UNIVERSIDADE D
COIMBRA

João Nuno Videira Rodrigues

Novos Modelos de Privacidade para a Internet das Coisas

Dissertação no âmbito do Mestrado Integrado de Engenharia Eletrotécnica e de Computadores orientado pelo Professor Doutor Jorge Miguel Sá Silva e apresentada ao departamento de Engenharia Eletrotécnica.

Junho de 2022

Agradecimentos

A presente dissertação só foi possível graças à cooperação e apoio de várias pessoas, às quais este espaço se dedica.

Primeiramente, gostaria de agradecer aos meus Pais e irmão pelo apoio, paciência e compreensão que demonstraram no decorrer deste longo trajeto. Acima de tudo quero agradecer o esforço feito para que tivesse a oportunidade de continuar o meu percurso.

Seguidamente, agradeço ao meu orientador Professor Jorge Miguel Sá Silva, pela oportunidade de poder participar neste projeto e pelo apoio demonstrado ao longo deste trajeto.

Posteriormente, agradeço ao meu orientador oficioso Mestre Jorge Eduardo Rivadeneira Muñoz, pelo constante apoio e acompanhamento durante toda a dissertação.

Finalmente, agradeço ao Professor José Ernesto Jiménez Merino pela sua ajuda com *Ethereum*.

Abstract

Nowadays with the increasing use of *Internet of Things* (IoT) devices, there are more vulnerabilities, many of them are unknown. All this amplifies the importance and need for security systems. Taking that into account, the topic of security and privacy in IoT systems is very prevailing and interesting.

This dissertation goes through the implementation and integration of innovative privacy and security mechanisms to be tested in the ISABELA and BATINA systems.

For this, it was necessary to acquire knowledge related to the underlying technologies used in the systems. Additionally, it was important to understand the technologies used to implement the security mechanisms, the state of the art, and the ISABELA and BATINA systems.

In conclusion, with this knowledge, it was possible to start the implementation of privacy mechanisms using blockchain technology

Resumo

Atualmente com o aumento do uso de dispositivos IoT, há cada vez mais vulnerabilidades, sendo muitas destas desconhecidas. Tudo isto amplifica a importância e a necessidade de sistemas de segurança, vindo daí a escolha do tema da segurança e privacidade em sistemas IoT.

Esta dissertação passa pela implementação e integração de mecanismos de privacidade e segurança inovadores que serão posteriormente testados nos sistemas ISABELA e BATINA.

Para isso foi necessário adquirir conhecimentos relativos a tecnologias subjacentes aos sistemas. Adicionalmente, foi imperativo o conhecimento das tecnologias usadas para implementar os mecanismos de segurança, do estado de arte e dos sistemas ISABELA e BATINA.

Concluindo, com a obtenção destes conhecimentos foi possível iniciar a implementação de mecanismos de privacidade fazendo uso da tecnologia de *blockchain*.

Índice

Índice de Figuras

Abreviaturas

AI *Artificial Intelligence*

BD *Base de Dados*

CA *Certificate Authority*

DC *Data Controllers*

GE *Generic Enabler*

HF *Hyperledger Fabric*

IoT *Internet of Things*

OS *Operating Systems*

QRRH *Query, Requests and Responses Handler*

SC *Smart Contract*

VM *Virtual Machine*

1 Introdução

1.1 Contexto

Como foi mencionado anteriormente, a IoT é cada vez mais usada, normalmente na aquisição de dados, sendo a transmissão desses dados um dos focos de interesse relativamente à segurança e privacidade.

Muito se fala na perda da privacidade, mas a cedência de dados não se faz, normalmente, sem a autorização do utilizador. Para isso usam-se formulários para apresentar os consentimentos do utilizador, formulários estes que são potenciais vulnerabilidades nos sistemas, sendo necessária a proteção da informação dos consentimentos para evitar fugas de informação e a perda de dados para entidades desconhecidas.

1.2 Objetivos

No contexto desta temática, este trabalho tem como objetivo a implementação de mecanismos de privacidade na plataforma apresentada em [1] denominada ISABELA (plataforma esta que será descrita com mais detalhe à frente), de modo a fazer a proteção dos dados dos utilizadores, através do controlo de quais entidades têm permissão de acesso aos dados dos utilizadores. Para isso ir-se-á utilizar *Hyperledger Fabric* (HF) tal como foi definido no projeto proposto. Adicionalmente pretende-se fazer uma breve introdução e implementação da possibilidade de fazer o mesmo usando *Ethereum*.

1.3 Estrutura do Documento

Este documento está dividido em seis partes, incluindo este primeiro capítulo de introdução ao trabalho. Partes estas que serão descritas de seguida:

- No capítulo 2 será feita uma descrição da metodologia, onde se explicita o grupo de trabalho, as tecnologias a serem utilizadas, o planeamento e uma análise comparativa das diferentes tecnologias existentes atualmente;
- No capítulo 3 analisar-se o estado da arte existente para dar a conhecer como outros grupos e plataformas utilizam as tecnologias no ensino;
- No capítulo 4 fala-se de segurança e privacidade, onde o foco estará na descrição da segurança da plataforma ISABELA e arquitetura do projeto;
- No capítulo 5 abordam-se todas as implementações realizadas no decorrer da dissertação. Onde se utilizaram as plataformas *Xamarin*, HF e *Ethereum*. O foco irá recair maioritariamente em HF e *Ethereum*, onde se vão dar a conhecer todas as informações necessárias para replicar as implementações. Adicionalmente estão registados alguns dos muitos problemas encontrados ao criar as implementações e como estes foram ultrapassados. No final faz-se uma avaliação comparativa das plataformas HF e *Ethereum*.
- No capítulo 6 refere-se o trabalho realizado (utilizando um gráfico de Gant) e possíveis desenvolvimentos a realizar no futuro.

2 Metodologia

2.1 Grupo de Trabalho

O grupo de trabalho em que me insiro é constituído por diversos alunos, investigadores e professores do Departamento de Engenharia Eletrotécnica e Computadores e do Departamento de Engenharia Informática. Todos trabalham nas áreas das Redes de Comunicação e da Internet das Coisas.

Este grupo de investigação tem desenvolvido o sistema ISABELA e o sistema BATINA com o objetivo de suportar métodos inovadores de aprendizagem, tirando partido das novas tecnologias nas áreas do IoT, da programação de telemóveis e da Inteligência Artificial.

2.2 Tecnologias usadas

Na implementação da dissertação existem diferentes plataformas usadas com as quais foi necessária a familiarização, incluindo as que foram utilizadas nos projetos ISABELA e BATINA, que irão ser descritas de seguida.

2.2.1 Android

Desde o lançamento do primeiro *smartphone* em 1992 até hoje, os telemóveis evoluíram bastante, tendo o seu mercado passado por diversas fases até chegar aos dias de hoje, onde existem essencialmente dois grandes Sistemas Operativos (*Operating Systems* – OS): Android e iOS.

Para esta dissertação, devido a existirem componentes já implementadas em Android nas plataformas ISABELA e BATINA, será essa a plataforma onde irá estar o foco. Apesar disto, dispositivos iOS também serão compatíveis.

2.2.2 Dash

Devido à necessidade de passar os dados recolhidos para um *dashboard* fez-se uso da plataforma *Dash* [2], que possibilitou colocar os dados reunidos, de maneira simples, à disposição do utilizador.

2.2.3 Xamarin

Devido à existência de dois OS dominantes, programadores de aplicações móveis tinham o problema de ser necessário implementar código para ambos. Para resolver essa dificuldade foram criadas diversas plataformas que produzem implementações para ambos OS de uma só vez, como por exemplo [3]:

- *Xamarin* - Criado pela *Microsoft*, utiliza linguagem C#, apresentando algumas limitações na migração para outras *frameworks* e com bibliotecas desatualizadas;
- *React Native* - Criado pelo *Facebook*, é necessário utilizar *JavaScript + React Native SDK, Kotlin + Android SDK e Swift + iOS SDK*. Adicionalmente é de difícil migração para outros *frameworks;*
- *Flutter* – Criado pelo *Google*, utiliza a linguagem *Dart*, criada pelo próprio Google, sendo necessário aprendê-la de raiz. Adicionalmente é importante ter alguns conhecimentos de iOS / Android, devido à possibilidade de uso de bibliotecas nativas. Em termos de migração para outras *frameworks* é necessário implementar tudo de raiz;
- *Kotplin Multiplatform* – Criado pela *JetBrains*, é uma ferramenta de partilha opcional de código entre plataformas, podendo-se escolher com grande flexibilidade o que partilhar. Esta ferramenta destina-se a *developers* de aplicações moveis que tenham uma grande compreensão de *Kotlin* (Android) ou *Swift* (iOS).

Para os projetos ISABELA e BATINA foi escolhido *Xamarin* devido à utilização de *C#*, que desta maneira oferece a possibilidade de aceder através *dessa* linguagem aos *Software Development Kits* e serviços dos OS, tanto Android como iOS através de *Xamarin.Android* e *Xamarin.iOS*, respetivamente. Para além disso ambas as *frameworks* permitem interagir com as *Base Class Libraries* de .NET. Usando a *framework Xamarin.Forms* é também possível criar *User Interfaces*, de forma bastante simples, apenas uma vez para ambos os OS.

Foi então necessária a familiarização com a plataforma *Xamarin*.

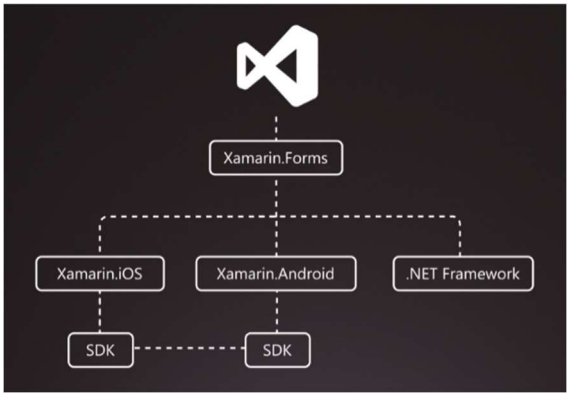


Figura 1 - Esquema do Xamarin [4]

2.2.4 Hyperledger Fabric

Devido à necessidade de fazer transações impossíveis de modificar, em 2008 foi criado o *Bitcoin* [5]. Com esta moeda digital passou a ser possível fazer transações *Peer to Peer* sem qualquer intervenção de bancos. Isto foi possível devido ao uso de criptografia nos nós da rede, de modo a verificar as transações e o consequente registo num *distributed ledger* público, ao qual se chamou *blockchain*.

Com o decorrer do tempo, apesar do *Bitcoin* e outras cripto-moedas serem bastante úteis, o facto de qualquer pessoa poder entrar e ter acesso anónimo era um grande obstáculo para a sua introdução no mundo empresarial, onde era fulcral criar ambientes com entrada controlada de utilizadores, sendo estes identificáveis. Para resolver esse problema foi criado, entre outros, o HF e passou a ser possível identificar e escolher os participantes de uma rede de *blockchain* privada, em que organizações participam e podem criar canais usando configurações de canais. Passando, deste modo, a ser impossível organizações na rede que não pertençam ao canal acederem à informação partilhada no canal.

A estrutura do HF é composta por organizações (onde é necessário usar *Certificate Authorities* [CAs] na sua génese, que emitem certificados digitais, confirmam a identidade do *owner* e fornece provas que o certificado é válido*)*, que têm *peers* que, quando pertencem a um ou mais canais, têm um ou mais *ledgers* (regista todas as transações do canal, sendo *tamper-resistant*) para cada canal. Para além disso, alguns dos *peers* têm instalado *chaincode* (define como as organizações *peer* interagem com o *ledger*), consoante seja necessário o *peer* interagir com o *ledger*.

Adicionalmente pode-se comunicar com canais fazendo uso de aplicações que se associam a organizações e comunicam através do envio de propostas de transação para os *peers* das organizações que participam no canal.

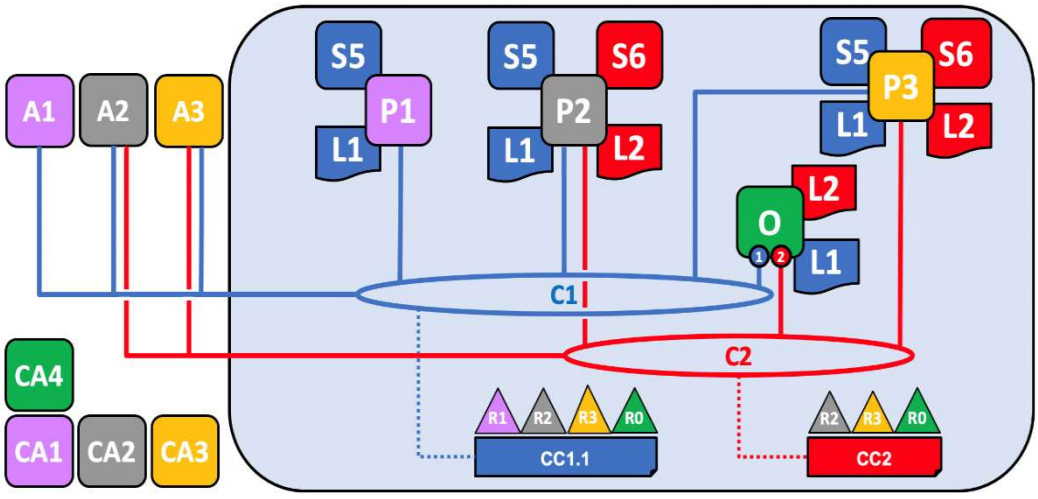


Figura 2 - Exemplo de Rede Hyperledger Fabric [6]

A Figura 2 exemplifica uma rede com quatro organizações (R0, R1, R2 e R3) criadas usando CAs (CA1, CA2, CA3, CA4), em que uma delas tem função de *orderer* ([O] onde se encontram *ledgers* dos dois canais [L1 e L2]) que cria dois canais (C1 e C2)). Ao canal um (C1) pertencem três organizações (R1, R2 e R3) (excluindo o *orderer* [R0]) que têm o *Smart Contract* (SC) e *ledger* de C1, podendo então comunicar todas no canal (assumindo que o SC assim o dita). No caso do *orderer*, apesar de ter a informação do canal no *ledger*, não pode comunicar porque não tem SC. Ao canal dois (C2) pertencem três organizações (R2, R3 e R0), ficando então R2 e R3 com dois pares de SC. Para as organizações R1, R2, R3 há as respetivas aplicações A1, A2 e A3.

2.2.5 Multipass

Tendo em conta que existe a necessidade de criação de Máquinas Virtuais (*Virtual Machines* - VM) com endereços de IP diferentes, para simular e testar as implementações criadas, utilizou-se o *Multipass* [7], uma plataforma focada na criação e gestão de VM em *Ubuntu* de maneira bastante simples.

2.2.6 Ethereum-Geth

Ethereum foi criado com o intuito de aumentar a usabilidade da tecnologia *blockchain*. Assim deixou-se de ter apenas um foco monetário e passou a ser possível criar *smart-contracts* (programa que determina o comportamento de cada *peer*) e *dApps* (aplicações descentralizadas presentes em redes de *blockchain*).

Uma das implementações do protocolo *Ethereum* fez-se usando a linguagem *Go*. Esta implementação encontra-se disponível usando o *standalone client* denominado *Geth*.

2.3 ISABELA/BATINA

Tal como foi mencionado anteriormente, a dissertação irá basear-se na implementação de mecanismos de privacidade para a plataforma ISABELA (IoT Student Advisor and Best Lifestyle Analyser), plataforma esta que recolhe informação de estudantes de modo a poder monitorizar o seu estilo de vida, com o intuito de maximizar o seu desempenho académico.

A arquitetura da ISABELA [8] faz uso de FIWARE para recolher, produzir, publicar e consumir informação, que pode ser proveniente de redes de sensores e telemóveis, utilizadores e sistemas pré-existentes. Para a recolha de dados são usadas:

- Caixas de IoT (desenvolvidas fazendo uso de *Arduino* e *Raspberry*) de modo a obter dados ambientais;
- Redes sociais com o propósito de entender melhor as preferências, relacionamentos e eventos frequentados, etc;
- Aplicações para *smartphones*/*smartwatches* (desenvolvidas em *Android*) tendo o intuito de averiguar a localização, horas de sono, batimento cardíaco, entre outros.

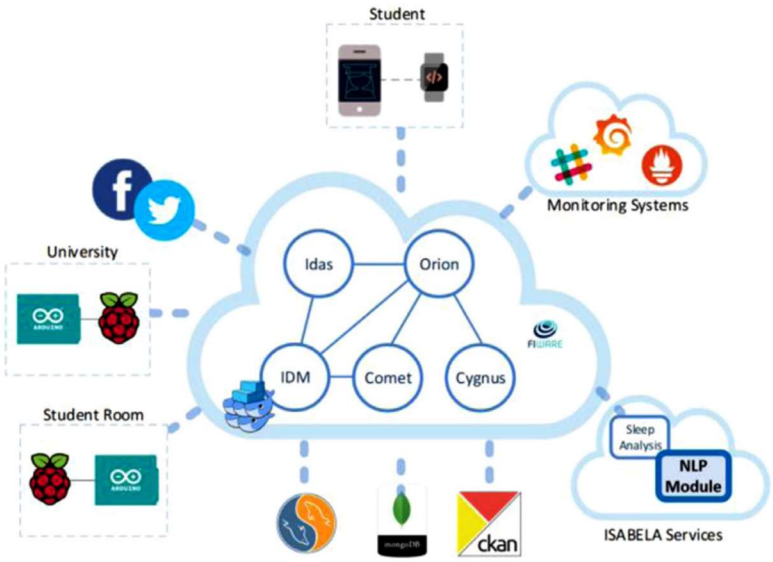


Figura 3 - Arquitetura da ISABELA [8]

No projeto foram usados diferentes *Generic Enablers* (GE) do *FIWARE project*, como se observa na Figura 3, que são descritos de seguida:

- IDAS – Usado na gestão de dispositivos IoT, através do qual se tem a capacidade de interagir entre os dispositivos e FIWARE.
- ORION – Usado na gestão das entidades na aplicação Android da ISABELA. Com esta API pode-se criar/apagar/obter/atualizar entidades. Adicionalmente, para melhorar a distribuição do sistema e a sua escalabilidade, o modulo atua como *broker*, permitindo os consumidores fazerem subscrições de dados com as entidades com regras e atributos específicos. O ORION envia notificações consoante as regras de subscrição são ativadas. Os dados provenientes de dispositivos IoT são associados com as entidades guardadas no ORION.
- Cygnus – Devido ao ORION não ser o GE indicado para gravar *historic data,* usa-se Cygnus para fazer persistir os dados. Este subscreve as entidades ORION e envia a informação para BDs (exemplo: MySQL, MongoDB e CKAN).
- STH-Comet – Tendo em conta que a maioria das BD (que recebem informação através de Cygnus) não têm APIs para aceder à sua informação usa-se este GE para poder ter acesso à informação de modo mais simples.
- IDM – Usado para proteger comunicações entre GEs, dispositivos IoT e a aplicação. Isto é feito com o adicionar de medidas de segurança e autenticação em cada um dos três casos.

Ainda na Figura 3 pode-se observar os dois serviços incluídos na ISABELA, nos quais se inclui o módulo da análise do sono, que faz o cálculo das horas dormidas através do uso dos sensores do smartphone e o módulo NLP faz processamento da linguagem das redes sociais para detetar sentimentos e emoções exprimidos.

Relativamente à plataforma BATINA, trata-se de um projeto com o intuito de fornecer um ensino de maior proximidade entre aluno e professor, tendo em conta a necessidade de privacidade individual.

Comparando ambas as plataformas, a diferença encontra-se na restrição imposta de apenas poder ser usada na sala de aula, no caso da plataforma BATINA. Além disso a plataforma BATINA é uma versão simplificada da ISABELA.

2.4 Planeamento

Devido à necessidade do projeto ISABELA ter acesso a informação pessoal dos utilizadores, quer seja através dos telemóveis ou das caixas IoT, é de extrema importância a procura de mecanismos de segurança que garantam a proteção dos dados segundo o Regulamento Geral de Proteção de Dados. Tendo isso em conta, nesta dissertação pretende-se fazer uso dos canais de HF para privatizar a troca de informação e encontrar outros mecanismos de segurança para conseguir criar um módulo de privacidade que será posteriormente integrado no projeto ISABELA. Adicionalmente pretende-se explorar a possibilidade de criar estes canais seguros fazendo uso de *Ethereum*.

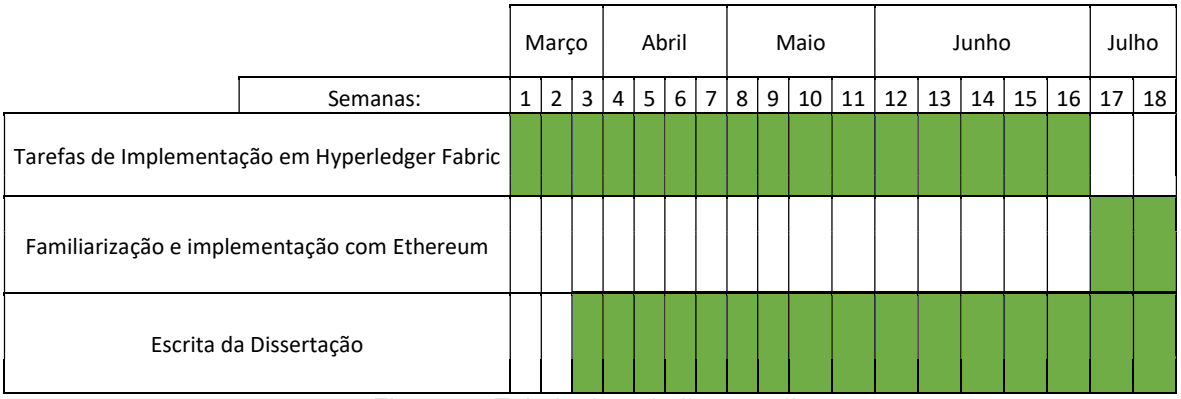


Figura 4 - Tabela do trabalho a realizar

3 Estado da arte

3.1 Plataformas Semelhantes

Dentro do mundo do ensino inovador, existem diferentes projetos que desenvolveram plataformas bastante interessantes, alguns dos quais irão ser descritos:

- *OpenCast* [9],[10] – Ferramenta *open-source* que permite partilhar, editar, gravar e fazer *upload* de vídeos, algo que é bastante útil tanto para alunos como professores. Adicionalmente, a vertente mais interessante passa por se poder integrar esta ferramenta em diversas ferramentas distintas, entre as quais se destaca a possível integração com *Matomo*, uma plataforma de análise da web que fornece informação dos utilizadores, como por exemplo a linguagem do utilizador, documentos/vídeos vistos ou que fizeram download, páginas que mais gostaram, etc.
- *BigBlueButton* [11],[12] – Plataforma de ensino *open-source* onde, por exemplo, é possível integrar o uso de múltiplos *whiteboards* para professor e aluno poderem interagir diretamente.
- *Aleks* [13] - Sistema de Inteligência Artificial (*Artificial Intelligence* – AI) usado para determinar o conhecimento dos alunos e ajudar os alunos a aprenderem os tópicos para os quais o sistema determina estarem prontos.

3.2 Grupos de Trabalho

Existem vários grupos de trabalho focados no uso das novas tecnologias para o ensino, nomeadamente o grupo coordenado por Kelly Parke [14] que numa das disciplinas que leciona propôs aos seus alunos encontrarem maneiras de melhorar o ensino fazendo uso de AI. Para além desse exemplo existe o *SthemBrasil* [15], onde um conjunto de Instituições do Ensino Superior desenvolvem trabalho conjunto para implementar inovações relativamente à qualidade do ensino. Um projeto a ser introduzido é o *Aleks* [16], mencionado anteriormente.

3.3 Análise Comparativa

Comparando os projetos ISABELA e BATINA aos pesquisados, o mais semelhante é o *Aleks* que usa os *inputs* dos alunos para maximizar a sua aprendizagem fazendo uso de AI. A diferença nesse caso é a utilização de AI enquanto na ISABELA é usado o processamento de linguagem natural proveniente das redes sociais. Adicionalmente *Aleks*, tal como o BATINA, apenas recolhe os seus dados na sala de aula.

4 Segurança e Privacidade

4.1 Segurança da ISABELA

Na versão corrente da ISABELA descrita em [1],[8], a segurança prende-se na utilização de técnicas de anonimização e agregação de dados, como por exemplo o uso de *hashes* para anonimizar dados relativos às chamadas feitas e mensagens enviadas. Para além disso alguma da informação trocada entre o utilizador e a entidade de atuação é guardada numa Base de Dados (BD) persistente nos smartphones, que é eliminada quando se desinstala a aplicação.

Adicionalmente emprega-se outro mecanismo de anonimização para o armazenamento de dados sensíveis, onde se utiliza o mecanismo de autenticação do Facebook para obter *hashed access tokens* dos seus servidores, que será novamente *hashed* utilizando o algoritmo SHA256.

Informação como a localização do GPS e o ID do WI-Fi estão personalizados para anonimizar os seus valores. No caso do GPS utiliza-se a distância percorrida e a velocidade média. Enquanto o ID do Wi-Fi é substituído pelo número de estudante.

Apesar de tudo, a plataforma ainda precisa de ter um sistema para notificar o utilizador dos processos de privacidade utilizados pelos serviços IoT, tal como dar autorização para *Data Controllers* (DC) poderem aceder às BD, sistema este que será o objetivo da dissertação.

4.2 Arquitetura do Projeto

Durante a realização da dissertação foram estudadas diversas formas de implementar um canal que permita o envio, de forma segura, da informação dos consentimentos necessários para o acesso aos dados do utilizador, para a aplicação desenvolvida.

