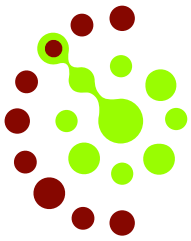

IPBeja

Instituto Politécnico de Beja

**Escola Superior de Tecnologia e Gestão**

**Mestrado em Engenharia de Segurança Informática**

# Blockchain para Dados Críticos com Hyperledger Fabric

## Uma aplicação para dados de qualidade da água adquiridos por técnicas IoT

*Carlos Alexandre Rijo Palma*

Beja, 1 de Fevereiro de 2022

**INSTITUTO POLITÉCNICO DE BEJA**

**Escola Superior de Tecnologia e Gestão**

**Mestrado em Engenharia de Segurança Informática**

# Blockchain para Dados Críticos com Hyperledger Fabric

## Uma aplicação para dados de qualidade da água adquiridos por técnicas IoT

Carlos Alexandre Rijo Palma

Orientado por :

Doutor José Jasnau Caeiro , IPBeja

Dissertação, realizada no Mestrado de Engenharia de Segurança Informática,
apresentado na
Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Beja

2022

# Resumo


## *Blockchain para Dados Críticos com Hyperledger Fabric*

***Uma aplicação para dados de qualidade da água adquiridos por técnicas IoT***

A dissertação apresenta um trabalho que teve como preocupação principal a construção duma aplicação com tecnologias *blockchain*, garantindo a segurança de dados em sistemas IoT. Os dados são de qualidade da água.

As tecnologias *blockchain* garantem a segurança e a fiabilidade dos dados armazenados. Esta tecnolgia é um sistema de blocos encadeados que representa um livro da razão que contém todas as transações. A arquitetura distribuída e descentralizada da tecnologia *blockchain* garante que as transações são imutáveis, uma vez que não podem ser alteradas.

Para o desenvolvimento da aplicação é usada a *framework* ***Hyperledger Fabric***. Este permite criar um *blockchain* privado e com permissões. Os participantes são inscritos no sistema por um provedor de serviços (MSP).

É apresentado um exemplo de como pode ser dividida a rede de barragens ao nível Nacional. Os participantes da rede são designados de organizações e associados a cada distrito de Portugal. Cada organização é composta por vários *peers* que correspondem a um recurso hídrico.

A recolha dos dados da água é feita através de sensores digitais que comunicam com um Micro Computador (Device IoT). Estes dados são enviados através de um protocolo de mensagens (MQTT).

Foi desenvolvido um cliente *Fabric* para obter os dados através do protocolo MQTT e enviar os mesmos para o *peer* para serem guardados no *blockchain*. Estes dados são guardados no *blockchain* de acordo com um conjunto de regras que foi definido na criação do contrato inteligente.



**Palavras-chave**: *Blockchain, IoT, Hyperledger Fabric, Qualidade da água, rede.*

# Abstract


## *Blockchain para Dados Críticos com Hyperledger Fabric*

***Uma aplicação para dados de qualidade da água adquiridos por técnicas IoT***

The dissertation presents a work whose main concern was the construction of an application with blockchain technologies, ensuring data security in IoT systems. The data is water quality.

Blockchaing technologies ensure the security and reliability of stored data. This technology is a system of linked blocks that represents a ledger that contains all transactions. The distributed and decentralized architecture of the blockchain technology means that transactions are immutable, as they cannot be changed. For application development the framework Hyperledger Fabric is used. This allows you to create a private and entitled blockchain. Participants are enrolled in the system by a service provider (MSP).

An example of how the dam network can be divided at the National level is presented. Network participants are designated associations and associated with each district of Portugal. Each organization is composed of several peers corresponding to a water resource.

The designation of water data is made through digital sensors that communicate with a Micro Computer (Device IoT). These data are sent from a message protocol (MQTT).

A Fabric client was developed to obtain the data through the MQTT protocol and send them to the peer to be saved in the blockchain. This data is saved in the blockchain according to a set of rules that was defined when creating the smart contract.



**Keywords**: *Blockchain, IoT, Hyperledger Fabric, water quality, network.*

# *Agradecimentos*

A presente dissertação não seria a mesma sem precioso apoio de várias pessoas.

Agradecer à minha mulher, Cátia Gusmão e aos meus filhos (Gonçalo e Madalena) pela paciência que tiveram, pelo apoio, força e energia que me fizeram passar ao longo do tempo no desenvolvimento da dissertação. Agradecer à minha mãe, ao meu pai e minha irmã por me apoiarem e encorajarem a nunca desistir.

Agradecer ao meu orientador Professor Doutor José Jasnau Caeiro por todo o empenho e disponibilidade com que sempre me orientou neste trabalho e por toda a confiança que em mim depositou.

Agradecer ao meu Amigo e colega António Baião o apoio e motivação tornando este trabalho uma enorme experiência de aprendizagem.

Por fim, mas não menos importante, um especial agradecimento a todos os professores que fizeram parte desta jornada.

Este trabalho é o resultado de todos os conhecimentos adquiridos juntamente com o apoio de todos, em cima enumerados.

# Índice

# Índice de Figuras

# Índice de Tabelas

# Índice de Listagens

# Capítulo 1

# Introdução

A dissertação tem como tema o desenvolvimento de uma aplicação usando as tecnologias *blockchain* para garantir a segurança de dados críticos em sistemas IoT. Esta aplicação teve como foco uma área critica como são os dados da qualidade da água. A água é um bem precioso e imprescindível à vida na Terra. Sem água em quantidade e qualidade, a sociedade humana não consegue garantir aos seus cidadãos qualidade de vida, nem manter os níveis mínimos de saúde pública.

Após uma analise ao local da internet do Sistema Nacional de Informação de Recursos Hídricos (SNIRH) em Portugal, pode-se verificar que existem muitos pontos de recolha de dados da água marcados no Mapa. Estes por vezes, não têm os dados disponíveis para certos períodos temporais. Isto pode ser um problema para as entidades que tratam os dados da água e verificam a qualidade da mesma. Uma outra questão que se levanta ao observar esta informação é a seguinte: Será que os dados são fiáveis?

Os dados estão numa base de dados do SNIRH. A aplicação desenvolvida nesta dissertação apresenta uma proposta de solução para parte destes problemas. Usa as tecnologias *blockchain* de forma a descentralizar os dados. Torna-os credíveis, transparentes, garante que os mesmos não podem ser alterados.

Com esta tecnologia ao não existir uma autoridade central responsável pelo processamento dos dados, vem dar mais segurança aos mesmos. Sendo que os dados estão distribuídos pelas várias identidades da rede blockchain. Assim não vai existir desconfiança sobre a autoridade central.

Os dados ao estarem distribuídos e encadeados são seguros uma vez que para serem alterados tinha que se alterar uma infinidade de registos e em diferentes base de dados. De referir que as transações são também seguras, devido ao fato de usarem uma chave privada para as criptografar e uma chave pública para poder ser acedidas pelo receptor.

A transparência dos dados é mantida devido ao fato de todos os membros que participam na rede participarem na validação das transações. E assim é mantida a confiança dos transmissores e receptores das transações.

No desenvolvimento desta aplicação é usada a *framework* ***Hyperledger Fabric***. Uma

*framework* que permite criar um *blockchain* privado. Restringe aos membros da rede certas permissões, quer para aceder, quer para gravar os dados.

A aplicação mostra um exemplo de como pode ser dividida a rede ao nível Nacional. Neste caso é mostrada uma rede de barragens do Sul de Portugal, dividida por organizações. Sendo que cada organização corresponde a um distrito [1], que poderá ser expandida a outros distritos. Cada organização é composta pelos vários *peers* que correspondem a um recurso hídrico em cada distrito. Cada *Peer* tem um livro razão [2] de forma a tornar os dados descentralizados e seguros.

Foi desenvolvido um contrato inteligente, onde foram criadas a regras de inserção e consulta dos dados. Assim cada membro da rede (consoante as suas permissões) pode chamar as funções do contrato inteligente de forma a inserir os dados e a consultar os mesmos.

A dissertação é dividida em cinco capítulos. O capítulo 1 que descreve o âmbito em que se integra a mesma, o trabalho desenvolvido pelo autor e a estrutura da dissertação. O capítulo 2 apresenta as tecnologias *blockchain* e a evolução que as mesmas foram tendo ao longo do tempo. Enumera alguns dos tipos existentes e mostra as várias áreas onde podem ser usadas as tecnologias *blockchain* para dispositivos IoT. O capítulo 3 fornece informação sobre a arquitetura do sistema e mostra as potencialidades e vantagens da *framework Hyperledger Fabric*. Apresenta ainda o modo como são recolhidos os dados da água, como são guardados no *blockchain* e ainda como podem ser consultados. O capítulo 4 debruça-se sobre a forma como foi implementada a aplicação usando o *Hyperledger Fabric*. Apresenta as várias fases para a sua construção. No capítulo 5 são apresentadas as conclusões finais e ainda as perspetivas de trabalho futuro.

[1] Beja, Évora, *etc*

[2] Ledger: https://hyperledger-fabric.readthedocs.io/en/latest/ledger.html?highlight=ledger

# Capítulo 2

# Estado da Arte

## 2.1 Introdução

Existem sistemas informáticos baseados em métodos e tecnologias associadas à Internet das Coisas (IoT) que produzem informação crítica. A área da qualidade da água é um dos exemplos de sistemas IoT que apresenta requisitos para a segurança dos dados. As tecnologias *Blockchain* permitem realizar estes requisitos.

Este capítulo apresenta o estudo da literatura que foi feito previamente para o desenvolvimento do projeto. Mostra a evolução da tecnologia *Blockchain*, descutem-se os vários tipos de *Blockchain* e apresentam-se as várias áreas onde se podem aplicar estas tecnologias usando os dispositivos IoT.

O capítulo é dividido nas seguintes secções: a secção 2.2 que explica o que é a Tecnologia *Blockchain*, a sua evolução e os vários tipos de *blockchain* existentes, onde pode ser aplicada esta tecnologia em IoT.

A secção 2.3 apresenta as conclusões do estado da arte.

## 2.2 A Tecnologia Blockchain

A tecnologia *Blockchain* é, hoje em dia, utilizada por um elevado número de empresas a fim de monitorizar e movimentar qualquer número de ativos em todo o mundo. É uma tecnologia revolucionária, resolve o da segurança e fiabilidade dos dados armazenados.

A tecnologia *Blockchain* (BC) teve início no ano de 2008, criada por um personagem que usou o nome fictício de Satoshi Nakamoto e a primeira aplicação em que foi testada foi no Bitcoin. Muitas vezes diz-se que o Bitcoin é o *blockchain* o que faria algum sentido devido ao facto de ter sido a primeira aplicação a usar a tecnologia BC. Na verdade a BC e Bitcoin são coisas distintas.

O BC é uma base dados distribuída e completamente descentralizada e usa uma aplicação que é executada em todas as máquinas associadas ao *blockchain*. É um sistema que funciona a partir de vários blocos encadeados, que representa um livro de razão consti-

tuído por todas as transações. Ao ser criado um novo bloco, este contem a informação do bloco anterior e é adicionado ao *blockchain* existente e replica-se por todos os membros pertencentes ao *blockchain*. Ao ser replicado por todos os membros faz com que o sistema se torne seguro e transparente para todos os membros [GW21].

O BC sendo um sistema distribuído, requer que exista um algoritmo de consenso, algoritmo este que é responsável por manter a integridade e segurança da rede *blockchain*. O primeiro algoritmo de consenso a ser criado na tecnologia *blockchain* foi o *Proof of Work* (PoW) de forma a solucionar o problema dos Generais bizantinos (*byzantine faults*[1]).

O problema do Generais bizantinos trata-se da dificuldade de confiar nas decisões num grupo de generais, onde alguns podem ser maliciosos. O problema consiste em que o império bizantino decide atacar uma cidade, na qual estão N exércitos liderados por N generais. De forma a terem sucesso no ataque todos os exércitos deviam atacar ao mesmo tempo. E cada general ao receber o comando de atacar ou recuar é responsável por transmitir o mesmo comando a cada general mais próximo. Como alguns exércitos tinham generais infiltrados do inimigo, os mesmos podiam alterar o comando e passar a informação errada para os generais mais próximos. A descrição deste problema é a dificuldade de se atingir um consenso descentralizado onde podem existir diversas pessoas que por sua vez podem ou não ser confiáveis.

O algoritmo *PoW* consiste na procura dum valor que quando codificado por um algoritmo como o SHA-256, comece por um número de bits a zero. Sendo que o trabalho médio é exponencialmente proporcional ao número de bits zero necessários. Este valor pode ser verificado executando uma operação de única *hash*. Na rede *bitcoin* a *Pow* incrementa um nonce[2] no bloco até encontrar um valor que produza o *hash* do bloco com os bits zero necessários. Após o esforço no processamento de forma a satisfazer a prova-de-trabalho, o bloco já não pode ser alterado sem que se tenha que refazer todo o trabalho. Sendo os blocos encadeados, modificar um bloco inclui refazer todos os blocos seguintes.

Um outro problema que o algoritmo *PoW* resolve é determinar o consenso da maioria. Se a maioria fosse baseada num caso em que cada endereço IP fosse um voto, o atacante poderia reservar uma grande quantidade de endereços IP e aí ia ter maior decisão dentro da rede. No entanto o algoritmo *PoW* determina que cada CPU tenha um voto. O que faz com que a decisão maioritária na rede sobre as transacções válidas corresponda à cadeia de bloco mais longa, sendo essa que prevalece nas decisões tomadas pelos nós sobre os próximos blocos a serem adicionados na cadeia.

O *hash* representa uma síntese duma enorme quantidade de dados que transforma numa pequena quantidade de informações. Considera-se a impressão digital de um bloco, que se torna fundamental num sistema de blocos encadeados.

[1] *byzantine faults*: https://medium.com/all-things-ledger/the-byzantine-generals-problem-168553f31480

[2] O nonce é uma abreviatura para um número que é usado apenas uma vez. No contexto da mineração das criptomoedas é um número adicionado a um bloco *hash*. É esse número que os mineradores do blockchain calculam para completar a *hash* do bloco.

Quando é gerado um novo bloco que contém o *hash* anterior, é criado uma espécie de selo, que permite verificar e sinalizar se algum bloco foi alterado, e consequentemente, proceder à sua invalidação.

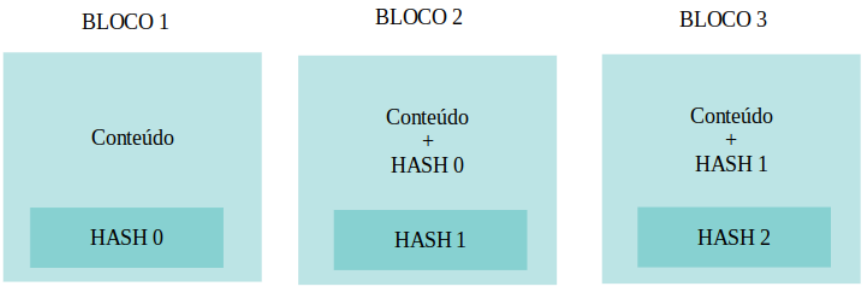


**Figura 2.1:** Exemplo tecnologia *blockchain*.

As informações nos blocos são escritas no *ledger* (livro-razão), onde depois de escritas, não poderão ser eliminadas. Cada rede de *blockchain* também agrupa participantes com o mesmo interesse, nos chamados nós, que podem ser **transaccionais**, escrevem ou geram blocos, ou **mineradores**, que verificam se o bloco é válido.

Os nós mineradores validam, autenticam, certificam e finalizam as transacções. Os nós mineradores recebem *bitcoins* como forma de recompensa pelo trabalho elaborado adicionando um novo valor ao sistema. Após nós mineradores terem criado um novo bloco aceite pelos outros participantes do sistema este já não pode ser modificado, tornando a informação permanente.

### 2.2.1 Definições

#### Blockchain

O Blockchain com a finalidade de registar transações que não possam ser alteradas retroativamente, de forma a mantê-las imutáveis, baseia-se numa arquitetura distribuída e descentralizada. A tecnologia *blockchain* pode ser comparável a um livro público, onde são armazenadas todas as transações ocorridas num sistema. Importa referir que não existe uma autoridade central responsável pelo processamento de transações. Após escrita neste livro-público, uma transação não pode ser alterada. É permitida a inserção de novas transações, no entanto a alteração ou exclusão de uma transação existente não é suportada. Tal operação não é suportada, uma vez que o *blockchain* possui um armazenamento imutável de dados. Antes que uma transação seja incluída no livro público, é necessário existir, através dos seus nós, um acordo em toda a rede. Esse acordo é conseguido através da aplicação de uma arquitetura de rede *peer-to-peer*.

### Flooding

Técnica utilizada para obter informação sobre os nós da rede através da marcação de pacotes como *flooding* transversalmente às interfaces de saída, isto é, para cada pacote recebido numa interface de entrada, são enviados pacotes por meio das interfaces de saída. É extremamente importante numa rede *blockchain*, uma vez que garante a entrega das requisições a todo o nó na rede. Em certos casos pode causar um efeito negativo na performance, no entanto na rede *blockchain* é relevante pois garante a transparência, notado que todos os nós recebem informações das transações realizadas. De acordo com Xu et al. (2016) «Uma vez criada, uma transação é assinada com a assinatura do iniciante da transação e também recebe um identificador único, como o bloco a qual pertence, que identifica a autorização para o gasto do valor monetário (no caso de transações envolvendo criptomoedas) A transação é então enviada para um nó da rede *blockchain* que sabe como validar a transação. Este, por sua vez, propaga a transação a um conjunto de nós conectados que também irão validar a transação e enviá-las a seus pares de nós até que se alcance todos os nós na rede» [Pau].

### Organizações

As organizações são conhecidas também por membros e são convidadas a entrar na rede *blockchain* através de um provedor de rede. Uma organização é associada a uma rede quando o seu provedor de serviços de associação (MSP) é adicionado à rede. É através do MSP que outros membros da rede podem verificar se as assinaturas foram geradas por uma entidade válida, emitida pela organização. O MSP possui direitos de acesso específicos de identidades que são geridos por políticas acordadas aquando da associação da organização à rede. O ponto final da transação de uma organização é um *peer* e um conjunto de organizações forma um consórcio, no entanto todas as organizações de uma rede são membros, mas nem todas farão parte de um consórcio.

### *Peer*

Uma entidade de rede que mantém um livro-razão e executa os códigos de cadeia para operações de leitura/gravação no livro-razão. Os *peers* fazem parte e são mantidos como membros.

### *Peer-to-peer*

Os nós funcionam como clientes e servidores para os restantes nós da rede. Os nós compartilham responsabilidades de servir aos outros nós. Desta forma não existe um ponto único de controlo. Essa troca de informação não ocorre sem que haja uma concordância de um conjunto de regras previamente definidas.

***Orderer***

Coletivo definido que ordena as transacções em bloco e distribui os blocos aos pares conectados para validação e confirmação. Todos os canais de rede possuem um serviço de pedidos independente dos processos pares e das transacções de pedidos por ordem de chegada. *Orderer* é fundamental no suporte às implementações plugáveis, bem como das variedades Kafka e Raft. É uma ligação comum para a rede geral onde contém o material criptográfico de identidade vinculado a cada membro.

### 2.2.2 Tipos de *Blockchain*

Existem actualmente três tipos de *blockchain* sendo eles os seguintes:

- *Blockchain* público

  Foi o primeiro tipo de *blockchain* e, como o próprio nome indica, refere-se a *blockchains* que são públicos e acessíveis. Os *blockchains* públicos permitem que qualquer pessoa faça parte deles, quer seja como utilizador, minerador ou administrador de um nó. O funcionamento da rede é totalmente aberto e transparente, ou seja, todos os dados, desde o início estão disponíveis sem qualquer restrição. Não existem entidades centralizadas, as redes públicas são completamente descentralizadas e não existe uma autoridade central que regule o funcionamento. Exemplos de *blockchain* público são: Bitcoin, Ethereum e Dash.

- *Blockchain* Privado

  A evolução da tecnologia *blockchain* fez com que, muitas empresas se interessassem por ela, dando origem a soluções de *blockchain* privadas. Este tipo de soluções tem os mesmo elementos de um *blockchain* público, mas ao contrario dos *blockchains* públicos, os *blockchains* privados dependem de um sistema/unidade central que controla todas as acções associadas ao *blockchain*. Esta unidade central controla o acesso aos utilizadores e as suas funções e permissões dentro do *blockchain*. É exemplo de *blockchain* privado o Hyperledger.

- *Blockchain* Híbrido

  O *blockchain* híbrido é uma união entre o *blockchain* público e o privado, numa tentativa de aproveitar o melhor de ambos. No *blockchain* híbrido a participação na rede é privada, o que faz que o acesso aos recursos da rede seja controlado por uma ou mais entidades. O livo razão é de acesso público, ou seja, qualquer um pode ver o que acontece nesse *blockchain*. Há exemplos de *blockchains* híbridos no sector da saúde, onde se está a utilizar para armazenar os dados das linhas de produção de medicamentos.

Sendo o *blockchain* uma tecnologia complexa ainda existem muitas partes que ainda não estão definitivamente aceites pela comunidade de utilizadores. Persiste-se em encontrar ideias para aplicar a tecnologia BC na segurança de dados.

Uma das vertentes em que se procura aplicar a tecnologia nesta dissertação é a utilização do *blockchain* de forma a ter segurança nos dados associados a sistemas baseados em IoT, para a utilização em áreas criticas, como é por exemplo, a área da qualidade da água ou a segurança alimentar.

### 2.2.3 Aplicação da tecnologia *blockchain* em IoT

Devido ao grande aumento do número de dispositivos IoT de dia para dia o *blockchain* pode ser a chave para a descentralização e democratização da comunicação com dispositivos IoT no futuro.

Um estudo feito pela IBM, revela que a democracia de dispositivos de baixo custo, surge para permitir novas economias digitais e criar um novo valor, de forma a oferecer melhores produtos e melhores experiências de utilizador. O estudo revela ainda que podem ser reduzidos os custos associados à manutenção e instalação de grandes centros de dados centralizados com a utilização da computação *peer-to-peer* de forma a processar centenas de bilhões de transacções IoT.

Existem varias áreas onde pode ser aplicada a tecnologia *blockchain*. Começou por ser aplicado no Bitcoin, onde foi designada de blockchain 1.0. Passando posteriormente para a *blockchain* 2.0, onde se começou a utilizar contratos inteligentes e de seguida passando para a *blockchain* 3.0, onde foi utilizada em aplicações de justiça, eficiência e coordenação. Uma das plataformas mais populares baseada no *blockchain* que usa contratos inteligentes é o Ethereum. Sendo que o Ethereum pode ainda usar outras aplicações distribuídas e interagir com mais do que um *blockchain*.

Não é só em criptomoedas e contratos inteligentes que o *blockchain* se aplica. Existem outras áreas onde o mesmo se pode aplicar, como da área IoT, Figura 2.2 [FF18].

O *Blockchain* pode ainda ser usado em aplicações ligadas à agricultura na área do IoT. Foi apresentado um sistema baseado na utilização *Radio Frequency Identification* (RFID) e na tecnologia *Blockchain*, de forma a melhorar a segurança e qualidade alimentar chinesa e ainda reduzir consideravelmente as perdas no processo logístico [Fen16].

Uma das áreas que pode beneficiar com o *blockchain*, o IoT e Internet da Energia (IoE) é o sector ligado à energia. Existe um estudo onde é proposto um sistema capaz de fazer o pagamento sem que exista intervenção humana, ou seja, o sistema contém um cabo inteligente que se conecta a uma tomada inteligente que por sua vez é capaz de pagar pela electricidade consumida.

