought about after this choice was made was doubling down on the idea of using a permissioned blockchain such as Fabric for a real implementation, such as for a MAN. The beauty of the idea is that its easily translatable to many different platform. For security network applications, I believe this could have great value in the realms of Public Key Infrastructure (PKI), Two-Factor Authentication (2FA), Distributed Denial of Service (DDoS) prevention, Domain Name System Security (DNSSec), and beyond! The Proof-of-Concept of his research project will be a single blockchain – or Channel in Hyperledger parlance – to be used as a distributed log aggregator. As anyone in blue team security can tell you, the logs are everything!

2.3 Project Objectives

The objectives of this project are:

- To demystify the Hyperledger Fabric Distributed Ledger Technology
- To test the applications of a permissioned blockchain system in the realm of Computer Security
- TODO: Add a couple more objectives.

2.4 Key Achievements

I am the Key Achievements.

3 Hyperledger Fabric

Hyperledger Fabric (or Fabric for short) is, at its core, a private, permissioned Blockchain technology. This Blockchain technology goes under a the differential term of Distributed Ledger Technology to (a) stay in line with the more "enterprise-like" Use Cases for which certain organizations will turn to it, and (b) stand out from other Blockchain technologies like that of Bitcoin and Ethereum. Unlike those open, permissionless offerings where nodes are typically anonymous, Fabric uses X509 certificates to establish identities on its network. Along with this difference, Fabric also employs a *deterministic* consensus algorithm whereas the open offerings use *probabilistic* versions.

However, Fabric does have a lot in common with the open offerings such as its commitment to open governance,

3.1 The Certificate Authority Servers

The Certificate Authority (CA) Servers of the Fabric System are a logical place to start as they are the first thing to get administered in a production environment. As mention above, nodes on the Fabric network all have identities associated with them. This is what enforces the *permissioned* quality of the system. These identities come in the form a x509 certificates. The certificates are created and issued by a couple of CA Servers[2]: The Organizational (Fabric) CA Server and the TLS CA Server.

The TLS CA Server is the one that secures communications of the different nodes using, surprisingly, the Transport Layer Security (TLS) Protocol. All nodes on the network need to have a certificate provided by the TLS Server to be able to communicate. Now, TLS isn't enable by default for testing purposes, but since I will be talking about my project as a production network, one can assume moving forward that every node has been registered and enrolled with the TLS Server. Other than that, these servers are relatively straight forward in the purpose, and not the more interesting of the two.

The more interesting of the two CA Servers is the Organizational (Fabric) Server. A quick comment on why I'm using "Organizational" and "Fabric" here: the

[2]While the Fabric system allows for the use of Intermediate CA Servers, my project won't be using them so they won't be mentioned further.

Hyperledger Fabric Guides refer to the server as the Organizational Server to differentiate it from the TLS Server. However, I find the term "Organizational" lacking (read: overloaded) as this term is used often when talking in relation to the other aspects of the participating Organizations. For this reason, I opted to use the term "Fabric CA" when talking about this CA. The word "Fabric" is, of course, itself an overloaded term in these guides as well, but far less in my estimation than that of "Organizational". That being said, moving forward, I will just be using the term "Fabric" or "Fab" CA Server to refer to this entity.

The Fabric CA Server is the server that manages the identities on the network as they are recognized by the Fabric Network. There are four types of Identities: admin, client, peer, and orderer. These are encoded into the certificates that an entity will get. Notice I didn't say just "node" as only two of those are used for nodes, specifically the peer and orderer. The other two are used for entities that act as administrators for either a given node or an organization and the end clients that will eventually be using the system. Certificate management and enforcement is done via a construct called Membership Service Providers (MSPs).