Uma das possibilidades estudadas (que acabou por não ser implementada, devido à falta de tempo) introduzida pelo Prof. Ernesto Jiménez, foi o uso de *blockchain* com *Etherium*. Apesar de ser uma tecnologia de *blockchain permissionless* (em que qualquer utilizador pode entrar e participar), ir-se-ia restringir o acesso ao canal com a definição de todos os utilizadores no bloco inicial do *blockchain* (bloco de génese) e com a encriptação do conteúdo, tendo apenas os utilizadores presentes no bloco de génese acesso a uma ou várias chaves privadas que iriam encriptar/desencriptar as mensagens. Caso fosse necessário adicionar utilizadores depois da inicialização do *blockchain* pensou-se em criar um subconjunto de nodos, que utilizaria Proof of Stake (normalmente funciona em redor de *cryptocurrencies*, mas neste caso poder-se-ia atribuir artificialmente um determinado valor de cryptocurrency, para a decisão de entrada ser realizada consoante a decisão da maioria dos participantes), de modo a permitirem a adição de novos utilizadores e atribuíssem uma chave privada para poder aceder ao conteúdo e outra chave privada para encriptar as mensagens a enviar. Adicionalmente, o subconjunto de nodos poderia conceder/retirar permissões de escrita no canal.

A outra possibilidade prendeu-se no uso de HF, já descrito em 2.2.4, tendo sido esta a alternativa usada para implementar o canal seguro.

Realizou-se então a integração dos módulos de privacidade na arquitetura já existente da ISABELA de maneira semelhante aos módulos já existentes em Docker *containers* usando-se HF.

Ou seja, do *Consent Manager* da Figura 5, utiliza-se o *Query, Requests and Responses Handler* (QRRH) para passar quem precisa de dar consentimento, que será enviado para a aplicação usando HF no *Data Aquisition Module* que, depois de receber o consentimento de volta da aplicação, vai passar para o *Data Dispatcher Module*, onde vai ser armazenado o novo utilizador que deu consentimento. Finalmente os DC utilizam o QRRH para ir buscar ao *Data Dispatcher Module*, sendo este processo controlado pela *Privacy Enforcement Bridge*.

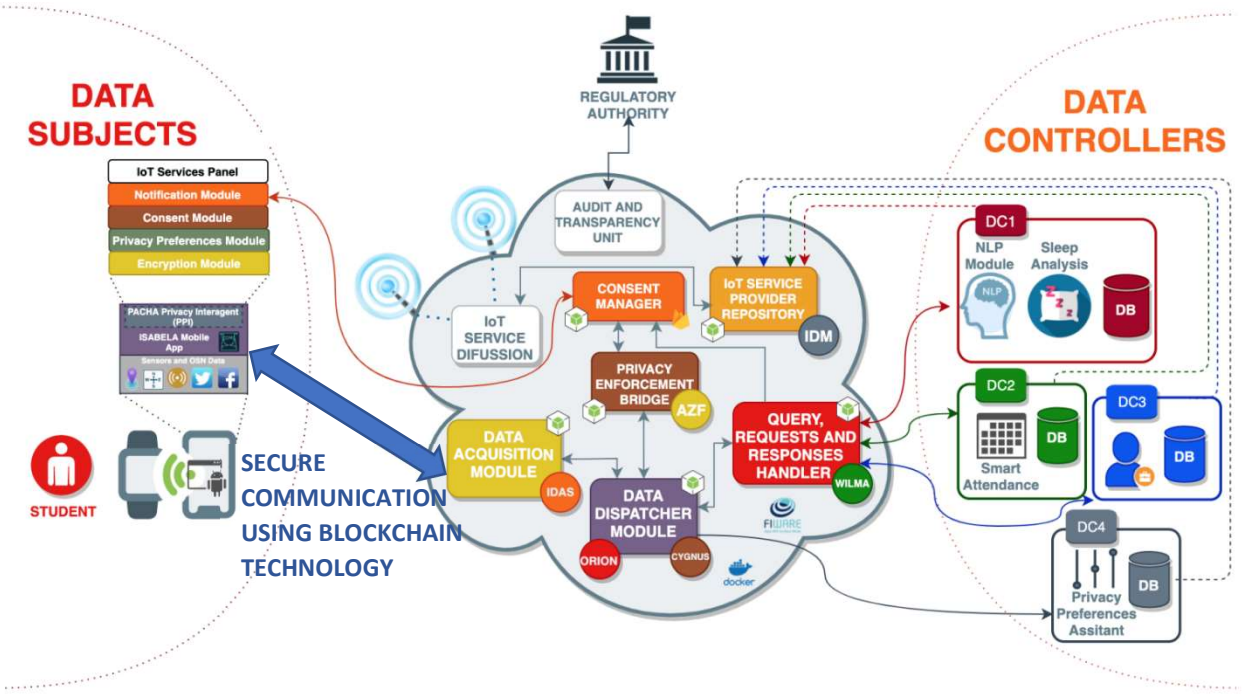


Figura 5 - Arquitetura da ISABELA na vertente de segurança

5 Implementação

Durante a dissertação foram realizadas diversas implementações que irão ser descritas detalhadamente neste capítulo.

5.1 Xamarin

No caso desta plataforma, foi realizado o tutorial de como criar aplicações móveis com o Xamarin.Forms, como se pode ver em [17]. Na implementação destes exemplos foi necessário rever C# e aprender um pouco de .NET, de modo a modificar algumas partes do tutorial que se encontravam desatualizadas.

5.2 Hyperledger Fabric

As implementações realizadas, os sistemas operativos utilizados e os pré-requisitos necessários para implementar em HF serão descritos passo a passo de seguida. Adicionalmente, é revelante ter em conta que foi bastante difícil colocar qualquer uma das implementações apresentadas a funcionar, isto deveu-se a esta tecnologia estar em continuo desenvolvimento, o que levou à desatualização da documentação. Adicionalmente, a relativa obscuridade de HF levou à escassez de recursos para ajudar a ultrapassar os problemas encontrados.

5.2.1 Sistemas operativos

5.2.1.1 Linux Mint 20.3 Una

Nas implementações que serão descritas mais à frente, com apenas um *host* e na criação dos ficheiros de configuração usa-se uma VM de *Linux Mint 20.3 Una*, versão *Cinnamon*, com o software de virtualização *VMware Workstation 16 Player*.

Foi feito download da imagem de *Linux Mint* em [18] (na data de escrita a referência encontra-se indisponível).

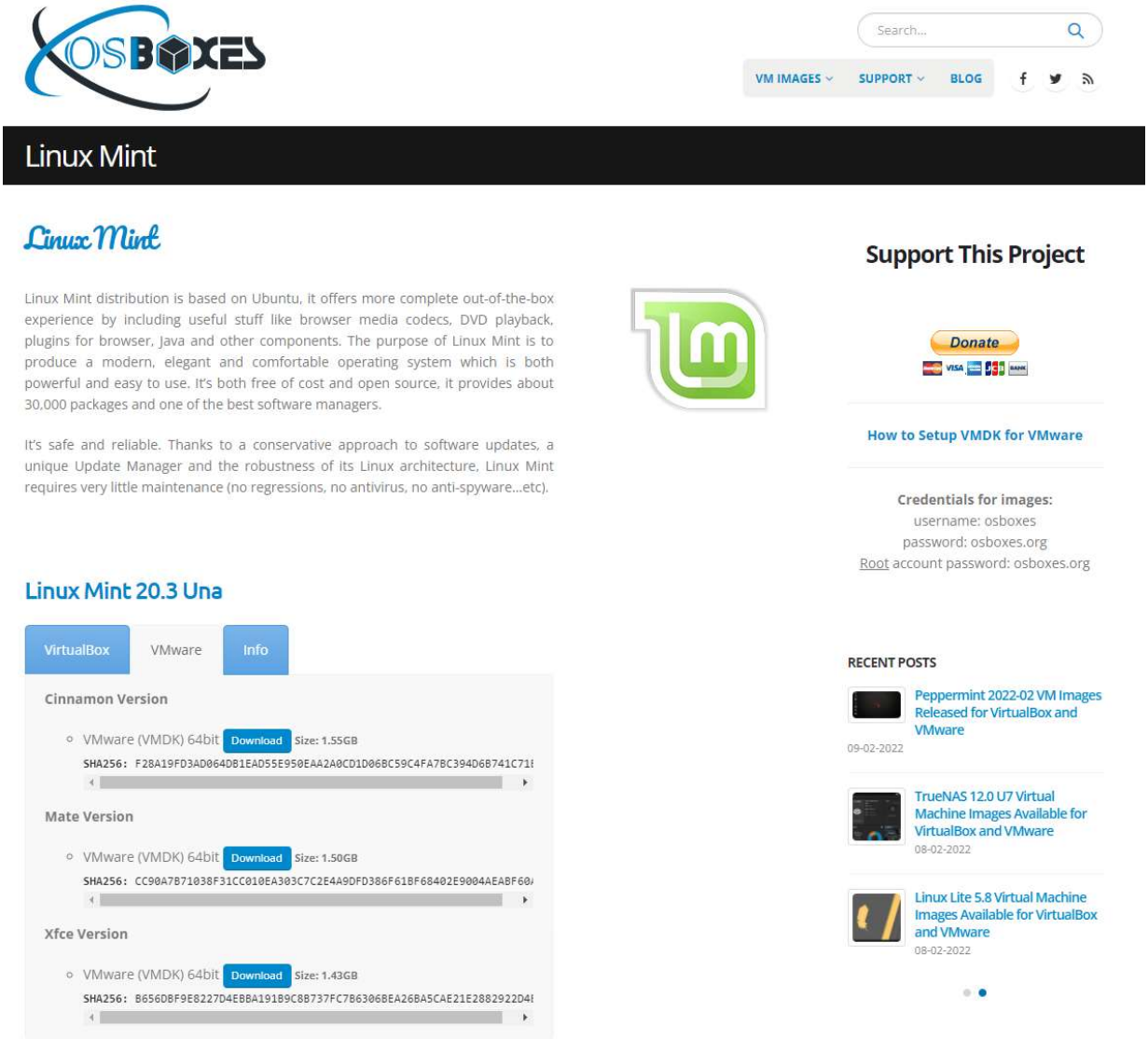


Figura 6 - Snapshot da webpage onde se realizou o download de Linux Mint [18]

5.2.1.2 Multipass - Ubuntu

Como foi mencionado anteriormente, usa-se a plataforma *Multipass*, de modo a poder criar VM de *Ubuntu* para as implementações *multihost*.

Instalou-se a plataforma seguindo as instruções presentes em [19] para o respetivo sistema operativo a ser utilizado. É fulcral fazer a instalação numa rede privada de modo a evitar problemas de comunicação.

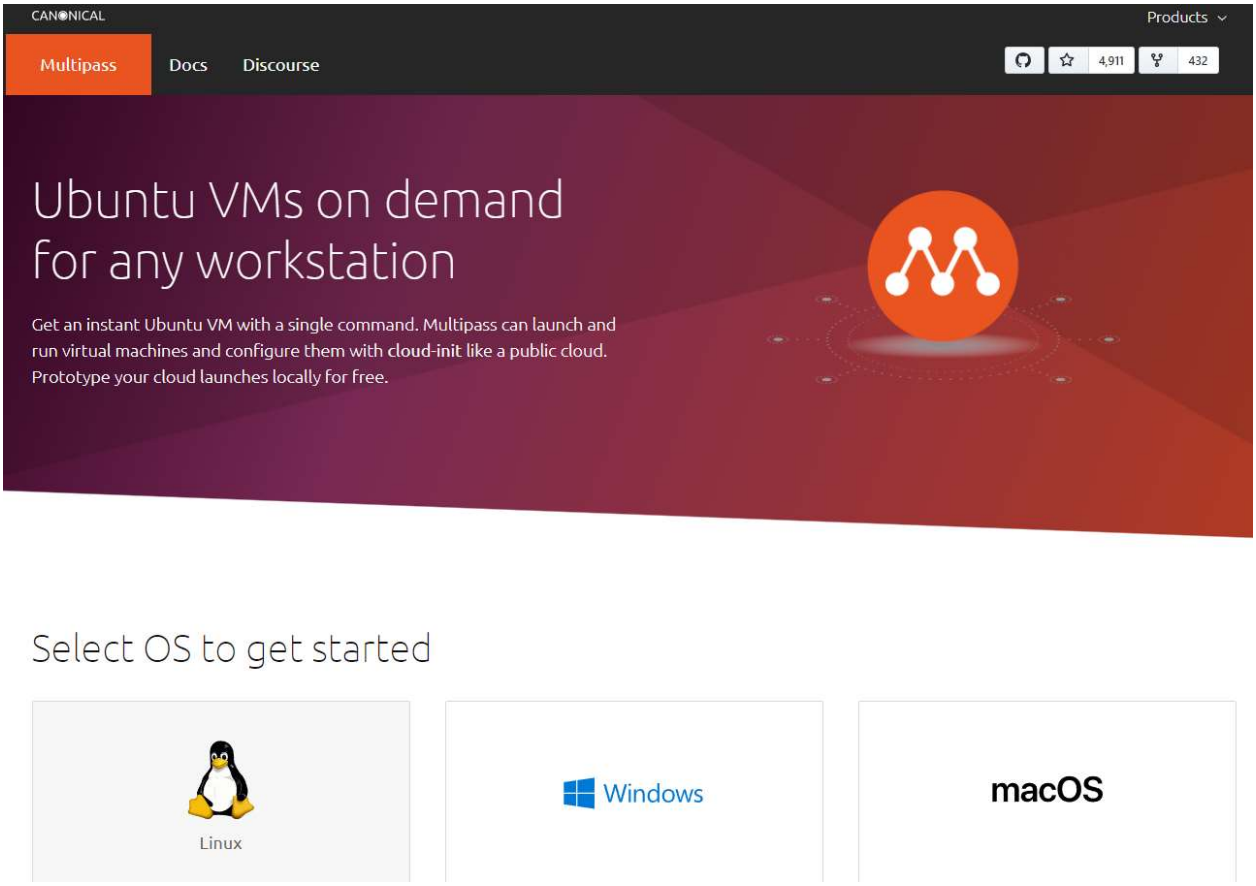


Figura 7 - Snapshot da webpage onde foi feito download de Multipass [19]

Após a instalação é possível o uso de comandos *Multipass* no terminal que permitem a criação de VM com diversas configurações, entre as quais: diferentes nomes, versões de *Ubuntu*, espaço de disco a utilizar, memoria de *RAM* e *CPUs* alocados, tipo de rede, entre outros.

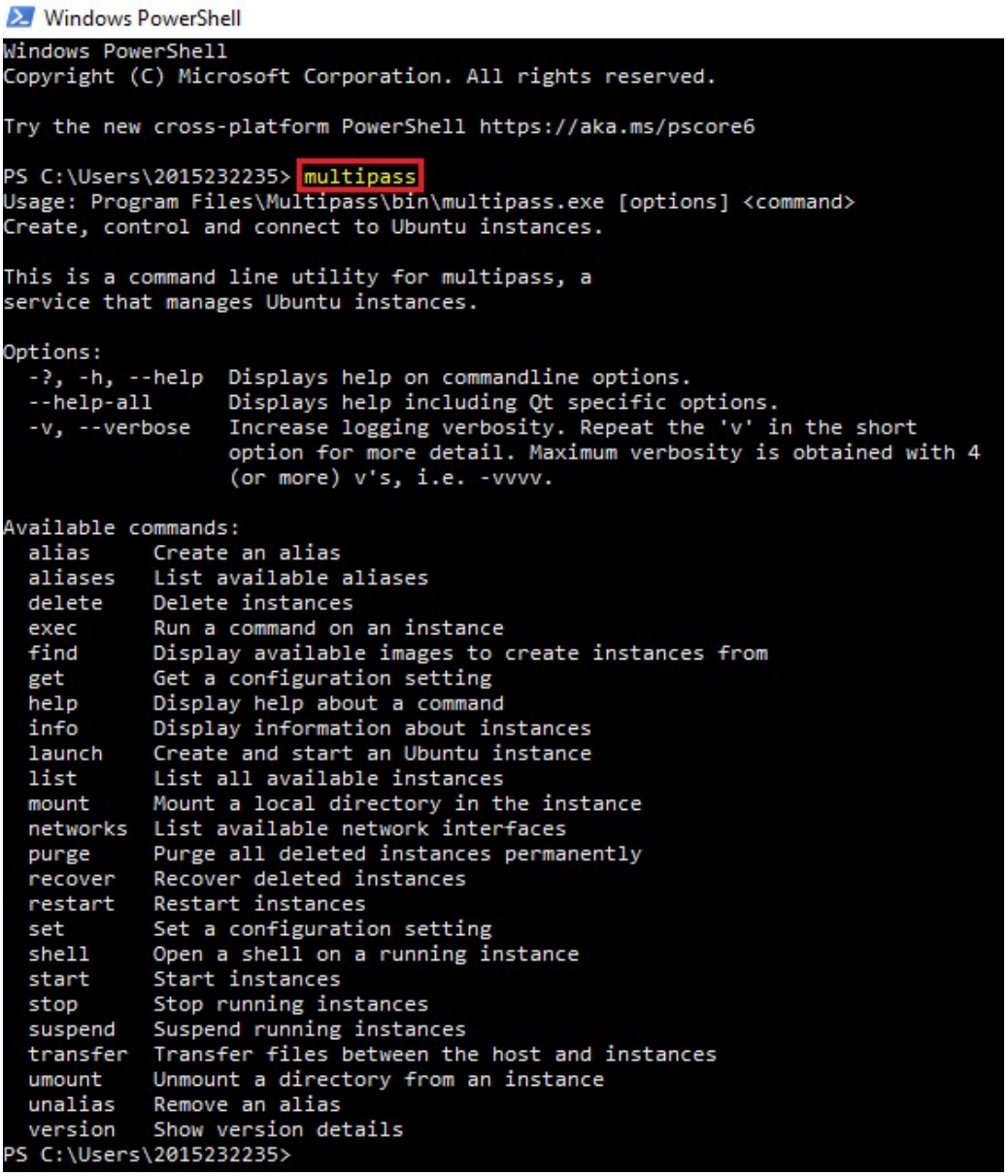


Figura 8 - Listagem de comandos Multipass

5.2.2 Pré-requisitos

Para poder correr as implementações foi necessária a instalação de diversos softwares, que irão ser enumerados e explicados de seguida. Nesta explicação é adquirido que se está a utilizar um sistema atualizado. Para isso corre-se o comando `sudo apt update && sudo apt upgrade`.

Adicionalmente, reitera-se que nestas implementações usou-se a rede em modo privado, de modo a ter mais permissões de acesso à comunicação, de modo a resolver erros de acesso que ocorreram durante a criação dos exemplos.

5.2.2.1 Git

Para instalar este conjunto de *utility programs* contidos em Git introduziu-se o comando `sudo apt-get install git`.

```
osboxes@osboxes:~$ sudo apt-get install git
[sudo] password for osboxes:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk
  gitweb git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 5,465 kB of archives.
After this operation, 38.4 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu focal/main amd64 liberror-perl all 0.17029-1 [26.5 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 git-man all 1:2.25.1-1ubuntu3.2 [884 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 git amd64 1:2.25.1-1ubuntu3.2 [4,554 kB]
Fetched 5,465 kB in 8s (710 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 312545 files and directories currently installed.)
Preparing to unpack .../liberror-perl_0.17029-1_all.deb ...
Unpacking liberror-perl (0.17029-1) ...
Selecting previously unselected package git-man.
Preparing to unpack .../git-man_1%3a2.25.1-1ubuntu3.2_all.deb ...
Unpacking git-man (1:2.25.1-1ubuntu3.2) ...
Selecting previously unselected package git.
Preparing to unpack .../git_1%3a2.25.1-1ubuntu3.2_amd64.deb ...
Unpacking git (1:2.25.1-1ubuntu3.2) ...
Setting up liberror-perl (0.17029-1) ...
Setting up git-man (1:2.25.1-1ubuntu3.2) ...
Setting up git (1:2.25.1-1ubuntu3.2) ...
Processing triggers for man-db (2.9.1-1) ...
osboxes@osboxes:~$
```

Figura 9 - Instalação de Git

5.2.2.2 cURL

Para fazer transferência de dados usou-se *cURL*, que foi instalado através do comando `sudo apt install curl`.

```
osboxes@osboxes:~$ sudo apt install curl
Reading package lists... Done
Building dependency tree
Reading state information... Done
curl is already the newest version (7.68.0-1ubuntu2.7).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
osboxes@osboxes:~$
```

Figura 10 - Instalação de cURL

Para verificar a versão instalada utilizar `curl --version`.

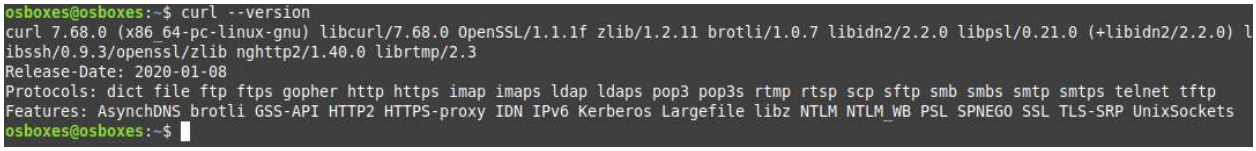


Figura 11 - Verificação da versão de cURL

5.2.2.3 Docker e Docker-Compose

Os *containers* onde a *rede* vai ser criada e onde se vai interagir com a mesma, vão ser criados usando *Docker* e D*ocker-Compose*.

Começa-se por instalar o *Docker*. Com o comando `sudo apt-get remove docker docker-engine docker.io` para desinstalar versões anteriores de Docker que poderiam estar instaladas anteriormente.

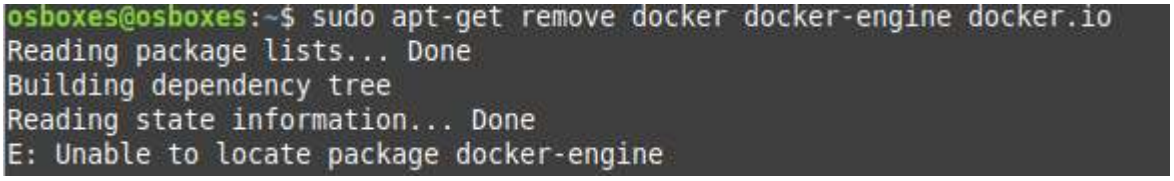


Figura 12 - Desinstalação de possíveis artefactos de Docker

Seguidamente usa-se `sudo apt install docker.io`, para instalar *Docker*.

```
osboxes@osboxes:~$ sudo apt install docker.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd pigz runc ubuntu-fan
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd docker.io pigz runc ubuntu-fan
0 upgraded, 6 newly installed, 0 to remove and 0 not upgraded.
Need to get 74.2 MB of archives.
After this operation, 360 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu focal/universe amd64 pigz amd64 2.4-1 [57.4 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal/main amd64 bridge-utils amd64 1.6-2ubuntu1 [30.5 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 runc amd64 1.0.1-0ubuntu2~20.04.1 [4,155 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 containerd amd64 1.5.5-0ubuntu3~20.04.2 [33.0 MB]
Get:5 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 docker.io amd64 20.10.7-0ubuntu5~20.04.2 [36.9 MB]
Get:6 http://archive.ubuntu.com/ubuntu focal/main amd64 ubuntu-fan all 0.12.13 [34.5 kB]
Fetched 74.2 MB in 1min 21s (919 kB/s)
Preconfiguring packages ...
Selecting previously unselected package pigz.
(Reading database ... 313479 files and directories currently installed.)
Preparing to unpack .../0-pigz_2.4-1_amd64.deb ...
Unpacking pigz (2.4-1) ...
Selecting previously unselected package bridge-utils.
Preparing to unpack .../1-bridge-utils_1.6-2ubuntu1_amd64.deb ...
Unpacking bridge-utils (1.6-2ubuntu1) ...
Selecting previously unselected package runc.
Preparing to unpack .../2-runc_1.0.1-0ubuntu2~20.04.1_amd64.deb ...
Unpacking runc (1.0.1-0ubuntu2~20.04.1) ...
Selecting previously unselected package containerd.
Preparing to unpack .../3-containerd_1.5.5-0ubuntu3~20.04.2_amd64.deb ...
Unpacking containerd (1.5.5-0ubuntu3~20.04.2) ...
Selecting previously unselected package docker.io.
Preparing to unpack .../4-docker.io_20.10.7-0ubuntu5~20.04.2_amd64.deb ...
Unpacking docker.io (20.10.7-0ubuntu5~20.04.2) ...
Selecting previously unselected package ubuntu-fan.
Preparing to unpack .../5-ubuntu-fan_0.12.13_all.deb ...
Unpacking ubuntu-fan (0.12.13) ...
Setting up runc (1.0.1-0ubuntu2~20.04.1) ...
Setting up bridge-utils (1.6-2ubuntu1) ...
Setting up pigz (2.4-1) ...
Setting up containerd (1.5.5-0ubuntu3~20.04.2) ...
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service → /lib/systemd/system/containerd.service.
Setting up ubuntu-fan (0.12.13) ...
Created symlink /etc/systemd/system/multi-user.target.wants/ubuntu-fan.service → /lib/systemd/system/ubuntu-fan.service.
Setting up docker.io (20.10.7-0ubuntu5~20.04.2) ...
Adding group `docker' (GID 135) ...
Done.
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /lib/systemd/system/docker.socket.
Processing triggers for systemd (245.4-4ubuntu3.15) ...
Processing triggers for man-db (2.9.1-1) ...
osboxes@osboxes:~$
```

Figura 13 - Instalação de Docker

Depois usa-se `sudo systemctl start docker` e `sudo systemctl enable docker` para o Docker inicializar quando se liga o computador.

```
osboxes@osboxes:~$ sudo systemctl start docker
osboxes@osboxes:~$ sudo systemctl enable docker
osboxes@osboxes:~$
```

Figura 14 - Configuração da inicialização de Docker no arranque da máquina

Finalmente, usa-se `sudo usermod -a -G docker <nome de utilizador>` (no caso apresentado o nome de utilizador é *osboxes*) para dar permissões de acesso ao utilizador.

```
osboxes@osboxes:~/Desktop$ sudo usermod -a -G docker osboxes
osboxes@osboxes:~/Desktop$
```

Figura 15 - Atribuição de permissões ao utilizador

Para verificar que versão do *Docker* foi instalada usar `docker --version`

Figura 16 - Verificação da versão de Docker

Passando para a instalação do *Docker-Compose*, usa-se o seguinte comando para fazer *download* e salvar o ficheiro em */usr/local/bin* com o nome *docker-compose.*

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.2/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose.
```

Figura 17 - Instalação de Docker-Compose

Seguidamente usa-se o comando apresentado para mudar as permissões do *Docker-Compose*.