De forma a reduzir as taxas de cada transacção de criptomoedas (Bitcoin), é apresentado um protocolo de micro pagamentos que faz agregação de vários pequenos pagamentos numa única transacção [LBA17]. Ainda no sector da Energia é apresentada uma plata-

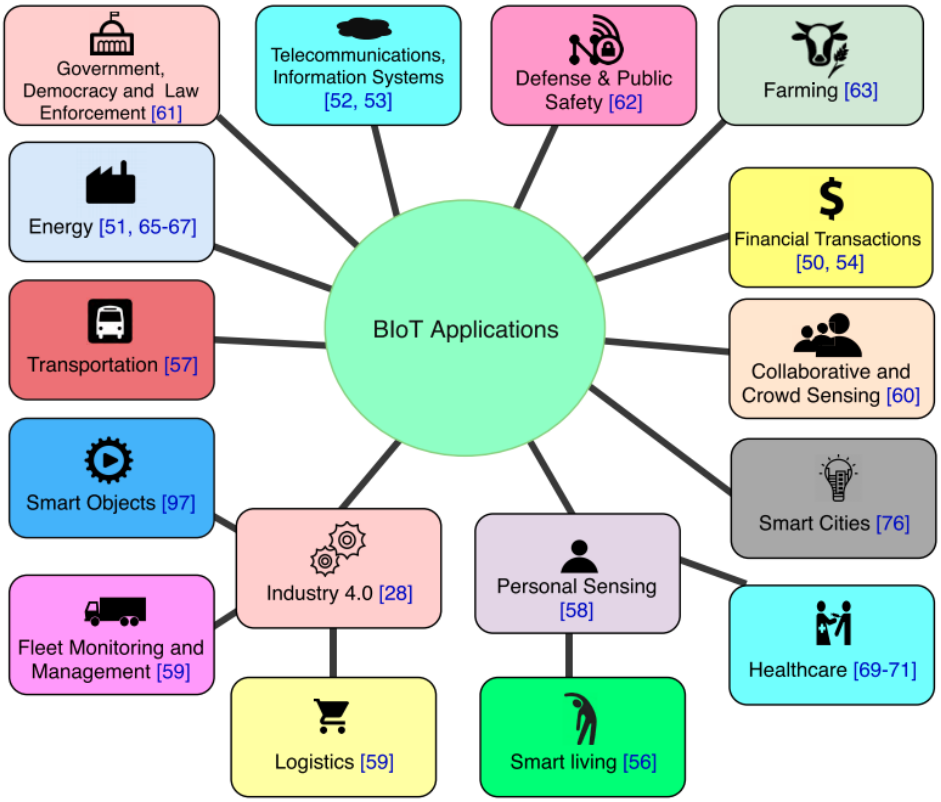


**Figura 2.2:** Aplicações BIot. [FF18]

forma de energia renovável onde os clientes compram e vende energia local e optimizam a rede ao nível da comunidade, Este sistema é apresentado pela LO3Energy e utiliza a tecnologia *blockchain* de forma a garantir a segurança dos dados pessoais e do sistema. (LO3Energy)

A saúde também é um dos exemplos onde pode ser aplicada a tecnologia *blockchain* na área do IoT. Há uma aplicação que se baseia em dados de sensores IoT e um *blockchain* para verificar a integridade dos dados para dar acesso público aos registos de temperatura numa cadeia de abastecimento de produtos farmacêuticos. Os dados da temperatura são críticos para o transporte dos produtos, de modo a garantir a qualidade e as condições ambientais quando é feito o transporte. Deve haver uma temperatura e humidade relativa ideal para os produtos terem a qualidade necessária. Cada pacote enviado tem um sensor que envia uma colecção de dados para o *blockchain*, onde contratos inteligentes validam se os dados enviados estão dentro dos parâmetros pretendidos. [Boc+17]

A tabela 2.1 mostra como a tecnologia *blockchain* foi evoluindo ao logo do tempo. Foram desenvolvidas novas aplicações em diversas áreas.

| Titulo | Ano | Blockchain - Framework | Observações - Sector |
|---|---|---|---|
| Criação Blockchain - Bitcoin | 01-2008 | Não se aplica | Não se aplica |
| Blockchain Version 1.0 (Bitcoin - Moeda) | 2009 | Não se aplica | Não se aplica |
| Blockchain Version 2.0 (Ethereum - Contratos Inteligentes) | 2013 | Não se aplica | Não se aplica |
| Blockchain Version 3.0 (utilizado em aplicações de justiça, eficiência e coordenação) | 2017 | Não se aplica | Não se aplica |
| Aplicação IoT (RFID e Blockchain) para melhorar a segurança e qualidade alimentar chinesa e reduzir perdas no processo logístico | 06-2016 | RFID (Radio-Frequency Identification) | Alimentar |
| Aplicação Blockchain e IoT ligada ao sector da saúde | 05-2017 | Ethereum | Saúde |
| Aplicação Blockchain e IoT para transformar pequenos pagamentos num único pagamento | 06-2017 | The smart cable and socket system | Energia |
| Comparação entre utilização de tecnologia com e sem blockchain em termos de segurança | 10-2018 | Ethereum | Implementação com blockchain é mais segura. |
| Comparação entre várias frameworks blockchain | 10-2019 | Hyperledger Fabric | Hyperledger Fabric a mais adequada para o IoT |

**Tabela 2.1:** Cronologia da tecnologia *blockchain*.

## 2.3 Conclusão

As tecnologias *blockcahin* não se aplicam apenas às criptomoedas. São aplicadas a várias áreas para garantir a segurança dos dados. Uma das áreas onde se aplica o *Blockchain* é o IoT. Notáveis avanços foram feitos nessa área, mostrando evoluções marcantes na segurança, privacidade e escalabilidade dos Sistemas IoT.

Ao utilizar a tecnologia *blockchain* em conjunto com IoT torna-se possível processar transações e coordenar comunicações com segurança num número muito grande de dispositivos conectados. Sendo um sistema descentralizado elimina pontos únicos de falha. Assim cria um sistema menos vulnerável para os dispositivos.

Os algortimos criptográficos usados pelo *blockchain* tornam os dados seguros e privados.

A tecnologia *blockchain* sendo um sistema que não pode ser manipulado por pessoas mal-intencionadas devido ao fato deste não existir apenas num único local e ter os seus blocos encadeados. Veio permitir segurança aos sistemas IoT. Os recursos descentralizados, autónomos e confiáveis são componentes ideais que a tecnologia *blockchain* disponibiliza para os sistemas IoT assegurarem todas as suas transações de dados.

A tecnologia *blockcahin* ao permitir manter um registo imutável, permite o funcionamento autonomo de dispositivos IoT sem a necessidade de uma autoridade central. A solução de blocos encadeados permite implementar uma série de cenários de IoT que eram notavelmente difíceis ou mesmo impossíveis de implementar.

A tecnologia *blockcahin* foi evoluindo de acordo com as aplicações.

Pode-se concluir que a tecnologia *blockchain* veio revolucionar o mundo das tecnologias e dar segurança e fiabilidade aos dados.

# Capítulo 3

# Arquitetura do Sistema

## 3.1 Introdução

Apresenta-se a arquitetura de uma aplicação para dados adquiridos por técnicas IoT em qualidade da água que usa a tecnologia *blockchain* para dados críticos na IoT com Hyperledger Fabric.

A garantia da segurança de dados críticos em sistemas IoT, pode ser assegurada pelas tecnologias *blockchain*. Estas disponibilizam informações imediatas, compartilhadas e completamente transparentes, armazenadas no livro-razão imutável. Este apenas pode ser acedido por membros da rede autorizada. Os membros compartilham uma visualização única dos fatos, o que permite o acesso a todos os detalhes de uma transação. Traduzindo-se numa maior confiança, eficiência e novas oportunidades.

Os elementos principais do *blockchain* são o livro-razão distribuído, registos imutáveis, e contratos inteligentes. No livro-razão distribuído todos os participantes da rede têm acesso ao mesmo e ao registo imutável de transacções. Com o livro-razão compartilhado, as transacções são registadas apenas uma vez, eliminando actividades duplicadas. Nos registos imutáveis nenhuma transacção poderá ser alterada ou corrompida depois de registada no livro-razão compartilhado. Se algum registo incluir um erro, terá que ser incluída uma nova transacção para reverter o mesmo e ambas ficarão visíveis. Os contratos inteligentes são um conjunto de regras armazenadas na *blockchain*. O mesmo é executado automaticamente de cada vez que se faz uma transacção.

As vantagens do *Blockchain* são:

**Maior confiança:** Com a *blockchain* é garantido que um membro apenas recebe dados corretos e pontuais, e que os seus registos confidenciais são compartilhados só com membros da rede a quem concedeu acesso específico.

**Maior segurança:** É exigido um consenso acerca da precisão dos dados e todas as transacções validadas são imutáveis e registadas permanentemente. Ninguém pode excluir uma transacção, nem mesmo um administrador.

**Mais eficiência:** Com a finalidade de acelerar as transacções, um contrato inteligente é armazenado e executado automaticamente.

No trabalho subjacente à dissertação é para guardar os dados da qualidade da água através da tecnologia *blockchain*, foi adoptada a plataforma ***Hyperledger Fabric***. Esta tem uma arquitectura modular e versátil. Adapta-se a uma vasta gama de casos de uso. Nomeadamente permite a segurança de dados críticos. O ***Hyperledger Fabric*** diferencia-se de outros sistemas de *blockchain* devido a de ser privado e com permissões.

Este capítulo fornece informação sobre a arquitectura do sistema. As várias fases para a implementação de uma rede *blockchain* utilizando o ***Hyperledger Fabric*** são apresentadas. A secção 3.2 mostra como apareceu o ***Hyperledger Fabric*** e as vantagens do mesmo. A secção 3.3 apresenta como os dados da água são recolhidos através dos dispositivos IoT. Na secção 3.4 explica-se como os dados vão ser inseridos no *blockchain*. A secção 3.5 mostra como podem ser consultados os dados no *blockchain*. A secção 3.6 apresenta as várias fases da construção dum sistema usando o ***Hyperledger Fabric***. A última secção 3.7 é a conclusão do capítulo.

## 3.2 Hyperledger Fabric para blockchain

Em 2015 a Linux Foundation fundou o projecto Hyperleger de forma a promover tecnologias *blockchains* de vários sectores. O Hyperledger Fabric é um dos projectos blockchain do Hyperledger. É uma plataforma que usa um livro razão distribuído que regista todas as transacções na rede. O Hyperledger Fabric diferencia-se de outros sistemas de *blockchain* devido ao facto de ser privado e com permissões [TNV18]. Enquanto que outros *blockchain* têm um sistema aberto e sem permissão. Esses *blockchain* permitem que entidades desconhecidas participem na rede, exigindo protocolos de prova de trabalho para a validação de transações e proteger a própria rede. No Hyperledger Fabric os participantes são inscritos no sistema por um provedor de serviços (MSP - Membership Service Provider) [she18].

No *blockchain* o livro razão é frequentemente descrito como descentralizado porque é replicado pelos vários participantes da rede, sendo que cada participante colabora na manutenção desse mesmo livro razão: uma grande vantagem para garantir a segurança dos dados.

Uma outra vantagem que o *blockchain* tem é que a informação registada num *blockchain* é adicionada usando técnicas criptográficas de forma a que quando uma transacção é adicionada esta não pode ser modificada. Esta propriedade de «imutabilidade», faz com que seja simples determinar a ordem com que cada transacção entrou no livro razão e os participantes na rede podem ter a certeza de que as informações não foram alteradas após terem sido feitas as transacções. É por isto que o *blockchain* pode ser designado como um sistema de prova.

A rede *blockchain* usa contratos inteligentes para que a informação fique guardada

no livro de razão de uma forma consistente. Os mesmos não têm apenas a função de encapsular a informação e a manter simples na rede, eles também podem ser escritos para permitir que os participantes executem algumas tarefas de forma automática.

## 3.3 Recolha dos Dados da Qualidade da Água

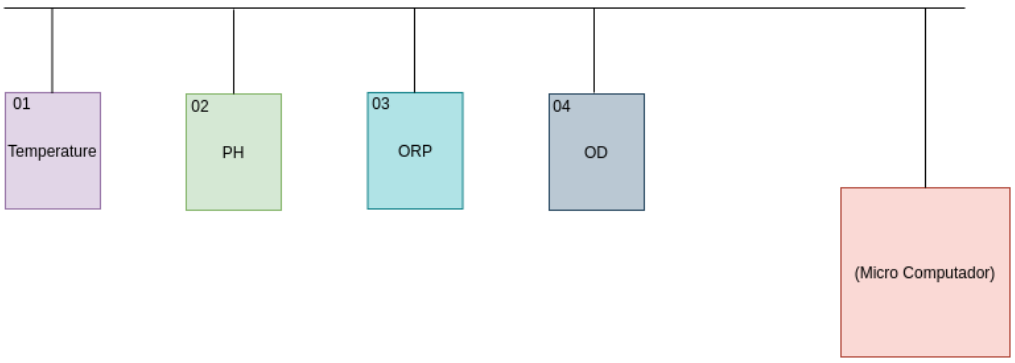


**Figura 3.1:** Ligação entre os sensores físico químicos e o Micro Computador.

Na Figura 3.1 apresenta-se a comunicação entre os sensores digitais e o Micro Computador (*Device* IoT) através de um barramento série. Os sensores e o Micro computador comunicam por um protocolo de ligação I2C (*Inter-Integrated Circuit*). Este é um protocolo de comunicação com interface de barramento incorporado em dispositivos electrónicos digitais para a comunicação série. Foi projetado pela *Philips* no ano de 1982. É um protocolo muito usado em comunicação em curtas distâncias. É conhecido como *Two Wired Interface* (TWI).

O protocolo I2C usa duas linhas de drenagem aberta bidireccionais para a comunicação. As linhas são SDA e SCL. *Serial Data* (SDA) é o pino onde ocorre a transferência dos dados. O Relógio serial (SCL) carrega o sinal de relógio. ([GG])


**Figura 3.2:** Dados da água adquiridos por sistemas IoT

Na Figura 3.2 é representado o processo de aquisição dos dados da água antes de serem guardados no *blockchain*. Apresentam-se duas das medidas que são obtidas na albufeira: a temperatura da água e o pH. Os sensores estão ligados ao dispositivo IoT. Esse dispositivo

vai enviar os dados através uma gateway IoT. Esta tem a finalidade de transformar os dados recolhidos pelo sensor num formato adequado (JSON[1]. Eventualmente pode ser YAML[2]). Os dados num destes formatos são enviados usando um protocolo de mensagens (MQTT[3]).

## 3.4 Inserção dos Dados no *Blockchain*

Na Figura 3.3 é apresentado a forma como os dados são inseridos no *blockchain*. O cliente fabric vai obter os dados através do protocolo MQTT. Seguidamente envia os dados para o *peer*. Este confirma a assinatura de acordo com o MSP e verifica se o cliente tem autorização para enviar a informação. O *peer* valida e verifica a informação recebida. O *peer* vai invocar o contrato inteligente com a proposta dos dados que são para inserir. O contrato inteligente vai propor ao livro da razão para simular o envio dos dados. Após a validação, o livro da razão envia ao contrato inteligente uma informação sobre a proposta. O contrato envia essa mesma informação ao *peer* que é devolvida ao cliente fabric. Essa informação devolve um conjunto de chave/valor de forma a efectuar alterações ao *blockchain*.

O cliente fabric, já com a informação validada, invoca o *orderer*. O *orderer* já não precisa validar a informação uma vez que a mesma já foi validada anteriormente. A informação é composta pelo **ID do canal**, pela **chave/valor** e a **assinatura digital dos peers** que processaram a transacção.

O *orderer* vai ordenar a transacção, cria os blocos da mesma e envia para todos os *peers* que pertencem ao canal. Neste caso, envia para o *peer* da Organização 1 e para o *peer* da Organização 2.

Os *peers* de cada organização, após receberem as transacções do *orderer*, voltam a validar se o *endorsement* foi feito de forma correcta segundo as políticas do canal. Seguidamente actualizam a respectiva cópia do livro razão.

## 3.5 Consulta dos dados no Blockchain

Na Figura 3.4 é apresentado como os dados são consultados no *blockchain*. O Cliente fabric liga-se ao canal enviando uma proposta de consulta ao *peer*. O *peer* confirma a assinatura do cliente fabric através do MSP e verifica se o mesmo tem autorização para fazer a consulta. O *peer* invoca o contrato inteligente com os dados da consulta. Seguidamente o contrato inteligente pede a sua cópia do livro razão. Os mesmo são enviados no sentido inverso, passando pelo contrato inteligente e *peer* até chegarem ao cliente fabric.

[1]JSON: https://www.json.org/json-en.html
[2]YAML: https://pt.wikipedia.org/wiki/YAML
[3]MQTT: https://mqtt.org

## 3.6 Sistema

A arquitetura do sistema é apresentada na Figura 3.5. Este é composto por um servidor TLS CA e por três organizações. Cada uma tem a sua autoridade de certificação (CA). Uma das três organizações contém o *orderer*. As outras duas organizações contêm os respectivos *peers*. O TLS CA é usado para emitir os certificados TLS. Estes são necessários para proteger a comunicação entre os vários processos do sistema.

### 3.6.1 Identidades

Numa rede *blockchain* existem vários atores como por exemplo, *peers*, *orderers*, aplicações de cliente, administradores. Cada um deles consomem serviços da rede. Vão possuir uma identidade encapsulada dentro dum certificado digital X.509. Estas identidades são importantes porque vão determinar as permissões exatas sobre os recursos que os actores têm dentro da rede *blockchain*.

Uma identidade digital tem alguns atributos principais que o *Hyperledger Fabric* utiliza para determinar as permissões. Sendo eles «usersIds» e «groupIds». Os atributos da identidade podem incluir mais propriedades, como por exemplo: a organização do ator, unidade organizacional, função ou identidade especifica do ator. São estas propriedades que por fim vão determinar as permissões do mesmo.

Uma identidade só pode ser verificada como válida se a mesma vier de uma autoridade confiável. No Fabric uma identidade confiável é designada de MSP (membership service provider). O mesmo é um componente que define as regras nas quais as identidades são válidas para a sua organização. De referir que o Fabric usa certificados X.509 como identidades. O mesmo adopta um modelo hierárquico de infraestruturas de chave pública. (PKI - Public Key Infrastructure) [Hypa].

Na figura 3.6 é apresentado um exemplo das identidades e dos MSP. Trata-se de uma pessoa que tem disponível na sua carteira vários cartões válidos. No entanto a caixa (organização) apenas permite que a pessoa pague as suas compras com uns determinados tipos de cartões. Não basta ter um cartão válido para pagar as compras se a caixa não permitir acesso a esse cartão. As autoridades de Certificação PKI podem funcionar como um provedor de cartões. Criam diferentes tipos de cartões válidos. No caso do MSP é a lista de cartões aceites pela caixa. Lista quais as identidades que são membros confiáveis e aceites pela rede de pagamento.

### 3.6.2 Infraestruturas de chave pública (PKI - public key infrastructure)

As infraestruturas de chave pública são o conjunto de *hardware*, *software*, politicas, processos e procedimentos necessários para criar, gerir, distribuir, usar, armazenar e revogar certificados digitais e chaves públicas. As PKI's são a base que permite o uso de tecnolo-

gias, como assinaturas digitais e criptografia, em grandes populações de utilizadores. Elas fornecem elementos essenciais para um ambiente de negócios seguro e confiável para a IoT.

As PKIs ajudam a estabelecer a identidade de pessoas, dispositivos e serviços. Permitindo acesso controlado a sistema, protecção de dados e responsabilidade nas transacções.

A forma mais comum de criptografia usada hoje em dia envolve uma chave pública e uma chave privada. A chave pública que pode ser usada por várias pessoas para cifrar uma mensagem e no caso da chave privada que deve ser usada apenas por uma única pessoa para decifrar a mensagem recebida. De referir que estas chaves podem ser usadas por pessoas, dispositivos e aplicações.

Ainda que uma rede *blockchain* seja mais do que uma rede de comunicação, a mesma depende das PKI. Com ela garante a comunicação segura entre os vários participantes da rede de forma a que as mensagens enviadas para o *blockchain* são devidamente autenticadas [Hypb].

Existem quatro elementos chaves para as PKI: Certificados Digitais, Chaves públicas e privadas, Autoridades de Certificação e lista de revogação de certificados.

### Certificados Digitais

O certificado digital é um documento que contém um conjunto de atributos relativos ao titular do certificado. O tipo de certificado mais usado é compatível com o padrão X.509. O mesmo permite a codificação dos detalhes de identificação na sua estrutura [Hypc].

Na Figura 3.7 podemos ver um exemplo dum certificado digital. No mesmo podemos ver os vários detalhes que identificam aquela pessoa. O certificado pode ser semelhante ao Cartão de Cidadão da pessoa em causa. Fornece informações que provam factos importantes sobre a mesma. Uma outra informação importante que os certificados disponibilizam é a chave pública da pessoa.

Estes certificados são gerados a partir duma técnica matemática designada por criptografia. Esta técnica faz com que alguma alteração feita ao certificado invalide o mesmo. A técnica da criptografia permite que o sujeito do certificado apresente aquele certificado a outras identidades para provar a sua identidade. Contudo as identidades só vão aprovar aquele certificado válido caso confiem na autoridade de certificação que emitiu o mesmo.

### Chaves públicas e privadas

É garantida a autenticação e a integridade das mensagens com criptografia de chave pública. O que torna as comunicações seguras entre as partes do sistema. A autenticação requer que as partes que trocam mensagens tenham a certeza da identidade que criou uma determinada mensagem. Já a integridade da mensagem quer dizer que a mensagem não foi alterada durante a sua transmissão.

Os mecanismos de autenticação tradicionais depende de assinatura digital. Os mesmos exigem que cada parte mantenha um par de chaves (Chave pública e Chave privada)

conectadas através de criptografia. A Chave pública que é disponibilizada e serve para autenticar as mensagens. A chave privada usada para produzir assinaturas digitais nas mensagens.

#### Autoridades de Certificação (CA)

Uma autoridade de certificação emite certificados para diferentes atores. Os certificados são assinados digitalmente pela CA e os mesmos ficam vinculados a essa CA. Como consequência disso, alguém que confie na CA, confia no certificado emitido pela mesma. Esse certificado vai conter a chave pública da CA.

Numa configuração de *blockchain*, cada ator que deseja interagir com a rede precisa de uma identidade. Logo pode-se afirmar que um ou mais CA podem ser usados para definir os membros de uma organização. Sendo o CA que fornece uma identidade aos atores para que os mesmos tenham uma identidade digital válida e que é verificada.

Podem existir CA raiz (CA Root) e CA intermediárias. Devido ao fato das CA ROOT terem de emitir milhões de certificados para os utilizadores de Internet, faz sentido haver as CA's intermediárias. As mesmas têm os seus certificados emitidos por uma CA Root ou por outra CA intermediária, fazendo com que exista uma cadeia de confiança. A possibilidade de rastrear um certificado até a raiz faz com que as funções dos CA seja escalável e garante uma maior segurança [Hypd].

#### Lista de revogação de certificados (Certificate Revocation List (CRL))

O CRL é uma lista de certificados que uma CA sabe que foram revogados [Hype].

Na Figura 3.8 é apresentado um exemplo de uma CRL para verificar se um certificado ainda é válido. No caso das validações em primeiro lugar deve se verificar nas CRL da CA emissora se o certificado é ou não válido. Um certificado revogado é diferente de um certificado expirado.

### 3.6.3 Membership Service Provider (MSP)

O MSP é um conjunto de pastas que são adicionadas à configuração da rede. É usado para definir uma organização internamente e externamente. Internamente porque as organizações decidem quem são os seus administradores. Externamente porque permite que outras organizações validem se essas identidades têm autoridade para o que estão a tentar fazer.

O MSP contém a lista de identidades permitidas. Identidades essas que foram geradas pelos CA. O MSP identifica quais as CA Root e as CA intermédias que são aceites para definir um domínio confiável. As identidades dos utilizadores são listadas pelo MSP. Identifica as CA autorizadas para os utilizadores.

Uma identidade é transformada numa função pelo MSP. Este identifica os privilégios específicos que um utilizador tem num nó ou num canal. É atribuída uma função (Admin, peer, cliente, orderer ou membro) ao utilizador da rede quando é registado com o Fabric CA [Hypf].

### 3.6.4 TLS - Transport Layer Security

O TLS é um protocolo de segurança. Este é projetado para facilitar a privacidade e a segurança de dados em comunicações na Internet. O principal caso de uso do TLS é cifrar a comunicação entre aplicações web se servidores. **O mesmo pode ainda cifrar outro tipo de comunicações.** Como por exemplo, transferências de ficheiros, conexões de rede privada (VPN), email e mensagens de voz sobre IP (VoiP).

A primeira versão do protocolo TLS foi publicada no ano de 1999. A versão mais recente é a TLS 1.3 e foi publicada em 2018. Foi proposta pela organização de padrões internacionais *Internet Engineering Task Force (IETF)*.

Existe três componentes principais que o protocolo assegura, sendo elas **Criptografia**, **Autenticação**, **Integridade**. A Criptografia faz com que os dados sejam cifrados quando é feita a comunicação. A Autenticação garante que uma entidade é quem diz ser. A Integridade garante que o conteúdo dos dados não foram falsificados ou alterados.

Para que uma aplicação ou site possa usar o TLS, tem que haver um certificado TLS instalado no servidor de origem. Uma entidade certificadora é responsável por emitir certificados TLS. O mesmo pode ser emitido para uma pessoa ou empresa que possui um domínio. O certificado tem informações importantes sobre quem é o proprietário do domínio, a chave publica do servidor. Ambas as informações são importantes para validar a identidade do servidor.

Uma conexão TLS é iniciada usando uma sequência conhecida como *handshake TLS*. Quando um utilizador navega para um site que usa TLS, o *handshake TLS* começa a trocar informação entre o dispositivo do utilizador, (também conhecido como dispositivo cliente) e o servidor da web. Durante o *handshake TLS* o cliente e o servidor trocam mensagens para se reconhecerem um ao outro. É especificada a versão do TLS usada. São estabelecidos os algoritmos de criptografia usados e são geradas as chaves de sessão para cifrar as mensagens trocadas entre eles quando o *handshake TLS* terminar [Clo].

Um *handshake TLS* envolve várias etapas. Estas variam dependendo do tipo de algoritmo de troca de chaves e dos conjuntos de criptografia suportada por ambas as partes. O algoritmo de troca de chaves mais usado é representado na Figura 3.9.

### 3.6.5 TLS CA

O TLS CA é um sistema que emite os certificados para garantir a segurança entre as várias partes do sistema numa rede *blockchain* do Hyperledger Fabric. Este pode ser disponibilizado através de tecnologias de *containers*.

Para usar a tecnologia de *containers* o Hyperledger Fabric disponibiliza uma imagem (Hyperledger/Fabric-CA). Após iniciar o *container* TLS CA, o mesmo cria um socket seguro. Este fica à espera de pedidos de forma a emitir certificados TLS.

Antes de se fazer os pedidos para emitir os certificados é preciso obter o certificado de assinatura do TLS CA. Este é necessário para se ligar ao socket usando conexões TLS. Após se ter copiado o certificado de assinatura do TLS CA para cada máquina onde foi iniciado o *container* TLS CA, é possível executar o binário com o cliente CA. Importa referir que este binário cliente CA, faz parte das ferramentas disponibilizadas pelo Hyperledger Fabric. Seguidamente podem ser inscritos o administrador do TLS CA e outras novas entidades no TLS CA. Entidades essa que podem ser os *peers* de cada organização ou mesmo o *orderer*.

### 3.6.6 Autoridade de certificação - CA

No sistema apresentado cada organização vai conter a sua própria autoridade de certificação (CA). As CA vão criar as entidades que vão pertencer a cada organização e emitir os certificados para cada uma das entidades criadas. Os certificados para as entidades são compostos por um par de chaves, uma chave pública e uma chave privada. Estas chaves vão permitir que todos os nós e entidades da organização possam assinar e verificar cada acção que façam. Contudo também vão ser entendidos por membros de outra organizações.