3.2 Membership Service Providers

Membership Service Providers (MSPs), unlike their name suggests, don't *actually* provide anything and instead are a well-defined way for a Fabric network to consume an identity. A lot of elements of the Fabric network need to know identities to make sure everything is correct with regard to its distributed nature from endorsing peers to orderer signing to admin management and more. To not get lost in the certificate nightmare one might imagine this to be, MSPs hold that cryptographic material. Every though I have only been mentioning certificates up until now, there are host of the crypto that the MSP holds such as the signing key and the root CA certificate. This is an example of a MSP listing for the Orderer Organization of my project:

```
organizations/
|-- ordererOrganizations/
    |-- org0.fabsec.com/
        |-- msp/
            |-- cacerts/
            |   |-- hypertest-7055.pem
            |-- IssuerPublicKey
            |-- IssuerRevocationPublicKey
            |-- keystore/
            |   |-- key.pem
            |-- signcerts/
            |   |-- cert.pem
            |-- user/
```

While each MSP may hold different material, their structure is pretty standard. Harkening back to the topic of CAs, the Fabric CA is responsible for generating and delivering these MSPs upon a successful enrollment command. There are informally two type of MSPs: Channel MSPs which belong to the Organizations that are to administer and participate in a given Channel (more on those below) and the Local MSPs which define what role the entity to whom that MSP belongs to will play.

3.3 Peers

A Peer Node is the node that contains, or hosts, the Ledgers and the Chaincode. More speficially they host *instances* of the Ledgers and Chaincode as every peer within a Channel will host their own copy. This is another way the decentralized nature of this system shows itself. Peers are the main mechanism by which clients, usually applications, and administrators interact with the Ledger through calls to their Chaincode. They will also endorse transactions, which is done by the act of signing with their private key, that will then get sent to an Orderer Node. TODO: Finish Peers.

3.4 Orderers

An Orderer Node is the node that collects the different transactions and *orderers* them into the blocks that will eventually go on the ledger, i.e. the blockchain. In a public blockchain system, every node has the chance to be "an orderer" so-to-speak however they call them Miners. While this isn't a paper on those types of systems, it is important to understand that that is another thing Fabric (and I'm sure other permissioned blockchain solutions) does: separates the role of the Peer from an Orderer (Miner) – whereas every node in a public blockchain has the potential to have all of the functional of a Peer and Orderer (Miner).

Fabric doesn't need "miners" since their consensus protocol is not based on any form of cryptographic mining. The main reason for this is the act of cryptographic mining is used to make sure the any node has a chance to propose blocks for the chain ensuring that no one node, or group of nodes, has control over the network and thus preserving its decentralized nature in a trustless environment. Fabric doesn't have the problem of a *trustless environment* to overcome. Every node has a known identity and a known parent organization.

So, Fabric has separated these roles of Peers – which talk to clients, execute Chaincode, and endorse transactions – from Orderers – which order the transactions into blocks and distributes those blocks to the Peers. An important thing to note is that an Orderer doesn't actually validate any of the transactions that they package into blocks as that is done by the Peers. They instead enforce the access controls in two ways. One, they maintain a list of Organizations that are participating in the Consortium. And two, they control the read/write access of a given Channel for which they are an Orderer. (More on Channels and how Orderer get these policies in the Channel section.)

Finally, the Orderers actually have a choice of consensus protocols from which to choose. Now, granted my project only has one Orderer, so it doesn't have to actually reach consensus with any other node than itself. However even for an Orderer of one, a Fabric network architect still has to choose, set-up, and maintain one of the consensus protocols. And further, Fabric is designed in such a way the Orderers can be added and removed from the Ordering Service[3] at any time meaning that the consensus protocol is just as important to an Ordering Service of one as it is to an Ordering Service of five. The one that is the default, and as of version 2.0 of Fabric the recommended, choice is that of Raft. So this is the protocol I will be going with.

3.4.1 The Raft Consensus Protocol

The Raft Consensus Protocol is an embedded, Crash-Fault Tolerated protocol. It uses a model in which a leader is elected among the set of Orderers based on random time intervals. TODO: Finish Raft.

[3]A group of Orderers is known as an Ordering Service.