```
sudo chmod +x /usr/local/bin/docker-compose
```

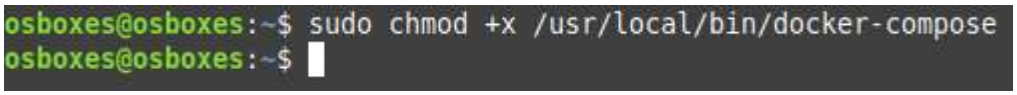


Figura 18 - Atribuição de permissões a Docker-Compose

Para verificar a versão de *Docker-Compose* utiliza-se `sudo docker-compose --version`.

Figura 19 - Verificação da versão de Docker-Compose

5.2.2.4 Documentos Fabric-Samples

Nos documentos Fabric-Samples estão presentes os requisitos que são utilizados na criação das redes em HF. Adicionalmente estão presentes alguns exemplos de redes HF.

Antes de tentar fazer download dos *Fabric-Samples* é necessário reiniciar a VM para o que foi instalado anteriormente, relativamente às permissões do *Docker*, entrarem em vigor.

O *Fabric-Samples* utilizado foi instalado seguindo as instruções fornecidas em [20].

Como o *Fabric-Samples* foi instalado na diretoria *Desktop*, foi utilizado `cd Desktop/`, seguido por `curl -sSL https://bit.ly/2ysbOFE | bash -s`, para fazer download e instalar a última versão de *Fabric-Samples*, que neste caso foram as versões 2.4.2 do *Fabric* e 1.5.2 do *Fabric-CA*. Para instalar uma versão especifica utilizar o comando `curl -sSL https://bit.ly/2ysbOFE | bash -s -- <fabric_version> <fabric-ca_version>`.

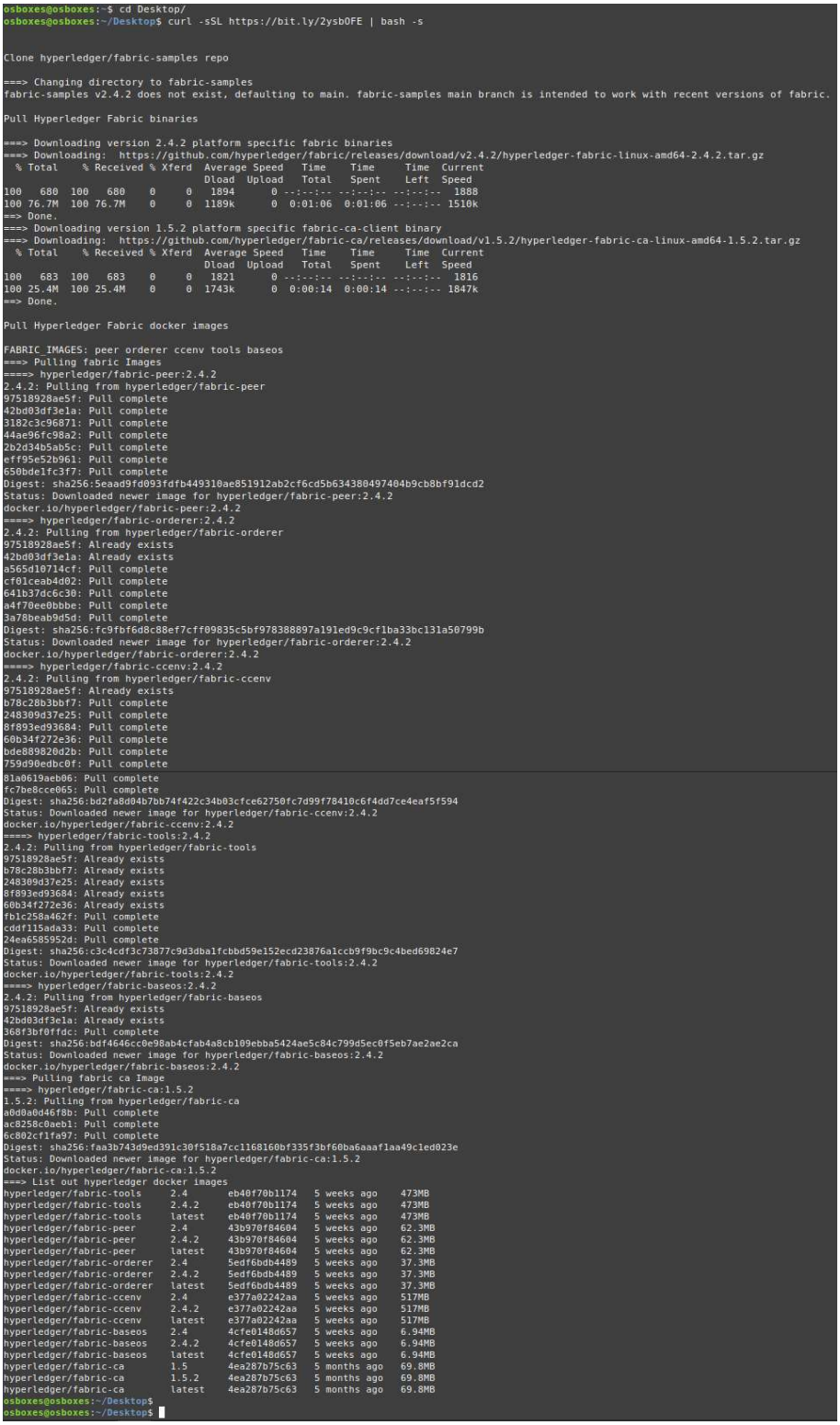


Figura 20 - Instalação de Fabric-Samples

Para verificar que tudo foi bem instalado, deverá mudar-se para a diretoria *test-network* dentro da pasta criada *fabric-samples* com `cd fabric-samples/test-network/` e executar `./network.sh down` para desligar a rede e remover ficheiros criados (num primeiro uso é esperado que nada seja removido, devido à rede nunca ter sido inicializada), seguido de `./network.sh up` para inicializar a rede. Se a inicialização correr sem problemas confirma-se que ficou instalado corretamente.

```
osboxes@osboxes:~/Desktop$ cd fabric-samples/test-network
osboxes@osboxes:~/Desktop/fabric-samples/test-network$ ./network.sh down
Using docker and docker-compose
Stopping network
Removing network fabric_test
WARNING: Network fabric_test not found.
Removing network compose_default
WARNING: Network compose_default not found.
Removing volume compose_orderer.example.com
WARNING: Volume compose_orderer.example.com not found.
Removing volume compose_peer0.org1.example.com
WARNING: Volume compose_peer0.org1.example.com not found.
Removing volume compose_peer0.org2.example.com
WARNING: Volume compose_peer0.org2.example.com not found.
Removing volume compose_peer0.org3.example.com
WARNING: Volume compose_peer0.org3.example.com not found.
Error: No such volume: docker_orderer.example.com
Error: No such volume: docker_peer0.org1.example.com
Error: No such volume: docker_peer0.org2.example.com
Removing remaining containers
Removing generated chaincode docker images
"docker kill" requires at least 1 argument.
See 'docker kill --help'.

Usage:  docker kill [OPTIONS] CONTAINER [CONTAINER...]

Kill one or more running containers
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
7e5209d2300f: Pull complete

Digest: sha256:34c3559bbdedefd67195e766e38cfbb0fcabff4241dbee3f390fd6e3310f5ebc
Status: Downloaded newer image for busybox:latest
```

Figura 21 – Encerramento da rede de exemplo

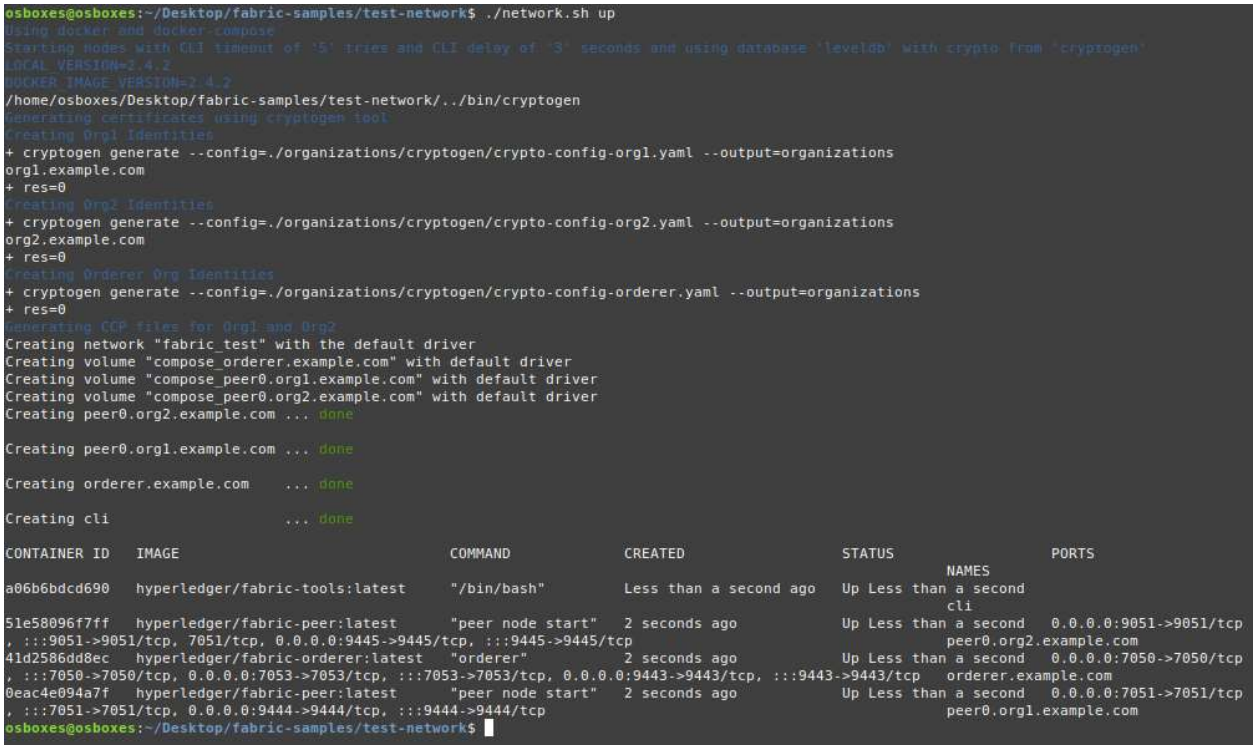


Figura 22 - Inicialização da rede de exemplo

Para poder correr os scripts existentes na diretoria *fabric-samples* é necessário utilizar o comando apresentado.

`export PATH=/home/<nome de utilizador>/Desktop/tese/fabric-samples/bin:$PATH`

No caso apresentado o utilizador é *osboxes*, ficando o comando com a seguinte forma.

`export PATH=/home/osboxes/Desktop/tese/fabric-samples/bin:$PATH`

5.2.2.5 Node.JS e NPM

Para utilizar a linguagem *Javascript* em *chaincodes* é necessário instalar *Node.js* e *npm*.

A instalação inicia-se com o comando.

`wget http://nodejs.org/dist/v16.14.0/node-v16.14.0-linux-x64.tar.gz` seguido por `sudo tar -C /usr/local --strip-components 1 -xzf node-v16.14.0-linux-x64.tar.gz`

Deste modo fica-se então com *Node.js* e *npm* instalados na diretoria */usr/local/bin*. Isto pode ser verificado utilizando o comando `ls -l /usr/local/bin/node ls -l /usr/local/bin/npm`.

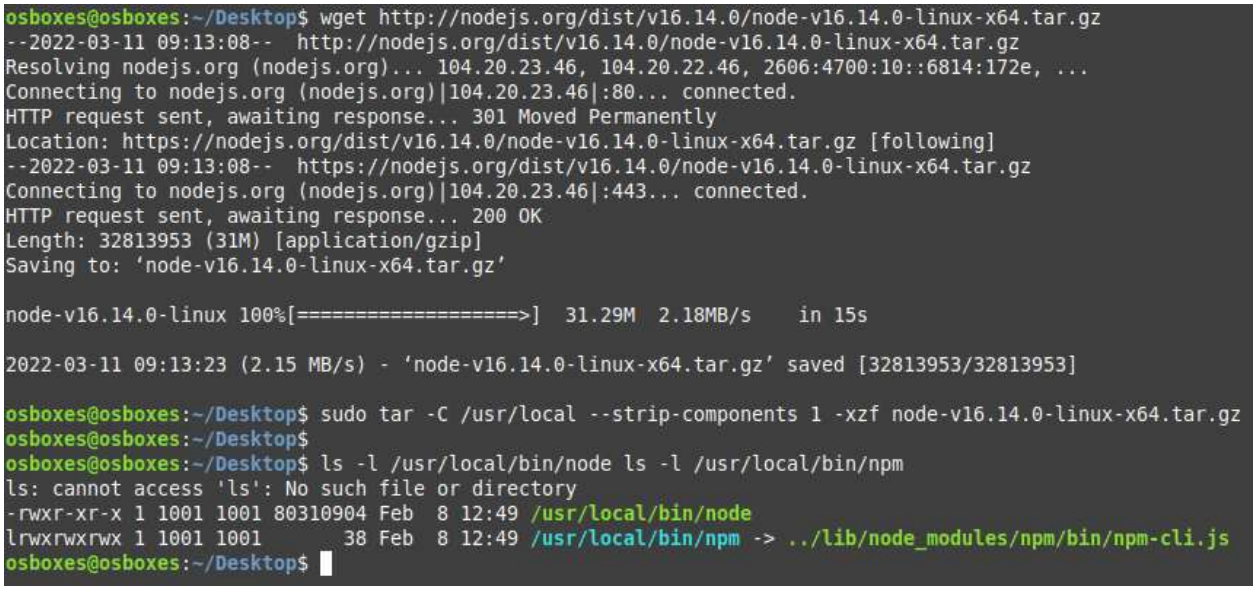


Figura 23 - Instalação de NPM e Node.JS

5.2.2.6 GO

Para utilizar *Go* na implementação do *chaincode* é necessário instalar a linguagem. Com esse intuito seguem-se as instruções presentes em [21].

Para verificar a versão instalada usa-se `go version`.

Figura 24 - Verificação da versão de Go

A diretoria onde os comandos são corridos é irrelevante.

5.2.2.7 jq

Comando usado para ler dados de um ficheiro *JSON*. Isto é necessário para a adição de organizações a uma rede pré-existente. Para instalar basta utilizar o comando `sudo apt-get install jq`.

5.2.2.8 Github Desktop

Apesar de ser bastante útil, a criação de um repositório de G*itHub* não é um pré-requisito proibitivo. Contudo numa das implementações, que irá ser descrita posteriormente, é vantajosa a criação de um repositório a partir de uma diretoria local.

No entanto, o processo de criação do repositório, não foi um processo linear.

Num primeiro momento existiu uma tentativa de seguir o tutorial presente em [22] onde é descrito como criar o repositório a partir do terminal, fazendo uso de comandos como `git init` (para criar um repositório), `git add` (para adicionar ficheiros ao repositório) e `git commit` (para guardar as modificações feitas ao repositório). Esta tentativa acabou por dar erros de permissão negada.

Para simplificar o processo e ter acesso a um modo mais visual de criação de repositórios, utilizou-se o *GitHub Desktop*. Para instalar foi-se a [23] fazer download e instalação consequente do executável.

Figura 25 - Snapshot da webpage de instalação de Github Desktop [23]

Após a instalação criou-se o repositório utilizando a diretoria (onde se localizam os materiais da rede). Mas ao clicar em *fetch* (de modo a passar as mudanças para o repositório no *GitHub*) apareceram erros de autenticação, não sendo mencionada explicitamente qual era a autenticação a causar problemas. Todavia, em algumas das linhas de erros era mencionado *SSH*.

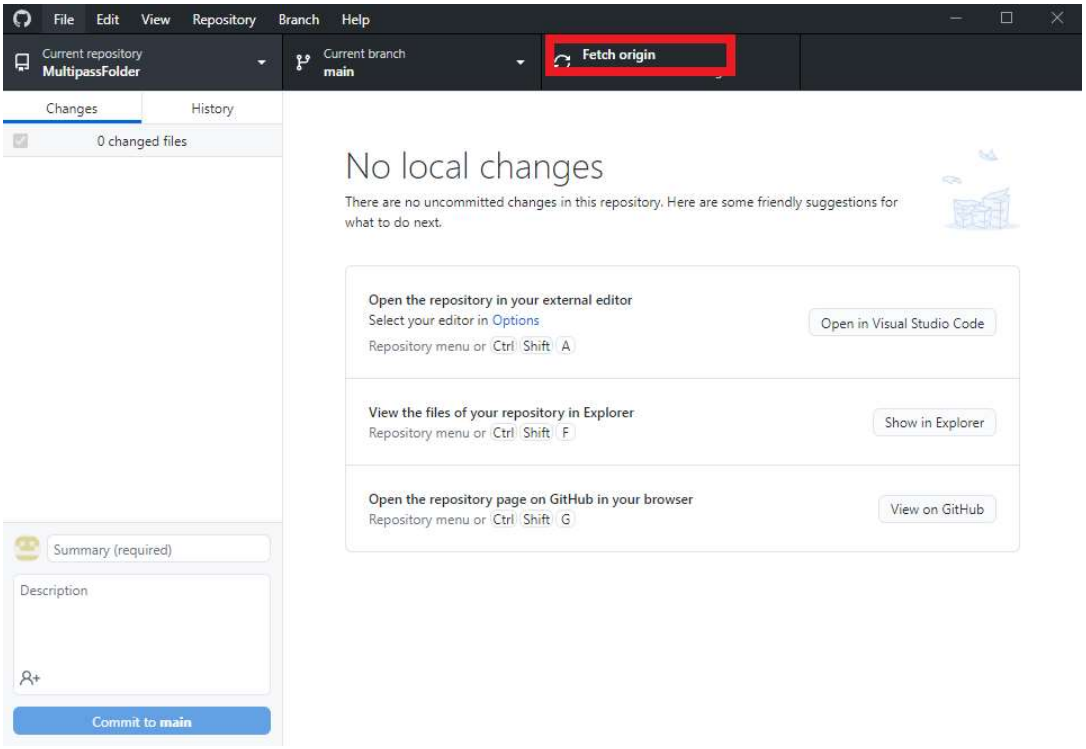


Figura 26 - Snapshot da página de Github Desktop

Numa tentativa de ultrapassar esta dificuldade utilizou-se [24] para criar uma chave *SSH* fazendo uso do comando `ssh-keygen` (usado na criação da chave publica e privada necessárias para a conexão *SSH*) e [25] para associar a chave publica à conta de *GitHub*.

Após percorrer os passos descritos continuou a aparecer o erro de autenticação. Nessa altura experimentou-se criar um repositório de raiz (sem nada), com o *Github Desktop* e seguidamente copiar os ficheiros necessários para a diretoria do repositório, conseguindo finalmente realizar o *fetch*.

Durante o *fetch* surgiu o *warning* `"LF will be replaced by CRLF"` que se deve ao facto de se terem *line endings* de *Linux* em *Windows*.

5.2.2.8 Subversion

Ferramenta normalmente utilizada na gestão e acompanhamento de mudanças de código em projetos. Neste caso vai ser utilizada para fazer download de uma diretoria de um repositório de *GitHub*.

Para instalar utilizou-se `sudo apt-get install subversion -y`

```
ubuntu@vm1:~$ sudo apt-get install subversion -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libapr1 libaprutil1 libserf-1-1 libsvn1
Suggested packages:
  db5.3-util libapache2-mod-svn subversion-tools
The following NEW packages will be installed:
  libapr1 libaprutil1 libserf-1-1 libsvn1 subversion
0 upgraded, 5 newly installed, 0 to remove and 0 not upgraded.
Need to get 2237 kB of archives.
After this operation, 9910 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/main amd64 libapr1 amd64 1.6.3-2 [90.9 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic/main amd64 libaprutil1 amd64 1.6.1-2 [84.4 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic/universe amd64 libserf-1-1 amd64 1.3.9-6 [44.4 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic/universe amd64 libsvn1 amd64 1.9.7-4ubuntu1 [1183 kB]
Get:5 http://archive.ubuntu.com/ubuntu bionic/universe amd64 subversion amd64 1.9.7-4ubuntu1 [834 kB]
Fetched 2237 kB in 4s (601 kB/s)
Selecting previously unselected package libapr1:amd64.
(Reading database ... 91423 files and directories currently installed.)
Preparing to unpack .../libapr1_1.6.3-2_amd64.deb ...
Unpacking libapr1:amd64 (1.6.3-2) ...
Selecting previously unselected package libaprutil1:amd64.
Preparing to unpack .../libaprutil1_1.6.1-2_amd64.deb ...
Unpacking libaprutil1:amd64 (1.6.1-2) ...
Selecting previously unselected package libserf-1-1:amd64.
Preparing to unpack .../libserf-1-1_1.3.9-6_amd64.deb ...
Unpacking libserf-1-1:amd64 (1.3.9-6) ...
Selecting previously unselected package libsvn1:amd64.
Preparing to unpack .../libsvn1_1.9.7-4ubuntu1_amd64.deb ...
Unpacking libsvn1:amd64 (1.9.7-4ubuntu1) ...
Selecting previously unselected package subversion.
Preparing to unpack .../subversion_1.9.7-4ubuntu1_amd64.deb ...
Unpacking subversion (1.9.7-4ubuntu1) ...
Setting up libapr1:amd64 (1.6.3-2) ...
Setting up libaprutil1:amd64 (1.6.1-2) ...
Setting up libserf-1-1:amd64 (1.3.9-6) ...
Setting up libsvn1:amd64 (1.9.7-4ubuntu1) ...
Setting up subversion (1.9.7-4ubuntu1) ...
Processing triggers for libc-bin (2.27-3ubuntu1.5) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

Figura 27 - Instalação de Subversion

5.2.2.8 yq

Versão de jq utilizada na edição de ficheiros do tipo yaml. Este comando foi utilizado na criação do script criador dos documentos consoante a configuração (secção 5.2.3.7). Para instalar utiliza-se o comando `sudo snap install yq`.

5.2.3 Criação, implementação e execução de diferentes configurações de rede

5.2.3.1 Implementação inicial de rede com três organizações

A implementação inicial foi realizada com a ajuda do MSc. Jorge Rivadeneira, onde usando [6], foi implementada em linguagem *go* uma rede onde existem três organizações, entre as quais uma das organizações apenas consegue ler a informação do canal, não podendo escrever para o canal, ao contrário das outras duas que podem ler e escrever.

O processo de criação da rede descrita irá ser explicado passo a passo de seguida:

1) Inicialmente recorreu-se a *cryptogen* [26], para gerar material chave do HF, onde se utilizou a configuração presente em *crypto-config.yaml*, que descreve quantas

organizações existem, número de utilizadores dentro das organizações, nome e domínio das organizações.