O Sistema do CA é criado como o TLS CA, através do serviço *docker* iniciando uma imagem disponibilizada pelo Hyperledger Fabric. Após iniciar o *container* é necessário inscrever o administrador da CA de cada organização e registar as respetivas identidades. Após a inscrição do administrados do CA é possível ver que é gerada uma estrutura de pastas, como por exemplo na Figura 3.11, esta tem o material criptográfico emitido pelo CA.

O fabric-ca-cliente.yaml é um ficheiro criado pelo cliente CA, que tem a configuração do mesmo. O ficheiro com o nome 0-0-0-0-7053.pem é o certificado público da CA que emitiu o certificado para esta entidade, que neste caso é a entidade administrador da CA. O ficheiro 60b6a16b8b5ba3fc3113c522cce86a724d7eb92d6c3961cfd9afbd27bf11c37f-sk é a chave privada que foi gerada. De referir que o nome deste ficheiro é variável e é sempre diferente cada vez que é gerada uma nova chave. O ficheiro cert.pem é o certificado do administrador que foi assinado e emitido pela CA.

Após serem criadas todas as autoridades de certificação para cada organização, já é possível inscrever os *peers* de cada organização nas suas CA. Essa inscrição é feita pelo administrador da organização. Após se ter o material criptográfico de cada entidade e de cada *peer* já, é possível iniciar os *container* para os *peer*. Com todas as organizações criadas, incluindo a organização do *orderer*, já é possível fazer a criação do canal e instalar o contrato.

### 3.6.7 Livro Razão (*Ledger* - Base Dados)

O Livro Razão (*ledger*) no Hyperledger Fabric consiste em duas partes distintas, que por sua vez estão relacionadas. Uma das partes representa um estado global e outra a parte do blockchain, onde vão estar as transacções. O estado global contém os valores atuais de um conjunto de estados do *ledger*. Este estado facilita a forma como um programa consegue aceder directamente ao valor actual de um estado sem ter que ir calcular esse estado percorrendo todo o *log* de transacções. Os estados do *ledger* são por norma expressos como pares de chave-valor. O *blockchain* apresenta a um *log* de transacções que regista todas as alterações resultantes no estado global actual.

O Estado global é fisicamente implementado como uma base dados. Com o objectivo de armazenar os dados e para que os mesmos sejam acedidos de forma fácil e simples no acesso aos estados do *Ledger*. Sendo que os estados *Ledger* podem ser armazenados de forma simples ou composta (Figura 3.12). O Hyperledger Fabric disponibiliza duas opções de Base dados, LevelDB e CouchDB. Sendo a base dados LevelDB adequada para quando os *ledger* são armazenados de forma simples, ou seja, pares de chave valor, optou-se antes para este projecto pela da base dados CouchDB. Isto porque é uma base dados apropriada quando os *ledger* são estruturados como documentos JSON, como é o caso dos dados recolhidos por sensores IoT. É ainda uma base dados que permite ter um bom suporte de consultas e uma boa atualização dos dados.

### 3.6.8 Canal (channel) Hyperledger Fabric

No sistema apresentado o mesmo só vai conter um único canal de forma a comunicar com todas as entidades por via desse canal. De referir que o Hyperledger Fabric permite que cada organização participe em simultâneo em várias redes *blockchain* separadas por diferentes canais.

### 3.6.9 Contrato Inteligente (Smart Contract)

O essencial de um sistema *blockchain* é constituído pelo conjunto de um contrato inteligente com o livro-razão.

O livro-razão contém factos sobre o estado actual e o histórico dum conjunto de objetos, enquanto que um contrato inteligente define a lógica executável que gera novos fatos que são adicionados ao livro-razão.

Contrariamente aos contratos comuns, um contrato inteligente é válido sem dependência das autoridades porque é um código visível para todos e inalterável. Este motivo confere-lhe um carácter descentralizado, imutável e transparente. É através do contrato inteligente que são definidas as regras entre as diferentes organizações no código executável.

Um contrato inteligente pode implementar regras de governança para qualquer tipo de objeto de negócios. Estas podem ser aplicadas de forma automática quando o contrato

inteligente é executado. O mesmo consegue aceder a duas partes distintas do livro razão. Sendo elas *blockchain* e o estado global. O *blockchain* que regista o histórico imutável de todas as transacções. O estado Global que armazena uma *cache* do valor actual desses estados.

As funções que os contratos inteligentes executam no estado global são ***put***, ***get*** e ***delete***. O ***get*** representa uma consulta para saber a informações sobre o estado actual de objecto de negócios. O ***put*** cria um novo objecto de negócio ou modifica um existente no estado global ou no livro da razão. O ***delete*** remove um objecto de negócio do estado actual do livro da razão, mas não do histórico.

Importa referir que a execução de um contrato inteligente é muito mais eficiente que um processo de regras manual.

Um ou mais contratos inteligentes podem ser definidos em um único *chaincode*. A implementação de um **chaincode** numa rede disponibiliza todos os contratos inteligentes para todas as organizações da rede.

Um *chaincode* é um programa, escrito em Go, nodejs ou em Java que implementa uma interface prescrita. O mesmo é executado num *container Docker* protegido e isolado do processo de endosso. O *chaincode* inicializa e gere o estado do livro da razão recebendo as transacções de aplicações. Devido ao facto deste lidar com a lógica de negócios acordada pelos membros da rede, pode ser considerado um contrato inteligente.

Quando um *chaincode* cria um estado, o mesmo não pode ser acedido por outro *chaincode*. Por sua vez dento da mesma rede existe a possibilidade de um *chaincode* invocar outro *chaincode* de forma a aceder ao seu estado.

Para o projecto foi desenvolvido um contrato inteligente através da linguagem de programação GO.

### 3.6.10 Endosso

O endosso é associado a todos os *chaincodes*. Segundo o Hyperledger Fabric, uma politica de endosso é muito importante, devido ao facto de indicar quais são as organizações em uma rede *blockchain* que podem assinar uma transacção gerada por um contrato inteligente. Isto de forma a que a transacção seja válida. Todas as transacções independentemente de serem válidas ou inválidas são adicionadas a um livro de razão distribuído. Contudo apenas as transacções válidas atualizam o estado global.

Se uma politica de endosso especificar que mais de uma organização deve assinar uma transacção, o contrato inteligente é executado por um conjunto suficiente de organizações para que uma transacção válida seja gerada.

São as politicas de endosso que fazem a diferença entre o Hyperledger Fabric e outros *blockchains* como Ethereum ou Bitcoin. Nesses *blockchains* as transacções válidas podem ser geradas por qualquer nó da rede. No caso do Hyperledger Fabric o *blockchain* aproxime-

se mais do mundo real, devido ao facto das transacções serem validadas por organizações confiáveis numa rede.

### 3.6.11 Protocolo de disseminação de dados Gossip

O Hyperledger Fabric implementou o protocolo Gossip de forma a garantir alguns requisitos. Requisitos esses que consigam garantir a disseminação dos dados de uma forma segura, confiável e escalável para garantir a integridade e consistência dos mesmos. Este protocolo fez com que o Hyperledger Fabric consiga optimizar o desempenho, a segurança e a escalabilidade da rede *blockchain*.

Os *peers* usam o protocolo *Gossip* para transmitir dados do livro razão e do canal de forma escalável. As mensagens do protocolo são contínuas e cada *peer* num canal está constantemente a receber dados atuais e consistentes de vários *peer*. As mensagens do *Gossip* são assinadas. Isto permite que participantes bizantinos que enviem mensagens falsas sejam facilmente identificados. Assim garante-se que a distribuição de mensagens para alvos indesejados é evitada. Os *peers* afetados por atrasos na rede ou por outras causas que resultam em blocos perdidos, podem sincronizar o estado do livro razão com outros pares que possuem os blocos perdidos.

Existem três funções principais que o protocolo *Gossip* executa numa rede Fabric.

- Descoberta de *peers* e membros do canal. Identifica continuamente os *peers* disponíveis. Deteta *peers* que ficaram *offline*.

- Dissemina os dados do livro da razão entre todos os *peers* do mesmo canal. Qualquer *peer* que tenha os dados dessincronizados com o resto do canal é identificado e os blocos ausentes são sincronizados copiando os dados corretos.

- Acelera a atualização dos dados do livro da razão com uma transferência *peer-to-peer*, nos *peers* recentemente ligados.

Os *peers online* indicam que estão disponíveis transmitindo continuamente mensagens a informar que estão «vivos». Cada mensagem é composta pelo ID da infraestrutura de chave pública (PKI) e a assinatura do remetente da mensagem. Os peers vão verificando essas mensagens e assim vão mantendo os membros no canal. Se nenhum *peer* receber uma mensagem «viva» de um *peer* especifico esse *peer* está «morto» e possivelmente é removido do canal. Como as mensagens são assinadas, os *peers* mal intencionados não podem passar por outros *peers*. Pois os mesmos não têm uma chave de assinatura autorizada por uma autoridade de certificação raiz.

## 3.7 Conclusão

De uma forma geral mostrou-se a potencialidade apresentada pelo Hyperledger Fabric. Este garante uma boa performance, devido ao seu processo de consenso e ser um blockchain privado. Dada a sua versatilidade, foi possível adaptar o mesmo para criar uma aplicação para garantir a segurança dos dados da água.

A performance do Hyperledger Fabric aumenta devido ao processo de consenso ser diferente do usado pelas criptmoedas. Este processo é ideal para ser aplicado aos sistemas IoT. O livro razão ser distribuído por todos os membros da rede *blockchain* assegura a veracidade dos dados da água. De referir que o registo dos dados da água só é aceite se a maioria dos peers o validarem.

O cliente Fabric ao ser conhecido pelo MSP faz com que apenas seja este a enviar os dados da água para o blockchain. O Ciente Fabric tem credenciais válidas para o fazer. Estas credenciais são válidas graças aos certificados digitais.

O chaincode inicializa e gere o estado livro razão recebendo as transações do cliente Fabric. É este que também lida com a lógica de negócios que foi acordada pelos membros da rede.

Contudo podemos concluir que o Hyperledger Fabric é uma boa solução para garantir a segurança dos dados em qualquer sistema IoT.