3.5 Channels

Channels are the main way that nodes associate themselves with not only other nodes, but with the Ledgers and Chaincode as well. They act rather similarly to a subnet keeping communications confidential for only those within a channel. As such, it is how an Orderer can do its job of access control. Each node can belong to multiple Channels which gives Fabric is claim to "multi-tenancy" fame. In fact, with my conception of how you would have Fabric be a security network, each of the Use Cases I mentioned PKI, DDoS protection, etc. would use their own Channel. This is a nice way to logically separate concerns since PKI management is not reliant on DDoS protection and vice versa, so why have them logically muddled on the same network? TODO: Finish Channels

3.5.1 The Genesis Block

The Genesis Block is a very important element of the Fabric network. As its name suggests, it is the first block on any given Blockchain on the network, and it is needed to *bootstrap* the first Orderer of an Ordering Service. TODO: Finish Gen Block.

3.6 Chaincode

Chaincode, as mentioned briefly before, is the Hyperledger equivalent to Smart Contracts. These are Turing-complete programs that hold the business logic for a given Use Case, such as the logging aggregator logic of this project. In that spirit, they are used to encapsulate the *shared processes* of the Fabric Network. TODO: Finish Chaincode.

3.7 Ledgers

Ledgers are immutables of all the transactions generated by the Chaincode. They encapsulate the *shared information* in a Fabric network. TODO: Finish Ledgers

4 System Design

This section will detail the design of my Logging Aggregator System. To better illustrate the network, I'll start with a image:

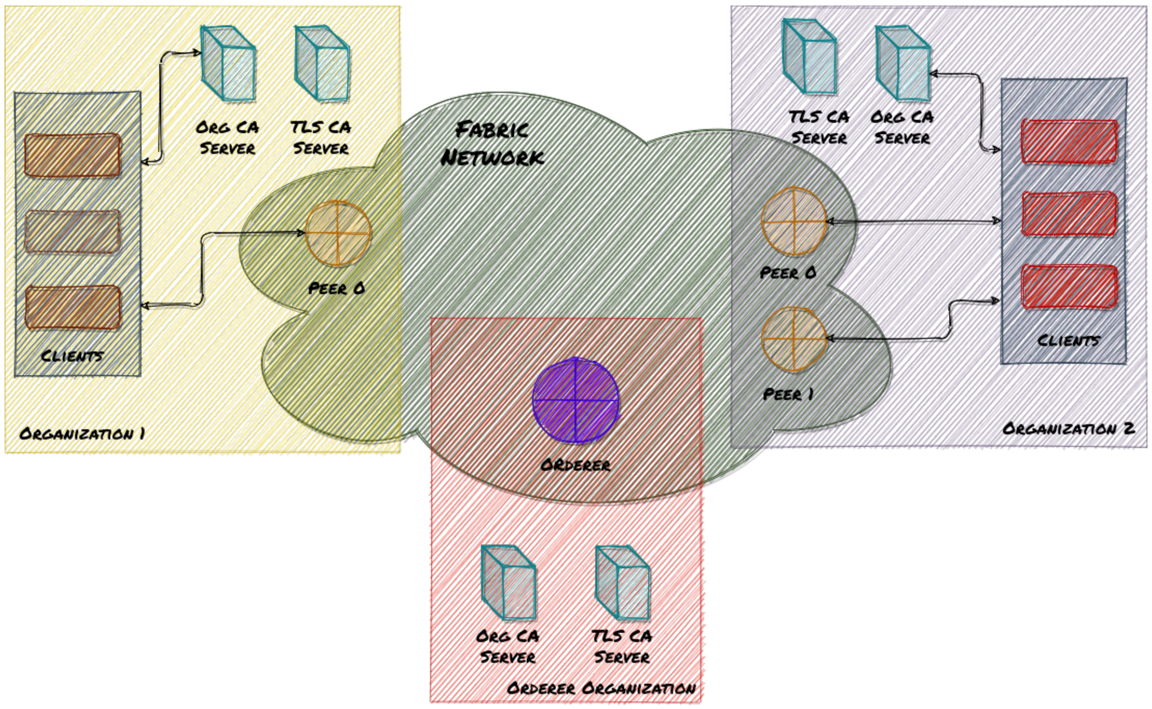

Figure 1: The FabSec Network Topography

At a high level this system will have three Organizations involved: two Peer Organizations and an Orderer Organization. The Peer Organizations will be the participants that will effective be the users of the system. In the context of the Log Aggregator network, they will be the ones collecting the log messages to send to the Orderer as messages in the transactions, and they are responsible for retrieving those message back at the appropriate time. The Orderer Organization acts as the consensus nodes. It will do the assembling of transactions into blocks, validates those transactions/blocks, and finally disseminates the blocks to the peers. Since this is a distributed network, the Orderer will be a "third-party" outside of those that are peers to a particular Channel.

The following is a directory listing view of how the structure for this project will look:

```
organizations/
|-- ordererOrganizations/
|   |-- org0.fabsec.com/
|       |-- ca-client/
|       |   |-- fab-ca/
|       |   |-- fabric-ca-client -> ../../../../bin/fabric-ca-client
|       |   |-- tls-ca/
|       |   |-- tls-root-cert/
|       |-- fab-ca-server/
|       |   |-- fabric-ca-server -> ../../../../bin/fabric-ca-server
|       |   |-- tls/
|       |-- orderers/
|       |   |-- orderer0.org0.fabsec.com/
|       |       |-- blockstore/
|       |       |-- configtx/
|       |       |   |-- configtxgen -> ../../../../../../bin/configtxgen
|       |       |   |-- configtx.yaml
|       |       |-- etcdraft/
|       |       |   |-- snapshot/
|       |       |   |-- wal/
|       |       |-- orderer -> ../../../../../bin/orderer
|       |       |-- PeerOrgsMSPs/
|       |       |   |-- org1/
|       |       |   |-- org2/
|       |       |-- system-genesis-block/
|       |-- tls-ca-server/
|           |-- fabric-ca-server -> ../../../../bin/fabric-ca-server
|-- peerOrganizations/
    |-- org1.fabsec.com/
    |   |-- ca-client/
    |   |   |-- fab-ca/
    |   |   |-- fabric-ca-client -> ../../../../bin/fabric-ca-client
    |   |   |-- tls-ca/
    |   |   |-- tls-root-cert/
    |   |-- fab-ca-server/
    |   |   |-- fabric-ca-server -> ../../../../bin/fabric-ca-server
    |   |   |-- tls/
    |   |-- peers
    |   |   |-- peer0.org1.fabsec.com/
    |   |       |-- blockstore/
    |   |       |-- ca-client/
    |   |       |   |-- fab-ca/
    |   |       |   |-- fabric-ca-client -> ../../../../../../bin/fabric-ca-client
    |   |       |   |-- tls-ca/
    |   |       |   |-- tls-root-cert/
    |   |       |-- channel-artifacts/
    |   |       |-- configtx/
    |   |       |   |-- configtxgen -> ../../../../../../bin/configtxgen
    |   |       |   |-- configtx.yaml
    |   |       |-- peer -> ../../../../../bin/peer
    |   |-- tls-ca-server/
    |       |-- fabric-ca-server -> ../../../../bin/fabric-ca-server
    |-- org2.fabsec.com/
        |-- ca-client/
        |   |-- fab-ca/
        |   |-- fabric-ca-client -> ../../../../bin/fabric-ca-client
        |   |-- tls-ca/
        |   |-- tls-root-cert
        |-- fab-ca-server/
        |   |-- fabric-ca-server -> ../../../../bin/fabric-ca-server
        |   |-- tls/
        |-- peers/
        |   |-- peer0.org2.fabsec.com/
        |   |   |-- blockstore/
        |   |   |-- ca-client/
        |   |   |   |-- fab-ca/
        |   |   |   |-- fabric-ca-client -> ../../../../../../bin/fabric-ca-client
        |   |   |   |-- tls-ca/
        |   |   |   |-- tls-root-cert/
        |   |   |-- channel-artifacts/
        |   |   |-- configtx/
        |   |   |   |-- configtxgen -> ../../../../../../bin/configtxgen
        |   |   |   |-- configtx.yaml
        |   |   |-- peer -> ../../../../../bin/peer
        |   |-- peer1.org2.fabsec.com/
        |       |-- blockstore/
        |       |-- ca-client/
        |       |   |-- fab-ca/
        |       |   |-- fabric-ca-client -> ../../../../../../bin/fabric-ca-client
        |       |   |-- tls-ca/
        |       |   |-- tls-root-cert/
        |       |-- channel-artifacts/
        |       |-- configtx/
        |       |   |-- configtxgen -> ../../../../../../bin/configtxgen
        |       |   |-- configtx.yaml
        |       |-- peer -> ../../../../../bin/peer
        |-- tls-ca-server/
            |-- fabric-ca-server -> ../../../../bin/fabric-ca-server
```