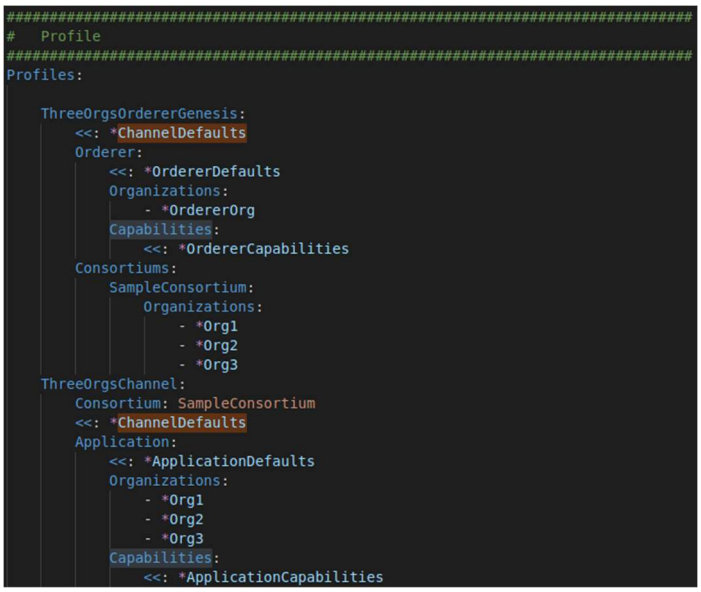
```
################################################################################
#   crypto-config.yaml
################################################################################
OrdererOrgs:
- Name: Orderer
  Domain: acme.com
  EnableNodeOUs: true
  Specs:
    - Hostname: orderer
      SANS:
        - localhost
PeerOrgs:
- Name: Org1
  Domain: org1.acme.com
  EnableNodeOUs: true
  Template:
    Count: 1
    SANS:
      - localhost
  Users:
    Count: 1
- Name: Org2
  Domain: org2.acme.com
  EnableNodeOUs: true
  Template:
    Count: 1
    SANS:
      - localhost
  Users:
    Count: 1
- Name: Org3
  Domain: org3.acme.com
  EnableNodeOUs: true
  Template:
    Count: 1
    SANS:
      - localhost
  Users:
    Count: 1
```

Figura 28 - Ficheiro crypto-config.yaml

```
linuxmint@linuxmint201:~/Desktop/tutorial/acme-network$ cryptogen generate --config=./crypto-config.yaml
org1.acme.com
org2.acme.com
org3.acme.com
```

Figura 29 - Comando cryptogen utilizado

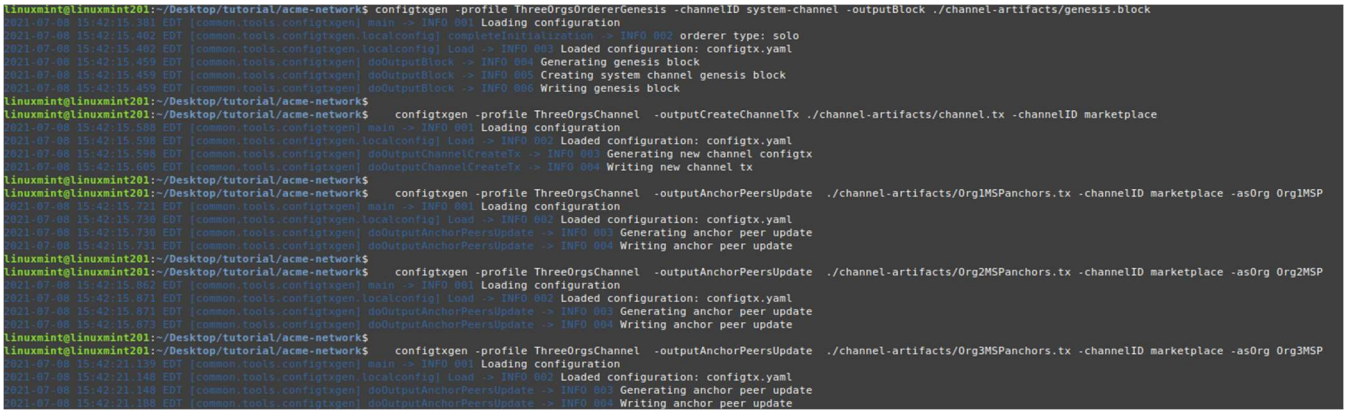
2) Seguidamente, usou-se *configtxgen* [27] que, fazendo uso dos perfis criados em *configtx.yaml* (onde se define informação necessária à criação do canal, como o comportamento das politicas, os *hosts* e os *ports* usados), se cria a configuração do canal para cada uma das organizações.



```
################################################################################
#   Profile
################################################################################
Profiles:

    ThreeOrgsOrdererGenesis:
        <<: *ChannelDefaults
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
            Capabilities:
                <<: *OrdererCapabilities
        Consortiums:
            SampleConsortium:
                Organizations:
                    - *Org1
                    - *Org2
                    - *Org3
    ThreeOrgsChannel:
        Consortium: SampleConsortium
        <<: *ChannelDefaults
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Org1
                - *Org2
                - *Org3
            Capabilities:
                <<: *ApplicationCapabilities
```

Figura 30 - Excerto de configtx.yaml relativo aos perfis



```
linuxmint@linuxmint201:~/Desktop/tutorial/acme-network$ configtxgen -profile ThreeOrgsOrdererGenesis -channelID system-channel -outputBlock ./channel-artifacts/genesis.block
2021-07-08 15:42:15.381 EDT [common.tools.configtxgen] main -> INFO 001 Loading configuration
2021-07-08 15:42:15.402 EDT [common.tools.configtxgen.localconfig] completeInitialization -> INFO 002 orderer type: solo
2021-07-08 15:42:15.402 EDT [common.tools.configtxgen.localconfig] Load -> INFO 003 Loaded configuration: configtx.yaml
2021-07-08 15:42:15.459 EDT [common.tools.configtxgen] doOutputBlock -> INFO 004 Generating genesis block
2021-07-08 15:42:15.459 EDT [common.tools.configtxgen] doOutputBlock -> INFO 005 Creating system channel genesis block
2021-07-08 15:42:15.459 EDT [common.tools.configtxgen] doOutputBlock -> INFO 006 Writing genesis block
linuxmint@linuxmint201:~/Desktop/tutorial/acme-network$
linuxmint@linuxmint201:~/Desktop/tutorial/acme-network$    configtxgen -profile ThreeOrgsChannel  -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID marketplace
2021-07-08 15:42:15.588 EDT [common.tools.configtxgen] main -> INFO 001 Loading configuration
2021-07-08 15:42:15.598 EDT [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: configtx.yaml
2021-07-08 15:42:15.598 EDT [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 003 Generating new channel configtx
2021-07-08 15:42:15.605 EDT [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 004 Writing new channel tx
linuxmint@linuxmint201:~/Desktop/tutorial/acme-network$
linuxmint@linuxmint201:~/Desktop/tutorial/acme-network$    configtxgen -profile ThreeOrgsChannel  -outputAnchorPeersUpdate  ./channel-artifacts/Org1MSPanchors.tx -channelID marketplace -asOrg Org1MSP
2021-07-08 15:42:15.721 EDT [common.tools.configtxgen] main -> INFO 001 Loading configuration
2021-07-08 15:42:15.730 EDT [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: configtx.yaml
2021-07-08 15:42:15.730 EDT [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Generating anchor peer update
2021-07-08 15:42:15.731 EDT [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 004 Writing anchor peer update
linuxmint@linuxmint201:~/Desktop/tutorial/acme-network$
linuxmint@linuxmint201:~/Desktop/tutorial/acme-network$    configtxgen -profile ThreeOrgsChannel  -outputAnchorPeersUpdate  ./channel-artifacts/Org2MSPanchors.tx -channelID marketplace -asOrg Org2MSP
2021-07-08 15:42:15.862 EDT [common.tools.configtxgen] main -> INFO 001 Loading configuration
2021-07-08 15:42:15.871 EDT [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: configtx.yaml
2021-07-08 15:42:15.871 EDT [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Generating anchor peer update
2021-07-08 15:42:15.873 EDT [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 004 Writing anchor peer update
linuxmint@linuxmint201:~/Desktop/tutorial/acme-network$
linuxmint@linuxmint201:~/Desktop/tutorial/acme-network$    configtxgen -profile ThreeOrgsChannel  -outputAnchorPeersUpdate  ./channel-artifacts/Org3MSPanchors.tx -channelID marketplace -asOrg Org3MSP
2021-07-08 15:42:21.139 EDT [common.tools.configtxgen] main -> INFO 001 Loading configuration
2021-07-08 15:42:21.148 EDT [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: configtx.yaml
2021-07-08 15:42:21.148 EDT [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Generating anchor peer update
2021-07-08 15:42:21.188 EDT [common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 004 Writing anchor peer update
```

Figura 31 - Comandos configtxgen utilizados

3) Depois, usou-se *Docker* para criar um *container* com uma BD *couchDB* para cada organização, usando a informação presente em *docker-compose-base.yaml* (informação relativa ao containers das organizações) e *docker-compose-clicouchdb.yaml* (informação dos containers, BD, *Certificate Authority* (CA) e *cli* que é o container onde se irá interagir com a rede).

```
# ---CHANGED--- The peer name is taken from the name generated by the "cryptogen" certs — it indicates the peer org 3 and one peer "peer0"
peer0.org3.acme.com:
  # ---CHANGED--- Container name — same as the peer name
  container_name: peer0.org3.acme.com
  extends:
    file: peer-base.yaml
    service: peer-base
  environment:
    # ---CHANGED--- changed to reflect peer name, org name and our company's domain
    - CORE_PEER_ID=peer0.org3.acme.com
    # ---CHANGED--- changed to reflect peer name, org name and our company's domain
    - CORE_PEER_ADDRESS=peer0.org3.acme.com:7051
    # ---CHANGED--- changed to reflect peer name, org name and our company's domain
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org3.acme.com:7051
    # ---CHANGED--- changed to reflect peer name, org name and our company's domain
    - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org3.acme.com:7051
    # ---CHANGED--- ensure that the MSP ID is correctly set of Org3
    - CORE_PEER_LOCALMSPID=Org3MSP
  volumes:
      - /var/run/:/host/var/run/
      # ---CHANGED--- changed to reflect peer name, org name and our company's domain
      - ../crypto-config/peerOrganizations/org3.acme.com/peers/peer0.org3.acme.com/msp:/etc/hyperledger/fabric/msp
      # ---CHANGED--- changed to reflect peer name, org name and our company's domain
      - ../crypto-config/peerOrganizations/org3.acme.com/peers/peer0.org3.acme.com/tls:/etc/hyperledger/fabric/tls
  ports:
    - 9051:7051
    - 9053:7053
```

Figura 32 - Excerto de docker-compose-base.yaml relativo à terceira organização

4) Utilizando o *container cli* previamente inicializado começou-se a criar o canal, recorrendo ao comando *peer channel create* [28] usando a organização *orderer* (*Org1*) para o gerar.

5) Posteriormente utiliza-se *peer channel join* em cada uma das organizações para as juntar ao canal criado.

6) Após isso, fez-se uso de *peer channel update* com o intuito de definir os *Anchor Peers* (usado para *peers* de diferentes organizações reconhecerem a existência uns dos outros). Neste caso para todas as organizações tendo em conta a necessidade de alta disponibilidade e redundância.

7) Passando para a implementação do *chaincode*, usa-*se peer lifecycle chaincode package*, onde utilizando *scripts* criados da linguagem *go* (que descrevem o comportamento no *chaincode*, o formato das mensagens e funções a serem invocadas mais à frente), cria-se um ficheiro comprimido do tipo *tar.gz* com o *chaincode*, que é instalado usando *peer lifecycle chaincode install* em cada uma das organizações.

```
func (s *SmartContract) Query(ctx contractapi.TransactionContextInterface, foodId string) (*Food, error) {

    foodAsBytes, err := ctx.GetStub().GetState(foodId)

    if err != nil {
        return nil, fmt.Errorf("Failed to read from world state. %s", err.Error())
    }

    if foodAsBytes == nil {
        return nil, fmt.Errorf("%s does not exist", foodId)
    }

    food := new(Food)

    err = json.Unmarshal(foodAsBytes, food)
    if err != nil {
        return nil, fmt.Errorf("Unmarshal error. %s", err.Error())
    }

    return food, nil
}
```

Figura 33 - Excerto de ficheiro .go usado na criação da função query

8) Após a instalação do *chaincode* em cada organização tomou-se a decisão de quais vão ter acesso de escrita. Para tal usa-se *peer lifecycle chaincode approveformyorg* para aprovar a definição do *chaincode*, em cada uma das organizações. Com este comando, ao definir as *signature policies* apenas para duas organizações, a terceira ficará impedida de escrever para o canal, algo que se pode observar com *peer lifecycle chaincode checkcommitreadiness*.

9) Finalmente para fazer *commit* do *chaincode* criado até aí, utiliza-se *peer lifecycle chaincode commit*.

10) Para testar o que foi implementado usam-se as funções presentes no *chaincode go* denominadas: *set* (para escrever no canal) e *query* (para ler do canal), onde se verificou que uma das organizações não tem autorização para escrever apesar de todas poderem ler do canal criado.

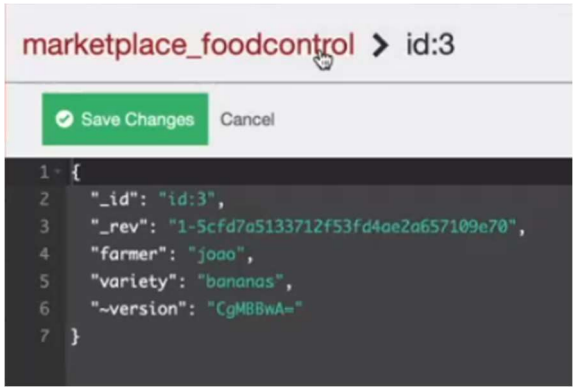


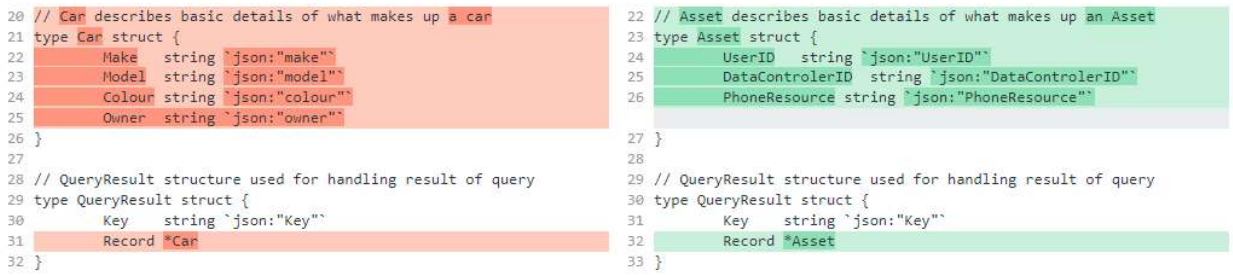
Figura 34 - Exemplo de valor introduzido na BD utilizando a função set

5.2.3.2 Implementação fazendo uso de scripts de fabric-samples com chaincode modificado

Nesta implementação existiu um foco na mudança do *chaincode*, tendo-se como objetivo correr a rede pré-existente em *fabric-samples* com esse *chaincode* novo, implementado em linguagem Go.

De seguida explica-se o processo:

1) Em *fabric-samples* deverá aceder-se à diretoria *./chaincode/fabcar/go* de modo a poder modificar o ficheiro do *chaincode* em linguagem *Go* denominado *fabcar.go* da maneira representada nas figuras 34-38 , sendo o código original o assinalado a vermelho e o novo código a verde.



```
// Car describes basic details of what makes up a car
type Car struct {
        Make   string `json:"make"`
        Model  string `json:"model"`
        Colour string `json:"colour"`
        Owner  string `json:"owner"`
}

// QueryResult structure used for handling result of query
type QueryResult struct {
        Key    string `json:"Key"`
        Record *Car
}
```

```
// Asset describes basic details of what makes up an Asset
type Asset struct {
        UserID   string `json:"UserID"`
        DataControlerID  string `json:"DataControlerID"`
        PhoneResource string `json:"PhoneResource"`

}

// QueryResult structure used for handling result of query
type QueryResult struct {
        Key    string `json:"Key"`
        Record *Asset
}
```

Figura 35 - Excerto do ficheiro fabcar.go relativo à estrutura dos dados

Ir-se-á modificar a estrutura e o seu nome de modo a passar a receber *UserID*, *DataControlerID* e *PhoneResource*.

```
// InitLedger adds a base set of cars to the ledger
func (s *SmartContract) InitLedger(ctx contractapi.TransactionContextInterface) error {
        cars := []Car{
                Car{Make: "Toyota", Model: "Prius", Colour: "blue", Owner: "Tomoko"},
                Car{Make: "Ford", Model: "Mustang", Colour: "red", Owner: "Brad"},
                Car{Make: "Hyundai", Model: "Tucson", Colour: "green", Owner: "Jin Soo"},
                Car{Make: "Volkswagen", Model: "Passat", Colour: "yellow", Owner: "Max"},
                Car{Make: "Tesla", Model: "S", Colour: "black", Owner: "Adriana"},
                Car{Make: "Peugeot", Model: "205", Colour: "purple", Owner: "Michel"},
                Car{Make: "Chery", Model: "S22L", Colour: "white", Owner: "Aarav"},
                Car{Make: "Fiat", Model: "Punto", Colour: "violet", Owner: "Pari"},
                Car{Make: "Tata", Model: "Nano", Colour: "indigo", Owner: "Valeria"},
                Car{Make: "Holden", Model: "Barina", Colour: "brown", Owner: "Shotaro"},
        }

        for i, car := range cars {
                carAsBytes, _ := json.Marshal(car)
                err := ctx.GetStub().PutState("CAR"+strconv.Itoa(i), carAsBytes)

                if err != nil {
                        return fmt.Errorf("Failed to put to world state. %s", err.Error())
                }
        }

        return nil
}
```

```
// InitLedger adds a base set of info to the ledger
func (s *SmartContract) InitLedger(ctx contractapi.TransactionContextInterface) error {
        db_info := []Asset{
                Asset{UserID: "1", DataControlerID: "1", PhoneResource: "camera & microphone"},
                Asset{UserID: "2", DataControlerID: "2", PhoneResource: "camera & microphone"},
                Asset{UserID: "3", DataControlerID: "3", PhoneResource: "microphone"},
                Asset{UserID: "4", DataControlerID: "4", PhoneResource: "camera & microphone"},
                Asset{UserID: "5", DataControlerID: "5", PhoneResource: "camera & microphone"},
                Asset{UserID: "6", DataControlerID: "3", PhoneResource: "camera & microphone"},
                Asset{UserID: "7", DataControlerID: "4", PhoneResource: "camera & microphone"},
                Asset{UserID: "8", DataControlerID: "2", PhoneResource: "camera & microphone"},
                Asset{UserID: "9", DataControlerID: "1", PhoneResource: "camera & microphone"},
                Asset{UserID: "10", DataControlerID: "3", PhoneResource: "camera & microphone"},
        }

        for i, asset := range db_info {
                assetAsBytes, _ := json.Marshal(asset)
                err := ctx.GetStub().PutState("ASSET"+strconv.Itoa(i), assetAsBytes)

                if err != nil {
                        return fmt.Errorf("Failed to put to world state. %s", err.Error())
                }
        }

        return nil
}
```

Figura 36 - Excerto do ficheiro fabcar.go relativo à inicialização do ledger

Modifica-se a inicialização para popular a base de dados com valores relevantes.

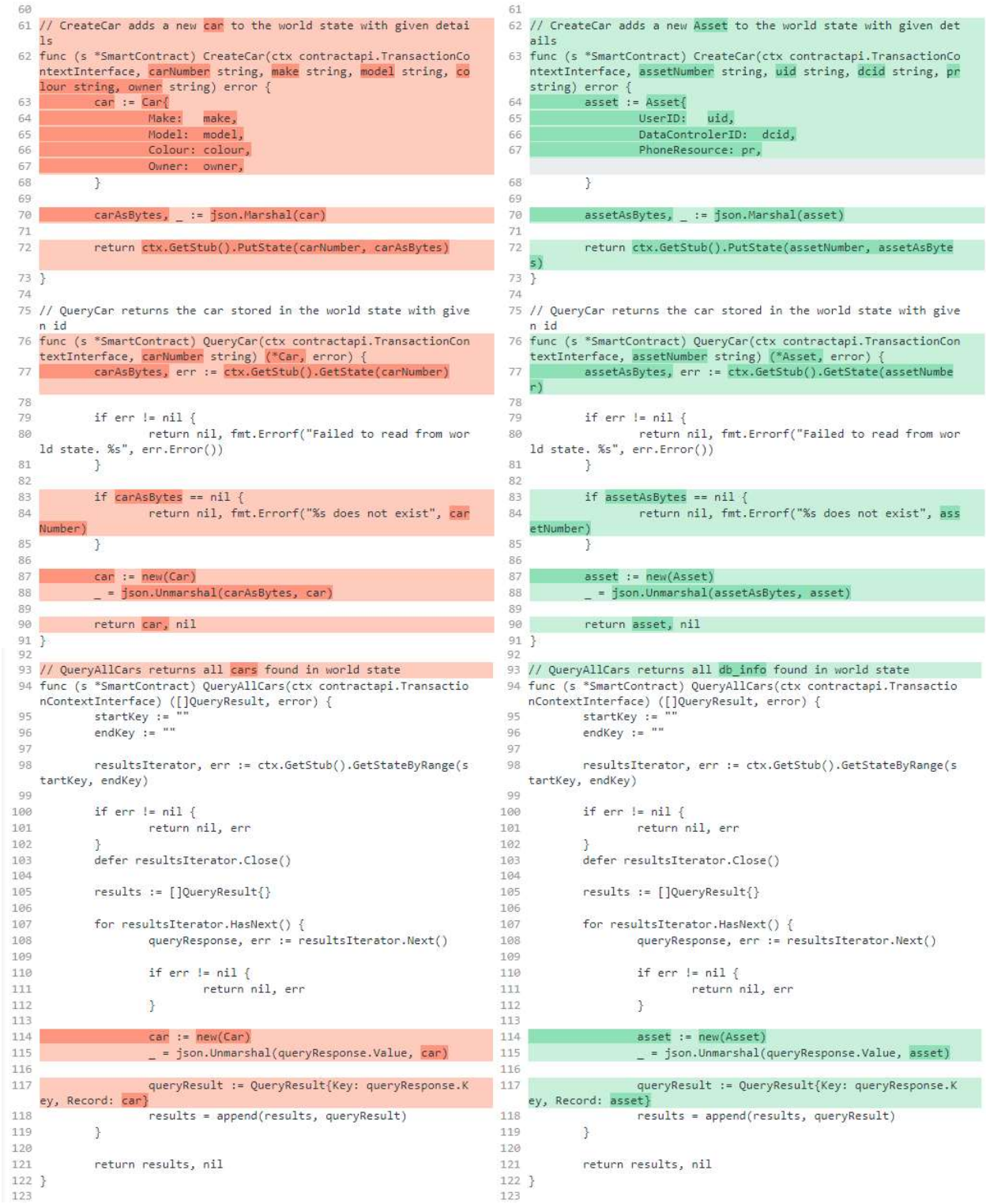


Figura 37 - Excerto do ficheiro fabcar.go com diversas funções

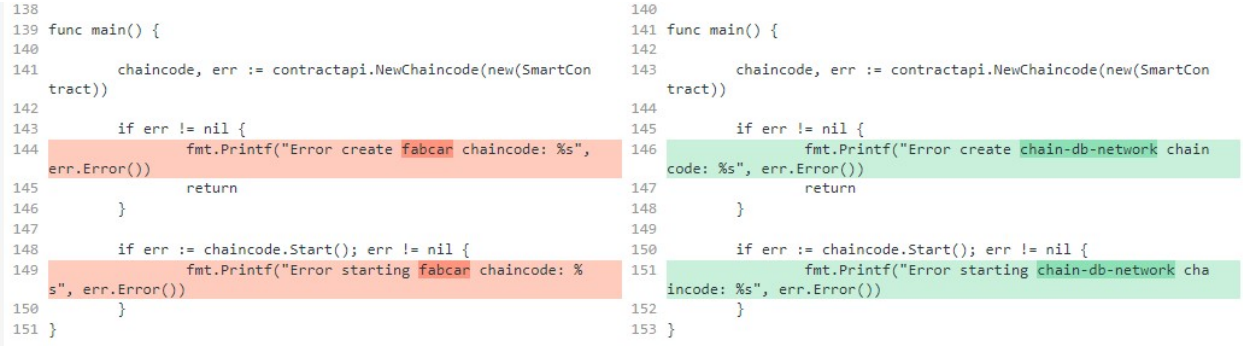
Muda-se a denominação das variáveis a utilizar nas diferentes funções.

```
123
124 // ChangeCarOwner updates the owner field of car with given id in world state
125 func (s *SmartContract) ChangeCarOwner(ctx contractapi.TransactionContextInterface, carNumber string, newOwner string) error {
126         car, err := s.QueryCar(ctx, carNumber)
127
128         if err != nil {
129                 return err
130         }
131
132         car.Owner = newOwner
133
134         carAsBytes, _ := json.Marshal(car)
135
136         return ctx.GetStub().PutState(carNumber, carAsBytes)
137 }
138
```

```
123
124 /* There are no owners
125 // ChangeCarOwner updates the owner field of car with given id in world state
126 func (s *SmartContract) ChangeCarOwner(ctx contractapi.TransactionContextInterface, assetNumber string, newOwner string) error {
127         asset, err := s.QueryCar(ctx, assetNumber)
128
129         if err != nil {
130                 return err
131         }
132
133         asset.Owner = newOwner
134
135         assetAsBytes, _ := json.Marshal(asset)
136
137         return ctx.GetStub().PutState(assetNumber, assetAsBytes)
138 }
139 */
140
```

Figura 38 - Excerto do ficheiro fabcar.go relativo à função que realiza a mudança de owner

No caso desta implementação não é necessário ter a função para mudar de *owner* (uma vez que, com a mudança da estrutura, esse valor já não é representado), sendo eliminada, ou seja, comentada.



```
138
139 func main() {
140
141         chaincode, err := contractapi.NewChaincode(new(SmartContract))
142
143         if err != nil {
144                 fmt.Printf("Error create fabcar chaincode: %s", err.Error())
145                 return
146         }
147
148         if err := chaincode.Start(); err != nil {
149                 fmt.Printf("Error starting fabcar chaincode: %s", err.Error())
150         }
151 }
```

```
140
141 func main() {
142
143         chaincode, err := contractapi.NewChaincode(new(SmartContract))
144
145         if err != nil {
146                 fmt.Printf("Error create chain-db-network chaincode: %s", err.Error())
147                 return
148         }
149
150         if err := chaincode.Start(); err != nil {
151                 fmt.Printf("Error starting chain-db-network chaincode: %s", err.Error())
152         }
153 }
```

Figura 39 - Excerto do final do ficheiro fabcar.go

Muda-se de novo os nomes das variáveis para apresentar mensagens de erro mais adequadas.