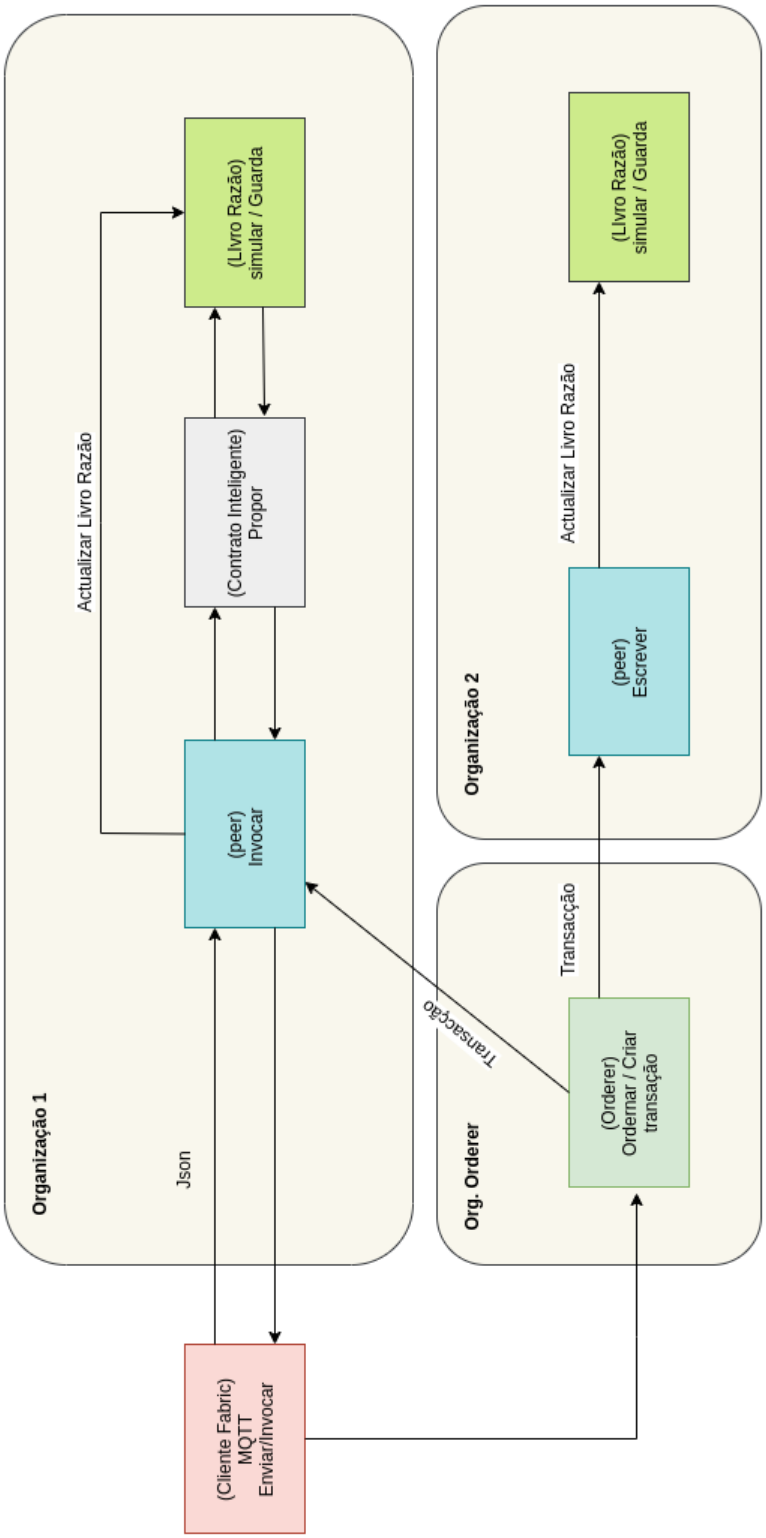


**Figura 3.3:** Sistema de armazenamento dos dados no *Blockchain*.

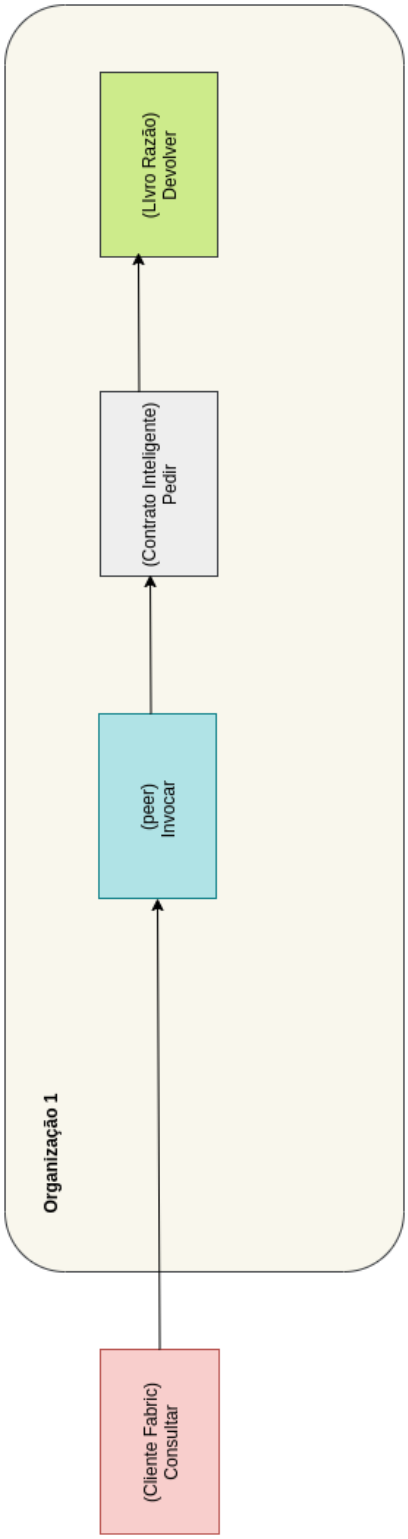


**Figura 3.4:** Consultar dados no *Blockchain*

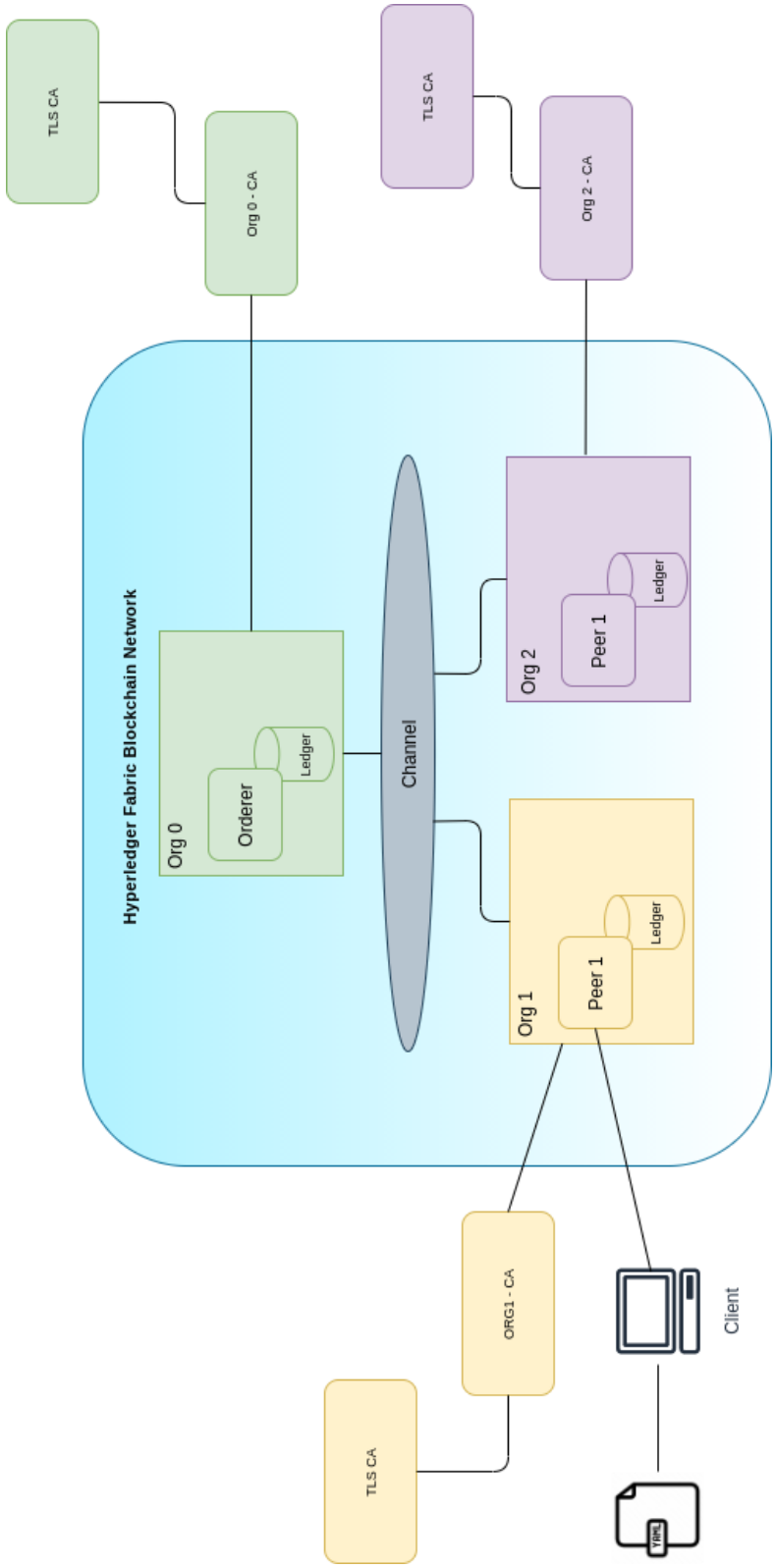


**Figura 3.5:** Exemplo da arquitetura do Sistema baseada em Hyperledger Fabric.


**Figura 3.6:** Exemplo da utilização das identidades e dos MSP (imagem retirada do site Hyperledger Fabric).

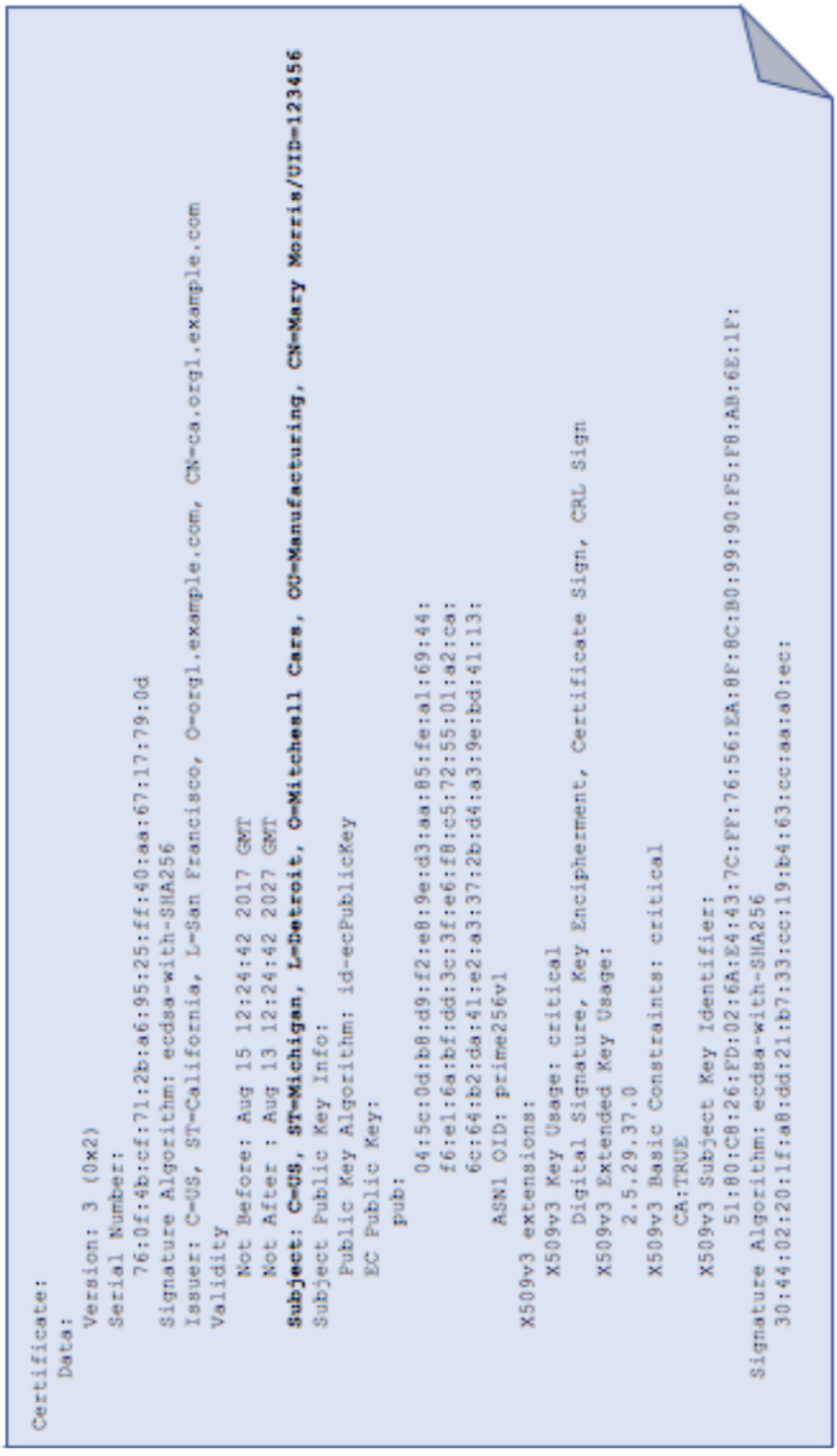


**Figura 3.7:** Exemplo de Certificado digital x.509 (imagem retirada do site Hyperledger Fabric).

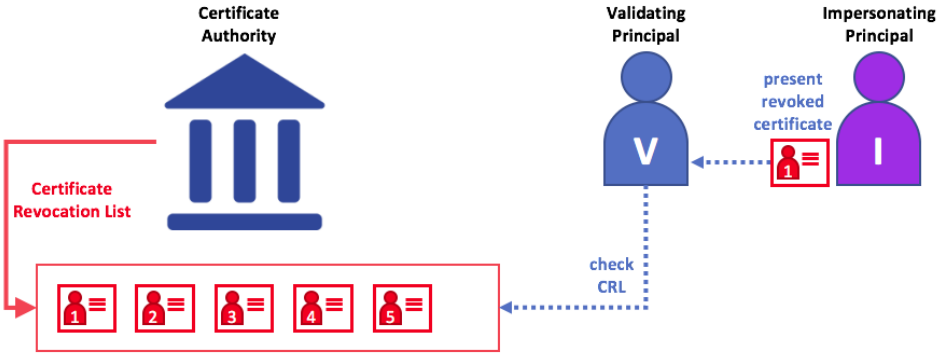


**Figura 3.8:** Exemplo de uma lista de revogação de certificado (imagem retirada do site Hyperledger Fabric).

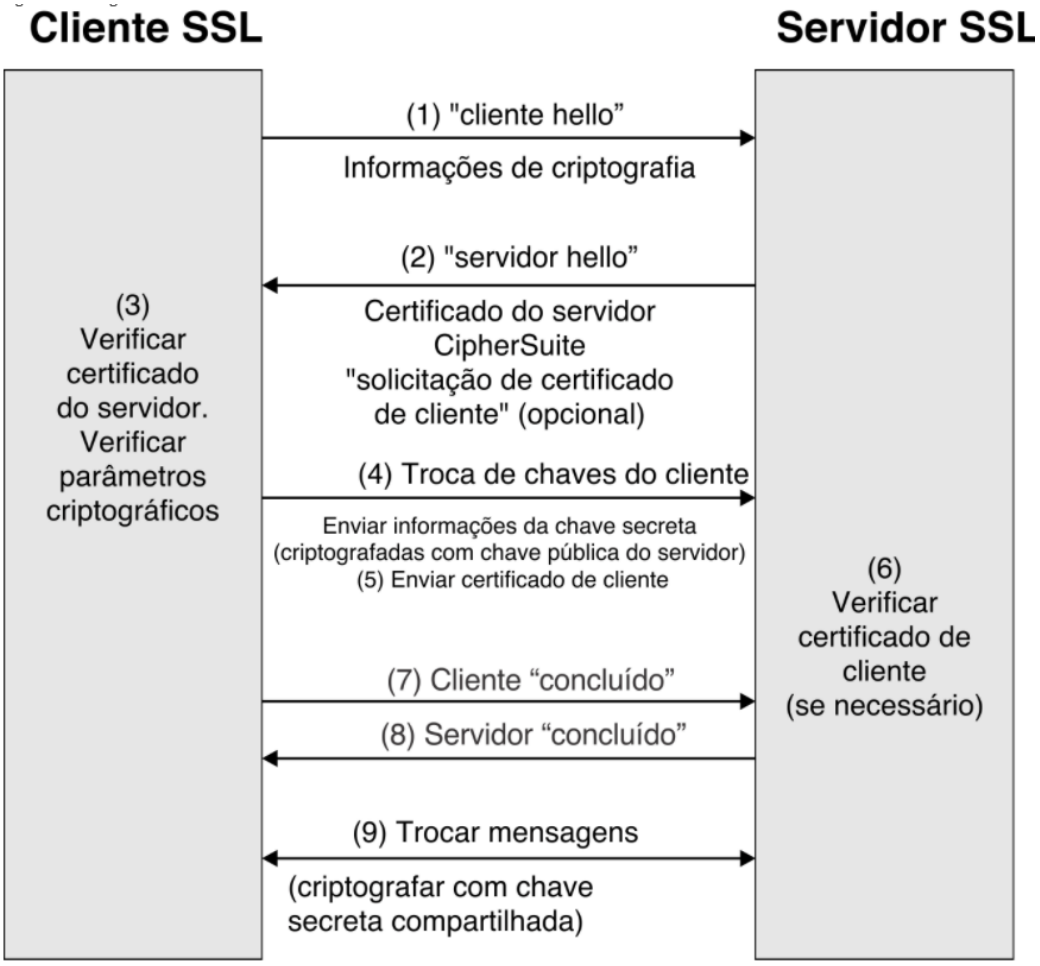


**Figura 3.9:** TLS handshake - (imagem retirada do site IBM).

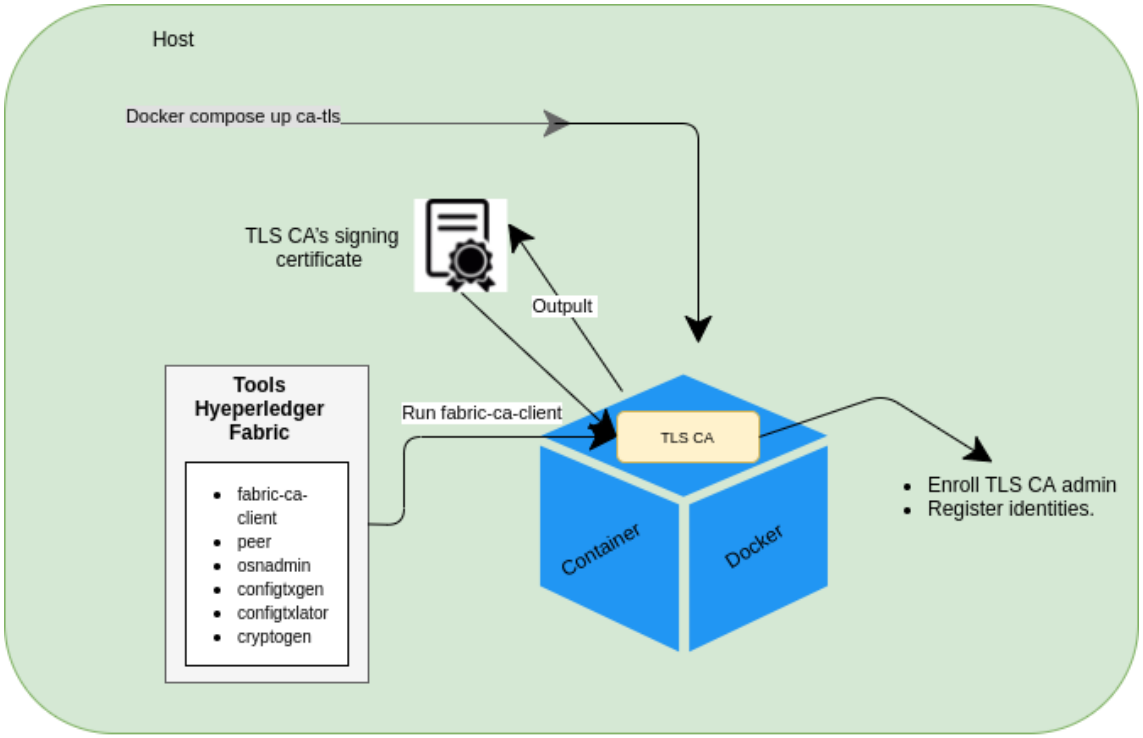


**Figura 3.10:** Criação TLS-CA Hyperledger Fabric.

```
admin
├── fabric-ca-client-config.yaml
└── msp
    ├── IssuerPublicKey
    ├── IssuerRevocationPublicKey
    ├── cacerts
    │   └── 0-0-0-0-7053.pem
    ├── keystore
    │   └── 60b6a16b8b5ba3fc3113c522cce86a724d7eb92d6c3961cfd9afbd27bf11c37f_sk
    ├── signcerts
    │   └── cert.pem
    └── user
```

**Figura 3.11:** Exemplo de material criptográfico do administrado CA.

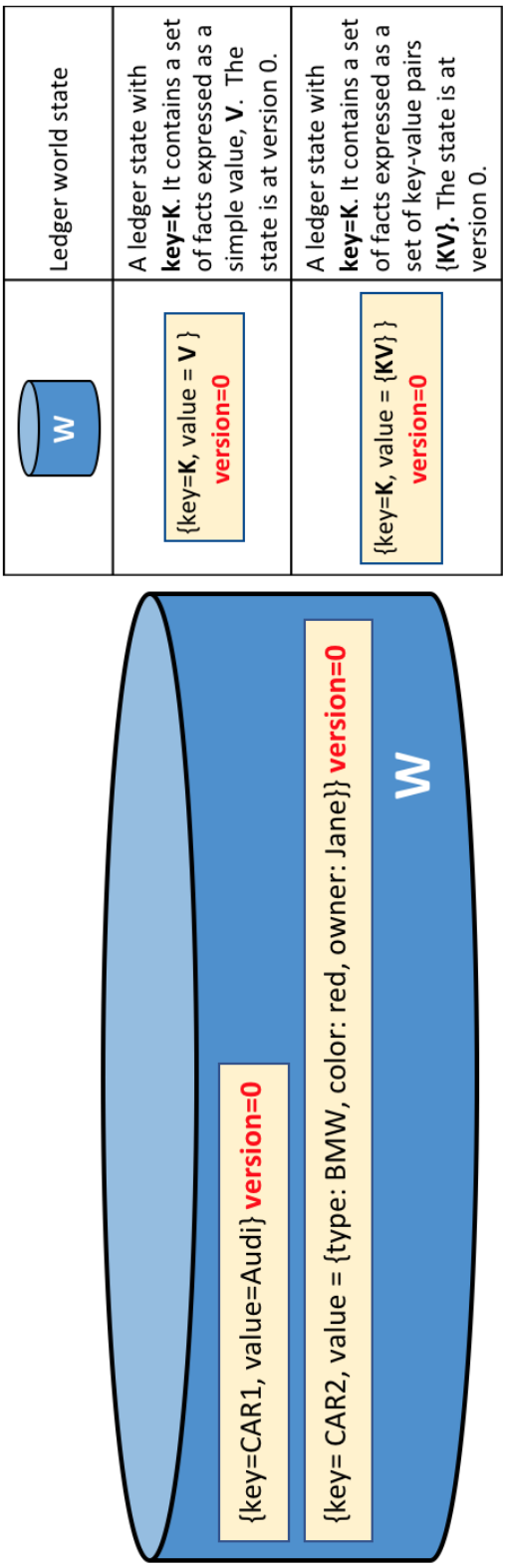


**Figura 3.12:** Estado global do Ledger (estado simples e estado composto - imagem retirada do site Hyperledger).

# Capítulo 4

# Realização experimental

## 4.1 Introdução

Neste capítulo é apresentada a implementação do sistema para guardar os dados da qualidade da água no *blockchain*, utilizando o Hyperledger Fabric. A versão do Hyperledger Fabric utilizada para a implementação foi a versão 2.3.

O capítulo encontra-se dividido em quatro secções. Na secção 4.2 é apresentado como foi criada a rede Hyperledger Fabric. Nessa secção é apresentada a forma como foram criados os CA, a criação das organizações, dos *peers* e ainda como foram criados os canais usados para fazer a comunicação entre os membros da rede.

Na secção 4.3 mostra o desenvolvimento do *Chaincode*. Esta secção mostra como é criado o *package* do *chaincode*, a forma como o mesmo é instalado nos *peers*, como é aprovado o *chaincode* e a confirmação do mesmo.

A secção 4.4 apresenta a criação do Cliente *Fabric*. Esta secção mostra como o Cliente *Fabric* foi criado, como os dispositivos IoT são registados no *blockchain* e a forma como os dados da água adquiridos pelos os dispositivos são guardados no *blockchain* e como são associados aos dispositivos.

A secção 4.5 mostra a forma como os dados são consultados no *blockchain*. É apresentado um web *service* para servir uma aplicação gráfica.

A secção 4.6 apresenta as conclusões tiradas do capítulo.

## 4.2 Criação da Rede Hyperledger Fabric

Para guardar no *blockchain* os dados adquiridos por técnicas IoT em Qualidade da Água a rede Hyperledger Fabric vai ser iniciada por três organizações. A Figura 4.1 apresenta um exemplo de como a divisão das organizações pode ser feita para Portugal. As organizações vão ser associadas a determinados distritos de Portugal. Cada organização vai conter um ou mais *peers*, que correspondem a um recurso Hídrico. Vai ter uma autoridade de certificação de forma a emitir os certificados para cada membro da organização.

### 4.2.1 Criação dos CA

```
version: '2.4'

networks:
  test:
    name: fabric_test

services:

  ca_org1:
    image: hyperledger/fabric-ca:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-org1
      - 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:
      - ../organizations/fabric-ca/org1:/etc/hyperledger/fabric-ca-
    server
    container_name: ca_org1
    networks:
      - test
```

**Listagem 4.1:** Exemplo de ficheiro YAML usado para criar o CA.

Os CA de cada organização foram criadas através de uma imagem do *Docker* que o Hyperledger Fabric disponibiliza para esse efeito (hyperledger/fabric-ca:latest). É criado um *container Docker* para cada CA. Na listagem 4.1 é apresentado um exemplo de um ficheiro YAML com as propriedades de cada CA para uma organização. Antes de iniciar os *containers* de cada CA, há um ficheiro de configuração que vai definir as propriedades do CA. Esse ficheiro vai ser passado ao *container* através da propriedade «Volumes». É nesse ficheiro de configurações que se pode definir que o CA criado é um Root Ca ou um CA intermediário.

```
ca:
  # Name of this CA
  name: Org1CA
  # Key file (is only used to import a private key into BCCSP)
  keyfile: /etc/hyperledger/fabric-ca-server/ica.key
  # Certificate file (default: ca-cert.pem)
  certfile: /etc/hyperledger/fabric-ca-server/ica.cert
  # Chain file
```

```
9    chainfile: /etc/hyperledger/fabric-ca-server/chain.cert
```

**Listagem 4.2:** Configuração dos CA (parte 1).

No Listagem 4.2 é apresentada a secção que contém as informações relacionadas ao CA. Pode ser configurado o nome do CA, o caminho para a chave, certificados, e o caminho para a cadeia de certificados.

O nome do CA é exclusivo para todos os membros da rede *blockchain*. A chave e o certificado são usados para emitir certificados de inscrição (ECerts) e transações de certificados (TCerts). A cadeia de certificados representa os certificados confiáveis para a CA, onde o primeiro certificado da cadeia é o Certificado CA Raiz.

```
1  csr:
2     cn: ca.org1.example.com
3     names:
4        - C: US
5          ST: "North Carolina"
6          L: "Durham"
7          O: org1.example.com
8          OU:
9     hosts:
10      - localhost
11      - org1.example.com
12    ca:
13       expiry: 131400h
14       pathlength: 1
```

**Listagem 4.3:** Configuração dos CA (parte 2).

Na Listagem 4.3 é apresentado a secção que contém as informações relativas a assinatura do certificado (CSR). Nesta secção pode ser configurado o tempo de expiração do certificado, a hierarquia do certificado. É o campo «pathlength»que define essa hierarquia e a mesma é descrita na secçao 4.2.1.9 do RFC 5280.

No caso do campo *pathlength* não ter nenhum valor associado define que não são impostos limites na hierarquia. No caso do valor ser igual a um, é o valor padrão para que este CA seja um CA Raiz. Logo a mesma pode emitir certificados CA intermediários. No entanto as CA intermediárias já não podem emitir outros certificados CA, podendo apenas emitir certificados de identidade final. No caso do valor do *pathlength* ser igual a zero define que a CA vai ser uma CA intermediária. Assim não pode emitir certificados CA, mas pode emitir certificados de identidade final.

Após alterado o ficheiro de configurações dos CA (Listagem 4.2 e Listagem 4.3) e criado o ficheiro YAML com os CA (Listagem 4.1) para todas as organizações, já é possível criar os containers com o seguinte comando:

```
docker-compose -f nome_ficheiro_yaml up -d
```

Na Figura 4.2 é apresentada uma listagem dos *containers* dos CA. Como se pode observar é apresentado um CA por cada organização criada.

### 4.2.2 Criação das Organizações

Antes de criar as organizações na rede Hyperledger Fabric é necessário gerar o material criptográfico para cada organização. É esse material criptográfico que vai definir as organizações na rede. Visto que Hyperledger Fabric é um *blockchain* com permissão para cada nó e para cada utilizador na rede. Os mesmos usam certificados e chaves para assinar e verificar cada acção. Cada utilizador na rede precisa pertencer a uma organização reconhecida como membro da rede.

#### Criação do Material Criptográfico para a Organização

- Inscrever Administrador CA

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

- Registar o *peer* para a organização

```
fabric-ca-client register
--caname ca-org1
--id.name peer0
--id.secret peer0pw
--id.type peer
--tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```

- Registar utilizador para a organização

```
fabric-ca-client register
--caname ca-org1
--id.name user1
--id.secret user1pw
--id.type client
```

```
--tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```

- Registar administrador da organização

```
fabric-ca-client register
--caname ca-org1
--id.name org1admin
--id.secret org1adminpw
--id.type admin
--tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```

- Gerar o MSP para o peer

```
fabric-ca-client enroll -u https://peer0:peer0pw@localhost:7054
--caname ca-org1
-M "${PWD}/organizations/peerOrganizations/
org1.example.com/peers/peer0.org1.example.com/msp"
--csr.hosts peer0.org1.example.com
--tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```

- Gerar os certificados TLS para o peer

```
fabric-ca-client enroll
-u https://peer0:peer0pw@localhost:7054
--caname ca-org1
-M "${PWD}/organizations/peerOrganizations/
org1.example.com/peers/peer0.org1.example.com/tls"
--enrollment.profile tls
--csr.hosts peer0.org1.example.com
--csr.hosts localhost
--tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```

- Gerar o MSP para o user

```
fabric-ca-client enroll
-u https://peer0:peer0pw@localhost:7054
--caname ca-org1
-M "${PWD}/organizations/peerOrganizations/
org1.example.com/peers/peer0.org1.example.com/tls"
--enrollment.profile tls
--csr.hosts peer0.org1.example.com
--csr.hosts localhost
--tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```

- Gerar o MSP para o administrador da Organização

```
fabric-ca-client enroll
-u https://org1admin:org1adminpw@localhost:7054
--caname ca-org1
-M "${PWD}/organizations/peerOrganizations/org1.example.com
/users/Admin@org1.example.com/msp"
--tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```

### 4.2.3 Criação dos *peers*

A criação do *peers* é feita após já ter sido gerado o material criptográfico de cada organização. Para se criar os *peers* para cada organização foi usado o docker-compose. Foi criado um ficheiro YAML que define as variáveis de ambiente e faz as ligações para o *container* apontar para o material criptográfico.

Através do seguinte comando pode ser criado os *peer* para as organizações.

```
docker-compose -f nome_ficheiro_yaml up -d
```

### 4.2.4 Criação do canal

Os canais são criados construindo uma transacção de criação de canal e enviando essa transacção ao serviço de ordenação. Essa transacção especifica a configuração inicial do canal e é usada pelo serviço de ordenação para escrever o bloco de geração do canal (*Genesis block*).

O primeiro canal criado na rede Hyperledger Fabric é o canal de sistema. Este define o conjunto de nós *orderer* que definem o serviço de ordenação e o conjunto de organizações que actuam como administradores do serviço de ordenação.

O canal de sistema também inclui as organizações que fazem parte do consórcio do *blockchain*. O consórcio é um conjunto de *peers* que pertence ao canal do sistema, mas que não são administradores do serviço de ordenação. Os membros do consórcio podem adicionar novos canais e ainda adicionar outras organizações do consórcio como membros do canal.

O Hyperledger Fabric disponibiliza uma ferramenta para facilitar a construção do ficheiro transação do *Block Genesis*. Esta ferramenta é nomeada de ***configtxgen***. A função dela é ler o ficheiro configtx.yaml e gravar as informações relevantes na transacção de criação do canal.

Antes da criação dos canais tem que se configurar o ficheiro **configtx.yaml**. Este ficheiro especifica a configuração dos canais. A ferramenta vai verificar os perfis de canal escritos no ficheiro configtx.yaml e grava as mesmas num formato protobuf.

Para a criação do *Block Genesis* foi usado um perfil de canal definido no ficheiro configtx.yaml que tem como nome "TwoOrgsOrdererGenesis". O comando executado foi o seguinte:

```
configtxgen
-profile TwoOrgsOrdererGenesis
-outputBlock ./channel-artifacts/nome_canal.block
-channelID nome_canal
```

A criação do canal foi feita usando o *peer* da Organização 1 com o comando osnadmin. É feito um pedido ao *endpoint* do *orderer* para ser criado e para o *peer* da Organização 1 se juntar ao canal.

```
osnadmin channel join
--channelID nome_canal
--config-block ./channel-artifacts/nome_canal.block
-o localhost:7053
--ca-file "$ORDERER_CA"
--client-cert "$ORDERER_ADMIN_TLS_SIGN_CERT"
--client-key "$ORDERER_ADMIN_TLS_PRIVATE_KEY"
```

Após o canal criado já é possível adicionar os *peers* de cada organização ao mesmo. Para isso em cada *peer* é executado o seguinte comando:

```
peer channel join -b ./channel-artifacts/nome_canal.block
```

Depois de os *peers* estarem adicionados ao canal, é preciso definir os *peer* âncora para cada organização. O protocolo *Gossip* usa os mesmos para garantir que os colegas em diferentes organizações se conheçam uns aos outros.

Sempre que um bloco de configuração tem uma nova atualização para os *peer* âncora, os outros *peers* do sistema vão procurar os âncora e com eles aprendem essas novas atualizações. Uma vez que a comunicação do protocolo *Gossip* é constante e os colegas estão sempre a pedir para serem actualizados sobre a existência de novos *peers* é estabelecida uma visão comum de associação para o canal.

Para definir o *peer* âncora em cada organização ou para atualizar a configuração do canal é necessário usar a ferramenta **configxlator**. Os processos para adicionar *peer* âncora são semelhante aos processos para fazer atualizações a um canal. Uma outra ferramenta necessária para a implementação dos *peer* âncora é o jq.

Para definir o *peer* âncora da Organização 1 deve-se seguir as seguintes etapas. A primeira etapa é extrair o bloco mais recente da configuração do canal. Deve ser executado o comando peer channel join e definir as variáveis de ambiente de forma a operar o peer CLI como sendo o administrador da Organização 1.

```
export FABRIC_CFG_PATH=config/
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=organizations/peerOrganizations
    /org1/peers/peer.org1/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=organizations/peerOrganizations/
    org1/users/Admin/msp
export CORE_PEER_ADDRESS=localhost:7051
peer channel join -b ./channel-artifacts/nome_canal.block
```

Após a execução do comando é retornada a informação do bloco. Neste caso, como o bloco de configuração de canal é o primeiro, é retornado o valor zero.

A segunda etapa é descodificar o bloco recebido em JSON de forma a trabalhar na configuração do canal. Para isso usa-se a ferramenta **configtxlator**. É ainda retirada a informação que não pertence às configurações do canal (config.json).

```
configtxlator proto_decode
    --input config_block.pb
    --type common.Block
    --output config_block.json
```

```
jq '.data.data[0].payload.data.config'
config_block.json > config.json
```

Posteriormente é feita uma cópia do ficheiro config.json para não se alterar diretamente no ficheiro original.

```
cp config.json config_copy.json
```

A terceira etapa é adicionar à configuração do canal a informação do par âncora para a Organização 1. Para isso é utilizada a ferramenta jq. Após ser executado o comando, a configuração do canal fica atualizada no ficheiro modified_config.json no formato JSON.

```
jq '.channel_group.groups.Application.groups.Org1MSP.values +=
    {"AnchorPeers":{"mod_policy":
        "Admins","value":{"anchor_peers":
    [{"host": "peer0.org1.example.com","port": 7051}]},
    "version": "0"}}'
config_copy.json > modified_config.json
```

A quarta etapa volta a converter as recentes configurações do canal novamente no formato protobuf. Essa actualização vai ficar guardada no ficheiro config_update.pb.

```
configtxlator proto_encode
    --input config.json
    --type common.Config
    --output config.pb
configtxlator proto_encode
    --input modified_config.json
    --type common.Config
    --output modified_config.pb
configtxlator compute_update
    --channel_id channel1
    --original config.pb
    --updated modified_config.pb
    --output config_update.pb
```

A quinta etapa é transformar a configuração do ficheiro anteriormente gerado num envelope de transação para criar a nova atualização de configuração do canal.

```
configtxlator proto_decode
    --input config_update.pb
    --type common.ConfigUpdate --output config_update.json
echo '{"payload":{"header":{"channel_header":
    {"channel_id":"channel1", "type":2}},
    "data":{"config_update":'$(cat config_update.json)'}}}'
    | jq . > config_update_in_envelope.json
configtxlator proto_encode
    --input config_update_in_envelope.json
    --type common.Envelope
    --output config_update_in_envelope.pb
```

Por fim a sexta etapa consiste em actualizar a configuração do canal. Após executar o comando é retornada uma informação sobre o estado da mesma.

```
peer channel update
    -f channel-artifacts/config_update_in_envelope.pb
    -c channel1
    -o localhost:7050
    --ordererTLSHostnameOverride orderer
    --tls --cafile organizations/ordererOrganizations/orderer
        /orderers/orderer/msp/tlscacerts/tlsca.cert.pem
```

O processo deve ser repetido para os outros peers de cada organização.

## 4.3 Desenvolvimento do *Chaincode*

O *Chaincode* foi desenvolvido usando a linguagem de programação Go. Este uma vez que lida com a lógica do negócio acordada pelos membros da rede, pode ser considerado um contrato inteligente. O ciclo de vida do *chaincode* no *Hyperledger Fabric* requer que as organizações concordem com o **nome**, **versão**, **politica de endosso** e os **parâmetros que definem o *chaincode***.

Para que exista um acordo há quatro etapas que os membros do canal precisam concluir. No entanto nem todas as etapas precisam ser concluídas por todas as organizações. As etapas são as seguintes: Criar um ***package* do *chaincode***; **instalar nos *peers***; **aprovar uma definição do *chaincode* para a organização** e **confirmar a definição do *chaincode* para o canal**.

### 4.3.1 Criar o *package* do *chaincode*

Para que um *chaincode* possa ser instalado nos *peers* o mesmo precisa ser empacotado num ficheiro *tar*. Para criar os ficheiros *tar* pode-se usar as ferramentas do Fabric ou ferramentas de terceiros, como é exemplo o GNU tar.

Para o projecto o *package* do *chaincode* foi criado usando as ferramentas do Fabric, executando o seguinte comando.

```
peer lifecycle chaincode package nameTar.tar.gz
--path caminhoTar
--lang go
--label nameTar_versao_contrato
```

**Estrutura do *package* do *chaincode***

```
{
  "path": "water/chaincode-go/contract",
  "type": "golang",
  "label": "basic_24.0"
}
```

**Listagem 4.4:** Estrutura do ficheiro metadata.

O ficheiro tar contem dois ficheiros: um de metadados, Lista 4.4 (metadata.json) e um outro tar "code.tar.gz". O ficheiro metadata.json é um ficheiro JSON que especifica a linguagem de programação usada para criar o *chaincode*, o caminho do código e o nome do *package* do *chaincode*.

### 4.3.2 Instalar o chaincode nos peers

O *chaincode* deve ser instalado em todos os *peers* que vão executar e endossar as transacções. O mesmo deve ser instalado usando um administrador da organização. O *peer* após a instalação vai fazer um *build* do *chaincode* e caso exista algum erro o mesmo vai devolver essa informação. Quando uma instalação é realizada corretamente é devolvida uma informação com um identificador do *package* do *chaincode*. Esse identificador é uma combinação do nome do *package* com uma *hash* do mesmo.

O identificador é usado para associar o *chaincode* instalado nos *peers* com uma definição *chaincode* aprovada pela organização.

Para o projecto o *chaincode* foi instalado usando as ferramentas do Fabric, executando o seguinte comando.