Note: To save space, this is not the entire listing with all of the different files that are needed to make the system work. However, it is a good guide illustration of the general structure of not only how it looks for, say, a node like orderer0.org0.fabsec.com vs an organization like org0.fabsec.com, but how the organizations are logically separated.

There will be a dedicated TLS CA and a dedicated Fabric CA for each Organization. This choice is to reflect a real-world scenario where geographically separated Organizations will have their own set of CAs. Speaking of CAs, while it is an option to deploy intermediate CAs (and a solid security decision to do so) for scalability, this project won't be using any. Another decision made in dealing with CA is that of credentials. Credentials, i.e. username and password, are used when register and enrolling entities that will eventually become identities. For now, I have made these static within the scripts that help fire up the network, however a future consideration would be to make them dynamic, allowing a script operator to feed them as commandline arguments to the script.

There are two options for, what's called, the State Database: CouchDB and LevelDB. The State Database is a database maintained by each Peer node and presents an indexed view of the current values for all of the assets on the Ledger. While a deep dive into this database is outside the scope of this project, let it be said that LevelDB is the default, and embedded, choice for this decision and the one used for this project.

Port management is a bit of a nightmare here. In a real world system, this will not be as hectic as an Organization will likely be running all of the nodes on separate physical machines. However, for this project, I'm hardcoded the ports that all of the different entities will run on. They are as follows:

- Org0, TLS CA – main: 7054, operations: 9443
- Org0, Fab CA – main: 7055, operations: 9444
- Org0, orderer0 – main: 6050, operations: 8443
- Org1, TLS CA – main: 7056, operations: 9445
- Org1, Fab CA – main: 7057, operations: 9446
- Org1, peer0 – main: 6051, chaincode: 6052, operations: 8446
- Org2, TLS CA – main: 7058, operations: 9447
- Org2, Fab CA – main: 7059, operations: 9448
- Org2, peer0 – main: 6053, chaincode: 6054, operations: 8447
- Org2, peer1 – main: 6055, chaincode: 6056, operations: 8448

5 Um, I dunno, a section like Transaction Flow or something?

I feel like I need another section here, but I don't have any quantitative data to show nor am I comparing it against anything (i.e. timing, performance, etc.) since it's a more of an exploratory project. Perhaps I can put something qualitative here once I have more data on how my chaincode operates. (Should be coming this week!)

6 Summary

In this project, I created a Hyperledger Fabric network from the bottom up with the idea in mind of creating a dedicated security network. The network consists of three Organizations: two Peers and an Orderer. This is to reflect a real world network on which to deploy my chaincode for a Distributed Logging Aggregator. Of course, this is just an illustive Use Case, and it is my hope to apply different security-focused Use Cases such as, but not limited to, Public Key Infrastructure (PKI), Two-Factor Authentication (2FA), Distributed Denial of Service (DDoS) prevention, Domain Name System Security (DNSSec), and others.

7 Conclusion

Not sure about my conclusion yet.