2) Voltando à diretoria *fabric-samples* vai-se para *fabcar* e modifica-se *startFabric.sh* (script usado na inicialização da rede) da seguinte maneira:

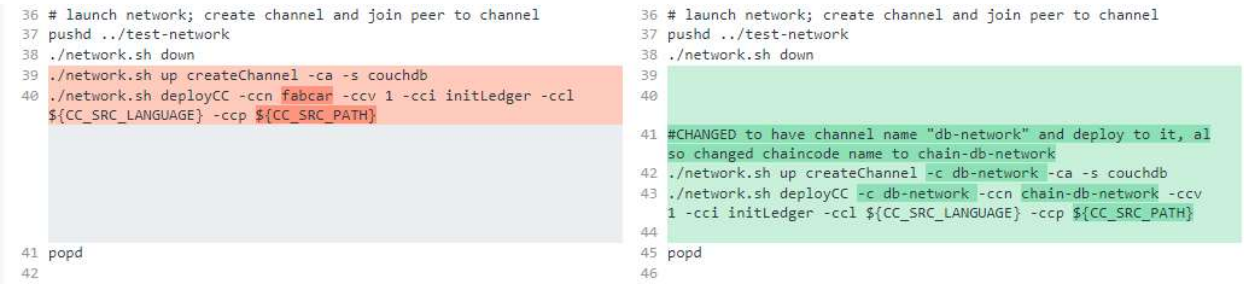


Figura 40 - Excerto do ficheiro startFabric.sh relativo à inicialização da network e do chaincode

Para atribuir o nome *db-network* ao *channel* e o nome *chain-db-network* ao *chaincode*. Caso se pretendam nomes diferentes é fulcral ter em conta que os nomes não podem incluir *underscore* (“_”).

```
cat <<EOF

Total setup execution time : $(($(date +%s) - starttime)) secs
...

Next, use the FabCar applications to interact with the deployed
FabCar contract.
The FabCar applications are available in multiple programming la
nguages.
Follow the instructions for the programming language of your cho
ice:

JavaScript:

  Start by changing into the "javascript" directory:
    cd javascript

  Next, install all required packages:
    npm install

  Then run the following applications to enroll the admin user,
and register a new user
  called appUser which will be used by the other applications to
interact with the deployed
  FabCar contract:
    node enrollAdmin
    node registerUser

  You can run the invoke application as follows. By default, the
invoke application will
  create a new car, but you can update the application to submit
other transactions:
    node invoke

  You can run the query application as follows. By default, the
query application will
  return all cars, but you can update the application to evaluat
e other transactions:
    node query

TypeScript:

  Start by changing into the "typescript" directory:
    cd typescript

  Next, install all required packages:

    npm install


  Next, compile the TypeScript code into JavaScript:
    npm run build


  Then run the following applications to enroll the admin user,
and register a new user
  called appUser which will be used by the other applications to
interact with the deployed
  FabCar contract:
    node dist/enrollAdmin
    node dist/registerUser

  You can run the invoke application as follows. By default, the
invoke application will
  create a new car, but you can update the application to submit
other transactions:
    node dist/invoke

  You can run the query application as follows. By default, the
query application will
  return all cars, but you can update the application to evaluat
e other transactions:
    node dist/query

Java:
```

```
cat <<EOF

Total setup execution time : $(($(date +%s) - starttime)) secs
...

Next, use the FabCar applications to interact with the deployed
chain-db-network contract.
Follow the instructions:


JavaScript:

  Start by changing into the "javascript" directory:
    cd javascript

  Next, install all required packages:
    npm install

  Then run the following applications to enroll the admin user,
and register a new user
  called appUser which will be used by the other applications to
interact with the deployed contract:
    node enrollAdmin.js
    node registerUser.js


  You can run the invoke application as follows. Which will crea
te a new database entry
  with the information provided by the user:
    node invoke.js <UserID> <DataControlerID> <PhoneResources>
    ex: node invoke.js 13 14 microphone

  You can run the query application as follows.
  In this function you can either query the whole database as it
is presented:
    node query.js
  Or you can query a specific Asset from the database:

    node query.js <Key>

    ex: node query ASSET4
```

Figura 41 - Excerto do ficheiro startFabric.sh relativo às instruções de uso da aplicação em JavaScript

Para apresentar diferentes instruções de uso ao correr o *script*. Adicionalmente retiram-se as instruções de uso relativas a todas as linguagens de programação, exceto *JavaScript*, uma vez que será essa a linguagem utilizada.

3) Ainda na diretoria *fabcar* corre-se o comando `./startFabric.sh`. Este script faz uso de *network.sh* presente na diretoria *fabric-samples/test-network* de modo a apagar conteúdos de redes pré-existentes e inicializar a nova rede com *channel* denominado

db-network. Adicionalmente, é instalado o *chaincode chain-db-network* em 2 organizações.

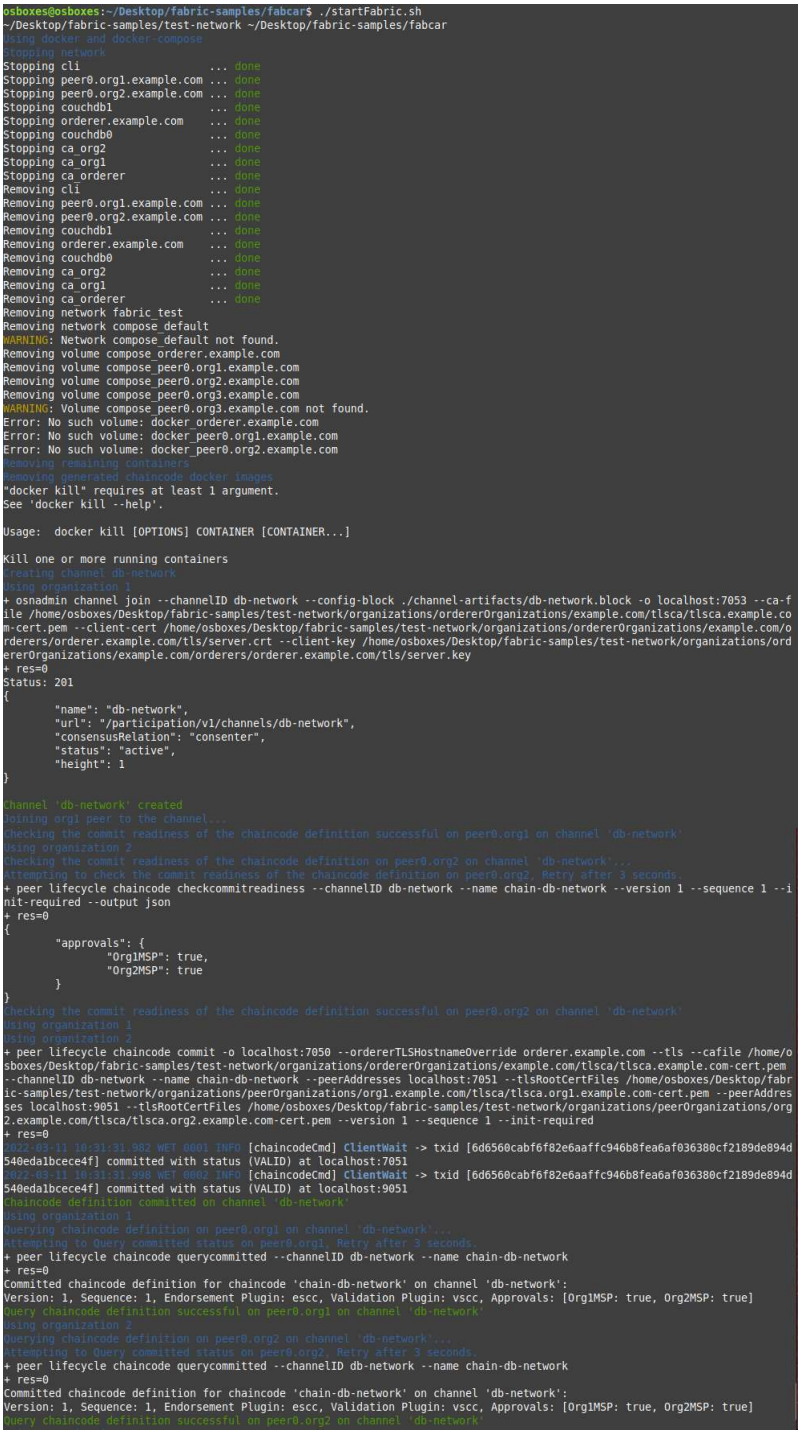


Figura 42 - Conjunto de excertos relativos à execução do script startFabric.sh

4) Depois da criação da rede, ainda em *fabcar*, na diretoria *JavaScript*, *query.js* modifica-se da seguinte maneira.

```
36
37        // Get the network (channel) our contract is deployed t
o.
38        const network = await gateway.getNetwork('mychannel');
39
40        // Get the contract from the network.
41        const contract = network.getContract('fabcar');
42
43        // Evaluate the specified transaction.
44        // queryCar transaction - requires 1 argument, ex: ('qu
eryCar', 'CAR4')
45        // queryAllCars transaction - requires no arguments, e
x: ('queryAllCars')
46        const result = await contract.evaluateTransaction('quer
yAllCars');
47        console.log(`Transaction has been evaluated, result is:
${result.toString()}`);
48
```

```
36
37        //CHANGED network name from mychannel to db-network
38        // Get the network (channel) our contract is deployed t
o.
39        const network = await gateway.getNetwork('db-network');
40
41        //CHANGED network name from fabcar to chain-db-network
42        // Get the contract from the network.
43        const contract = network.getContract('chain-db-networ
k');
44
45        // Evaluate the specified transaction.
46        // queryCar transaction - requires 1 argument, ex: node
query.js ASSET4
47        // queryAllCars transaction - requires no arguments, no
de query.js
48
49        //CHANGED in order to be able to choose what query to u
se in command line
50        const myArgs = process.argv.slice(2);
51        if (myArgs[0] !=null){
52            const result = await contract.evaluateTransaction
('QueryCar', myArgs[0]);
53            console.log(`Transaction has been evaluated, result
is: ${result.toString()}`);
54        } else if (myArgs[0] ==null){
55            const result = await contract.evaluateTransaction
('QueryAllCars');
56            console.log(`Transaction has been evaluated, result
is: ${result.toString()}`);
57        }
58        else{
59            console.log("To evaluate the specified transactio
n");
60            console.log("queryCar transaction - requires 1 argu
ment, ex: node query.js ASSET4");
61            console.log("queryAllCars transaction - requires no
arguments, node query.js");
62            console.log("Run query.js as specified above");
63        }
64
65
```

Figura 43 - Excerto do ficheiro query.js

Muda-se o nome da rede e *chaincode* para o que foi definido anteriormente. Modifica-se também o comportamento do código de modo a poder definir que tipo de *query* fazer e especificar o *ASSET* a ser pesquisado, tudo isto na linha de comandos.

5) Ainda na diretoria *JavaScript*, muda-se o ficheiro invoke.js consoante o apresentado de seguida.

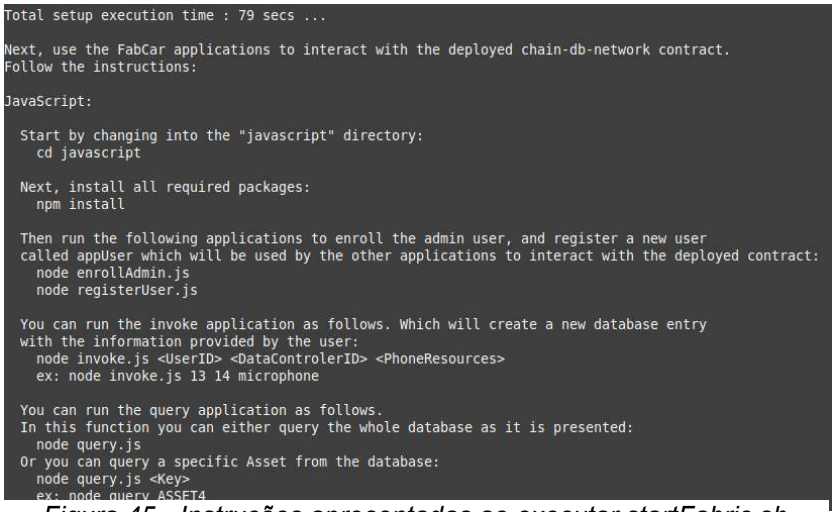
```
35
36        // Get the network (channel) our contract is deployed t
o.
37        const network = await gateway.getNetwork('mychannel');
38
39        // Get the contract from the network.
40        const contract = network.getContract('fabcar');
41
42        // Submit the specified transaction.
43        // createCar transaction - requires 5 argument, ex: ('cr
eateCar', 'CAR12', 'Honda', 'Accord', 'Black', 'Tom')
44        // changeCarOwner transaction - requires 2 args , ex:
('changeCarOwner', 'CAR12', 'Dave')
45        await contract.submitTransaction('createCar', 'CAR12',
'Honda', 'Accord', 'Black', 'Tom');
46        console.log('Transaction has been submitted');
47
48        // Disconnect from the gateway.
49        await gateway.disconnect();
50
51    } catch (error) {
52        console.error(`Failed to submit transaction: ${error}`);
53        process.exit(1);
54    }
55 }
```

```
36
37        //CHANGED network name from mychannel to db-network
38        // Get the network (channel) our contract is deployed t
o.
39        const network = await gateway.getNetwork('db-network');
40
41        //CHANGED network name from fabcar to chain-db-network
42        // Get the contract from the network.
43        const contract = network.getContract('chain-db-networ
k');
44
45        // Submit the specified transaction.
46        // createCar transaction - requires 3 arguments, ex: nod
e invoke.js 13 14 microphone)
47
48        //CHANGED to automatically choose a new key value taking
into account the last ASSET in the database
49        const result = await contract.evaluateTransaction('Query
AllCars');
50        let s_result=result.toString();
51        //Registers the number of times the string 'Key' occurs
in string 's_result'
52        var count=s_result.match(/Key/g).length;
53        let new_asset="ASSET"+ count;
54
55        //DEBUG CONSOLES
56        //console.log(result.toString());
57        //console.log(count);
58        //console.log(new_asset);
59
60        //CHANGED to take UserID, DatacontrolerID and phoneResou
rces directly in the terminal
61        const myArgs = process.argv.slice(2);
62        await contract.submitTransaction('createCar', new_asset,
myArgs[0], myArgs[1], myArgs[2]);
63        console.log('Transaction has been submitted');
64
65        // Disconnect from the gateway.
66        await gateway.disconnect();
67
68    } catch (error) {
69        console.error(`Failed to submit transaction: ${error}`);
70        process.exit(1);
71    }
72 }
```

Figura 44 - Excerto do ficheiro invoke.js

Em que se modifica o código de modo a criar uma *key* para o novo valor na base de dados e para poder especificar as informações do novo *ASSET* a ser criado no terminal.

6) Finalmente, basta apenas seguir as instruções apresentadas no fim de correr o script *startFabric.sh*.



```
Total setup execution time : 79 secs ...

Next, use the FabCar applications to interact with the deployed chain-db-network contract.
Follow the instructions:

JavaScript:

  Start by changing into the "javascript" directory:
    cd javascript

  Next, install all required packages:
    npm install

  Then run the following applications to enroll the admin user, and register a new user
  called appUser which will be used by the other applications to interact with the deployed contract:
    node enrollAdmin.js
    node registerUser.js

  You can run the invoke application as follows. Which will create a new database entry
  with the information provided by the user:
    node invoke.js <UserID> <DataControlerID> <PhoneResources>
    ex: node invoke.js 13 14 microphone

  You can run the query application as follows.
  In this function you can either query the whole database as it is presented:
    node query.js
  Or you can query a specific Asset from the database:
    node query.js <Key>
    ex: node query ASSET4
```

Figura 45 - Instruções apresentadas ao executar startFabric.sh

7) Após fazer `npm install`, na diretoria *./fabcar/javascript* (para instalar os packages necessários para correr os passos seguintes) utilizar `node enrollAdmin.js` e `node registerUser.js` para registar o *admin* e criar um novo utilizador para interagir com a aplicação.

```
osboxes@osboxes:~/Desktop/fabric-samples/fabcar$ cd javascript/
osboxes@osboxes:~/Desktop/fabric-samples/fabcar/javascript$ npm install
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'fabric-ca-client@2.2.11',
npm WARN EBADENGINE   required: { node: '^10.15.3 || ^12.13.1 || ^14.13.1', npm: '^6.4.1' },
npm WARN EBADENGINE   current: { node: 'v16.14.0', npm: '8.3.1' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'fabric-common@2.2.11',
npm WARN EBADENGINE   required: { node: '^10.15.3 || ^12.13.1 || ^14.13.1', npm: '^6.4.1' },
npm WARN EBADENGINE   current: { node: 'v16.14.0', npm: '8.3.1' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'fabric-network@2.2.11',
npm WARN EBADENGINE   required: { node: '^10.15.3 || ^12.13.1 || ^14.13.1', npm: '^6.4.1' },
npm WARN EBADENGINE   current: { node: 'v16.14.0', npm: '8.3.1' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'fabric-protos@2.2.11',
npm WARN EBADENGINE   required: { node: '^10.15.3 || ^12.13.1 || ^14.13.1', npm: '^6.4.1' },
npm WARN EBADENGINE   current: { node: 'v16.14.0', npm: '8.3.1' }
npm WARN EBADENGINE }

up to date, audited 345 packages in 1s

19 packages are looking for funding
  run `npm fund` for details

13 moderate severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
osboxes@osboxes:~/Desktop/fabric-samples/fabcar/javascript$ node enrollAdmin.js
Wallet path: /home/osboxes/Desktop/fabric-samples/fabcar/javascript/wallet
Successfully enrolled admin user "admin" and imported it into the wallet
osboxes@osboxes:~/Desktop/fabric-samples/fabcar/javascript$ node registerUser.js
Wallet path: /home/osboxes/Desktop/fabric-samples/fabcar/javascript/wallet
Successfully registered and enrolled admin user "appUser" and imported it into the wallet
osboxes@osboxes:~/Desktop/fabric-samples/fabcar/javascript$
```

Figura 46 - Execução de npm install, registo de admin e criação de novo utilizador

8) Finalmente pode-se usar node query.js e node invoke.js para utilizar as funcionalidades descritas anteriormente.

```
osboxes@osboxes:~/Desktop/fabric-samples/fabcar/javascript$ node query.js
Wallet path: /home/osboxes/Desktop/fabric-samples/fabcar/javascript/wallet
Transaction has been evaluated, result is: [{"Key":"ASSET0","Record":{"UserID":"1","DataControlerID":"1","PhoneResour
ce":"camera \u0026 microphone"}},{"Key":"ASSET1","Record":{"UserID":"2","DataControlerID":"2","PhoneResource":"camera
 \u0026 microphone"}},{"Key":"ASSET2","Record":{"UserID":"3","DataControlerID":"3","PhoneResource":"microphone"}},{"K
ey":"ASSET3","Record":{"UserID":"4","DataControlerID":"4","PhoneResource":"camera \u0026 microphone"}},{"Key":"ASSET4
","Record":{"UserID":"5","DataControlerID":"5","PhoneResource":"camera \u0026 microphone"}},{"Key":"ASSET5","Record":
{"UserID":"6","DataControlerID":"3","PhoneResource":"camera \u0026 microphone"}},{"Key":"ASSET6","Record":{"UserID":"
7","DataControlerID":"4","PhoneResource":"camera \u0026 microphone"}},{"Key":"ASSET7","Record":{"UserID":"8","DataCon
trolerID":"2","PhoneResource":"camera \u0026 microphone"}},{"Key":"ASSET8","Record":{"UserID":"9","DataControlerID":"
1","PhoneResource":"camera \u0026 microphone"}},{"Key":"ASSET9","Record":{"UserID":"10","DataControlerID":"3","PhoneR
esource":"camera \u0026 microphone"}}]
osboxes@osboxes:~/Desktop/fabric-samples/fabcar/javascript$ node invoke.js 13 14 microphone
Wallet path: /home/osboxes/Desktop/fabric-samples/fabcar/javascript/wallet
Transaction has been submitted
osboxes@osboxes:~/Desktop/fabric-samples/fabcar/javascript$ node query.js ASSET10
Wallet path: /home/osboxes/Desktop/fabric-samples/fabcar/javascript/wallet
Transaction has been evaluated, result is: {"UserID":"13","DataControlerID":"14","PhoneResource":"microphone"}
osboxes@osboxes:~/Desktop/fabric-samples/fabcar/javascript$
```

Figura 47 - Execução de query.js e invoke.js

5.2.3.3 Implementação de uma rede com quatro organizações sem scripts

Nesta implementação criou-se a seguinte estrutura de diretorias.

Figura 48 - Estrutura de diretorias criada

A diretoria *chaincode* contém o código *chaincode* a correr em cada organização, passo esse que será descrito mais á frente.

Na diretoria *jnvr-network* está toda a configuração da rede. Entrando nessa diretoria observa-se o seguinte:

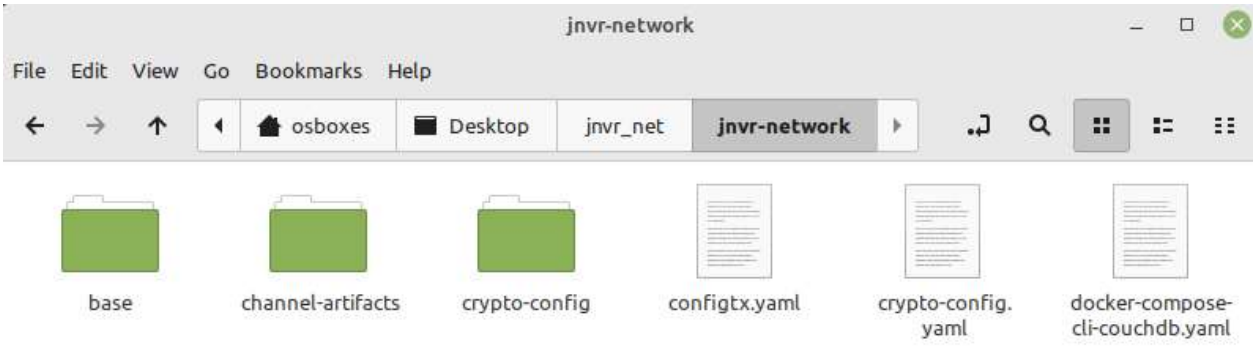


Figura 49 - Estrutura da diretoria jnvr-network

A configuração é feita seguindo os seguintes passos a partir da diretoria apresentada anteriormente:

1) Inicialmente irá ser necessário definir o comportamento do ficheiro *crypto-config.yaml*, da seguinte forma.

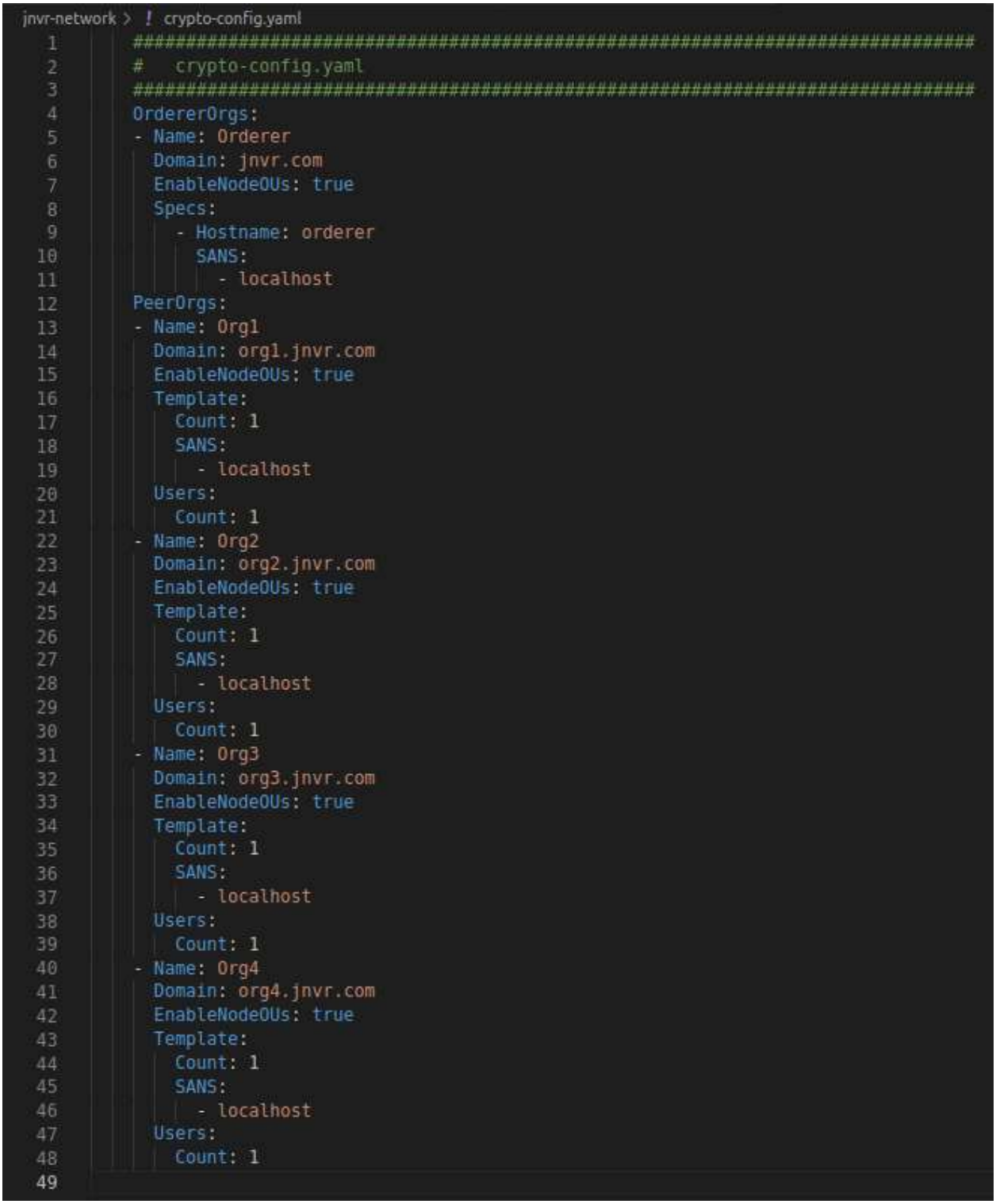


Figura 50 - Ficheiro crypto-config.yaml

No ficheiro *crypto-config.yaml* define-se as organizações da rede. No caso da implementação existe uma organização *Orderer* e quatro organizações *Peer*. Em cada um explicita-se o nome, o domínio da organização, o número de utilizadores e *EnableNodeOUs* é colocado *true* para permitir a classificação de identidades. Neste caso são utilizados *peer*, *client*, *admin* e *member* [29], como se verifica futuramente.