```
peer lifecycle chaincode install nameTar.tar.gz
```

### 4.3.3 Aprovar uma definição de chaincode para a organização

Antes que um *chaincode* possa ser usado num canal o mesmo tem que ter a sua definição de *chaincode* aprovada. A mesma é aprovada pelos membros do canal e cada aprovação atua como um voto por cada organização. A aprovação deve ser feita pelo administrador de cada organização. Após a transação de aprovação ser feita com sucesso, a definição aprovada é guardada e está disponível para todos os *peers* da organização. Uma organização apenas precisa aprovar uma definição de *chaincode* uma única vez e é independente de ter um ou mais *peers*.

Existem parâmetros que tem que ser consistentes em todas as organizações para que uma definição seja aprovada, sendo eles os seguintes.

#### Nome

É o nome que as aplicações vão usar para invocar o *chaincode*.

#### Versão

É um número versão ou um valor associado ao *chaincode*. Cada vez que se atualiza os ficheiros do *chaincode* a versão também deve ser alterada.

#### Sequência

A sequência é o número de vezes que o *chaincode* é definido. É um valor inteiro e que controla as atualizações efectuadas.

#### Politica de endosso

É quais as organizações que precisam executar e validar a saída da transacção. A mesma pode ser enviada através de uma *string* ou fazer referência a uma politica na configuração do canal. Por norma é definida como Canal/Aplicação/Endosso, o que faz com que a maioria das organizações do canal endosse uma transação.

#### Configuração da *Collection*

É um caminho para um ficheiro da definição de dados privados associados ao *chaincode*.

#### Plug-ins ESCC / VSCC

É o nome de um endosso personalizado ou um *plug-in* de validação que o *chaincode* deve usar.

#### Inicialização

No caso de usar a API Fabric Chaincode Shim o chaincode precisa ter uma função init para inicializar o mesmo. É uma função exigida pela interface do *chaincode*, mas que não precisa ser executada. Ao aprovar a definição de *chaincode* é especificado se o init é ou não chamado antes do invoke. No caso de ser definido que a função init tem que ser executada o Fabric garante que a mesma é executada antes de qualquer outra função e ainda que é apenas executada uma única vez.

A função init permite implementar alguma lógica quando o *chaincode* é inicializado. Exemplo disso é implementar algum estado inicial.

No caso de se usar o Fabric peer CLI pode-se enviar um parâmetro –init-required ao aprovar e confirmar a definição de chaincode para indicar se a função init deve ou não ser chamada.

### 4.3.4 Confirmar a definição do chaincode para o canal

Uma vez que um número suficiente de membros do canal tenha aprovado uma definição do *chaincode*, uma organização pode confirmar a definição do *chaincode*. Usando o comando *CheckCommitReadiness* pode-se verificar a confirmação da definição do *chaincode*.

A politica que rege o número de organizações que são precisas para aprovar uma definição de *chaincode* antes da confirmação é ***Channel/Application/LifecycleEndorsement***.

## 4.4 Criação do Cliente Fabric

Para interagir com a rede *blockchain* foi desenvolvido uma aplicação, designada por **Cliente Fabric**. Esta foi desenvolvida com a linguagem de programação Node.js. O Cliente fabric liga-se a um protocolo MQTT de forma a obter os dados para posteriormente serem guardados no *blockchain*. O cliente usa o Fabric SDK de forma a interagir com a rede do Fabric e assim poder fazer as solicitações ao contracto inteligente.

### 4.4.1 Guardar os *Devices* no *Blockchain*

Antes de o *Blockchain* começar a receber os dados da qualidade da água, foram inseridos no *blockchain* todos os *devices*. Assim todos os *devices* ficam identificados e apenas os dados «capturados» por esses *devices* podem ser inseridos no *blockchain*. Na Figura 4.3 pode-se ver a ligação entre os *devices* e a informação recolhida pelos mesmos. Existe uma relação de 1-n entre os *devices* e os dados da água.

Os *devices* podem ser inseridos no *blockchain* usando uma função definida no contrato inteligente, nomeada de InitDevice. Essa função tem o objectivo de iniciar o livro da razão com os primeiros dados. A mesma é iniciada após a instalação do contrato. Nó Código 4.5

é apresentada a função InitDevice definida no contrato. A função e iniciada executando o seguinte comando:

```
peer chaincode invoke
    -o localhost:7050
    --ordererTLSHostnameOverride orderer.example.com
    --tls
    --cafile ./organizations/ordererOrganizations/order/orderers/orderer
        /msp/tlscacerts/tlsca.example.com-cert.pem
    -C mychannel -n basic --peerAddresses localhost:7051
    --tlsRootCertFiles ./organizations/peerOrganizations/org1/
        peers/peer0/tls/ca.crt
    --peerAddresses localhost:9051
    --tlsRootCertFiles ./organizations/peerOrganizations/org2/
        peers/peer0/tls/ca.crt
    -c '{"function":"InitDevice","Args":[]}'
```

```
1  func (s *SmartContract) InitDevice(ctx contractapi.
       TransactionContextInterface) error {
2   devices := []Device{
3       {
4        ID: "D-1",
5        Type_data: "device",
6        DeviceId: "7d7de2b9-5937-49b6-a968-a893c918fcab",
7        DeviceName: "water-analysis-1",
8        CoordX: 37.956938 ,
9        CoordY: -8.067635,
10      },
11      {
12       ID: "D-2",
13       Type_data: "device",
14       DeviceId: "43cd2f42-7859-448a-b43a-938bd66bada7",
15       DeviceName: "water-analysis-2",
16       CoordX: 37.962100 ,
17       CoordY: -8.067820,
18      },
19     }
20     for _, device := range devices {
21      deviceJSON, err := json.Marshal(device)
22      if err != nil {
23       return err
24      }
25      err = ctx.GetStub().PutState(device.ID, deviceJSON)
26      if err != nil {
```

```
27          return fmt.Errorf("failed to put to world state. %v", err)
28        }
29      }
30      return nil
31  }
```

**Listagem 4.5:** Lista de Devices iniciados no Ledger

Existe uma outra função definida no contrato que permite adicionar novos *devices* ao *blockchain*. Essa função é nomeada ***CreateDevice***. A mesma recebe como parâmetros dados do device. (ID, DeviceId, DeviceName, CoordX e o CoordY). Essa função pode ser executada usando o seguinte comando:

```
peer chaincode invoke
-o localhost:7050
--ordererTLSHostnameOverride orderer.example.com
--tls
--cafile ./organizations/ordererOrganizations/order/orderers/orderer
    /msp/tlscacerts/tlsca.example.com-cert.pem
-C mychannel
-n basic
--peerAddresses localhost:7051
--tlsRootCertFiles ./organizations/peerOrganizations/org1/
    peers/peer0/tls/ca.crt
--peerAddresses localhost:9051
--tlsRootCertFiles ./organizations/peerOrganizations/org2/
    peers/peer0/tls/ca.crt
-c '{"Args":["CreateDevice", "1",
    "7d7de2b9-5937-49b6-a968-a893c918fcab",
    "water-analysis-1", "37.956938", "8.067635"]}'
```

### 4.4.2 Processo para guardar no *Blockchain*

O processo para guardar os dados no *blockchain* é representado em seis etapas.

- Seleção da identidade de uma carteira

  A aplicação usa a *class Fabric Wallet* para localizar uma carteira na estrutura de pastas onde a mesma é executada. Após encontrar a carteira, a mesma contém os certificados digitais X.509 para aceder a uma organização. A *class Fabric Wallet* é usada da seguinte forma.

```
1     // walletPath é o caminho para as directorias da carteira
```

```
2     wallet = await Wallets.newFileSystemWallet(walletPath);
3
```

- Fazer a ligação/conexão a uma *gateway*.

  Uma outra *classe* usada pela aplicação é o *Fabric Gateway*. É esta *classe* que indica um ou mais *peers* para dar acesso a rede. A função recebe dois parâmetros de entrada sendo os ***connectionProfile*** e ***connectionOptions***. O primeiro é a localização de um ficheiro que define um perfil de ligação. Este perfil de ligação identifica quais os *peers* a que se pode ligar a aplicação. O segundo parâmetro é um conjunto de opções usadas para controlar como aplicação interage com a organização. A *class Fabric Gateway* é usada da seguinte forma.

```
1     await gateway.connect(ccp, {
2       wallet,
3       identity: org1UserId,
4       discovery: { enabled: true, asLocalhost: true }
5      });
6
```

- Aceder a rede desejada

  Nesta etapa é possível escolher o canal a que a aplicação se vai ligar. De referir que aqui a aplicação podia ligar-se a vários canais em simultâneo. A ligação ao canal é apresentada da seguinte forma.

```
1     // channelName é igual ao nome do canal
2     const network = await gateway.getNetwork(channelName);
3
```

- Construir uma solicitação de transação ao Contrato Inteligente

  Nesta etapa é possível adquirir o acesso ao contrato inteligente.

```
1     const contract = network.getContract(chaincodeName);
2
```

- Enviar as transacções para a rede

  Uma das solicitações que podem ser feitas ao contrato inteligente é a de inserir os dados da água no *blockchain*. A aplicação usa a função **submitTransaction**. Essa função recebe como parâmetros o nome da função a executar no contrato e os parâmetros que essa função recebe. Neste caso como é para inserir os dados da água. Os parâmetros que vai receber são: o id transacção; o device id (é o identificador único do *device*); a data e hora em que o dado foi recolhido; o valor da temperatura da agua e o valor do pH da água. A transação é submetida da seguinte forma.

```
result = await contract.submitTransaction('CreateData',
        10, "7d7de2b9-5937-49b6-a968-a893c918fcab",
        "2021-09-02T10:00:00", "20", "9");

```

- Processar a Resposta

  A ultima etapa do processo para guardar os dados no *blockchain* é o processo da resposta. Nesta etapa é apresentada uma informação devolvida pelo contrato a informar se a transacção foi o não efectuada com sucesso. Embora possa parecer que a aplicação recebe logo a resposta após a conclusão do contrato inteligente, pode-se afirmar que não é o caso. Antes de se obter a resposta o SDK gerência o processo de consenso e notifica a aplicação quando é concluído o processo.

## 4.5 Consultar dados no *Blockchain*

### 4.5.1 Web Service

Para consultar os dados inseridos no *blockchain* foi desenvolvido um web service. Este foi desenvolvido através da *framework* *Express*. É uma *framework* que usa a linguagem de programação Node.js e fornece um conjunto robusto de recursos para o desenvolvimento de aplicações web e móveis.

Este *web service* tal como o Cliente Fabric descrito anteriormente, interage com a rede *blockchain*. Para isso usa o Fabric SDK.

#### Consultar todos os dispositivos IoT inseridos no blockchain

Um dos pedidos desenvolvidos no web service é de consultar todos os dispositivos IoT que estão no blockchain. O pedido é requerido através do protocolo HTTPS usando o método GET. Após receber o pedido, o *web service* vai interagir com o contrato inteligente executando a função "GetAllDevices"definida no mesmo. Esta função vai consultar todos os dispositivos IoT ao *blockchain* e devolve uma estrutura de dados com os mesmos.

#### Consultar Dados da Água no *blockchain*

No *web service* foi implementado um pedido para consultar os dados da água. O pedido é requerido através do protocolo HTTPS usando o método GET e que recebe como parâmetro o ID do dispositivo IoT que se deseja consultar. Este pedido pode ainda consultar os dados entre intervalos de tempo. O pedido vai interagir com o contrato inteligente executando a função "GetAllDataToDevice". Esta função vai consultar os dados da água referente ao dispositivo IoT que recebe como parâmetro.

### 4.5.2 Interface Gráfica

Para apresentar os dados das consulta foi desenvolvida uma interface gráfica. Esta comunica com o *web service* anteriormente descrito e apresenta os dados aos utilizadores.

Na Figura 4.4 é apresentado um *Dashboard*. O mesmo mostra o número de dispositivos IoT contidos no *blockchain*, o número de transacções relativas aos dados da água e um gráfico onde mostra os dados relativos a propriedades da água em casa dispositivo IoT.

Na Figura 4.5 é apresentada uma listagem de todos os dispositivos IoT inseridos no *blockchain* e as suas propriedades.

Na Figura 4.6 é apresentado a listagem dos dados da água adquirido pelo dispositivo IoT que foi seleccionado.

## 4.6 Conclusão

A *framework* Hyperledger Fabric corresponde aos requisitos necessários para a realização de uma aplicação para guardar os dados da água de uma forma segura e fiável. O Hyperledger Fabric requer uma curva de aprendizagem acentuada, devido a envolver muito trabalho manual em termos de configurações de ficheiros. Sendo um trabalho complexo e moroso.

Após definir a criação da rede é possível criar os CA e os MSP para cada membro das organizações. Já com o material criptográfico para cada membro foi possível criar os *peers* de cada organização. Após a rede criada, é desenvolvido o *chaincode*.

O *chaincode* permite criar as regras com a lógica do negócio. Uma vantagem do Hyperledger Fabric na criação do *chaincode* é que o mesmo permite desenvolver o mesmo em várias linguagens de programação.

O *chaincode* é instalado e aprovado em cada *peer*. Assim com toda a estrutura montada, já é possível criar um Cliente Fabric de forma obter os dados e a guardar no *blockchain*.

O Cliente Fabric obtém os dados através de um protocolo de mensagens. Após ter os dados, vai interagir com a rede blockchain para guardar os dados.

É possível ver os dados no *blockchain* através dum *web service* com permissões para se ligar ao *blockchain* e disponibilizar a informação. Foi desenvolvida uma interface gráfica para apresentar aos utilizadores a informação devolvida pelo *web service*.

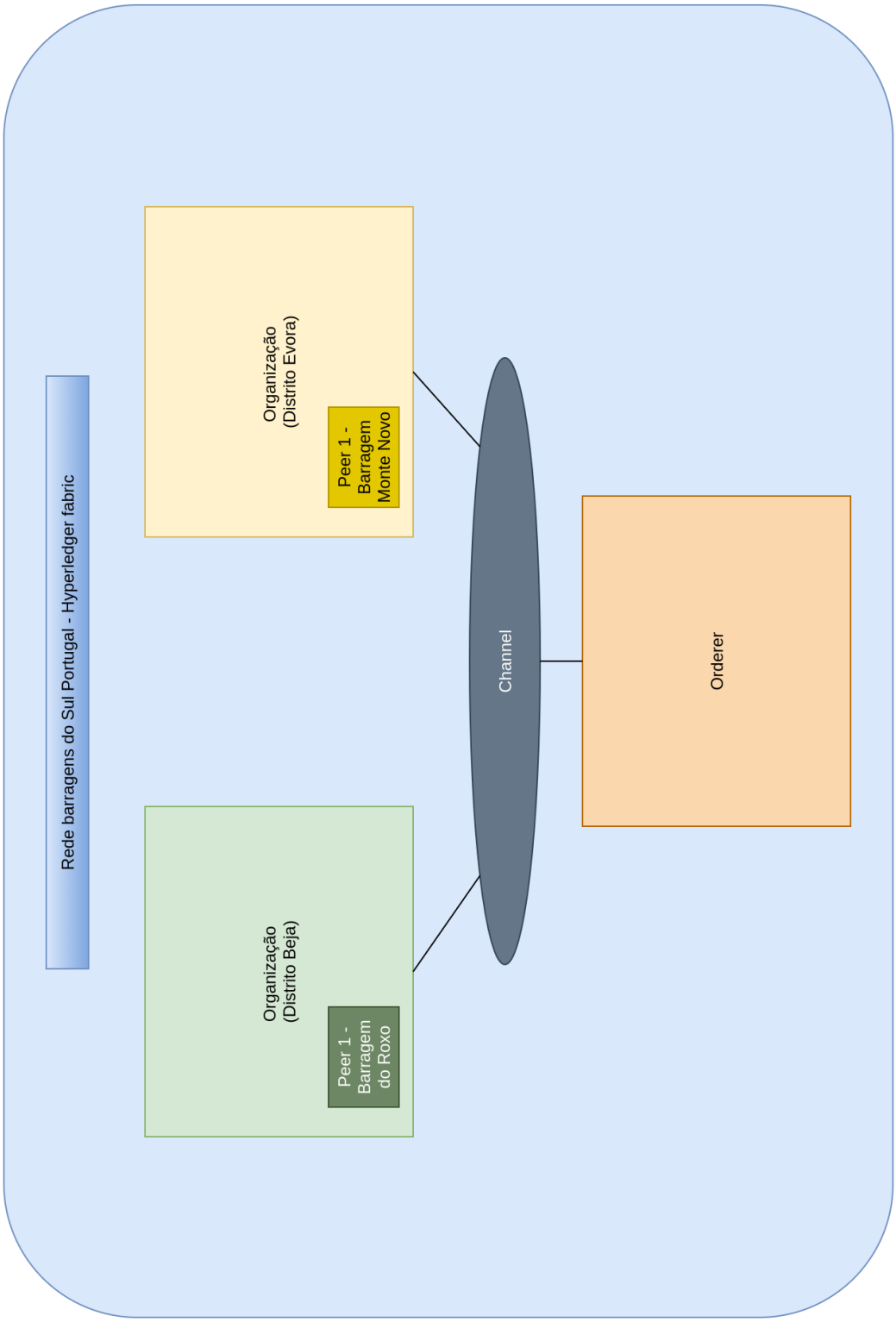


**Figura 4.1:** Exemplo da Rede Hyperledger Fabric representando as organizações em distritos de Portugal.

```
CONTAINER ID   IMAGE                           COMMAND                  CREATED       STATUS       PORTS                                                NAMES
39a45d94988c   hyperledger/fabric-ca:latest    "sh -c 'fabric-ca-se…"   3 weeks ago   Up 20 hours   7054/tcp, 0.0.0.0:8054->8054/tcp, :::8054->8054/tcp   ca_org2
9e11d857aa7b   hyperledger/fabric-ca:latest    "sh -c 'fabric-ca-se…"   3 weeks ago   Up 20 hours   0.0.0.0:7054->7054/tcp, :::7054->7054/tcp             ca_org1
2a5e8fb39b98   hyperledger/fabric-ca:latest    "sh -c 'fabric-ca-se…"   3 weeks ago   Up 20 hours   7054/tcp, 0.0.0.0:9054->9054/tcp, :::9054->9054/tcp   ca_orderer
```

**Figura 4.2:** Lista dos containers dos CA.

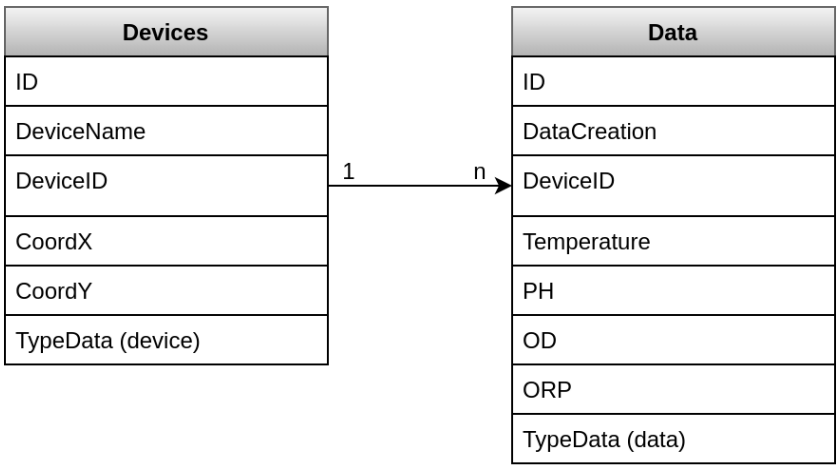


**Figura 4.3:** Relação entre os dados a inserir no *blockchain*.

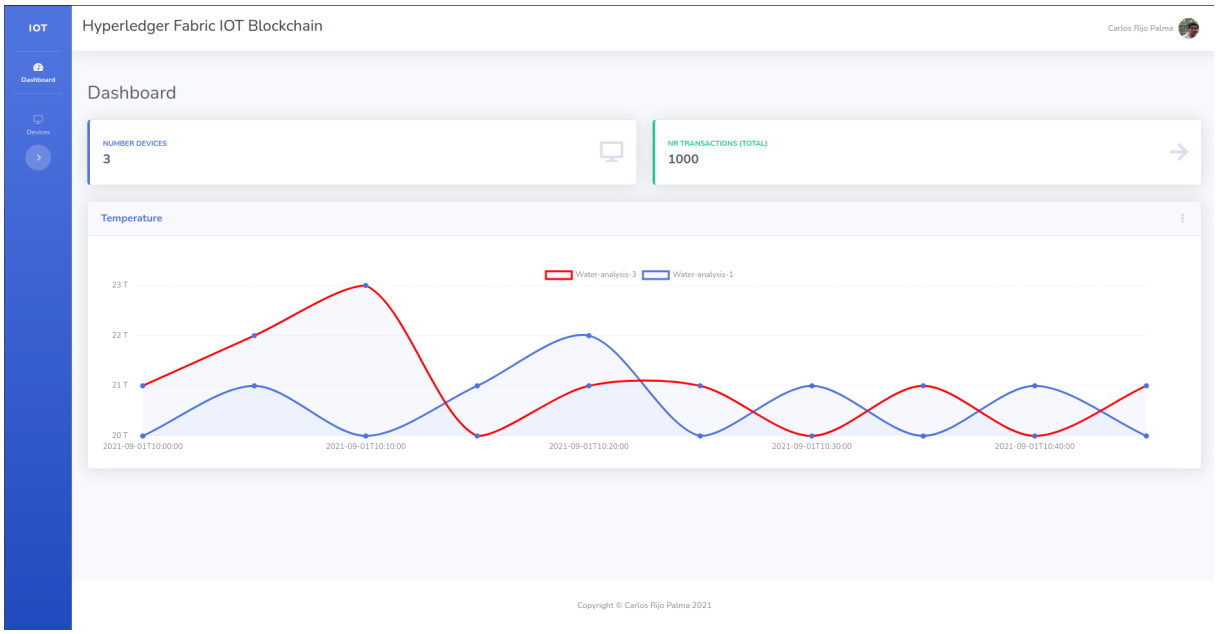


**Figura 4.4:** *Dashboard* Dados *Blockchain*.

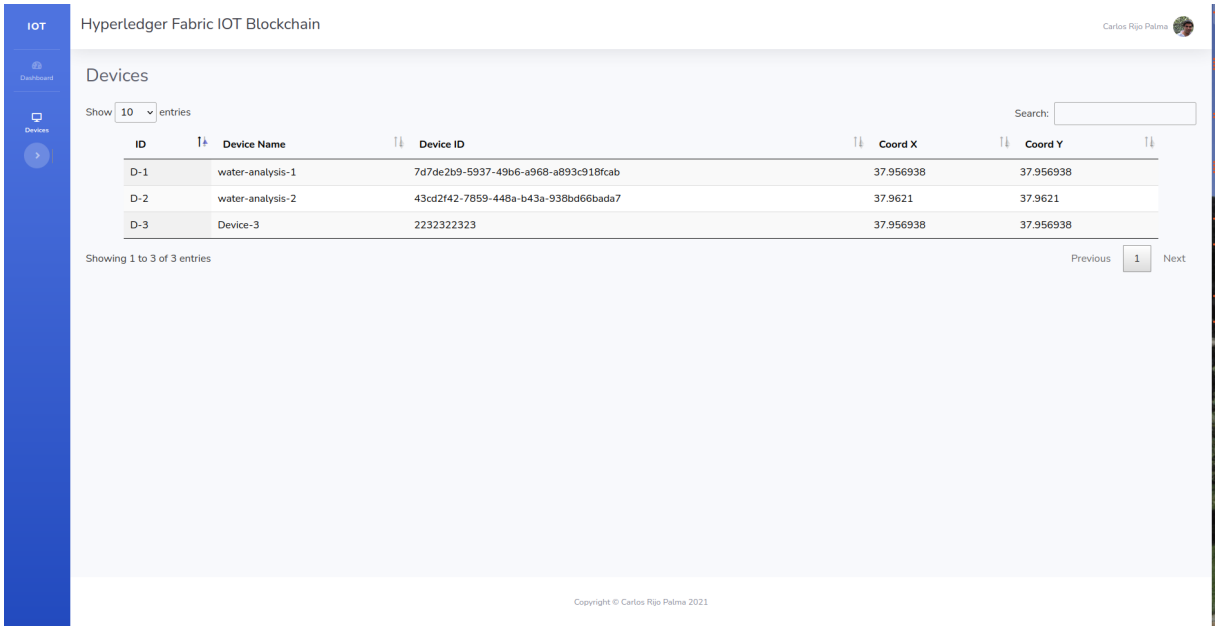


**Figura 4.5:** Lista de dispositivos IoT no *blockchain*.

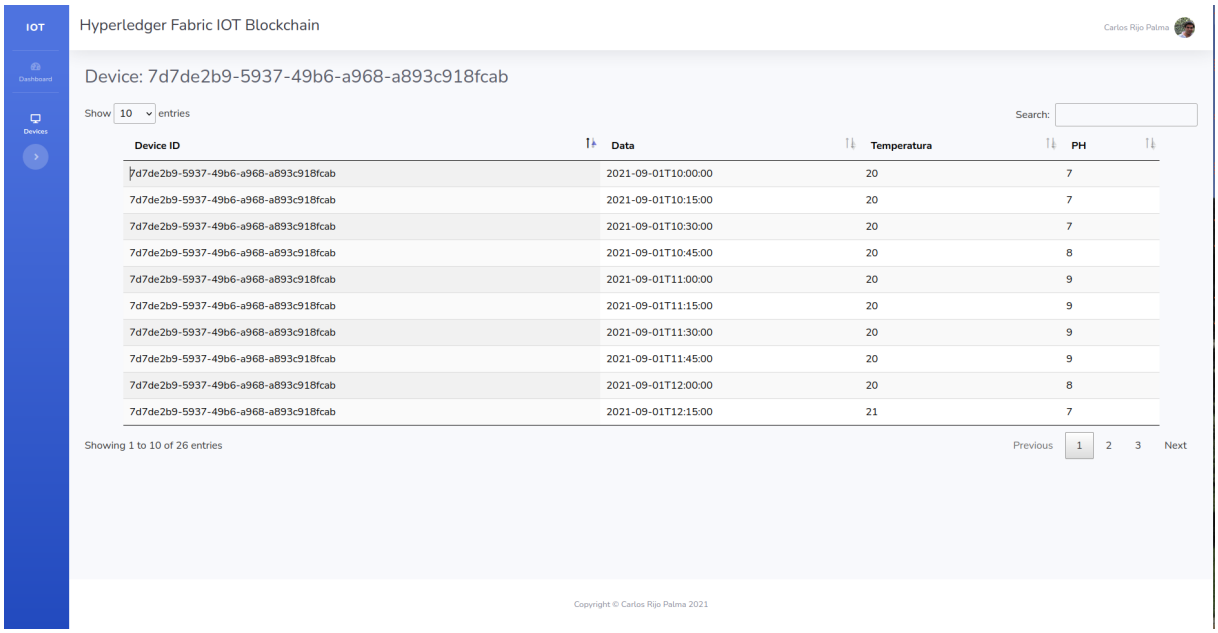


**Figura 4.6:** Lista de Dados Água no *blockchain* por Dispositivo.

# Capítulo 5

# Conclusões

As tecnologias *blockchain* têm início devido às criptomoedas. Pode-se dizer dada a sua potencialidade que está cada vez mais a emergir em várias áreas. Sendo exemplo a área do IoT, onde já existem vários projetos a usar esta tecnologia. Uma tecnologia que provou resolver muitas das preocupações associadas à segurança, privacidade e escalabilidade dos sistemas IoT.

Nesta dissertação podemos referir que ao desenvolver uma aplicação utilizando esta tecnologia, para guardar os dados da qualidade da água, estamos a confirmar que os mesmos podem ser guardados de uma forma segura, tornando-se imutáveis após a sua gravação no *blockchain*.

Para a criação desta aplicação foi utilizada uma *framework Hyperledger Fabric*. A mesma facilitou a criação da rede, dada a sua estrutura modular e a capacidade de se adaptar a diferentes casos de uso.

Para esta aplicação cada organização da rede foi associada a um distrito de Portugal e cada recurso hídrico associado a um *peer*. Foi desenvolvido um contrato inteligente e instalado em cada *peer*. No mesmo foram definidas as regras para receber os dados. A criação do Cliente Fabric e a utilização do Fabric SDK permite interagir com a rede do Fabric e fazer os pedidos ao contrato inteligente.

A criação da interface gráfica ajuda os utilizadores na visualização e consulta dos dados guardados no *blockchain*.

Como trabalho futuro considera-se importante rever a forma de consulta dos dados e criar um módulo para fazer a integração do Chirpstack com o Hyperledger Fabric.

No que diz respeito à consulta dos dados. Estes podiam estar sincronizados com uma base dados PostgreSQL para melhorar a sua consulta. Esta base dados deve ter instalada a extensão PostGIS.

Assim é possível ter as coordenadas dos dispositivos IoT «georreferenciadas»em base de dados. É possível também utilizar o Qgis ligada a essa base de dados.

Esta dissertação integrou-se com os trabalhos realizados no projeto AquaQ2. Projecto n.o 039494, com as referências ALT20-03-0145-FEDER-039494(SAICT-ALT/39494/2018),

financiado pelo Programa Operacional Regional do Alentejo/Portugal 2020/FEDER.

# Bibliografia


[Boc+17] Thomas Bocek et al. «Blockchains everywhere - a use-case of blockchains in the pharma supply-chain». en. Em: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Lisbon, Portugal: IEEE, mai. de 2017, pp. 772–777. ISBN: 978-3-901882-89-0. DOI: 10.23919/INM.2017.7987376. URL: http://ieeexplore.ieee.org/document/7987376/ (acedido em 06/02/2021) (citado na página 9).

[Clo] Cloudflare. *Cloudflare*. URL: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/. (accessed: 02.12.2021) (citado na página 20).

[Fen16] Feng Tian. «An agri-food supply chain traceability system for China based on RFID & blockchain technology». en. Em: *2016 13th International Conference on Service Systems and Service Management (ICSSSM)*. Kunming, China: IEEE, jun. de 2016, pp. 1–6. ISBN: 978-1-5090-2842-9. DOI: 10.1109/ICSSSM.2016.7538424. URL: http://ieeexplore.ieee.org/document/7538424/ (acedido em 19/01/2021) (citado na página 8).

[FF18] Tiago M. Fernandez-Carames e Paula Fraga-Lamas. «A Review on the Use of Blockchain for the Internet of Things». en. Em: *IEEE Access* 6 (2018), pp. 32979–33001. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2842685. URL: https://ieeexplore.ieee.org/document/8370027/ (acedido em 25/01/2021) (citado nas páginas 8, 9).

[GG] Narendra Gupta e Vipin Gupta. «The Performance and Optimization of I2C Protocol using Verilog on Xilinx Tool». en. Em: (), p. 4 (citado na página 15).

[GW21] Jianxiong Guo e Weili Wu. «Differential Privacy-Based Online Allocations towards Integrating Blockchain and Edge Computing». Em: *arXiv:2101.02834 [cs]* (7 de jan. de 2021). arXiv: 2101.02834. URL: http://arxiv.org/abs/2101.02834 (acedido em 20/01/2021) (citado na página 4).

[Hypa] Hyperledger Fabric. *Hyperledger*. URL: https://hyperledger-fabric.readthedocs.io/en/latest/identity/identity.html?highlight=PKI#what-is-an-identity. (accessed: 25.11.2021) (citado na página 17).

[Hypb] Hyperledger Fabric. *Hyperledger*. URL: https://hyperledger-fabric.readthedocs.io/en/latest/identity/identity.html?highlight=PKI#what-are-pkis. (accessed: 25.11.2021) (citado na página 18).

[Hypc] Hyperledger Fabric. *Hyperledger*. URL: https://hyperledger-fabric.readthedocs.io/en/latest/identity/identity.html?highlight=PKI#digital-certificates. (accessed: 25.11.2021) (citado na página 18).

[Hypd] Hyperledger Fabric. *Hyperledger*. URL: https://hyperledger-fabric.readthedocs.io/en/latest/identity/identity.html?highlight=PKI#certificate-authorities. (accessed: 25.11.2021) (citado na página 19).

[Hype] Hyperledger Fabric. *Hyperledger*. URL: https://hyperledger-fabric.readthedocs.io/en/latest/identity/identity.html?highlight=PKI#certificate-revocation-lists. (accessed: 25.11.2021) (citado na página 19).

[Hypf] Hyperledger Fabric. *Hyperledger*. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.2/membership/membership.html?highlight=MSP#membership-service-provider-msp. (accessed: 02.12.2021) (citado na página 20).

[LBA17] Thomas Lundqvist, Andreas de Blanche e H. Robert H. Andersson. «Thing-to-thing electricity micro payments using blockchain technology». en. Em: *2017 Global Internet of Things Summit (GIoTS)*. Geneva, Switzerland: IEEE, jun. de 2017, pp. 1–6. ISBN: 978-1-5090-5873-0. DOI: 10.1109/GIOTS.2017.8016254. URL: http://ieeexplore.ieee.org/document/8016254/ (acedido em 27/01/2021) (citado na página 8).

[Pau] Paulo Henrique Alves, Rodrigo Laigner, Rafael Nasser, Gustavo Robichez, Hélio Lopes, Marcos Kalinowski. *Desmistificando Blockchain: Conceitos e Aplicações*. URL: http://www-di.inf.puc-rio.br/~kalinowski//publications/AlvesLNRLK20.pd. (accessed: 25.11.2021) (citado na página 6).

[she18] Sunith shetty. *Hands-On Blockchain with Hyperledger*. International series of monographs on physics. Packt Publishing Ltd, 2018. ISBN: 978-1-78899-452-1 (citado na página 14).

[TNV18] Parth Thakkar, Senthil Nathan e Balaji Viswanathan. «Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform». Em: *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). Milwaukee, WI: IEEE, set. de 2018, pp. 264–276. ISBN: 978-1-5386-6886-3. DOI: 10.1109/MASCOTS.2018.00034. URL: https://ieeexplore.ieee.org/document/8526892/ (acedido em 09/01/2021) (citado na página 14).

# Apêndices

# Apêndice I

# Configurações do Hyperledger Fabric