Após ter o ficheiro preparado, corre-se o comando `cryptogen generate --config=./crypto-config.yaml` para criar materiais chave da rede.

```
osboxes@osboxes:~/Desktop/jnvr_net/jnvr-network$ cryptogen generate --config=./crypto-config.yaml
org1.jnvr.com
org2.jnvr.com
org3.jnvr.com
org4.jnvr.com
```

Figura 51 - Execução do comando cryptogen generate

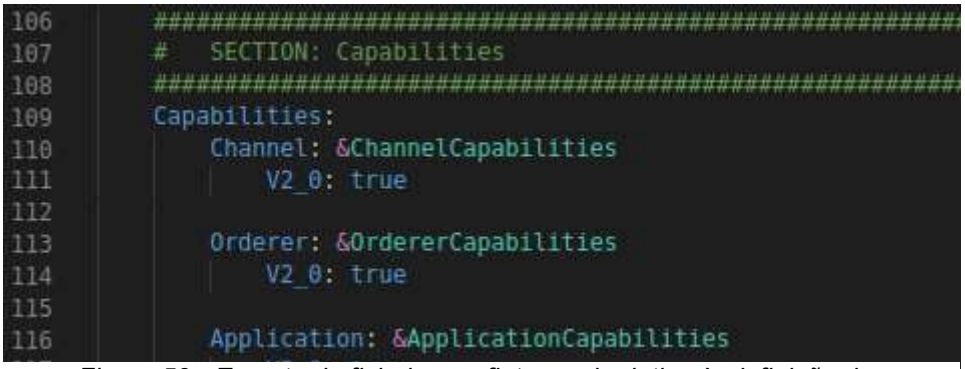
2) No próximo passo ir-se-á criar artefactos relacionados à configuração do canal. Para isso, vai ser utilizado o comando *configtxgen* de diversas maneiras. Para utilizar estes comandos irá ser necessária a criação de *configtx.yaml*, que descreve toda a rede e o seu comportamento.

   Identifica-se cada organização e denomina-se-lhes um ID.

```
22
23          - &Org1
24              Name: Org1MSP
25              ID: Org1MSP
26              MSPDir: crypto-config/peerOrganizations/org1.jnvr.com/msp
27              Policies:
28                  Readers:
29                      Type: Signature
30                      Rule: "OR('Org1MSP.admin', 'Org1MSP.peer', 'Org1MSP.client')"
31                  Writers:
32                      Type: Signature
33                      Rule: "OR('Org1MSP.admin', 'Org1MSP.client')"
34                  Admins:
35                      Type: Signature
36                      Rule: "OR('Org1MSP.admin')"
37                  Endorsement:
38                      Type: Signature
39                      Rule: "OR('Org1MSP.peer')"
40              AnchorPeers:
41                  - Host: peer0.org1.jnvr.com
42                    Port: 7051
```

Figura 52 - Excerto do ficheiro configtx.yaml relativo à identificação da organização 1

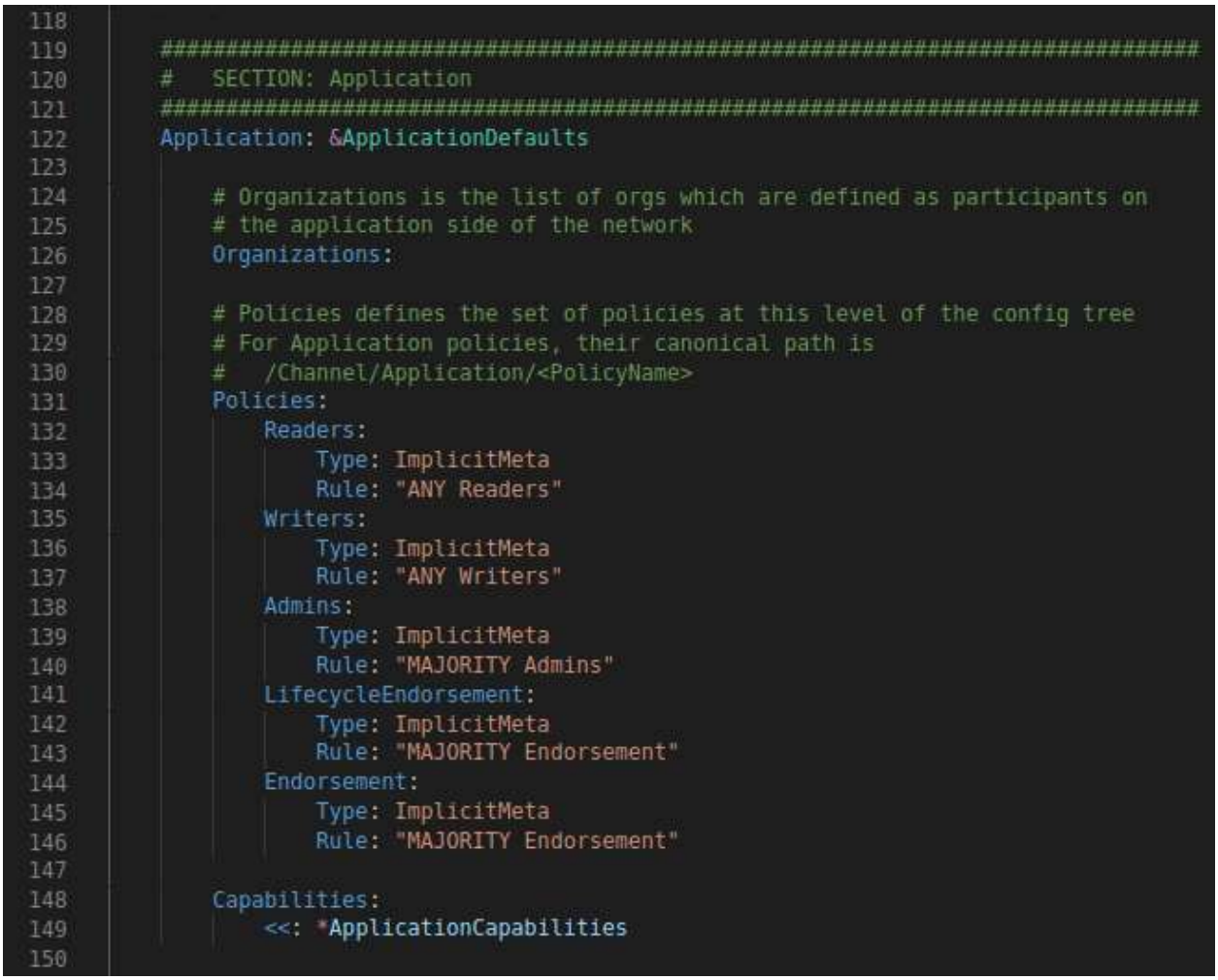
Apresentam-se de seguida as capacidades do canal, que irão ser definidas nas próximas figuras 53-55.



```
106     ################################################################
107     #   SECTION: Capabilities
108     ################################################################
109     Capabilities:
110         Channel: &ChannelCapabilities
111             V2_0: true
112
113         Orderer: &OrdererCapabilities
114             V2_0: true
115
116         Application: &ApplicationCapabilities
```

Figura 53 - Excerto do ficheiro configtx.yaml relativo à definição das capacidades do canal

Uma das definições de capacidades é a secção *Application*, que identifica os recursos usados pelos *peer nodes*.



```
118
119     ################################################################################
120     #   SECTION: Application
121     ################################################################################
122     Application: &ApplicationDefaults
123
124         # Organizations is the list of orgs which are defined as participants on
125         # the application side of the network
126         Organizations:
127
128         # Policies defines the set of policies at this level of the config tree
129         # For Application policies, their canonical path is
130         #   /Channel/Application/<PolicyName>
131         Policies:
132             Readers:
133                 Type: ImplicitMeta
134                 Rule: "ANY Readers"
135             Writers:
136                 Type: ImplicitMeta
137                 Rule: "ANY Writers"
138             Admins:
139                 Type: ImplicitMeta
140                 Rule: "MAJORITY Admins"
141             LifecycleEndorsement:
142                 Type: ImplicitMeta
143                 Rule: "MAJORITY Endorsement"
144             Endorsement:
145                 Type: ImplicitMeta
146                 Rule: "MAJORITY Endorsement"
147
148         Capabilities:
149             <<: *ApplicationCapabilities
150
```

Figura 54 - Excerto do ficheiro configtx.yaml relativo à definição da secção da aplicação

Outra das definições de capacidades é a secção *Orderer*, usada para identificar os recursos usados pelos *orderer nodes*.

```
150
151     ################################################################################
152     #   SECTION: Orderer
153     ################################################################################
154     Orderer: &OrdererDefaults
155         # Orderer Type: The orderer implementation to start
156         OrdererType: solo
157
158     #    OrdererType: etcdraft
159
160         EtcdRaft:
161             Consenters:
162             - Host: orderer.jnvr.com
163               Port: 7050
164               ClientTLSCert: ../organizations/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/tls/server.crt
165               ServerTLSCert: ../organizations/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/tls/server.crt
166
167         Addresses:
168             - orderer.jnvr.com:7050
169         BatchTimeout: 2s
170         BatchSize:
171             MaxMessageCount: 10
172             AbsoluteMaxBytes: 99 MB
173             PreferredMaxBytes: 512 KB
174
175         Kafka:
176             Brokers:
177                 - 127.0.0.1:9092
178         Organizations:
179
180         Policies:
181             Readers:
182                 Type: ImplicitMeta
183                 Rule: "ANY Readers"
184             Writers:
185                 Type: ImplicitMeta
186                 Rule: "ANY Writers"
187             Admins:
188                 Type: ImplicitMeta
189                 Rule: "MAJORITY Admins"
190             # BlockValidation specifies what signatures must be included in the block
191             # from the orderer for the peer to validate it.
192             BlockValidation:
193                 Type: ImplicitMeta
194                 Rule: "ANY Writers"
195
```

Figura 55 - Excerto do ficheiro configtx.yaml relativo à definição dos orderers

A última definição das capacidades do canal é a secção *Channel*, que define os valores a codificar para parâmetros relativos ao canal.

```
196  ################################################################################
197  #
198  #   CHANNEL
199  #
200  #   This section defines the values to encode into a config transaction or
201  #   genesis block for channel related parameters.
202  #
203  ################################################################################
204  Channel: &ChannelDefaults
205      # Policies defines the set of policies at this level of the config tree
206      # For Channel policies, their canonical path is
207      #   /Channel/<PolicyName>
208      Policies:
209          # Who may invoke the 'Deliver' API
210          Readers:
211              Type: ImplicitMeta
212              Rule: "ANY Readers"
213          # Who may invoke the 'Broadcast' API
214          Writers:
215              Type: ImplicitMeta
216              Rule: "ANY Writers"
217          # By default, who may modify elements at this config level
218          Admins:
219              Type: ImplicitMeta
220              Rule: "MAJORITY Admins"
221
222      # Capabilities describes the channel level capabilities, see the
223      # dedicated Capabilities section elsewhere in this file for a full
224      # description
225      Capabilities:
226          <<: *ChannelCapabilities
```

Figura 56 - Excerto do ficheiro configtx.yaml relativo à definição do canal

Finalmente define-se as políticas que são utilizadas pelo comando *configtxgen* na criação do canal e seus elementos.

```
227
228      ################################################################################
229      #   Profile
230      ################################################################################
231      Profiles:
232
233          FourOrgsOrdererGenesis:
234              <<: *ChannelDefaults
235              Orderer:
236                  <<: *OrdererDefaults
237                  Organizations:
238                      - *OrdererOrg
239                  Capabilities:
240                      <<: *OrdererCapabilities
241              Consortiums:
242                  SampleConsortium:
243                      Organizations:
244                          - *Org1
245                          - *Org2
246                          - *Org3
247                          - *Org4
248          FourOrgsChannel:
249              Consortium: SampleConsortium
250              <<: *ChannelDefaults
251              Application:
252                  <<: *ApplicationDefaults
253                  Organizations:
254                      - *Org1
255                      - *Org2
256                      - *Org3
257                      - *Org4
258                  Capabilities:
259                      <<: *ApplicationCapabilities
```

Figura 57 - Excerto do ficheiro configtx.yaml relativo à definição das politicas do canal

Após a configuração de *configtx.yaml* usa-se o seguinte comando , para criar o bloco de génese do canal.

```
configtxgen -profile FourOrgsOrdererGenesis -channelID system-channel
-outputBlock ./channel-artifacts/genesis.block
```

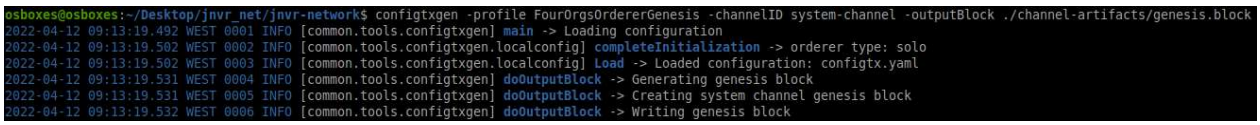


Figura 58 - Criação do bloco de génese

O comando seguinte irá ser utilizado para criar o canal, que neste caso se denomina *channel1*.

```
configtxgen -profile FourOrgsChannel -outputCreateChannelTx
./channel-artifacts/channel.tx -channelID channel1,
```

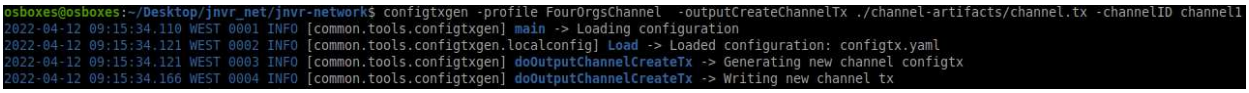


Figura 59 - Criação do canal channel1

Os comandos apresentados subsequentemente são usados para criar os *anchor peers* de cada organização.

```
configtxgen -profile FourOrgsChannel -outputAnchorPeersUpdate
./channel-artifacts/Org1MSPanchors.tx -channelID channel1 -asOrg
Org1MSP
```

```
configtxgen -profile FourOrgsChannel -outputAnchorPeersUpdate
./channel-artifacts/Org2MSPanchors.tx -channelID channel1 -asOrg
Org2MSP
```

```
configtxgen -profile FourOrgsChannel -outputAnchorPeersUpdate
./channel-artifacts/Org3MSPanchors.tx -channelID channel1 -asOrg
Org3MSP
```

```
configtxgen -profile FourOrgsChannel -outputAnchorPeersUpdate
./channel-artifacts/Org4MSPanchors.tx -channelID channel1 -asOrg
Org4MSP
```

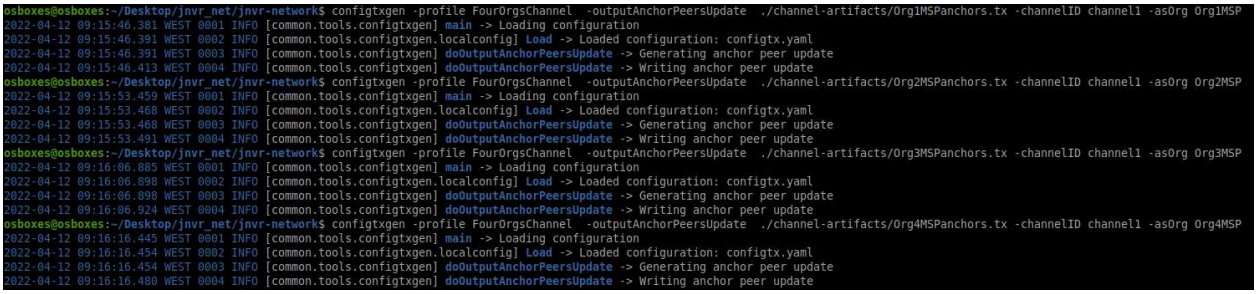


Figura 60 - Criação dos anchor peers em todas as organizações

3) Passando para a criação dos containers *Docker* onde a rede e o *channel* correm. É necessário criar o documento *docker-compose-cli-couchdb.yaml*, que define o nome da rede, estabelece informação relativa a todos os elementos da rede, inclui todas as organizações, fixa o CA da *Org1*, determina os *couchdb* (utilizados para armazenar a informação de cada organização) e descreve o container *cli* (usado na interação com a rede) que estará a correr na *Org1*.

```
85    peer0.org4.jnvr.com:
86      # ---CHANGED--- Container name - same as the peer name
87      container_name: peer0.org4.jnvr.com
88      extends:
89        file:  base/docker-compose-base.yaml
90        # ---CHANGED--- Refers to peer name
91        service: peer0.org4.jnvr.com
92      environment:
93        - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
94        - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb3:5984
95        - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
96        - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
97      depends_on:
98        - orderer.jnvr.com
99        - couchdb3
100     networks:
101       # ---CHANGED--- our network is called "basic"
102       - basic
```

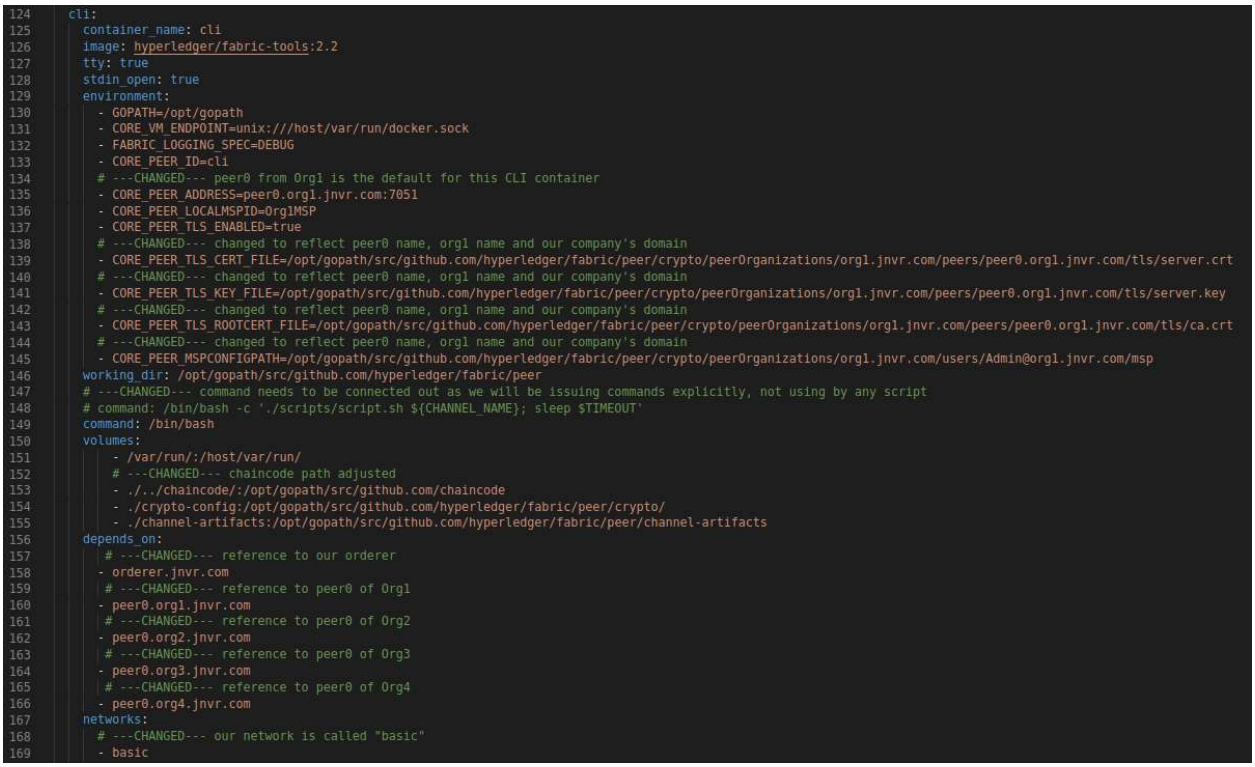
Figura 61 - Excerto do ficheiro docker-compose-cli-couchdb.yaml a definir um peer de uma organização

```
105   ca.org1.jnvr.com:
106     image: hyperledger/fabric-ca:1.4.8
107     environment:
108       - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
109       - FABRIC_CA_SERVER_CA_NAME=ca.org1.jnvr.com
110       - FABRIC_CA_SERVER_TLS_ENABLED=true
111       - FABRIC_CA_SERVER_TLS_CERTFILE=/etc/hyperledger/fabric-ca-server-config/ca.org1.jnvr.com-cert.pem
112       - FABRIC_CA_SERVER_TLS_KEYFILE=/etc/hyperledger/fabric-ca-server-config/priv_sk
113       - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-ca-server-config/ca.org1.jnvr.com-cert.pem
114       - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-config/priv_sk
115     ports:
116       - "7054:7054"
117     command: sh -c 'fabric-ca-server start -b admin:adminpw'
118     volumes:
119       - ./crypto-config/peerOrganizations/org1.jnvr.com/ca/:/etc/hyperledger/fabric-ca-server-config
120     container_name: ca.org1.jnvr.com
121     networks:
122       - basic
```

Figura 62 - Excerto do ficheiro docker-compose-cli-couchdb.yaml a definir a configuração do CA da organização 1

```
170   couchdb0:
171     image: couchdb:3.1
172     environment:
173       - COUCHDB_USER=admin
174       - COUCHDB_PASSWORD=adminpw
175     ports:
176       - 5984:5984
177     container_name: couchdb0
178     networks:
179       - basic
```

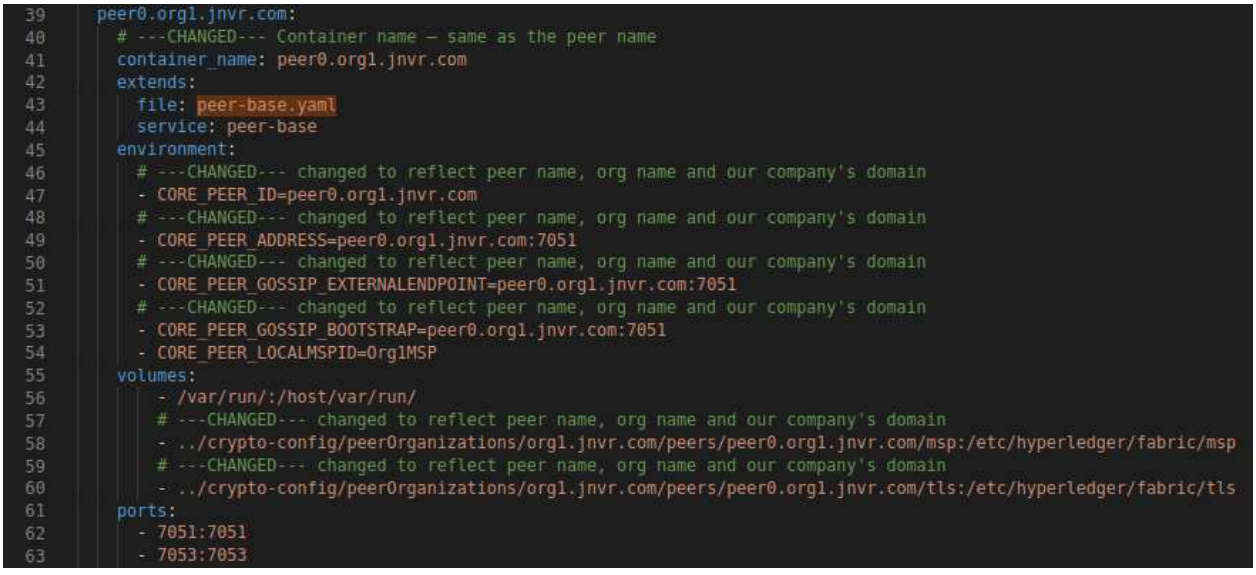
Figura 63 - Excerto do ficheiro docker-compose-cli-couchdb.yaml a definir uma das bases de dados



```
cli:
  container_name: cli
  image: hyperledger/fabric-tools:2.2
  tty: true
  stdin_open: true
  environment:
    - GOPATH=/opt/gopath
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - FABRIC_LOGGING_SPEC=DEBUG
    - CORE_PEER_ID=cli
    # ---CHANGED--- peer0 from Org1 is the default for this CLI container
    - CORE_PEER_ADDRESS=peer0.org1.jnvr.com:7051
    - CORE_PEER_LOCALMSPID=Org1MSP
    - CORE_PEER_TLS_ENABLED=true
    # ---CHANGED--- changed to reflect peer0 name, org1 name and our company's domain
    - CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.jnvr.com/peers/peer0.org1.jnvr.com/tls/server.crt
    # ---CHANGED--- changed to reflect peer0 name, org1 name and our company's domain
    - CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.jnvr.com/peers/peer0.org1.jnvr.com/tls/server.key
    # ---CHANGED--- changed to reflect peer0 name, org1 name and our company's domain
    - CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.jnvr.com/peers/peer0.org1.jnvr.com/tls/ca.crt
    # ---CHANGED--- changed to reflect peer0 name, org1 name and our company's domain
    - CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.jnvr.com/users/Admin@org1.jnvr.com/msp
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
  # ---CHANGED--- command needs to be connected out as we will be issuing commands explicitly, not using by any script
  # command: /bin/bash -c './scripts/script.sh ${CHANNEL_NAME}; sleep $TIMEOUT'
  command: /bin/bash
  volumes:
      - /var/run/:/host/var/run/
      # ---CHANGED--- chaincode path adjusted
      - ./../chaincode/:/opt/gopath/src/github.com/chaincode
      - ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
      - ./channel-artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel-artifacts
  depends_on:
    # ---CHANGED--- reference to our orderer
    - orderer.jnvr.com
    # ---CHANGED--- reference to peer0 of Org1
    - peer0.org1.jnvr.com
    # ---CHANGED--- reference to peer0 of Org2
    - peer0.org2.jnvr.com
    # ---CHANGED--- reference to peer0 of Org3
    - peer0.org3.jnvr.com
    # ---CHANGED--- reference to peer0 of Org4
    - peer0.org4.jnvr.com
  networks:
    # ---CHANGED--- our network is called "basic"
    - basic
```

Figura 64 - Excerto do ficheiro docker-compose-cli-couchdb.yaml a definir configuração de cli

4) Adicionalmente é necessário criar o documento *docker-compose-base.yaml* na diretoria base, que é mencionado anteriormente na definição dos containers das organizações. Neste documento é definido em maior detalhe informações sobre a inicialização dos *dockers* das organizações e onde são guardados os materiais criados.



```
peer0.org1.jnvr.com:
  # ---CHANGED--- Container name - same as the peer name
  container_name: peer0.org1.jnvr.com
  extends:
    file: peer-base.yaml
    service: peer-base
  environment:
    # ---CHANGED--- changed to reflect peer name, org name and our company's domain
    - CORE_PEER_ID=peer0.org1.jnvr.com
    # ---CHANGED--- changed to reflect peer name, org name and our company's domain
    - CORE_PEER_ADDRESS=peer0.org1.jnvr.com:7051
    # ---CHANGED--- changed to reflect peer name, org name and our company's domain
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org1.jnvr.com:7051
    # ---CHANGED--- changed to reflect peer name, org name and our company's domain
    - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org1.jnvr.com:7051
    - CORE_PEER_LOCALMSPID=Org1MSP
  volumes:
      - /var/run/:/host/var/run/
      # ---CHANGED--- changed to reflect peer name, org name and our company's domain
      - ../crypto-config/peerOrganizations/org1.jnvr.com/peers/peer0.org1.jnvr.com/msp:/etc/hyperledger/fabric/msp
      # ---CHANGED--- changed to reflect peer name, org name and our company's domain
      - ../crypto-config/peerOrganizations/org1.jnvr.com/peers/peer0.org1.jnvr.com/tls:/etc/hyperledger/fabric/tls
  ports:
    - 7051:7051
    - 7053:7053
```

Figura 65 - Excerto do ficheiro docker-compose-base.yaml a definir um peer de uma organização

5) Novamente, menciona-se outro ficheiro denominado *peer-base.yaml*, que vai ser necessário criar, ainda na diretoria *base*. Este ficheiro é usado para estabelecer os diferentes modos de trabalho, a diretoria onde a rede trabalha e a localização de ficheiros chave para a rede.

```
version: '2'

services:
  peer-base:
    image: hyperledger/fabric-peer:2.2.0
    environment:
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      # the following setting starts chaincode containers on the same
      # bridge network as the peers
      # https://docs.docker.com/compose/networking/
      # ---CHANGED---
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=jnvr-network_basic
      - FABRIC_LOGGING_SPEC=INFO
     # - FABRIC_LOGGING_SPEC=DEBUG
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_GOSSIP_USELEADERELECTION=true
      - CORE_PEER_GOSSIP_ORGLEADER=false
      - CORE_PEER_PROFILE_ENABLED=true
      - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
      - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    command: peer node start
```

Figura 66 - Excerto do ficheiro peer-base.yaml a definir o comportamento de um peer

6) Tendo os ficheiros criados, correm-se os comandos

`export CHANNEL_NAME=channel1`

`export VERBOSE=false`

`export FABRIC_CFG_PATH=$PWD`

Seguidos por `CHANNEL_NAME=$CHANNEL_NAME docker-compose -f docker-compose-cli-couchdb.yaml up -d`, que irá iniciar os containers da rede.

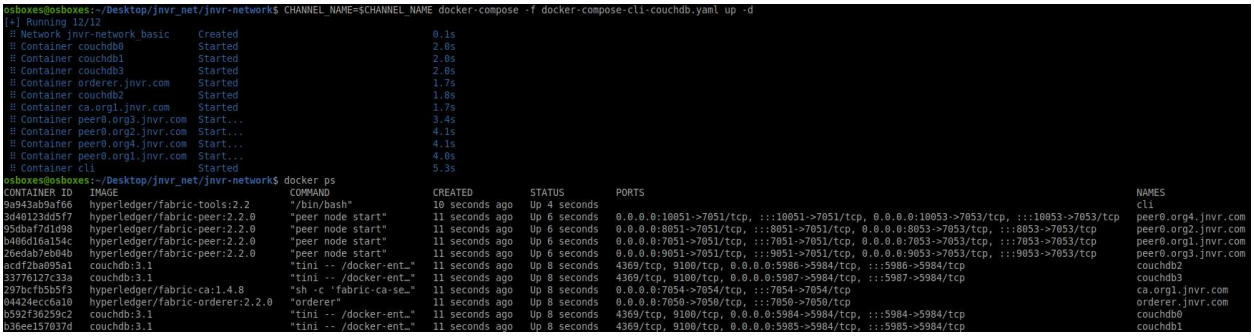


Figura 67 - Inicialização dos containers da rede

7) Após o início dos containers da rede, ir-se-á entrar no container *cli*, criado para ser possível interagir com a rede, com o comando `docker exec -it cli /bin/sh`.

8) Dentro de *cli* é necessário voltar a exportar o nome do canal com o comando `export CHANNEL_NAME=channel1`.

9) Depois cria-se o canal usando o seguinte comando.

```
peer channel create -o orderer.jnvr.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/channel.tx --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem
```

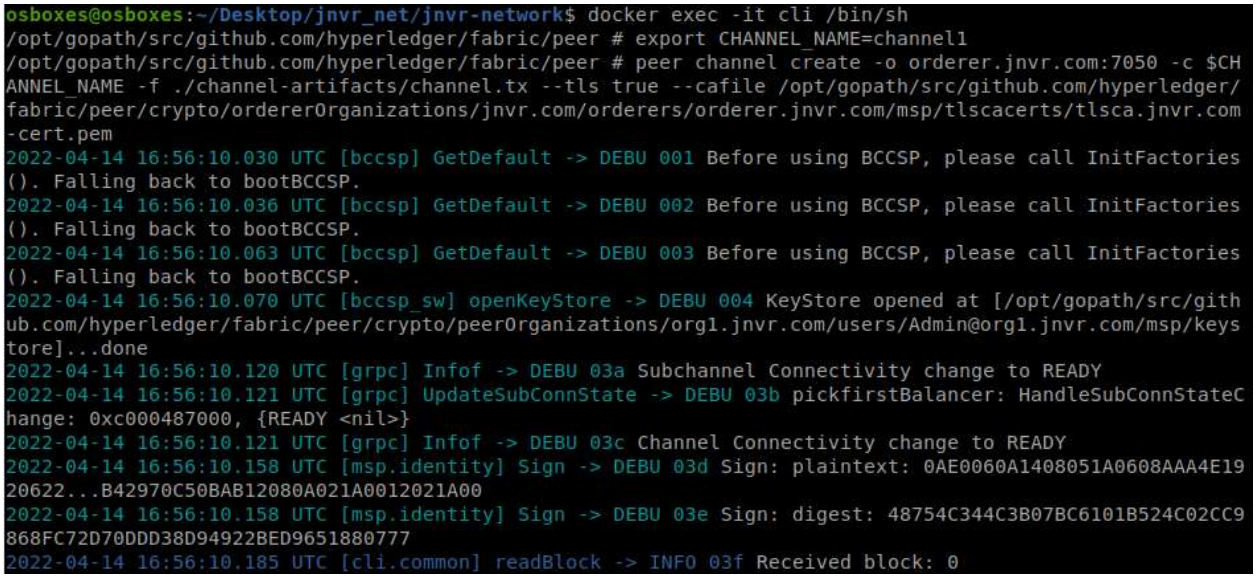


Figura 68 - Excertos da criação do canal

Para verificar que o comando executou corretamente basta utilizar o comando `ls` e verificar que se criou o ficheiro, neste caso, *channel1.*block. Se o nome do canal fosse diferente, aparecia o nome de canal escolhido.

10) Seguidamente juntam-se as organizações 1, 2, 3 e 4 ao canal através dos seguintes comandos, onde se utiliza o ficheiro do tipo *.block* obtido anteriormente.

```
peer channel join -b channel1.block
```

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org2.jnvr.com/users/Admin@org2.jnvr.
com/msp/ CORE_PEER_ADDRESS=peer0.org2.jnvr.com:7051
CORE_PEER_LOCALMSPID="Org2MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org2.jnvr.com/peers/peer0.org2.j
nvr.com/tls/ca.crt peer channel join -b channel1.block
```

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org3.jnvr.com/users/Admin@org3.jnvr.
com/msp/ CORE_PEER_ADDRESS=peer0.org3.jnvr.com:7051
CORE_PEER_LOCALMSPID="Org3MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org3.jnvr.com/peers/peer0.org3.j
nvr.com/tls/ca.crt peer channel join -b channel1.block
```

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org4.jnvr.com/users/Admin@org4.jnvr.
com/msp/ CORE_PEER_ADDRESS=peer0.org4.jnvr.com:7051
CORE_PEER_LOCALMSPID="Org4MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org4.jnvr.com/peers/peer0.org4.j
nvr.com/tls/ca.crt peer channel join -b channel1.block
```

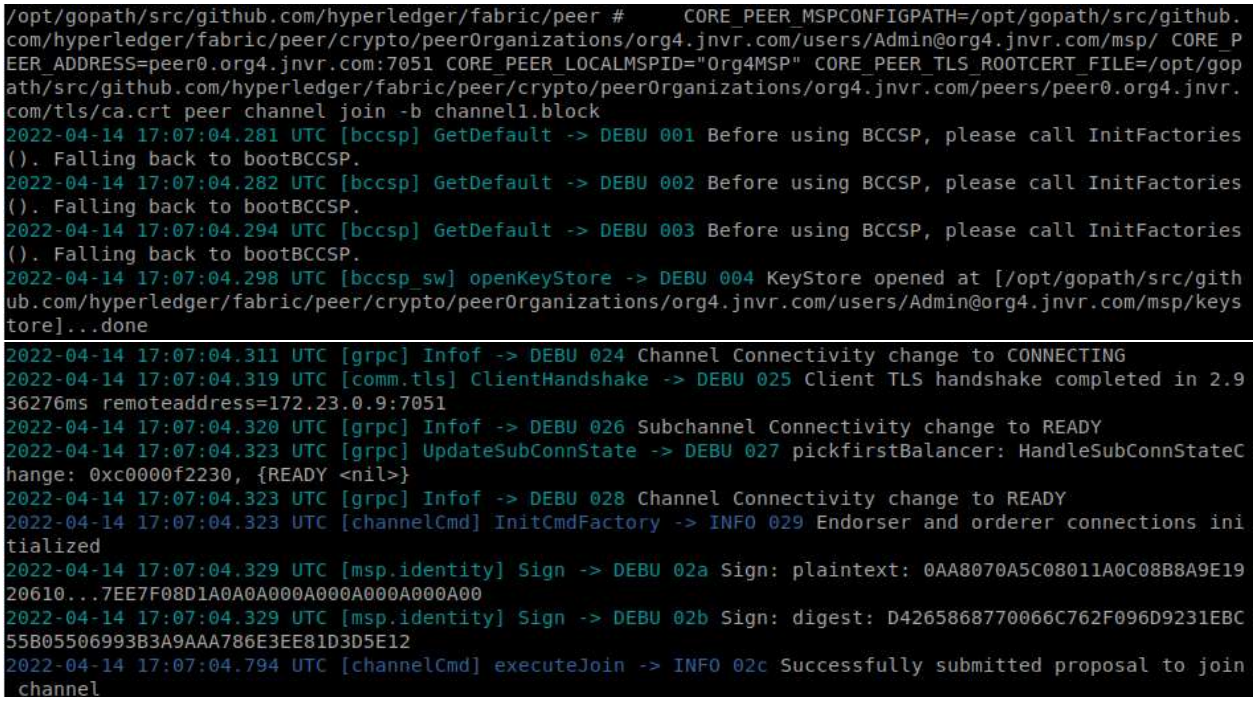


Figura 69 - Excertos da junção de uma organização ao canal

Deve-se notar que devido a *cli* estar a correr na organização 1, não é necessária toda a definição de variáveis que se verifica para as outras organizações.

11) De seguida utilizam-se os seguintes comandos para criar os *anchor peers* para cada uma das organizações.

```
peer channel update -o orderer.jnvr.com:7050 -c $CHANNEL_NAME -f
./channel-artifacts/Org1MSPanchors.tx --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem
```

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.jnvr.com/users/Admin@org2.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org2.jnvr.com:7051
CORE_PEER_LOCALMSPID="Org2MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.jnvr.com/peers/peer0.org2.jnvr.com/tls/ca.crt peer channel update -o orderer.jnvr.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/Org2MSPanchors.tx --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem
```

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/users/Admin@org3.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org3.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org3MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/peers/peer0.org3.jnvr.com/tls/ca.crt peer channel update -o orderer.jnvr.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/Org3MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem
```

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/users/Admin@org4.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org4.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org4MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/peers/peer0.org4.jnvr.com/tls/ca.crt peer channel update -o orderer.jnvr.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/Org4MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem
```

```
/opt/gopath/src/github.com/hyperledger/fabric/peer #    CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/users/Admin@org4.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org4.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org4MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/peers/peer0.org4.jnvr.com/tls/ca.crt peer channel update -o orderer.jnvr.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/Org4MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem
2022-04-14 17:09:41.966 UTC [bccsp] GetDefault -> DEBU 001 Before using BCCSP, please call InitFactories(). Falling back to bootBCCSP.
2022-04-14 17:09:41.968 UTC [bccsp] GetDefault -> DEBU 002 Before using BCCSP, please call InitFactories(). Falling back to bootBCCSP.
2022-04-14 17:09:41.983 UTC [bccsp] GetDefault -> DEBU 003 Before using BCCSP, please call InitFactories(). Falling back to bootBCCSP.
2022-04-14 17:09:41.989 UTC [bccsp_sw] openKeyStore -> DEBU 004 KeyStore opened at [/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/users/Admin@org4.jnvr.com/msp/keystore]...done
2022-04-14 17:09:41.997 UTC [grpc] Infof -> DEBU 025 ClientConn switching balancer to "pick_first"
2022-04-14 17:09:41.997 UTC [grpc] Infof -> DEBU 026 Channel switches to new LB policy "pick_first"
2022-04-14 17:09:41.997 UTC [grpc] Infof -> DEBU 027 Subchannel Connectivity change to CONNECTING
2022-04-14 17:09:41.998 UTC [grpc] Infof -> DEBU 028 Subchannel picks a new address "orderer.jnvr.com:7050" to connect
2022-04-14 17:09:42.000 UTC [grpc] UpdateSubConnState -> DEBU 029 pickfirstBalancer: HandleSubConnStateChange: 0xc00002fa40, {CONNECTING <nil>}
2022-04-14 17:09:42.000 UTC [grpc] Infof -> DEBU 02a Channel Connectivity change to CONNECTING
2022-04-14 17:09:42.006 UTC [comm.tls] ClientHandshake -> DEBU 02b Client TLS handshake completed in 3.101788ms remoteaddress=172.23.0.7:7050
2022-04-14 17:09:42.009 UTC [grpc] Infof -> DEBU 02c Subchannel Connectivity change to READY
2022-04-14 17:09:42.010 UTC [grpc] UpdateSubConnState -> DEBU 02d pickfirstBalancer: HandleSubConnStateChange: 0xc00002fa40, {READY <nil>}
2022-04-14 17:09:42.010 UTC [grpc] Infof -> DEBU 02e Channel Connectivity change to READY
2022-04-14 17:09:42.039 UTC [channelCmd] update -> INFO 02f Successfully submitted channel update
```

Figura 70 - Excertos da criação de um anchor peer numa organização

Volta-se a observar que na organização 1 não é necessário a definição das variáveis, que se verificam em todas as outras organizações, tal como quando se juntaram as organizações ao canal, anteriormente.

12) Neste exemplo vai ser utilizado um *chaincode* em linguagem *go*, que está localizado na diretoria *chaincode* mencionada no início do exemplo. Nessa diretoria cria-se uma pasta com o nome do *chaincode*, que neste caso se denomina *foodcontrol*.

Dentro da pasta anterior cria-se um ficheiro do tipo .*go* com o nome do *chaincode*, ou seja, foodcontrol.go com o seguinte conteúdo, onde estão funções para inicializar valores em cada *peer*, permitir realização de pesquisas, introdução e edição de novos valores.

```go
package main

import (
    "encoding/json"
    "fmt"
    "github.com/hyperledger/fabric-contract-api-go/contractapi"
)

// SmartContract provides functions for control the food
type SmartContract struct {
    contractapi.Contract
}

//Food describes basic details of what makes up a food
type Food struct {
    Farmer  string `json:"farmer"`
    Variety string `json:"variety"`
}

func (s *SmartContract) Set(ctx contractapi.TransactionContextInterface, foodId string, farmer string, variety string) error {

    //Validaciones de sintaxis

    //validaciones de negocio

    food := Food{
        Farmer:  farmer,
        Variety: variety,
    }

    foodAsBytes, err := json.Marshal(food)
    if err != nil {
        fmt.Printf("Marshal error: %s", err.Error())
        return err
    }

    return ctx.GetStub().PutState(foodId, foodAsBytes)
}

func (s *SmartContract) Query(ctx contractapi.TransactionContextInterface, foodId string) (*Food, error) {

    foodAsBytes, err := ctx.GetStub().GetState(foodId)

    if err != nil {
        return nil, fmt.Errorf("Failed to read from world state. %s", err.Error())
    }

    if foodAsBytes == nil {
        return nil, fmt.Errorf("%s does not exist", foodId)
    }

    food := new(Food)

    err = json.Unmarshal(foodAsBytes, food)
    if err != nil {
        return nil, fmt.Errorf("Unmarshal error. %s", err.Error())
    }

    return food, nil
}

func main() {

    chaincode, err := contractapi.NewChaincode(new(SmartContract))

    if err != nil {
        fmt.Printf("Error al crear el chaincode: %s", err.Error())
        return
    }

    if err := chaincode.Start(); err != nil {
        fmt.Printf("Error al crear el: %s", err.Error())
    }
}
```

Figura 71 - Documento foodcontrol.go

13) Depois cria-se go.mod que identifica as versões e *packages* necessários para correr o ficheiro *.go*, neste caso *foodcontrol.go*.

Figura 72 - Documento go.mod

14) Após a criação dos ficheiros fazem-se os seguintes exports no terminal, ainda dentro de cli.

```
export CHANNEL_NAME=channel1

export CHAINCODE_NAME=foodcontrol

export CHAINCODE_VERSION=1

export CC_RUNTIME_LANGUAGE=golang

export CC_SRC_PATH="../../../chaincode/$CHAINCODE_NAME/"

export
ORDERER_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto
/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscace
rts/tlsca.jnvr.com-cert.pem
```

```
/opt/gopath/src/github.com/hyperledger/fabric/peer # export CHANNEL_NAME=channel1
/opt/gopath/src/github.com/hyperledger/fabric/peer # export CHAINCODE_NAME=js_chain
/opt/gopath/src/github.com/hyperledger/fabric/peer # export CHAINCODE_VERSION=1
/opt/gopath/src/github.com/hyperledger/fabric/peer # export CC_RUNTIME_LANGUAGE=node
/opt/gopath/src/github.com/hyperledger/fabric/peer # export CC_SRC_PATH="../../../chaincod
e/$CHAINCODE_NAME/"
```

Figura 73 - Execução dos exports dentro de cli

15) Passando para a génese do package com o *chaincode* a ser instalado em cada peer utiliza-se o seguinte comando, que faz uso dos *exports* realizados anteriormente.

```
peer lifecycle chaincode package ${CHAINCODE_NAME}.tar.gz --path
${CC_SRC_PATH} --lang ${CC_RUNTIME_LANGUAGE} --label
${CHAINCODE_NAME}_${CHAINCODE_VERSION} >&log.txt
```

```
/opt/gopath/src/github.com/hyperledger/fabric/peer # peer lifecycle chaincode package ${CH
AINCODE_NAME}.tar.gz --path ${CC_SRC_PATH} --lang ${CC_RUNTIME_LANGUAGE} --label ${CHAINCO
DE_NAME}_${CHAINCODE_VERSION} >&log.txt
```

Figura 74 – Criação do package com o chaincode a instalar nos peers

16) Seguidamente pode-se verificar os resultados utilizando o comando `ls`, de onde se verifica a criação de dois ficheiros, um denominado *log.txt* (onde estão os *logs* do funcionamento do comando) e outro chamado *<nome do chaincode>.tar.gz*, neste caso foodcontrol.tar.gz (*chaincode* a ser instalado em cada *peer*).

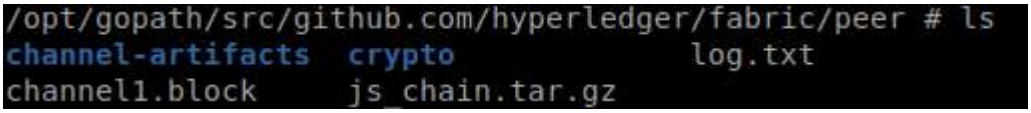


Figura 75 - Verificação da criação do package do chaincode

17) Finalmente instala-se o *chaincode* em cada *peer* com os seguintes comandos.

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

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org2.jnvr.com/users/Admin@org2.jnvr.
com/msp/ CORE_PEER_ADDRESS=peer0.org2.jnvr.com:7051
CORE_PEER_LOCALMSPID="Org2MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org2.jnvr.com/peers/peer0.org2.j
nvr.com/tls/ca.crt peer lifecycle chaincode install
foodcontrol.tar.gz
```

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org3.jnvr.com/users/Admin@org3.jnvr.
com/msp/ CORE_PEER_ADDRESS=peer0.org3.jnvr.com:7051
CORE_PEER_LOCALMSPID="Org3MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org3.jnvr.com/peers/peer0.org3.j
nvr.com/tls/ca.crt peer lifecycle chaincode install
foodcontrol.tar.gz
```

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org4.jnvr.com/users/Admin@org4.jnvr.
com/msp/ CORE_PEER_ADDRESS=peer0.org4.jnvr.com:7051
CORE_PEER_LOCALMSPID="Org4MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org4.jnvr.com/peers/peer0.org4.j
nvr.com/tls/ca.crt peer lifecycle chaincode install
foodcontrol.tar.gz
```

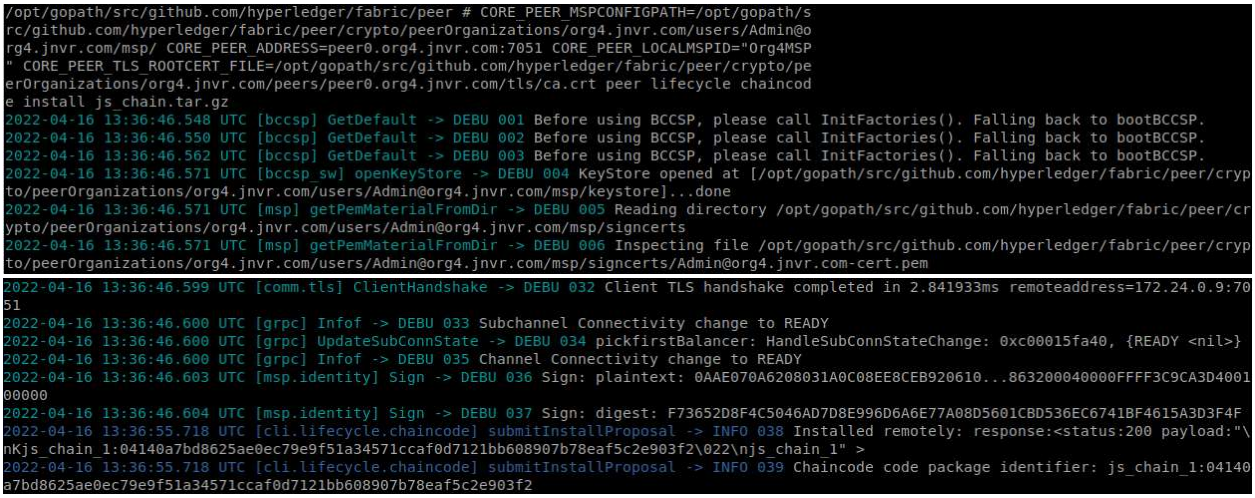


Figura 76 - Excertos da instalação do chaincode em um dos peers

Destes comandos é importante registar o *package identifier* que aparece no final do comando. No caso desta implementação é:
`foodcontrol_1:04140a7bd8625ae0ec79e9f51a34571ccaf0d7121bb608907b78eaf5c2e903f2`

18) Tendo o *package identifier* pode-se aprovar a definição do *chaincode* para cada organização. Neste caso como se pretende manter a Organização 2 em modo *read-only*, esta organização não irá ter o seu *chaincode* aprovado. Isto vai ser realizado com os seguintes comandos.

```
peer lifecycle chaincode approveformyorg --tls --cafile $ORDERER_CA
--channelID $CHANNEL_NAME --name $CHAINCODE_NAME --version
$CHAINCODE_VERSION --sequence 1 --waitForEvent --signature-policy
"OR ('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')" --package-id
<package identifier>
```

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/users/Admin@org3.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org3.jnvr.com:7051
CORE_PEER_LOCALMSPID="Org3MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/peers/peer0.org3.jnvr.com/tls/ca.crt peer lifecycle chaincode approveformyorg --tls --cafile $ORDERER_CA --channelID $CHANNEL_NAME --name $CHAINCODE_NAME
--version $CHAINCODE_VERSION --sequence 1 --waitForEvent --
signature-policy "OR ('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')"
--package-id <package identifier>
```

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/users/Admin@org4.jnvr.
```

```
com/msp/ CORE_PEER_ADDRESS=peer0.org4.jnvr.com:7051
CORE_PEER_LOCALMSPID="Org4MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org4.jnvr.com/peers/peer0.org4.j
nvr.com/tls/ca.crt peer lifecycle chaincode approveformyorg --tls --
cafile $ORDERER_CA --channelID $CHANNEL_NAME --name $CHAINCODE_NAME
--version $CHAINCODE_VERSION --sequence 1 --waitForEvent --
signature-policy "OR ('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')"
--package-id <package identifier>
```

Sabendo o valor do *package identifier* do passo anterior, o comando fica da seguinte forma.