```
# Copyright IBM Corp. All Rights Reserved.
#
# SPDX-License-Identifier: Apache-2.0
#

---
################################################################
#
#   Section: Organizations
#
#   - This section defines the different organizational identities which will
#   be referenced later in the configuration.
#
################################################################
Organizations:

    # SampleOrg defines an MSP using the sampleconfig.  It should never be used
    # in production but may be used as a template for other definitions
    - &OrdererOrg
        # DefaultOrg defines the organization which is used in the sampleconfig
        # of the fabric.git development environment
        Name: OrdererOrg

        # ID to load the MSP definition as
        ID: OrdererMSP

```

```
27          # MSPDir is the filesystem path which contains the MSP
    configuration
28          MSPDir: ../organizations/ordererOrganizations/example.com/msp
29
30          # Policies defines the set of policies at this level of the
    config tree
31          # For organization policies, their canonical path is usually
32          #   /Channel/<Application|Orderer>/<OrgName>/<PolicyName>
33          Policies:
34              Readers:
35                  Type: Signature
36                  Rule: "OR('OrdererMSP.member')"
37              Writers:
38                  Type: Signature
39                  Rule: "OR('OrdererMSP.member')"
40              Admins:
41                  Type: Signature
42                  Rule: "OR('OrdererMSP.admin')"
43
44          OrdererEndpoints:
45              - orderer.example.com:7050
46
47      - &Org1
48          # DefaultOrg defines the organization which is used in the
    sampleconfig
49          # of the fabric.git development environment
50          Name: Org1MSP
51
52          # ID to load the MSP definition as
53          ID: Org1MSP
54
55          MSPDir: ../organizations/peerOrganizations/org1.example.com/
    msp
56
57          # Policies defines the set of policies at this level of the
    config tree
58          # For organization policies, their canonical path is usually
59          #   /Channel/<Application|Orderer>/<OrgName>/<PolicyName>
60          Policies:
61              Readers:
62                  Type: Signature
63                  Rule: "OR('Org1MSP.admin', 'Org1MSP.peer', 'Org1MSP.
    client')"
64              Writers:
65                  Type: Signature
66                  Rule: "OR('Org1MSP.admin', 'Org1MSP.client')"
67              Admins:
68                  Type: Signature
69                  Rule: "OR('Org1MSP.admin')"
```

```
70              Endorsement:
71                  Type: Signature
72                  Rule: "OR('Org1MSP.peer')"
73
74      - &Org2
75          # DefaultOrg defines the organization which is used in the
        sampleconfig
76          # of the fabric.git development environment
77          Name: Org2MSP
78
79          # ID to load the MSP definition as
80          ID: Org2MSP
81
82          MSPDir: ../organizations/peerOrganizations/org2.example.com/
        msp
83
84          # Policies defines the set of policies at this level of the
        config tree
85          # For organization policies, their canonical path is usually
86          #   /Channel/<Application|Orderer>/<OrgName>/<PolicyName>
87          Policies:
88              Readers:
89                  Type: Signature
90                  Rule: "OR('Org2MSP.admin', 'Org2MSP.peer', 'Org2MSP.
        client')"
91              Writers:
92                  Type: Signature
93                  Rule: "OR('Org2MSP.admin', 'Org2MSP.client')"
94              Admins:
95                  Type: Signature
96                  Rule: "OR('Org2MSP.admin')"
97              Endorsement:
98                  Type: Signature
99                  Rule: "OR('Org2MSP.peer')"
100
101 ################################################################
102 #
103 #   SECTION: Capabilities
104 #
105 #   - This section defines the capabilities of fabric network. This is
         a new
106 #   concept as of v1.1.0 and should not be utilized in mixed networks
        with
107 #   v1.0.x peers and orderers.  Capabilities define features which
        must be
108 #   present in a fabric binary for that binary to safely participate
        in the
109 #   fabric network.  For instance, if a new MSP type is added, newer
        binaries
```

```
110 #   might recognize and validate the signatures from this type, while
        older
111 #   binaries without this support would be unable to validate those
112 #   transactions.  This could lead to different versions of the fabric
        binaries
113 #   having different world states.  Instead, defining a capability for
        a channel
114 #   informs those binaries without this capability that they must
        cease
115 #   processing transactions until they have been upgraded.  For v1.0.x
        if any
116 #   capabilities are defined (including a map with all capabilities
        turned off)
117 #   then the v1.0.x peer will deliberately crash.
118 #
119 ################################################################
120 Capabilities:
121     # Channel capabilities apply to both the orderers and the peers
        and must be
122     # supported by both.
123     # Set the value of the capability to true to require it.
124     Channel: &ChannelCapabilities
125         # V2_0 capability ensures that orderers and peers behave
        according
126         # to v2.0 channel capabilities. Orderers and peers from
127         # prior releases would behave in an incompatible way, and are
        therefore
128         # not able to participate in channels at v2.0 capability.
129         # Prior to enabling V2.0 channel capabilities, ensure that all
130         # orderers and peers on a channel are at v2.0.0 or later.
131         V2_0: true
132
133     # Orderer capabilities apply only to the orderers, and may be
        safely
134     # used with prior release peers.
135     # Set the value of the capability to true to require it.
136     Orderer: &OrdererCapabilities
137         # V2_0 orderer capability ensures that orderers behave
        according
138         # to v2.0 orderer capabilities. Orderers from
139         # prior releases would behave in an incompatible way, and are
        therefore
140         # not able to participate in channels at v2.0 orderer
        capability.
141         # Prior to enabling V2.0 orderer capabilities, ensure that all
142         # orderers on channel are at v2.0.0 or later.
143         V2_0: true
144
```

```
145     # Application capabilities apply only to the peer network, and may
        be safely
146     # used with prior release orderers.
147     # Set the value of the capability to true to require it.
148     Application: &ApplicationCapabilities
149         # V2_0 application capability ensures that peers behave
        according
150         # to v2.0 application capabilities. Peers from
151         # prior releases would behave in an incompatible way, and are
        therefore
152         # not able to participate in channels at v2.0 application
        capability.
153         # Prior to enabling V2.0 application capabilities, ensure that
         all
154         # peers on channel are at v2.0.0 or later.
155         V2_0: true
156
157 ################################################################
158 #
159 #   SECTION: Application
160 #
161 #   - This section defines the values to encode into a config
        transaction or
162 #   genesis block for application related parameters
163 #
164 ################################################################
165 Application: &ApplicationDefaults
166
167     # Organizations is the list of orgs which are defined as
        participants on
168     # the application side of the network
169     Organizations:
170
171     # Policies defines the set of policies at this level of the config
         tree
172     # For Application policies, their canonical path is
173     #   /Channel/Application/<PolicyName>
174     Policies:
175         Readers:
176             Type: ImplicitMeta
177             Rule: "ANY Readers"
178         Writers:
179             Type: ImplicitMeta
180             Rule: "ANY Writers"
181         Admins:
182             Type: ImplicitMeta
183             Rule: "MAJORITY Admins"
184         LifecycleEndorsement:
185             Type: ImplicitMeta
```

```
186                 Rule: "MAJORITY Endorsement"
187             Endorsement:
188                 Type: ImplicitMeta
189                 Rule: "MAJORITY Endorsement"
190
191         Capabilities:
192             <<: *ApplicationCapabilities
193 ################################################################
194 #
195 #   SECTION: Orderer
196 #
197 #   - This section defines the values to encode into a config
        transaction or
198 #   genesis block for orderer related parameters
199 #
200 ################################################################
201 Orderer: &OrdererDefaults
202
203     # Orderer Type: The orderer implementation to start
204     OrdererType: etcdraft
205
206     # Addresses used to be the list of orderer addresses that clients
        and peers
207     # could connect to.  However, this does not allow clients to
        associate orderer
208     # addresses and orderer organizations which can be useful for
        things such
209     # as TLS validation.  The preferred way to specify orderer
        addresses is now
210     # to include the OrdererEndpoints item in your org definition
211     Addresses:
212         - orderer.example.com:7050
213
214     EtcdRaft:
215         Consenters:
216         - Host: orderer.example.com
217           Port: 7050
218           ClientTLSCert: ../organizations/ordererOrganizations/example
        .com/orderers/orderer.example.com/tls/server.crt
219           ServerTLSCert: ../organizations/ordererOrganizations/example
        .com/orderers/orderer.example.com/tls/server.crt
220
221     # Batch Timeout: The amount of time to wait before creating a
        batch
222     BatchTimeout: 2s
223
224     # Batch Size: Controls the number of messages batched into a block
225     BatchSize:
226
```

```
227         # Max Message Count: The maximum number of messages to permit
    in a batch
228         MaxMessageCount: 10
229
230         # Absolute Max Bytes: The absolute maximum number of bytes
    allowed for
231         # the serialized messages in a batch.
232         AbsoluteMaxBytes: 99 MB
233
234         # Preferred Max Bytes: The preferred maximum number of bytes
    allowed for
235         # the serialized messages in a batch. A message larger than
    the preferred
236         # max bytes will result in a batch larger than preferred max
    bytes.
237         PreferredMaxBytes: 512 KB
238
239     # Organizations is the list of orgs which are defined as
    participants on
240     # the orderer side of the network
241     Organizations:
242
243     # Policies defines the set of policies at this level of the config
     tree
244     # For Orderer policies, their canonical path is
245     #   /Channel/Orderer/<PolicyName>
246     Policies:
247         Readers:
248             Type: ImplicitMeta
249             Rule: "ANY Readers"
250         Writers:
251             Type: ImplicitMeta
252             Rule: "ANY Writers"
253         Admins:
254             Type: ImplicitMeta
255             Rule: "MAJORITY Admins"
256         # BlockValidation specifies what signatures must be included
    in the block
257         # from the orderer for the peer to validate it.
258         BlockValidation:
259             Type: ImplicitMeta
260             Rule: "ANY Writers"
261
262################################################################
263#
264#   CHANNEL
265#
266#   This section defines the values to encode into a config
    transaction or
```

```
267 #   genesis block for channel related parameters.
268 #
269 ################################################################
270 Channel: &ChannelDefaults
271     # Policies defines the set of policies at this level of the config
        tree
272     # For Channel policies, their canonical path is
273     #   /Channel/<PolicyName>
274     Policies:
275         # Who may invoke the 'Deliver' API
276         Readers:
277             Type: ImplicitMeta
278             Rule: "ANY Readers"
279         # Who may invoke the 'Broadcast' API
280         Writers:
281             Type: ImplicitMeta
282             Rule: "ANY Writers"
283         # By default, who may modify elements at this config level
284         Admins:
285             Type: ImplicitMeta
286             Rule: "MAJORITY Admins"
287
288     # Capabilities describes the channel level capabilities, see the
289     # dedicated Capabilities section elsewhere in this file for a full
290     # description
291     Capabilities:
292         <<: *ChannelCapabilities
293
294 ################################################################
295 #
296 #   Profile
297 #
298 #   - Different configuration profiles may be encoded here to be
        specified
299 #   as parameters to the configtxgen tool
300 #
301 ################################################################
302 Profiles:
303
304     TwoOrgsApplicationGenesis:
305         <<: *ChannelDefaults
306         Orderer:
307             <<: *OrdererDefaults
308             Organizations:
309                 - *OrdererOrg
310             Capabilities:
311                 <<: *OrdererCapabilities
312         Application:
313             <<: *ApplicationDefaults
```

```
            Organizations:
                - *Org1
                - *Org2
            Capabilities:
                <<: *ApplicationCapabilities
```

**Listagem I.1:** Ficheiro de configuração configtx.

```
1
2
3 import os
4
5
6 def gen_tls_ca(organization, serial, crlnumber, data_Cert_ROOTCA,
      data_Cert_ICA, dir_create_crt , dir_main = "/tmp"):
7
8     path_dir_crt = "{0}/{1}/".format(dir_main, dir_create_crt)
9
10    os.system("rm -r {0}".format(path_dir_crt))
11
12    # remove cert to organization
13    os.system('rm organizations/fabric-ca/{organization}/ica.*'.format
      (organization = organization))
14    os.system('rm organizations/fabric-ca/{organization}/chain.cert'.
      format(organization = organization))
15
16
17    os.system("mkdir -p {0}".format(path_dir_crt))
18
19    # create files
20    os.system("touch {0}index.txt {0}index.txt.attr".format(
      path_dir_crt))
21
22    os.system("echo {0} > {1}serial".format(serial, path_dir_crt))
23    os.system("echo {0} > {1}crlnumber".format(crlnumber, path_dir_crt
      ))
24
25
26    # Generate self-signed Root CA
27
28    # 1 - generate a private key for RCA
29    # 2 . copy confs to dir_create_crt
30    # 3 - generate a self-signed certificate from the private key
      based on the configuration.
31
32    os.system("cd {path_dir_crt}; openssl ecparam -name prime256v1 -
      genkey -noout -out rca.key".format(path_dir_crt = path_dir_crt))
33    os.system("cp ca.conf {0}ca.cnf".format(path_dir_crt))
```

```
34      os.system('cd {path_dir_crt}; openssl req -config ca.cnf -new -
        x509 -sha256 -extensions \
35                v3_ca -key rca.key -out rca.cert -days 3650 \
36                -subj "/C={C}/ST={ST}/L={L}/O={O}/CN={CN}"'.format(
37              path_dir_crt = path_dir_crt,
38              C = data_Cert_ROOTCA['C'],
39              ST = data_Cert_ROOTCA['ST'],
40              L = data_Cert_ROOTCA['L'],
41              O = data_Cert_ROOTCA['O'],
42              CN = data_Cert_ROOTCA['CN']
43          ))
44
45
46      # 2.3 - Generate ICA issued by Root CA
47      #
48      # 1 - generate a private key for ICA
49      # 2 - generate a CSR based on this ICA with proper information
50      # 3 - RCA issues certificate to ICA based on the CSR
51
52      os.system('cd {path_dir_crt}; openssl ecparam -name prime256v1 -
        genkey \
53          -noout -out ica.key'.format(path_dir_crt = path_dir_crt))
54      os.system('cd {path_dir_crt}; openssl req -new -sha256 -key ica.
        key \
55                -out ica.csr \
56                -subj "/C={C}/ST={ST}/L={L}/O={O}/CN={CN}"'.format(
57                 path_dir_crt = path_dir_crt,
58                 C = data_Cert_ICA['C'],
59                 ST = data_Cert_ICA['ST'],
60                 L = data_Cert_ICA['L'],
61                 O = data_Cert_ICA['O'],
62                 CN = data_Cert_ICA['CN']
63                ))
64
65      os.system('cd {path_dir_crt}; openssl ca -batch -config ca.cnf \
66                -extensions v3_intermediate_ca -days 365 \
67                -notext -md sha256 -in ica.csr -out ica.cert'.format(
68                    path_dir_crt = path_dir_crt))
69
70      os.system('cat {0}ica.cert {0}rca.cert > {0}chain.cert'.format(
        path_dir_crt))
71
72
73      os.system('cp {path_dir_crt}ica.key organizations/fabric-ca/{
        organization}'\
74          .format(
75                path_dir_crt = path_dir_crt,
76                organization = organization
77      ))
```

```
78
79      os.system('cp {path_dir_crt}ica.cert organizations/fabric-ca/{
        organization}'\
80          .format(
81                  path_dir_crt = path_dir_crt,
82                  organization = organization
83      ))
84
85      os.system('cp {path_dir_crt}chain.cert organizations/fabric-ca/{
        organization}'\
86          .format(
87                  path_dir_crt = path_dir_crt,
88                  organization = organization
89      ))
90
91      os.system('openssl x509 -in organizations/fabric-ca/{organization
        }/ica.cert \
92          --noout -subject -issuer'.format(organization = organization))
93
94
95
96
97  def createOrgs(orgs):
98      for organization in orgs:
99          gen_tls_ca(
100             organization['name'],
101             organization['serial'],
102             organization['crlnumber'],
103             organization['data_cert_root_ca'],
104             organization['data_cert_ica'],
105             organization['dir_create_crts'],
106             organization['dir_main'],
107         )
108
109
110
111 if __name__ == '__main__':
112     orgs = [
113         {
114             "name": "org1",
115             "data_cert_root_ca": {
116                 "C": "US",
117                 "ST": "North Carolina",
118                 "L": "Durham",
119                 "O": "org1.example.com",
120                 "CN": "ca.org1.example.com"
121             },
122             "data_cert_ica": {
123                 "C": "US",
```

```
124                 "ST": "North Carolina",
125                 "L": "Durham",
126                 "O": "org1.example.com",
127                 "CN": "ica.org1.example.com"
128             },
129             "serial": 1000,
130             "crlnumber": 1000,
131             "dir_main": "./tmp_cert",
132             "dir_create_crts": "gen-ica_1"
133         },
134         {
135             "name": "org2",
136                 "data_cert_root_ca": {
137                 "C": "US",
138                 "ST": "North Carolina",
139                 "L": "Durham",
140                 "O": "org2.example.com",
141                 "CN": "ca.org2.example.com"
142             },
143             "data_cert_ica": {
144                 "C": "US",
145                 "ST": "North Carolina",
146                 "L": "Durham",
147                 "O": "org2.example.com",
148                 "CN": "ica.org2.example.com"
149             },
150             "serial": 1000,
151             "crlnumber": 1000,
152             "dir_main": "./tmp_cert",
153             "dir_create_crts": "gen-ica_2"
154         },
155         {
156             "name": "ordererOrg",
157                 "data_cert_root_ca": {
158                 "C": "US",
159                 "ST": "North Carolina",
160                 "L": "Durham",
161                 "O": "orderer.example.com",
162                 "CN": "ca.orderer.example.com"
163             },
164             "data_cert_ica": {
165                 "C": "US",
166                 "ST": "North Carolina",
167                 "L": "Durham",
168                 "O": "orderer.example.com",
169                 "CN": "ica.orderer.example.com"
170             },
171             "serial": 1000,
172             "crlnumber": 1000,
```

```
173             "dir_main": "./tmp_cert",
174             "dir_create_crts": "gen-ica_orderer"
175         }
176     ]
177     createOrgs(orgs)
```

**Listagem I.2:** *Script* para criar os Root CA.

```
1  ########################################
2  #   This is a configuration file for the fabric-ca-server command.
3  #
4  #   COMMAND LINE ARGUMENTS AND ENVIRONMENT VARIABLES
5  #   -----------------------------------------------
6  #   Each configuration element can be overridden via command line
7  #   arguments or environment variables.  The precedence for
       determining
8  #   the value of each element is as follows:
9  #   1) command line argument
10 #      Examples:
11 #      a) --port 443
12 #         To set the listening port
13 #      b) --ca.keyfile ../mykey.pem
14 #         To set the "keyfile" element in the "ca" section below;
15 #         note the '.' separator character.
16 #   2) environment variable
17 #      Examples:
18 #      a) FABRIC_CA_SERVER_PORT=443
19 #         To set the listening port
20 #      b) FABRIC_CA_SERVER_CA_KEYFILE="../mykey.pem"
21 #         To set the "keyfile" element in the "ca" section below;
22 #         note the '_' separator character.
23 #   3) configuration file
24 #   4) default value (if there is one)
25 #      All default values are shown beside each element below.
26 #
27 #   FILE NAME ELEMENTS
28 #   ------------------
29 #   The value of all fields whose name ends with "file" or "files" are
30 #   name or names of other files.
31 #   For example, see "tls.certfile" and "tls.clientauth.certfiles".
32 #   The value of each of these fields can be a simple filename, a
33 #   relative path, or an absolute path.  If the value is not an
34 #   absolute path, it is interpretted as being relative to the
       location
35 #   of this configuration file.
36 #
37 ########################################
38
39 # Version of config file
```

```
40 version: 1.2.0
41
42 # Server's listening port (default: 7054)
43 port: 7054
44
45 # Enables debug logging (default: false)
46 debug: false
47
48 # Size limit of an acceptable CRL in bytes (default: 512000)
49 crlsizelimit: 512000
50
51 ##########################################
52 #  TLS section for the server's listening port
53 #
54 #  The following types are supported for client authentication:
       NoClientCert,
55 #  RequestClientCert, RequireAnyClientCert, VerifyClientCertIfGiven,
56 #  and RequireAndVerifyClientCert.
57 #
58 #  Certfiles is a list of root certificate authorities that the server
       uses
59 #  when verifying client certificates.
60 ##########################################
61 tls:
62   # Enable TLS (default: false)
63   enabled: true
64   # TLS for the server's listening port
65   certfile:
66   keyfile:
67   clientauth:
68     type: noclientcert
69     certfiles:
70
71 ##########################################
72 #  The CA section contains information related to the Certificate
       Authority
73 #  including the name of the CA, which should be unique for all
       members
74 #  of a blockchain network.  It also includes the key and certificate
       files
75 #  used when issuing enrollment certificates (ECerts) and transaction
76 #  certificates (TCerts).
77 #  The chainfile (if it exists) contains the certificate chain which
78 #  should be trusted for this CA, where the 1st in the chain is always
       the
79 #  root CA certificate.
80 ##########################################
81 ca:
82   # Name of this CA
```

```
83    name: OrdererCA
84    # Key file (is only used to import a private key into BCCSP)
85    keyfile: /etc/hyperledger/fabric-ca-server/ica.key
86    # Certificate file (default: ca-cert.pem)
87    certfile: /etc/hyperledger/fabric-ca-server/ica.cert
88    # Chain file
89    chainfile: /etc/hyperledger/fabric-ca-server/chain.cert
90
91 ############################################
92 #  The gencrl REST endpoint is used to generate a CRL that contains
       revoked
93 #  certificates. This section contains configuration options that are
       used
94 #  during gencrl request processing.
95 ############################################
96  crl:
97    # Specifies expiration for the generated CRL. The number of hours
98    # specified by this property is added to the UTC time, the resulting
         time
99    # is used to set the 'Next Update' date of the CRL.
100   expiry: 24h
101
102 ############################################
103 #  The registry section controls how the fabric-ca-server does two
       things:
104 #  1) authenticates enrollment requests which contain a username and
       password
105 #     (also known as an enrollment ID and secret).
106 #  2) once authenticated, retrieves the identity's attribute names and
107 #     values which the fabric-ca-server optionally puts into TCerts
108 #     which it issues for transacting on the Hyperledger Fabric
       blockchain.
109 #     These attributes are useful for making access control decisions
       in
110 #     chaincode.
111 #  There are two main configuration options:
112 #  1) The fabric-ca-server is the registry.
113 #     This is true if "ldap.enabled" in the ldap section below is
       false.
114 #  2) An LDAP server is the registry, in which case the fabric-ca-
       server
115 #     calls the LDAP server to perform these tasks.
116 #     This is true if "ldap.enabled" in the ldap section below is true
       ,
117 #     which means this "registry" section is ignored.
118 ############################################
119  registry:
120    # Maximum number of times a password/secret can be reused for
       enrollment
```

```
121   # (default: -1, which means there is no limit)
122   maxenrollments: -1
123
124   # Contains identity information which is used when LDAP is disabled
125   identities:
126      - name: admin
127        pass: adminpw
128        type: client
129        affiliation: ""
130        attrs:
131           hf.Registrar.Roles: "*"
132           hf.Registrar.DelegateRoles: "*"
133           hf.Revoker: true
134           hf.IntermediateCA: true
135           hf.GenCRL: true
136           hf.Registrar.Attributes: "*"
137           hf.AffiliationMgr: true
138
139 ############################################
140 #  Database section
141 #  Supported types are: "sqlite3", "postgres", and "mysql".
142 #  The datasource value depends on the type.
143 #  If the type is "sqlite3", the datasource value is a file name to
      use
144 #  as the database store.  Since "sqlite3" is an embedded database, it
145 #  may not be used if you want to run the fabric-ca-server in a
      cluster.
146 #  To run the fabric-ca-server in a cluster, you must choose "postgres
      "
147 #  or "mysql".
148 ############################################
149 db:
150   type: sqlite3
151   datasource: fabric-ca-server.db
152   tls:
153       enabled: false
154       certfiles:
155       client:
156         certfile:
157         keyfile:
158
159 ############################################
160 #  LDAP section
161 #  If LDAP is enabled, the fabric-ca-server calls LDAP to:
162 #  1) authenticate enrollment ID and secret (i.e. username and
      password)
163 #     for enrollment requests;
164 #  2) To retrieve identity attributes
165 ############################################
```

```
ldap:
   # Enables or disables the LDAP client (default: false)
   # If this is set to true, the "registry" section is ignored.
   enabled: false
   # The URL of the LDAP server
   url: ldap://<adminDN>:<adminPassword>@<host>:<port>/<base>
   # TLS configuration for the client connection to the LDAP server
   tls:
      certfiles:
      client:
         certfile:
         keyfile:
   # Attribute related configuration for mapping from LDAP entries to Fabric CA attributes
   attribute:
      # 'names' is an array of strings containing the LDAP attribute names which are
      # requested from the LDAP server for an LDAP identity's entry
      names: ['uid','member']
      # The 'converters' section is used to convert an LDAP entry to the value of
      # a fabric CA attribute.
      # For example, the following converts an LDAP 'uid' attribute
      # whose value begins with 'revoker' to a fabric CA attribute
      # named "hf.Revoker" with a value of "true" (because the boolean expression
      # evaluates to true).
      #    converters:
      #       - name: hf.Revoker
      #         value: attr("uid") =~ "revoker*"
      converters:
         - name:
           value:
      # The 'maps' section contains named maps which may be referenced by the 'map'
      # function in the 'converters' section to map LDAP responses to arbitrary values.
      # For example, assume a user has an LDAP attribute named 'member' which has multiple
      # values which are each a distinguished name (i.e. a DN). For simplicity, assume the
      # values of the 'member' attribute are 'dn1', 'dn2', and 'dn3'.
      # Further assume the following configuration.
      #    converters:
      #       - name: hf.Registrar.Roles
      #         value: map(attr("member"),"groups")
      #    maps:
      #       groups:
      #          - name: dn1
```

```
207       #             value: peer
208       #           - name: dn2
209       #             value: client
210       # The value of the user's 'hf.Registrar.Roles' attribute is then
          computed to be
211       # "peer,client,dn3".  This is because the value of 'attr("member
          ")' is
212       # "dn1,dn2,dn3", and the call to 'map' with a 2nd argument of
213       # "group" replaces "dn1" with "peer" and "dn2" with "client".
214       maps:
215           groups:
216              - name:
217                value:
218
219 ###########################################
220 # Affiliations section. Fabric CA server can be bootstrapped with the
221 # affiliations specified in this section. Affiliations are specified
      as maps.
222 # For example:
223 #   businessunit1:
224 #     department1:
225 #       - team1
226 #   businessunit2:
227 #     - department2
228 #     - department3
229 #
230 # Affiliations are hierarchical in nature. In the above example,
231 # department1 (used as businessunit1.department1) is the child of
      businessunit1.
232 # team1 (used as businessunit1.department1.team1) is the child of
      department1.
233 # department2 (used as businessunit2.department2) and department3 (
      businessunit2.department3)
234 # are children of businessunit2.
235 # Note: Affiliations are case sensitive except for the non-leaf
      affiliations
236 # (like businessunit1, department1, businessunit2) that are specified
      in the configuration file,
237 # which are always stored in lower case.
238 ###########################################
239 affiliations:
240    org1:
241       - department1
242       - department2
243    org2:
244       - department1
245
246 ###########################################
247 #  Signing section
```

```
248 #
249 #  The "default" subsection is used to sign enrollment certificates;
250 #  the default expiration ("expiry" field) is "8760h", which is 1 year
       in hours.
251 #
252 #  The "ca" profile subsection is used to sign intermediate CA
       certificates;
253 #  the default expiration ("expiry" field) is "43800h" which is 5
       years in hours.
254 #  Note that "isca" is true, meaning that it issues a CA certificate.
255 #  A maxpathlen of 0 means that the intermediate CA cannot issue other
256 #  intermediate CA certificates, though it can still issue end entity
       certificates.
257 #  (See RFC 5280, section 4.2.1.9)
258 #
259 #  The "tls" profile subsection is used to sign TLS certificate
       requests;
260 #  the default expiration ("expiry" field) is "8760h", which is 1 year
       in hours.
261 ########################################
262 signing:
263     default:
264       usage:
265         - digital signature
266       expiry: 8760h
267     profiles:
268       ca:
269          usage:
270            - cert sign
271            - crl sign
272          expiry: 43800h
273          caconstraint:
274            isca: true
275            maxpathlen: 0
276       tls:
277          usage:
278             - signing
279             - key encipherment
280             - server auth
281             - client auth
282             - key agreement
283          expiry: 8760h
284
285 ########################################
286 #  Certificate Signing Request (CSR) section.
287 #  This controls the creation of the root CA certificate.
288 #  The expiration for the root CA certificate is configured with the
289 #  "ca.expiry" field below, whose default value is "131400h" which is
290 #  15 years in hours.
```

```
291 #  The pathlength field is used to limit CA certificate hierarchy as
        described
292 #  in section 4.2.1.9 of RFC 5280.
293 #  Examples:
294 #  1) No pathlength value means no limit is requested.
295 #  2) pathlength == 1 means a limit of 1 is requested which is the
        default for
296 #     a root CA.  This means the root CA can issue intermediate CA
       certificates,
297 #     but these intermediate CAs may not in turn issue other CA
       certificates
298 #     though they can still issue end entity certificates.
299 #  3) pathlength == 0 means a limit of 0 is requested;
300 #     this is the default for an intermediate CA, which means it can
       not issue
301 #     CA certificates though it can still issue end entity
       certificates.
302 ###############################################
303 csr:
304     cn: ca.example.com
305     names:
306         - C: US
307           ST: "New York"
308           L: "New York"
309           O: example.com
310           OU:
311     hosts:
312       - localhost
313       - example.com
314     ca:
315         expiry: 131400h
316         pathlength: 1
317
318 ###############################################
319 # BCCSP (BlockChain Crypto Service Provider) section is used to select
        which
320 # crypto library implementation to use
321 ###############################################
322 bccsp:
323     default: SW
324     sw:
325         hash: SHA2
326         security: 256
327         filekeystore:
328             # The directory used for the software file-based keystore
329             keystore: msp/keystore
330
331 ###############################################
332 # Multi CA section
```

```
333 #
334 # Each Fabric CA server contains one CA by default.  This section is
        used
335 # to configure multiple CAs in a single server.
336 #
337 # 1) --cacount <number-of-CAs>
338 # Automatically generate <number-of-CAs> non-default CAs.  The names
        of these
339 # additional CAs are "ca1", "ca2", ... "caN", where "N" is <number-of-
        CAs>
340 # This is particularly useful in a development environment to quickly
        set up
341 # multiple CAs. Note that, this config option is not applicable to
        intermediate CA server
342 # i.e., Fabric CA server that is started with intermediate.
        parentserver.url config
343 # option (-u command line option)
344 #
345 # 2) --cafiles <CA-config-files>
346 # For each CA config file in the list, generate a separate signing CA.
         Each CA
347 # config file in this list MAY contain all of the same elements as are
         found in
348 # the server config file except port, debug, and tls sections.
349 #
350 # Examples:
351 # fabric-ca-server start -b admin:adminpw --cacount 2
352 #
353 # fabric-ca-server start -b admin:adminpw --cafiles ca/ca1/fabric-ca-
        server-config.yaml
354 # --cafiles ca/ca2/fabric-ca-server-config.yaml
355 #
356 ##########################################
357
358 cacount:
359
360 cafiles:
361
362 ##########################################
363 # Intermediate CA section
364 #
365 # The relationship between servers and CAs is as follows:
366 #   1) A single server process may contain or function as one or more
        CAs.
367 #      This is configured by the "Multi CA section" above.
368 #   2) Each CA is either a root CA or an intermediate CA.
369 #   3) Each intermediate CA has a parent CA which is either a root CA
        or another intermediate CA.
370 #
```

```
371 # This section pertains to configuration of #2 and #3.
372 # If the "intermediate.parentserver.url" property is set,
373 # then this is an intermediate CA with the specified parent
374 # CA.
375 #
376 # parentserver section
377 #    url - The URL of the parent server
378 #    caname - Name of the CA to enroll within the server
379 #
380 # enrollment section used to enroll intermediate CA with parent CA
381 #    profile - Name of the signing profile to use in issuing the
        certificate
382 #    label - Label to use in HSM operations
383 #
384 # tls section for secure socket connection
385 #   certfiles - PEM-encoded list of trusted root certificate files
386 #   client:
387 #     certfile - PEM-encoded certificate file for when client
        authentication
388 #     is enabled on server
389 #     keyfile - PEM-encoded key file for when client authentication
390 #     is enabled on server
391 ###########################################
392 intermediate:
393   parentserver:
394     url:
395     caname:
396
397   enrollment:
398     hosts:
399     profile:
400     label:
401
402   tls:
403     certfiles:
404     client:
405       certfile:
406       keyfile:
```

**Listagem I.3:** Ficheiro de configuração dos CA (exemplo do *orderer*

```
1 # Copyright IBM Corp. All Rights Reserved.
2 #
3 # SPDX-License-Identifier: Apache-2.0
4 #
5
6 version: '2.4'
7
8 networks:
```

```
  test:
    name: fabric_test

services:

  ca_org1:
    image: hyperledger/fabric-ca:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-org1
      - 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:
      - ../organizations/fabric-ca/org1:/etc/hyperledger/fabric-ca-
    server
    container_name: ca_org1
    networks:
      - test

  ca_org2:
    image: hyperledger/fabric-ca:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-org2
      - FABRIC_CA_SERVER_TLS_ENABLED=true
      - FABRIC_CA_SERVER_PORT=8054
    ports:
      - "8054:8054"
    command: sh -c 'fabric-ca-server start -b admin:adminpw -d'
    volumes:
      - ../organizations/fabric-ca/org2:/etc/hyperledger/fabric-ca-
    server
    container_name: ca_org2
    networks:
      - test

  ca_orderer:
    image: hyperledger/fabric-ca:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
```

```
56        - FABRIC_CA_SERVER_CA_NAME=ca-orderer
57        - FABRIC_CA_SERVER_TLS_ENABLED=true
58        - FABRIC_CA_SERVER_PORT=9054
59      ports:
60        - "9054:9054"
61      command: sh -c 'fabric-ca-server start -b admin:adminpw -d'
62      volumes:
63        - ../organizations/fabric-ca/ordererOrg:/etc/hyperledger/fabric-
      ca-server
64      container_name: ca_orderer
65      networks:
66        - test
```

**Listagem I.4:** Ficheiro de criação dos docker CA.

```
1 # Copyright IBM Corp. All Rights Reserved.
2 #
3 # SPDX-License-Identifier: Apache-2.0
4 #
5
6 version: '2.4'
7
8 volumes:
9   orderer.example.com:
10  peer0.org1.example.com:
11  peer0.org2.example.com:
12
13 networks:
14  test:
15    name: fabric_test
16
17 services:
18
19  orderer.example.com:
20    container_name: orderer.example.com
21    image: hyperledger/fabric-orderer:latest
22    labels:
23      service: hyperledger-fabric
24    environment:
25      - FABRIC_LOGGING_SPEC=INFO
26      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
27      - ORDERER_GENERAL_LISTENPORT=7050
28      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
29      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
30      # enabled TLS
31      - ORDERER_GENERAL_TLS_ENABLED=true
32      - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/
      server.key
```

```
33        - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/
      server.crt
34        - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.
      crt]
35        - ORDERER_KAFKA_TOPIC_REPLICATIONFACTOR=1
36        - ORDERER_KAFKA_VERBOSE=true
37        - ORDERER_GENERAL_CLUSTER_CLIENTCERTIFICATE=/var/hyperledger/
      orderer/tls/server.crt
38        - ORDERER_GENERAL_CLUSTER_CLIENTPRIVATEKEY=/var/hyperledger/
      orderer/tls/server.key
39        - ORDERER_GENERAL_CLUSTER_ROOTCAS=[/var/hyperledger/orderer/tls/
      ca.crt]
40        - ORDERER_GENERAL_BOOTSTRAPMETHOD=none
41        - ORDERER_CHANNELPARTICIPATION_ENABLED=true
42        - ORDERER_ADMIN_TLS_ENABLED=true
43        - ORDERER_ADMIN_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/
      server.crt
44        - ORDERER_ADMIN_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/
      server.key
45        - ORDERER_ADMIN_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt
      ]
46        - ORDERER_ADMIN_TLS_CLIENTROOTCAS=[/var/hyperledger/orderer/tls/
      ca.crt]
47        - ORDERER_ADMIN_LISTENADDRESS=0.0.0.0:7053
48      working_dir: /opt/gopath/src/github.com/hyperledger/fabric
49      command: orderer
50      volumes:
51          - ../system-genesis-block/genesis.block:/var/hyperledger/
      orderer/orderer.genesis.block
52          - ../organizations/ordererOrganizations/example.com/orderers/
      orderer.example.com/msp:/var/hyperledger/orderer/msp
53          - ../organizations/ordererOrganizations/example.com/orderers/
      orderer.example.com/tls/:/var/hyperledger/orderer/tls
54          - orderer.example.com:/var/hyperledger/production/orderer
55      ports:
56        - 7050:7050
57        - 7053:7053
58      networks:
59        - test
60
61    peer0.org1.example.com:
62      container_name: peer0.org1.example.com
63      image: hyperledger/fabric-peer:latest
64      labels:
65        service: hyperledger-fabric
66      environment:
67        #Generic peer variables
68        - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
69        - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=fabric_test
```

```
70        - FABRIC_LOGGING_SPEC=INFO
71        #- FABRIC_LOGGING_SPEC=DEBUG
72        - CORE_PEER_TLS_ENABLED=true
73        - CORE_PEER_PROFILE_ENABLED=true
74        - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
75        - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
76        - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
77        # Peer specific variabes
78        - CORE_PEER_ID=peer0.org1.example.com
79        - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
80        - CORE_PEER_LISTENADDRESS=0.0.0.0:7051
81        - CORE_PEER_CHAINCODEADDRESS=peer0.org1.example.com:7052
82        - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
83        - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org1.example.com:7051
84        - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org1.example.com:7051
85        - CORE_PEER_LOCALMSPID=Org1MSP
86      volumes:
87          - /var/run/docker.sock:/host/var/run/docker.sock
88          - ../organizations/peerOrganizations/org1.example.com/peers/
      peer0.org1.example.com/msp:/etc/hyperledger/fabric/msp
89          - ../organizations/peerOrganizations/org1.example.com/peers/
      peer0.org1.example.com/tls:/etc/hyperledger/fabric/tls
90          - peer0.org1.example.com:/var/hyperledger/production
91      working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
92      command: peer node start
93      ports:
94        - 7051:7051
95      networks:
96        - test
97
98    peer0.org2.example.com:
99      container_name: peer0.org2.example.com
100     image: hyperledger/fabric-peer:latest
101     labels:
102       service: hyperledger-fabric
103     environment:
104       #Generic peer variables
105       - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
106       - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=fabric_test
107       - FABRIC_LOGGING_SPEC=INFO
108       #- FABRIC_LOGGING_SPEC=DEBUG
109       - CORE_PEER_TLS_ENABLED=true
110       - CORE_PEER_PROFILE_ENABLED=true
111       - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
112       - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
113       - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
114       # Peer specific variabes
115       - CORE_PEER_ID=peer0.org2.example.com
116       - CORE_PEER_ADDRESS=peer0.org2.example.com:9051
```

```
117       - CORE_PEER_LISTENADDRESS=0.0.0.0:9051
118       - CORE_PEER_CHAINCODEADDRESS=peer0.org2.example.com:9052
119       - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:9052
120       - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org2.example.com:9051
121       - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org2.example.com:9051
122       - CORE_PEER_LOCALMSPID=Org2MSP
123     volumes:
124         - /var/run/docker.sock:/host/var/run/docker.sock
125         - ../organizations/peerOrganizations/org2.example.com/peers/
      peer0.org2.example.com/msp:/etc/hyperledger/fabric/msp
126         - ../organizations/peerOrganizations/org2.example.com/peers/
      peer0.org2.example.com/tls:/etc/hyperledger/fabric/tls
127         - peer0.org2.example.com:/var/hyperledger/production
128     working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
129     command: peer node start
130     ports:
131       - 9051:9051
132     networks:
133       - test
134
135   cli:
136     container_name: cli
137     image: hyperledger/fabric-tools:latest
138     labels:
139       service: hyperledger-fabric
140     tty: true
141     stdin_open: true
142     environment:
143       - GOPATH=/opt/gopath
144       - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
145       - FABRIC_LOGGING_SPEC=INFO
146       #- FABRIC_LOGGING_SPEC=DEBUG
147     working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
148     command: /bin/bash
149     volumes:
150         - /var/run/:/host/var/run/
151         - ../organizations:/opt/gopath/src/github.com/hyperledger/
      fabric/peer/organizations
152         - ../scripts:/opt/gopath/src/github.com/hyperledger/fabric/
      peer/scripts/
153     depends_on:
154       - peer0.org1.example.com
155       - peer0.org2.example.com
156     networks:
157       - test
```

**Listagem I.5:** Ficheiro de criação das Organizações e *peers*

```
1 # Copyright IBM Corp. All Rights Reserved.
```

```
2 #
3 # SPDX-License-Identifier: Apache-2.0
4 #
5
6 version: '2.4'
7
8 networks:
9   test:
10     name: fabric_test
11
12 services:
13   couchdb0:
14     container_name: couchdb0
15     image: couchdb:3.1.1
16     labels:
17       service: hyperledger-fabric
18     # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin
       user and password
19     # for CouchDB. This will prevent CouchDB from operating in an "
       Admin Party" mode.
20     environment:
21       - COUCHDB_USER=admin
22       - COUCHDB_PASSWORD=adminpw
23     # Comment/Uncomment the port mapping if you want to hide/expose
       the CouchDB service,
24     # for example map it to utilize Fauxton User Interface in dev
       environments.
25     ports:
26       - "5984:5984"
27     networks:
28       - test
29
30   peer0.org1.example.com:
31     environment:
32       - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
33       - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb0:5984
34       # The CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME and
      CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD
35       # provide the credentials for ledger to connect to CouchDB. The
       username and password must
36       # match the username and password set for the associated CouchDB
      .
37       - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
38       - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
39     depends_on:
40       - couchdb0
41
42   couchdb1:
43     container_name: couchdb1
```

```
44      image: couchdb:3.1.1
45      labels:
46        service: hyperledger-fabric
47      # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin
        user and password
48      # for CouchDB.  This will prevent CouchDB from operating in an "
        Admin Party" mode.
49      environment:
50        - COUCHDB_USER=admin
51        - COUCHDB_PASSWORD=adminpw
52      # Comment/Uncomment the port mapping if you want to hide/expose
        the CouchDB service,
53      # for example map it to utilize Fauxton User Interface in dev
        environments.
54      ports:
55        - "7984:5984"
56      networks:
57        - test
58
59    peer0.org2.example.com:
60      environment:
61        - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
62        - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb1:5984
63        # The CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME and
       CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD
64        # provide the credentials for ledger to connect to CouchDB.  The
        username and password must
65        # match the username and password set for the associated CouchDB
       .
66        - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
67        - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
68      depends_on:
69        - couchdb1
```

**Listagem I.6:** Ficheiro de criação da base de dados CouchDB para cada *peer*

# Apêndice II

# Contrato Inteligente

```
1  package main
2
3  import (
4   "log"
5
6   "github.com/hyperledger/fabric-contract-api-go/contractapi"
7   "water/chaincode-go/contract/chaincode"
8  )
9
10 func main() {
11  abacSmartContract, err := contractapi.NewChaincode(&chaincode.
      SmartContract{})
12  if err != nil {
13   log.Panicf("Error creating abac chaincode: %v", err)
14  }
15
16  if err := abacSmartContract.Start(); err != nil {
17   log.Panicf("Error starting abac chaincode: %v", err)
18  }
19 }
```

**Listagem II.1:** Main do contrato inteligente.

```
1
2  package chaincode
3
4  import (
5   "encoding/json"
6   "fmt"
7
8   "github.com/hyperledger/fabric-contract-api-go/contractapi"
9  )
10
11 // SmartContract provides functions for managing an Asset
```

```
12 type SmartContract struct {
13  contractapi.Contract
14 }
15
16
17 type Device struct {
18  ID              string `json:"ID"`
19  Type_data       string `json:"type_data"`
20  DeviceId        string `json:"deviceId"`
21  DeviceName      string `json:"deviceName"`
22  CoordX          float64 `json:"coordX"`
23  CoordY          float64 `json:"coordY"`
24 }
25
26
27 // Asset describes basic details of what makes up a simple asset
28 type Asset struct {
29  ID              string `json:"ID"`
30  Type_data       string `json:"type_data"`
31  DeviceId        string `json:"deviceId"`
32  DataCreation    string `json:"dataCreation"`
33  Temperature     string `json:"Temperature"`
34  PH              string `json:"PH"`
35 }
36
37
38 func (s *SmartContract) InitDevice(ctx contractapi.
       TransactionContextInterface) error {
39  devices := []Device{
40   {
41    ID: "D-1", Type_data: "device", DeviceId: "7d7de2b9-5937-49b6-a968-
       a893c918fcab", DeviceName: "water-analysis-1", CoordX: 37.956938 ,
       CoordY: -8.067635,
42   },
43   {
44    ID: "D-2", Type_data: "device", DeviceId: "43cd2f42-7859-448a-b43a
       -938bd66bada7", DeviceName: "water-analysis-2", CoordX: 37.962100 ,
        CoordY: -8.067820,
45   },
46  }
47
48  for _, device := range devices {
49   deviceJSON, err := json.Marshal(device)
50   if err != nil {
51    return err
52   }
53
54   err = ctx.GetStub().PutState(device.ID, deviceJSON)
55   if err != nil {
```

```
56      return fmt.Errorf("failed to put to world state. %v", err)
57     }
58    }
59
60    return nil
61  }
62
63
64  // GET ALL DEVICES
65  func (s *SmartContract) GetAllDevices(ctx contractapi.
        TransactionContextInterface) ([]*Device, error) {
66
67    queryString := fmt.Sprintf(`{"selector":{"type_data":"device"}}`)
68
69    resultsIterator, err := ctx.GetStub().GetQueryResult(queryString)
70
71    if err != nil {
72     return nil, err
73    }
74    defer resultsIterator.Close()
75
76    var devices []*Device
77    for resultsIterator.HasNext() {
78     queryResponse, err := resultsIterator.Next()
79     if err != nil {
80      return nil, err
81     }
82
83     var device Device
84     err = json.Unmarshal(queryResponse.Value, &device)
85     if err != nil {
86      return nil, err
87     }
88     devices = append(devices, &device)
89    }
90
91    return devices, nil
92  }
93
94
95  //////////// DATA  Water
96
97  func (s *SmartContract) DeviceIdExists(ctx contractapi.
        TransactionContextInterface, deviceId string) (bool, error) {
98    assetJSON, err := ctx.GetStub().GetState(deviceId)
99    if err != nil {
100    return false, fmt.Errorf("failed to read from world state: %v", err)
101   }
102
```

```
103  return assetJSON != nil, nil
104 }
105
106
107
108 // // DeviceExists returns true when asset with given ID exists in
        world state
109 // func (s *SmartContract) DeviceExists(ctx contractapi.
        TransactionContextInterface, deviceId string) (bool, error) {
110 //  queryString := fmt.Sprintf(`{"selector":{"type_data":"device","
        deviceId":"%s"}}`, deviceId)
111 //  deviceJSON, err := ctx.GetStub().GetQueryResult(queryString)
112 //  if err != nil {
113 //   return false, fmt.Errorf("failed to read from world state: %v",
        err)
114 //  }
115 //  fmt.Print(deviceJSON)
116 //  return deviceJSON != nil, nil
117 // }
118
119
120
121 // Create Device
122 func (s *SmartContract) CreateDevice(ctx contractapi.
        TransactionContextInterface, id string, deviceId string,
123       deviceName string, coordX float64, coordY float64) error {
124
125  exists, err := s.DeviceIdExists(ctx, id)
126  if err != nil {
127   return err
128  }
129
130  if exists{
131   return fmt.Errorf("the asset %s already exists", id)
132  }
133
134
135  asset := Device{
136   ID:              id,
137   Type_data:       "device",
138   DeviceId:        deviceId,
139   DeviceName:     deviceName,
140   CoordX:        coordX,
141   CoordY:          coordY,
142  }
143
144  assetJSON, err := json.Marshal(asset)
145  if err != nil {
146   return err
```

```
147  }
148  return ctx.GetStub().PutState(id, assetJSON)
149 }
150
151
152
153
154
155 // Create data water
156 func (s *SmartContract) CreateData(ctx contractapi.
        TransactionContextInterface, id string, deviceId string,
157             dataCreation string, temperature string, ph string) error {
158
159  exists, err := s.DeviceIdExists(ctx, deviceId)
160  if err != nil {
161   return err
162  }
163
164  if !exists{
165   return fmt.Errorf("the device %s not exists", deviceId)
166  }
167
168  if exists {
169   exists_id, err := s.AssetExists(ctx, id)
170   if err != nil {
171    return err
172   }
173   if exists_id {
174    return fmt.Errorf("the asset %s already exists", id)
175   }
176
177   asset := Asset{
178    ID:              id,
179    Type_data:       "data",
180    DeviceId:        deviceId,
181    DataCreation:    dataCreation,
182    Temperature:     temperature,
183    PH:              ph,
184   }
185
186   assetJSON, err := json.Marshal(asset)
187   if err != nil {
188    return err
189   }
190   return ctx.GetStub().PutState(id, assetJSON)
191  }
192
193  return fmt.Errorf("the device %s not exist11111111111111111111s", id
       )
```

```
194 }
195
196
197
198 // GET ALL DATA TO DEVICE
199 func (s *SmartContract) GetAllDataToDevice(ctx contractapi.
        TransactionContextInterface, deviceId string) ([]*Asset, error) {
200  queryString := fmt.Sprintf(`{"selector":{"type_data":"data", "
        deviceId":"%s"}}`, deviceId)
201
202  resultsIterator, err := ctx.GetStub().GetQueryResult(queryString)
203
204  if err != nil {
205   return nil, err
206  }
207  defer resultsIterator.Close()
208
209  var assets []*Asset
210  for resultsIterator.HasNext() {
211   queryResponse, err := resultsIterator.Next()
212   if err != nil {
213    return nil, err
214   }
215
216   var asset Asset
217   err = json.Unmarshal(queryResponse.Value, &asset)
218   if err != nil {
219    return nil, err
220   }
221   assets = append(assets, &asset)
222  }
223
224  return assets, nil
225
226 }
227
228
229
230 // // InitLedger adds a base set of assets to the ledger
231 // func (s *SmartContract) InitLedger(ctx contractapi.
        TransactionContextInterface) error {
232 //  assets := []Asset{
233 //   {
234 //    ID: "1", DeviceId: "7d7de2b9-5937-49b6-a968-a893c918fcab",
        DeviceName: "water-analysis-1",
235 //    DataCreation: "2021-09-01T10:00:00", Temperature: "20", PH: "9",
236 //    CoordX: 37.956938 , CoordY: -8.067635,
237 //   },
238 //   {
```

```
239 //     ID: "2", DeviceId: "7d7de2b9-5937-49b6-a968-a893c918fcab",
        DeviceName: "water-analysis-1",
240 //     DataCreation: "2021-09-01T10:05:00", Temperature: "22", PH: "7",
241 //     CoordX: 37.956938 , CoordY: -8.067635,
242 //    },
243 //    {
244 //     ID: "3", DeviceId: "43cd2f42-7859-448a-b43a-938bd66bada7",
        DeviceName: "water-analysis-2",
245 //     DataCreation: "2021-09-01T10:00:00", Temperature: "21", PH: "5",
246 //     CoordX: 37.962100 , CoordY: -8.067820,
247 //    },
248 //  }
249
250 //  for _, asset := range assets {
251 //   assetJSON, err := json.Marshal(asset)
252 //   if err != nil {
253 //    return err
254 //   }
255
256 //   err = ctx.GetStub().PutState(asset.ID, assetJSON)
257 //   if err != nil {
258 //    return fmt.Errorf("failed to put to world state. %v", err)
259 //   }
260 //  }
261
262 //  return nil
263 // }
264
265 // // CreateAsset issues a new asset to the world state with given
        details.
266 // func (s *SmartContract) CreateAsset(ctx contractapi.
        TransactionContextInterface, id string, deviceId string,
267 //          deviceName string, dataCreation string, temperature string
        , ph string,
268 //          coordX float64, coordY float64) error {
269 //  exists, err := s.AssetExists(ctx, id)
270 //  if err != nil {
271 //   return err
272 //  }
273 //  if exists {
274 //   return fmt.Errorf("the asset %s already exists", id)
275 //  }
276
277 //  asset := Asset{
278 //   ID:             id,
279 //   DeviceId:       deviceId,
280 //   DeviceName:     deviceName,
281 //   DataCreation:   dataCreation,
282 //   Temperature:    temperature,
```

```
283 //    PH:               ph,
284 //    CoordX:           coordX,
285 //    CoordY:           coordY,
286 //   }
287 //   assetJSON, err := json.Marshal(asset)
288 //   if err != nil {
289 //    return err
290 //   }
291
292 //   return ctx.GetStub().PutState(id, assetJSON)
293 // }
294
295 // // ReadAsset returns the asset stored in the world state with given
        id.
296 // func (s *SmartContract) ReadAsset(ctx contractapi.
      TransactionContextInterface, id string) (*Asset, error) {
297 //   assetJSON, err := ctx.GetStub().GetState(id)
298 //   if err != nil {
299 //    return nil, fmt.Errorf("failed to read from world state: %v", err
      )
300 //   }
301 //   if assetJSON == nil {
302 //    return nil, fmt.Errorf("the asset %s does not exist", id)
303 //   }
304
305 //   var asset Asset
306 //   err = json.Unmarshal(assetJSON, &asset)
307 //   if err != nil {
308 //    return nil, err
309 //   }
310
311 //   return &asset, nil
312 // }
313
314 // // // UpdateAsset updates an existing asset in the world state with
        provided parameters.
315 // // func (s *SmartContract) UpdateAsset(ctx contractapi.
      TransactionContextInterface, id string, temperature string, ph
      string) error {
316 // //   exists, err := s.AssetExists(ctx, id)
317 // //   if err != nil {
318 // //    return err
319 // //   }
320 // //   if !exists {
321 // //    return fmt.Errorf("the asset %s does not exist", id)
322 // //   }
323
324 // //   // overwriting original asset with new asset
325 // //   asset := Asset{
```

```
326 // //    ID:              id,
327 // //    Temperature:     temperature,
328 // //    PH:              ph,
329 // //   }
330 // //   assetJSON, err := json.Marshal(asset)
331 // //   if err != nil {
332 // //    return err
333 // //   }
334
335 // //   return ctx.GetStub().PutState(id, assetJSON)
336 // // }
337
338 // // DeleteAsset deletes an given asset from the world state.
339 // func (s *SmartContract) DeleteAsset(ctx contractapi.
        TransactionContextInterface, id string) error {
340 //   exists, err := s.AssetExists(ctx, id)
341 //   if err != nil {
342 //    return err
343 //   }
344 //   if !exists {
345 //    return fmt.Errorf("the asset %s does not exist", id)
346 //   }
347
348 //   return ctx.GetStub().DelState(id)
349 // }
350
351
352 // AssetExists returns true when asset with given ID exists in world
        state
353 func (s *SmartContract) AssetExists(ctx contractapi.
        TransactionContextInterface, id string) (bool, error) {
354  assetJSON, err := ctx.GetStub().GetState(id)
355  if err != nil {
356   return false, fmt.Errorf("failed to read from world state: %v", err)
357  }
358
359  return assetJSON != nil, nil
360 }
361
362 // // // // TransferAsset updates the owner field of asset with given
        id in world state.
363 // // // func (s *SmartContract) TransferAsset(ctx contractapi.
        TransactionContextInterface, id string, newOwner string) error {
364 // // //   asset, err := s.ReadAsset(ctx, id)
365 // // //   if err != nil {
366 // // //    return err
367 // // //   }
368
369 // // //   asset.Owner = newOwner
```

```
370 // // //  assetJSON, err := json.Marshal(asset)
371 // // //  if err != nil {
372 // // //   return err
373 // // //  }
374
375 // // //  return ctx.GetStub().PutState(id, assetJSON)
376 // // // }
377
378 // // GetAllAssets returns all assets found in world state
379 // func (s *SmartContract) GetAllAssets(ctx contractapi.
        TransactionContextInterface) ([]*Asset, error) {
380 //  // range query with empty string for startKey and endKey does an
381 //  // open-ended query of all assets in the chaincode namespace.
382 //  resultsIterator, err := ctx.GetStub().GetStateByRange("", "")
383 //  if err != nil {
384 //   return nil, err
385 //  }
386 //  defer resultsIterator.Close()
387
388 //  var assets []*Asset
389 //  for resultsIterator.HasNext() {
390 //   queryResponse, err := resultsIterator.Next()
391 //   if err != nil {
392 //    return nil, err
393 //   }
394
395 //   var asset Asset
396 //   err = json.Unmarshal(queryResponse.Value, &asset)
397 //   if err != nil {
398 //    return nil, err
399 //   }
400 //   assets = append(assets, &asset)
401 //  }
402
403 //  return assets, nil
404 // }
```

**Listagem II.2:** Contrato inteligente.

# Apêndice III

# Cliente Fabric

```
1  /*
2   * Copyright IBM Corp. All Rights Reserved.
3   *
4   * SPDX-License-Identifier: Apache-2.0
5   */
6
7  'use strict';
8
9  const fs = require('fs');
10 const path = require('path');
11
12 exports.buildCCPOrg1 = () => {
13  // load the common connection configuration file
14  //const ccpPath = path.resolve(__dirname, '..', '..', 'test-network',
        'organizations', 'peerOrganizations', 'org1.example.com', '
       connection-org1.json');
15  //const ccpPath = path.resolve(__dirname, '..', '..', 'test-network-
       ica', 'organizations', 'peerOrganizations', 'org1.example.com', '
       connection-org1.json');
16  const ccpPath = path.resolve(__dirname, '..', '..', 'test-network-ica
       ', 'organizations', 'peerOrganizations', 'org1.example.com', '
       connection-org1.json');
17  const fileExists = fs.existsSync(ccpPath);
18  if (!fileExists) {
19   throw new Error(`no such file or directory: ${ccpPath}`);
20  }
21  const contents = fs.readFileSync(ccpPath, 'utf8');
22
23  // build a JSON object from the file contents
24  const ccp = JSON.parse(contents);
25
26  console.log(`Loaded the network configuration located at ${ccpPath}`)
       ;
27  return ccp;
```

```
28 };
29
30 exports.buildCCPOrg2 = () => {
31  // load the common connection configuration file
32  const ccpPath = path.resolve(__dirname, '..', '..', 'test-network-ica
       ',
33   'organizations', 'peerOrganizations', 'org2.example.com', '
       connection-org2.json');
34  const fileExists = fs.existsSync(ccpPath);
35  if (!fileExists) {
36   throw new Error(`no such file or directory: ${ccpPath}`);
37  }
38  const contents = fs.readFileSync(ccpPath, 'utf8');
39
40  // build a JSON object from the file contents
41  const ccp = JSON.parse(contents);
42
43  console.log(`Loaded the network configuration located at ${ccpPath}`)
       ;
44  return ccp;
45 };
46
47 exports.buildWallet = async (Wallets, walletPath) => {
48  // Create a new  wallet : Note that wallet is for managing identities
       .
49  let wallet;
50  if (walletPath) {
51   wallet = await Wallets.newFileSystemWallet(walletPath);
52   console.log(`Built a file system wallet at ${walletPath}`);
53  } else {
54   wallet = await Wallets.newInMemoryWallet();
55   console.log('Built an in memory wallet');
56  }
57
58  return wallet;
59 };
60
61 exports.prettyJSONString = (inputString) => {
62  if (inputString) {
63    return JSON.stringify(JSON.parse(inputString), null, 2);
64  }
65  else {
66    return inputString;
67  }
68 }
```

**Listagem III.1:** Ficheiro auxiliar para aplicação.

```
1 /*
```

```
 * Copyright IBM Corp. All Rights Reserved.
 *
 * SPDX-License-Identifier: Apache-2.0
 */

'use strict';

const adminUserId = 'admin';
const adminUserPasswd = 'adminpw';

/**
 *
 * @param {*} FabricCAServices
 * @param {*} ccp
 */
exports.buildCAClient = (FabricCAServices, ccp, caHostName) => {
 // Create a new CA client for interacting with the CA.
 const caInfo = ccp.certificateAuthorities[caHostName]; //lookup CA
    details from config
 const caTLSCACerts = caInfo.tlsCACerts.pem;
 const caClient = new FabricCAServices(caInfo.url, { trustedRoots:
    caTLSCACerts, verify: false }, caInfo.caName);

 console.log(`Built a CA Client named ${caInfo.caName}`);
 return caClient;
};

exports.enrollAdmin = async (caClient, wallet, orgMspId) => {
 try {
  // Check to see if we've already enrolled the admin user.
  const identity = await wallet.get(adminUserId);
  if (identity) {
   console.log('An identity for the admin user already exists in the
    wallet');
   return;
  }

  // Enroll the admin user, and import the new identity into the
    wallet.
  const enrollment = await caClient.enroll({ enrollmentID: adminUserId
    , enrollmentSecret: adminUserPasswd });
  const x509Identity = {
   credentials: {
    certificate: enrollment.certificate,
    privateKey: enrollment.key.toBytes(),
   },
   mspId: orgMspId,
   type: 'X.509',
  };
```

```
46    await wallet.put(adminUserId, x509Identity);
47    console.log('Successfully enrolled admin user and imported it into
        the wallet');
48   } catch (error) {
49    console.error(`Failed to enroll admin user : ${error}`);
50   }
51  };
52
53  exports.registerAndEnrollUser = async (caClient, wallet, orgMspId,
        userId, affiliation) => {
54   try {
55    // Check to see if we've already enrolled the user
56    const userIdentity = await wallet.get(userId);
57    if (userIdentity) {
58     console.log(`An identity for the user ${userId} already exists in
        the wallet`);
59     return;
60    }
61
62    // Must use an admin to register a new user
63    const adminIdentity = await wallet.get(adminUserId);
64    if (!adminIdentity) {
65     console.log('An identity for the admin user does not exist in the
        wallet');
66     console.log('Enroll the admin user before retrying');
67     return;
68    }
69
70    // build a user object for authenticating with the CA
71    const provider = wallet.getProviderRegistry().getProvider(
        adminIdentity.type);
72    const adminUser = await provider.getUserContext(adminIdentity,
        adminUserId);
73
74    // Register the user, enroll the user, and import the new identity
        into the wallet.
75    // if affiliation is specified by client, the affiliation value must
         be configured in CA
76    const secret = await caClient.register({
77     affiliation: affiliation,
78     enrollmentID: userId,
79     role: 'client'
80    }, adminUser);
81    const enrollment = await caClient.enroll({
82     enrollmentID: userId,
83     enrollmentSecret: secret
84    });
85    const x509Identity = {
86     credentials: {
```

```
87      certificate: enrollment.certificate,
88      privateKey: enrollment.key.toBytes(),
89     },
90     mspId: orgMspId,
91     type: 'X.509',
92    };
93    await wallet.put(userId, x509Identity);
94    console.log(`Successfully registered and enrolled user ${userId} and
        imported it into the wallet`);
95   } catch (error) {
96    console.error(`Failed to register user : ${error}`);
97   }
98 };
```

**Listagem III.2:** Ficheiro auxiliar para aplicação comunicar com os CA.

```
1
2
3
4  const express = require('express');
5  var cors = require('cors')
6  var app = express()
7
8  app.use(cors())
9
10 'use strict';
11
12 const { Gateway, Wallets } = require('fabric-network');
13 const FabricCAServices = require('fabric-ca-client');
14 const path = require('path');
15 const { buildCAClient, registerAndEnrollUser, enrollAdmin } = require(
       '../../test-application/javascript/CAUtil.js');
16 const { buildCCPOrg1, buildWallet } = require('../../test-application/
       javascript/AppUtil.js');
17
18 const channelName = 'mychannel';
19 const chaincodeName = 'basic';
20 const mspOrg1 = 'Org1MSP';
21 const walletPath = path.join(__dirname, 'wallet');
22 const org1UserId = 'Auser100';
23
24
25
26 function prettyJSONString(inputString) {
27  return JSON.parse(inputString);
28 }
29
30
31
```

```
32 app.get('/getAllDevices', async (req, res) => {
33     try {
34   // build an in memory object with the network configuration (also
       known as a connection profile)
35   const ccp = buildCCPOrg1();
36   // build an instance of the fabric ca services client based on
37   // the information in the network configuration
38   const caClient = buildCAClient(FabricCAServices, ccp, 'ca.org1.
       example.com');
39   // setup the wallet to hold the credentials of the application user
40   const wallet = await buildWallet(Wallets, walletPath);
41   // in a real application this would be done on an administrative
       flow, and only once
42   await enrollAdmin(caClient, wallet, mspOrg1);
43   // in a real application this would be done only when a new user was
        required to be added
44   // and would be part of an administrative flow
45   await registerAndEnrollUser(caClient, wallet, mspOrg1, org1UserId, '
       org1.department1');
46   // Create a new gateway instance for interacting with the fabric
       network.
47   // In a real application this would be done as the backend server
       session is setup for
48   // a user that has been verified.
49   const gateway = new Gateway();
50
51   try {
52    // setup the gateway instance
53    // The user will now be able to create connections to the fabric
       network and be able to
54    // submit transactions and query. All transactions submitted by
       this gateway will be
55    // signed by this user using the credentials stored in the wallet.
56    await gateway.connect(ccp, {
57     wallet,
58     identity: org1UserId,
59     discovery: { enabled: true, asLocalhost: true } // using
       asLocalhost as this gateway is using a fabric network deployed
       locally
60    });
61
62    // Build a network instance based on the channel where the smart
       contract is deployed
63    const network = await gateway.getNetwork(channelName);
64
65    // Get the contract from the network.
66    const contract = network.getContract(chaincodeName);
67
68
```

```
69      // Let's try a query type operation (function).
70      // This will be sent to just one peer and the results will be shown
        .
71      //console.log('\n--> Evaluate Transaction: GetAllAssets, function
        returns all the current assets on the ledger');
72      let result = await contract.evaluateTransaction('GetAllDevices');
73      //console.log(`*** Result: ${prettyJSONString(result.toString())}`)
        ;
74              res.json(prettyJSONString(result.toString()));
75
76
77    } finally {
78      // Disconnect from the gateway when the application is closing
79      // This will close all connections to the network
80      gateway.disconnect();
81    }
82  } catch (error) {
83    console.error(`******** FAILED to run the application: ${error}`);
84  }
85
86 });
87
88
89 app.get('/getDataToDevice/:deviceID', async (req, res) => {
90      try {
91
92    var deviceID = req.params.deviceID
93    // build an in memory object with the network configuration (also
        known as a connection profile)
94    const ccp = buildCCPOrg1();
95    // build an instance of the fabric ca services client based on
96    // the information in the network configuration
97    const caClient = buildCAClient(FabricCAServices, ccp, 'ca.org1.
        example.com');
98    // setup the wallet to hold the credentials of the application user
99    const wallet = await buildWallet(Wallets, walletPath);
100   // in a real application this would be done on an administrative
        flow, and only once
101   //await enrollAdmin(caClient, wallet, mspOrg1);
102
103   // in a real application this would be done only when a new user was
        required to be added
104   // and would be part of an administrative flow
105   //await registerAndEnrollUser(caClient, wallet, mspOrg1, org1UserId,
        'org1.department1');
106
107   // Create a new gateway instance for interacting with the fabric
        network.
108   // In a real application this would be done as the backend server
```

```
    session is setup for
109  // a user that has been verified.
110  const gateway = new Gateway();
111
112  try {
113   // setup the gateway instance
114   // The user will now be able to create connections to the fabric
      network and be able to
115   // submit transactions and query. All transactions submitted by
      this gateway will be
116   // signed by this user using the credentials stored in the wallet.
117   await gateway.connect(ccp, {
118    wallet,
119    identity: org1UserId,
120    discovery: { enabled: true, asLocalhost: true } // using
      asLocalhost as this gateway is using a fabric network deployed
      locally
121   });
122
123   // Build a network instance based on the channel where the smart
      contract is deployed
124   const network = await gateway.getNetwork(channelName);
125
126   // Get the contract from the network.
127   const contract = network.getContract(chaincodeName);
128
129
130   // Let's try a query type operation (function).
131   // This will be sent to just one peer and the results will be shown
      .
132   //console.log('\n--> Evaluate Transaction: GetAllAssets, function
      returns all the current assets on the ledger');
133   let result = await contract.evaluateTransaction('GetAllDataToDevice
      ', deviceID);
134
135   if (`${result}` !== '') {
136    console.log(`*** Result: ${prettyJSONString(result.toString())}`);
137    res.json(prettyJSONString(result.toString()));
138   }
139   else{
140    console.log("EEEEEEE")
141    res.json([]);
142   }
143
144  } finally {
145   // Disconnect from the gateway when the application is closing
146   // This will close all connections to the network
147   gateway.disconnect();
148  }
```

```
149  } catch (error) {
150    console.error(`******** FAILED to run the application: ${error}`);
151  }
152
153 });
154
155 // Listen to the App Engine-specified port, or 8080 otherwise
156 const PORT = process.env.PORT || 8080;
157 app.listen(PORT, () => {
158   console.log(`Server listening on port ${PORT}...`);
159 });
```

**Listagem III.3:** Código do Cliente Fabric.