```
peer lifecycle chaincode approveformyorg --tls --cafile $ORDERER_CA
--channelID $CHANNEL_NAME --name $CHAINCODE_NAME --version
$CHAINCODE_VERSION --sequence 1 --waitForEvent --signature-policy
"OR ('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')" --package-id
foodcontrol_1:04140a7bd8625ae0ec79e9f51a34571ccaf0d7121bb608907b78ea
f5c2e903f2
```

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org3.jnvr.com/users/Admin@org3.jnvr.
com/msp/ CORE_PEER_ADDRESS=peer0.org3.jnvr.com:7051
CORE_PEER_LOCALMSPID="Org3MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org3.jnvr.com/peers/peer0.org3.j
nvr.com/tls/ca.crt peer lifecycle chaincode approveformyorg --tls --
cafile $ORDERER_CA --channelID $CHANNEL_NAME --name $CHAINCODE_NAME
--version $CHAINCODE_VERSION --sequence 1 --waitForEvent --
signature-policy "OR ('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')"
--package-id
foodcontrol_1:04140a7bd8625ae0ec79e9f51a34571ccaf0d7121bb608907b78ea
f5c2e903f2
```

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org4.jnvr.com/users/Admin@org4.jnvr.
com/msp/ CORE_PEER_ADDRESS=peer0.org4.jnvr.com:7051
CORE_PEER_LOCALMSPID="Org4MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org4.jnvr.com/peers/peer0.org4.j
nvr.com/tls/ca.crt peer lifecycle chaincode approveformyorg --tls --
cafile $ORDERER_CA --channelID $CHANNEL_NAME --name $CHAINCODE_NAME
--version $CHAINCODE_VERSION --sequence 1 --waitForEvent --
signature-policy "OR ('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')"
--package-id
```

```
foodcontrol_1:04140a7bd8625ae0ec79e9f51a34571ccaf0d7121bb608907b78ea
f5c2e903f2
```

Para verificar que apenas a Organização 2 não está aprovada corre-se o comando.

```
peer lifecycle chaincode checkcommitreadiness --channelID
$CHANNEL_NAME --name $CHAINCODE_NAME --version $CHAINCODE_VERSION --
sequence 1 --signature-policy "OR
('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')" --output json.
```

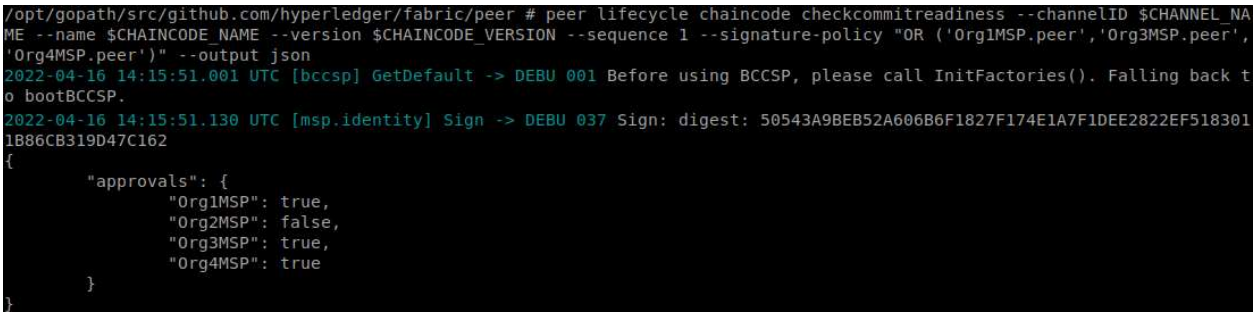


Figura 77 - Excertos da verificação da aprovação das organizações

Onde se verifica que a Organização 2 não tem *approval*.

19) Para fazer o *commit* do *chaincode* usa-se o seguinte comando, onde se volta a verificar a ausência da Organização 2 devido a esta ser *read-only*.

```
peer lifecycle chaincode commit -o orderer.jnvr.com:7050 --tls --
cafile $ORDERER_CA --peerAddresses peer0.org1.jnvr.com:7051 --
tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrgani
zations/org1.jnvr.com/peers/peer0.org1.jnvr.com/tls/ca.crt --
peerAddresses peer0.org3.jnvr.com:7051 --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrgani
zations/org3.jnvr.com/peers/peer0.org3.jnvr.com/tls/ca.crt --
peerAddresses peer0.org4.jnvr.com:7051 --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrgani
zations/org4.jnvr.com/peers/peer0.org4.jnvr.com/tls/ca.crt --
channelID $CHANNEL_NAME --name $CHAINCODE_NAME --version
$CHAINCODE_VERSION --sequence 1 --signature-policy "OR
('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')"
```

```
/opt/gopath/src/github.com/hyperledger/fabric/peer # peer lifecycle chaincode commit -o orderer.jnvr.com:7050 --tls --cafi
le $ORDERER_CA --peerAddresses peer0.org1.jnvr.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/p
eer/crypto/peerOrganizations/org1.jnvr.com/peers/peer0.org1.jnvr.com/tls/ca.crt --peerAddresses peer0.org3.jnvr.com:7051 -
-tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/peers/peer0.or
g3.jnvr.com/tls/ca.crt --peerAddresses peer0.org4.jnvr.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/
fabric/peer/crypto/peerOrganizations/org4.jnvr.com/peers/peer0.org4.jnvr.com/tls/ca.crt --channelID $CHANNEL_NAME --name $
CHAINCODE_NAME --version $CHAINCODE_VERSION --sequence 1 --signature-policy "OR ('Org1MSP.peer','Org3MSP.peer','Org4MSP.pe
er')"
2022-04-16 14:42:42.009 UTC [msp.identity] Sign -> DEBU 081 Sign: digest: 4D702565538E8448E322EF40B8828A65FB1350E05CE4B1AB
F402A12CC4272146
2022-04-16 14:42:44.298 UTC [chaincodeCmd] ClientWait -> INFO 082 txid [a767edfd001464c7f58a420e06a59787b4fad114b9d94f5a46
91834263dcfd02] committed with status (VALID) at peer0.org3.jnvr.com:7051
2022-04-16 14:42:44.614 UTC [chaincodeCmd] ClientWait -> INFO 083 txid [a767edfd001464c7f58a420e06a59787b4fad114b9d94f5a46
91834263dcfd02] committed with status (VALID) at peer0.org4.jnvr.com:7051
2022-04-16 14:42:44.676 UTC [chaincodeCmd] ClientWait -> INFO 084 txid [a767edfd001464c7f58a420e06a59787b4fad114b9d94f5a46
91834263dcfd02] committed with status (VALID) at peer0.org1.jnvr.com:7051
```

Figura 78 - Excertos do commit do chaincode

20) Após ter o *chaincode* preparado utiliza-se o seguinte comando para utilizar a função *Set* definida no *chaincode* e para criar no *ledger* o valor de *id:3*.

```
peer chaincode invoke -o orderer.jnvr.com:7050 --tls --cafile
$ORDERER_CA -C $CHANNEL_NAME -n $CHAINCODE_NAME -c
'{"Args":["Set","id:3","jorge","naranja"]}'
```

```
/opt/gopath/src/github.com/hyperledger/fabric/peer # peer chaincode invoke -o orderer.jnvr.com:7050 --tls --cafile $OR
DERER_CA -C $CHANNEL_NAME -n $CHAINCODE_NAME -c '{"Args":["Set","id:3","jorge","naranja"]}'
```

Figura 79 - Utilização da função set do chaincode

21) De modo a observar os resultados deste comando pode-se ir ao url `localhost:5984/_utils/` em qualquer navegador de Internet, podendo assim aceder à base de dados *couchdb* do *peer1*.

Nessa página será necessário introduzir o nome de utilizador e password do *couchdb*, que foram definidos no *setup* da rede.

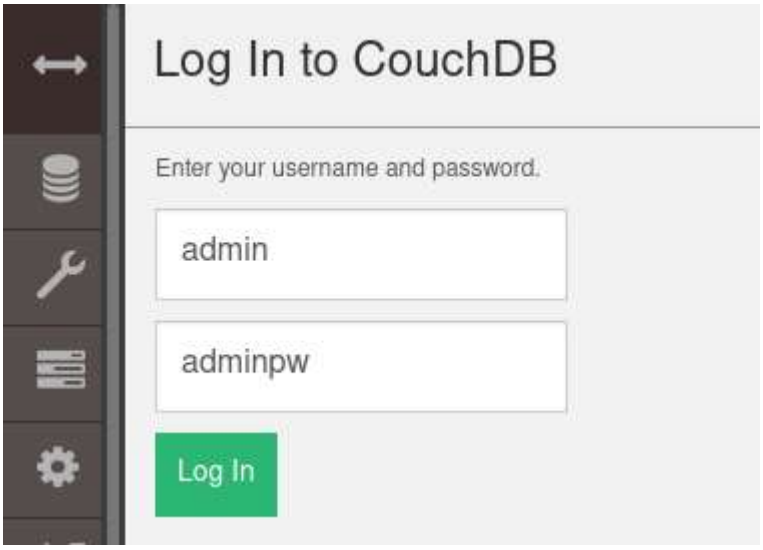


Figura 80 - Página inicial de couchdb

Após o log in no *couchdb* irá ser apresentada a seguinte página onde se observa o *chaincode* instalado.

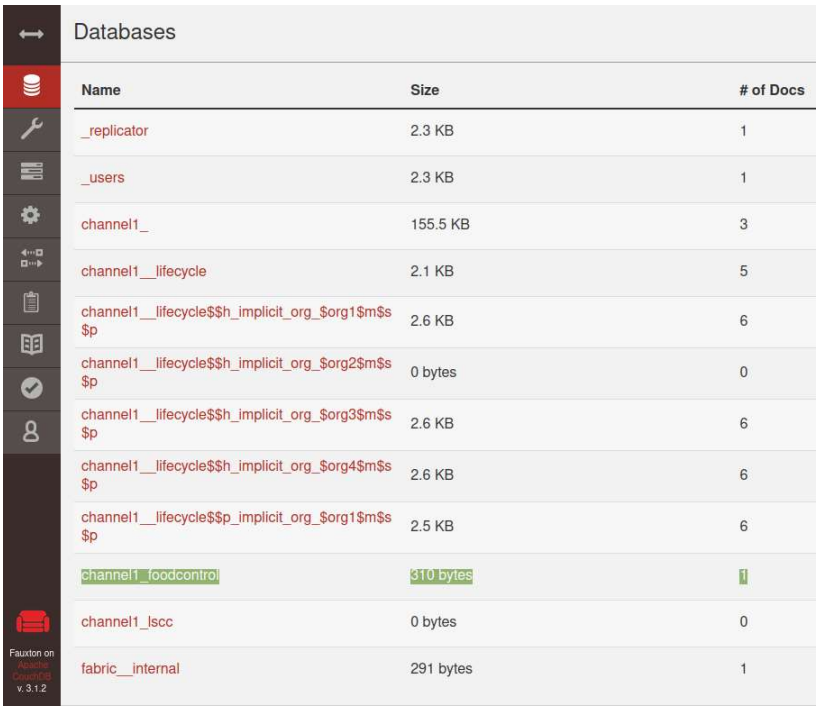


Figura 81 - Chaincode em couchdb

Ao entrar em *channel1_foodcontrol* observa-se o *id:3* e os valores definidos criados com o comando.

Figura 82 - Valor previamente criado, guardado em couchdb

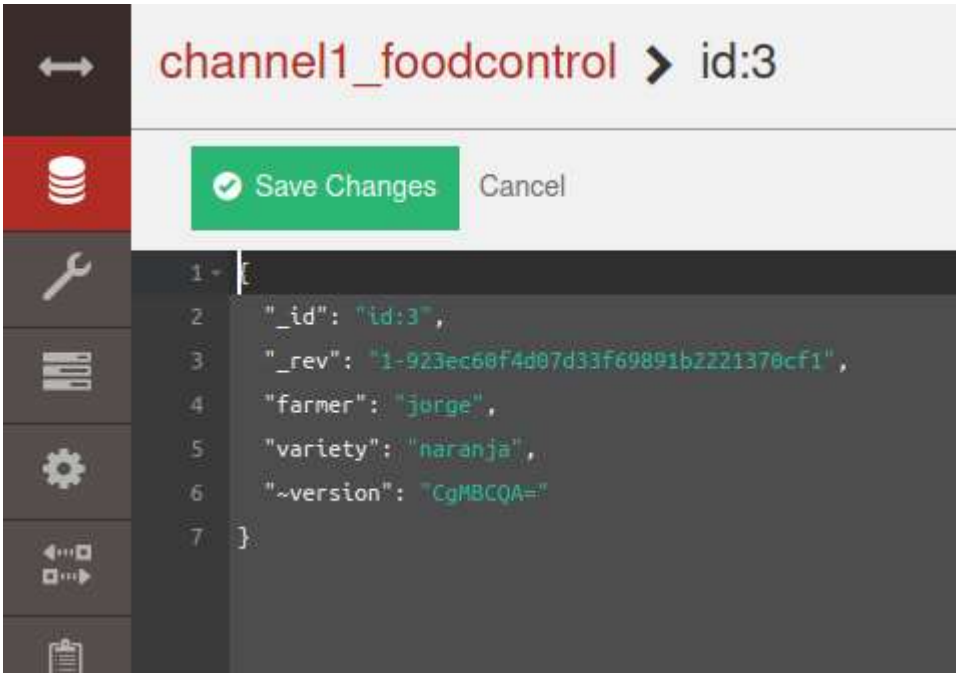


Figura 83 - Detalhes do valor criado, guardado em couchdb

5.2.3.4 Criação de scripts para conceber uma rede com quatro organizações

Neste caso usa-se a estrutura de pastas apresentadas na implementação anterior (secção 5.2.3.3) com os mesmos documentos *.yaml* de configuração.

Tendo estes documentos de configuração cria-se scripts de modo a inicializar uma rede com quatro organizações, que é explicada de seguida:

1) Inicialmente irá ser criado o ficheiro *script_jnvr_network.sh* na diretoria *jnvr_network* que se pode observar de seguida.

home > osboxes > Desktop > jnvr_net > jnvr-network > $ script-jnvr-network.sh

```
#!/bin/sh

#This scripts assumes it is being run in the jnvr_network directory.
#There also is the assumption that the structure of the example is being followed.

#################################!!!!!ATENTION!!!!!#################################
#This script eliminates ALL DOCKER CONTAINERS AND VOLUMES IN YOUR MACHINE, meaning if
#there is a network already running, THAT NETWORK'S CONTAINERS AND VOLUMES will be DELETED
#####################################################################################

rm -r ./channel-artifacts
mkdir ./channel-artifacts
sudo rm -r ./crypto-config
mkdir ./crypto-config
docker rm -f $(docker ps -aq)
docker volume rm -f $(docker volume ls -q)
NETWORK_NAME=$(grep 'CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE' /home/osboxes/Desktop/jnvr_net/jnvr-network/base/peer-base.yaml)
#IFS variable is changed in order to modify how Bash recognizes word boundaries while splitting a sequence of character strings, normaly it is set to ' ' as I've defined it is set to '='
old_IFS=$IFS
IFS==
#Heredoc is being used here to pass the value of NETWORK_NAME as an input to the read comand, EOF here functions as a delimiter token
read -r thing NETWORK_NAME <<EOF
$NETWORK_NAME
EOF
IFS=$old_IFS
docker network rm $NETWORK_NAME


#Generate Cryptographic Material:
  cryptogen generate --config=./crypto-config.yaml

#Create Genesis Block
   configtxgen -profile FourOrgsOrdererGenesis -channelID system-channel -outputBlock ./channel-artifacts/genesis.block
   configtxgen -profile FourOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID channel1
   configtxgen -profile FourOrgsChannel -outputAnchorPeersUpdate  ./channel-artifacts/Org1MSPanchors.tx -channelID channel1 -asOrg Org1MSP
   configtxgen -profile FourOrgsChannel -outputAnchorPeersUpdate  ./channel-artifacts/Org2MSPanchors.tx -channelID channel1 -asOrg Org2MSP
   configtxgen -profile FourOrgsChannel -outputAnchorPeersUpdate  ./channel-artifacts/Org3MSPanchors.tx -channelID channel1 -asOrg Org3MSP
   configtxgen -profile FourOrgsChannel -outputAnchorPeersUpdate  ./channel-artifacts/Org4MSPanchors.tx -channelID channel1 -asOrg Org4MSP

#Start Docker containers
   export CHANNEL_NAME=channel1
   export VERBOSE=false
   export FABRIC_CFG_PATH=$PWD
   CHANNEL_NAME=$CHANNEL_NAME docker-compose -f docker-compose-cli-couchdb.yaml up -d

#Copy Docker Script into container cli
   docker cp ./script-jnvr-network-docker.sh cli:/opt/gopath/src/github.com/hyperledger/fabric/peer

#Initiate cli shell
   docker exec -it cli /bin/sh
```

Figura 84 - Documento script-jnvr-network.sh

Neste script removem-se quaisquer ficheiros previamente criados na configuração da *jnvr_network*, todos os containers e volumes presentes na máquina e irá ser removida a rede previamente criada com a configuração presente na diretoria. Caso o script corra numa máquina nova, nada será feito nesta fase inicial.

De seguida irão ser corridos os comandos apresentados na implementação anterior (secção 5.2.3.3) até à criação dos *containers*.

Finalmente, o *script_jnvr_network_docker.sh* irá ser copiado para o *container cli* e entra-se no container.

2) Passando para a criação de *script_jnvr_network_docker.sh*, que também se encontra na diretoria *jnvr_network*, observa-se de seguida a sua constituição.

```
#!/bin/sh

#Create the Channel
   export CHANNEL_NAME=channel1
   peer channel create -o orderer.jnvr.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/channel.tx --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem

#Join Org1 Org2 Org3 and Org4 to the Channel
   peer channel join -b ${CHANNEL_NAME}.block
   CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.jnvr.com/users/Admin@org2.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org2.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org2MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.jnvr.com/peers/peer0.org2.jnvr.com/tls/ca.crt peer channel join -b ${CHANNEL_NAME}.block
   CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/users/Admin@org3.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org3.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org3MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/peers/peer0.org3.jnvr.com/tls/ca.crt peer channel join -b ${CHANNEL_NAME}.block
   CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/users/Admin@org4.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org4.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org4MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/peers/peer0.org4.jnvr.com/tls/ca.crt peer channel join -b ${CHANNEL_NAME}.block

#Set the Anchors Peers in all Orgs
   peer channel update -o orderer.jnvr.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/Org1MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem
   CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.jnvr.com/users/Admin@org2.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org2.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org2MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.jnvr.com/peers/peer0.org2.jnvr.com/tls/ca.crt peer channel update -o orderer.jnvr.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/Org2MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem
   CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/users/Admin@org3.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org3.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org3MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/peers/peer0.org3.jnvr.com/tls/ca.crt peer channel update -o orderer.jnvr.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/Org3MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem
   CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/users/Admin@org4.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org4.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org4MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/peers/peer0.org4.jnvr.com/tls/ca.crt peer channel update -o orderer.jnvr.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/Org4MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem

#Make chaincode package
   export CHANNEL_NAME=channel1
   export CHAINCODE_NAME=foodcontrol
   export CHAINCODE_VERSION=1
   export CC_RUNTIME_LANGUAGE=golang
   export CC_SRC_PATH="../../../chaincode/$CHAINCODE_NAME/"
   export ORDERER_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem
   peer lifecycle chaincode package ${CHAINCODE_NAME}.tar.gz --path ${CC_SRC_PATH} --lang ${CC_RUNTIME_LANGUAGE} --label ${CHAINCODE_NAME}_${CHAINCODE_VERSION} >&log.txt

#Install Chaincode
   peer lifecycle chaincode install ${CHAINCODE_NAME}.tar.gz &> pckID.txt
   pckID=$(grep -o 'identifier.*' pckID.txt|cut -f2,3 -d:) #grab sring 'identifier' and forwards, after, the output is cut with delimeter ':' in this case there are 3 parts, where parts 2 and 3 are choosen
   rm pckID.txt
   CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.jnvr.com/users/Admin@org2.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org2.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org2MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.jnvr.com/peers/peer0.org2.jnvr.com/tls/ca.crt peer lifecycle chaincode install ${CHAINCODE_NAME}.tar.gz
   CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/users/Admin@org3.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org3.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org3MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/peers/peer0.org3.jnvr.com/tls/ca.crt peer lifecycle chaincode install ${CHAINCODE_NAME}.tar.gz
   CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/users/Admin@org4.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org4.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org4MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/peers/peer0.org4.jnvr.com/tls/ca.crt peer lifecycle chaincode install ${CHAINCODE_NAME}.tar.gz

#Endorsing Policies
   peer lifecycle chaincode approveformyorg --tls --cafile $ORDERER_CA --channelID $CHANNEL_NAME --name $CHAINCODE_NAME --version $CHAINCODE_VERSION --sequence 1 --waitForEvent --signature-policy "OR ('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')" --package-id $pckID
   CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/users/Admin@org3.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org3.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org3MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/peers/peer0.org3.jnvr.com/tls/ca.crt peer lifecycle chaincode approveformyorg --tls --cafile $ORDERER_CA --channelID $CHANNEL_NAME --name $CHAINCODE_NAME --version $CHAINCODE_VERSION --sequence 1 --waitForEvent --signature-policy "OR ('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')" --package-id $pckID
   CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/users/Admin@org4.jnvr.com/msp/ CORE_PEER_ADDRESS=peer0.org4.jnvr.com:7051 CORE_PEER_LOCALMSPID="Org4MSP" CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/peers/peer0.org4.jnvr.com/tls/ca.crt peer lifecycle chaincode approveformyorg --tls --cafile $ORDERER_CA --channelID $CHANNEL_NAME --name $CHAINCODE_NAME --version $CHAINCODE_VERSION --sequence 1 --waitForEvent --signature-policy "OR ('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')" --package-id $pckID

#Check Endorsing Policies
   peer lifecycle chaincode checkcommitreadiness --channelID $CHANNEL_NAME --name $CHAINCODE_NAME --version $CHAINCODE_VERSION --sequence 1 --signature-policy "OR ('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')" --output json

#Commit chaincode
   peer lifecycle chaincode commit -o orderer.jnvr.com:7050 --tls --cafile $ORDERER_CA --peerAddresses peer0.org1.jnvr.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.jnvr.com/peers/peer0.org1.jnvr.com/tls/ca.crt --peerAddresses peer0.org3.jnvr.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com/peers/peer0.org3.jnvr.com/tls/ca.crt --peerAddresses peer0.org4.jnvr.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com/peers/peer0.org4.jnvr.com/tls/ca.crt --channelID $CHANNEL_NAME --name $CHAINCODE_NAME --version $CHAINCODE_VERSION --sequence 1 --signature-policy "OR ('Org1MSP.peer','Org3MSP.peer','Org4MSP.peer')"

#Using chaincode functions
   peer chaincode invoke -o orderer.jnvr.com:7050 --tls --cafile $ORDERER_CA -C $CHANNEL_NAME -n $CHAINCODE_NAME -c '{"Args":["Set","id:3","jorge","naranja"]}'
```

Figura 85 - Documento script_jnvr_network_docker.sh

Este script irá correr os comandos dentro de *cli*, presentes na implementação anterior (secção 5.2.3.3). Onde se cria o canal, adicionam-se as organizações ao canal, definem-se os *Anchor Peers* para cada organização (de modo a poder existir comunicação entre organizações), faz-se o *setup* do *chaincode* e coloca-se um valor no *ledger* utilizando a função *Set* presente no *chaincode*.

3) Após ter os ficheiros prontos basta executar *script_jnvr_network.sh* a partir da diretoria *jnvr_network* com o comando `./script_jnvr_network.sh`.

Figura 86 - Excertos da execução de script_jnvr_network.sh

4) Depois, verifica-se que o *script_jnvr_network_docker.sh* foi enviado para o container utilizando o comando `ls`.

Figura 87 - Verificação da presença de script_jnvr_network_docker.sh

5) Finalmente utilizando o comando `./script_jnvr_network_docker.sh` será terminada a configuração da rede, obtendo-se o mesmo resultado da implementação anterior (secção 5.2.3.3).

Durante a criação dos scripts foram encontrados problemas de acesso quando se utilizava o comando

```
docker exec --user root cli
/opt/gopath/src/github.com/hyperledger/fabric/peer/script_jnvr_netwo
rk_docker.sh
```

Este comando iria permitir a execução automática de *script_jnvr_network_docker.sh* ao executar *script_jnvr_network.sh*. O facto deste comando não funcionar é a razão de ser necessário correr o *script* dentro do container *cli*.

Adicionalmente, é importante ter em conta que os *exports* estão a ser declarados dentro do *script*. Caso seja necessário de correr novos comandos que usem variáveis exportadas nos scripts, é necessário voltar a exportá-las.

5.2.3.5 Adicionar uma organização a uma rede em funcionamento

A tentativa de implementar a adição de uma organização a uma rede em funcionamento foi baseada no tutorial presente em [30] onde é utilizada a rede descrita na implementação anterior (secção 5.2.3.4), que contém quatro organizações de raiz, tendo-se o objetivo de adicionar uma quinta organização. O processo percorrido é descrito de seguida:

0) Antes de qualquer passo é necessário inicializar a rede, para isso utiliza-se os scripts descritos anteriormente (secção 5.2.3.4), ou colocar os comandos manualmente (secção 5.2.3.3).

1) Tendo a rede a correr cria-se a diretoria *AddOrg5* em *jnvr-network*. Depois começa-se por criar o ficheiro *org5-crypto-config.yaml*, que descreve a nova organização a ser criada, como se observa de seguida.

Figura 88 - Ficheiro org5-crypto-config.yaml

2) Com o ficheiro obtido anteriormente utiliza-se o comando `cryptogen generate --config=./org5-crypto-config.yaml`, na diretoria acabada de criar (AddOrg5) para gerar os materiais chave da organização 5 na diretoria *crypto-config*.

```
osboxes@osboxes:~/Desktop/jnvr_net/jnvr-network$ cd AddOrg5/
osboxes@osboxes:~/Desktop/jnvr_net/jnvr-network/AddOrg5$ cryptogen generate
--config=./org5-crypto-config.yaml
org5.jnvr.com
```

Figura 89 - Execução do comando cryptogen generate para a organização 5

3) Depois copia-se a diretoria *crypto-config* da rede inicializada anteriormente para o *crypto-config* da organização cinco, acabado de criar, com o comando `cp -r ../crypto-config/ordererOrganizations crypto-config/`.

4) De seguida utiliza-se o seguinte comando para criar o ficheiro *org5.json* na diretoria *channel-artifacts* da rede, onde se descreve a configuração da organização cinco em formato *json*.

```
export FABRIC_CFG_PATH=$PWD && configtxgen -printOrg Org5MSP > ../channel-artifacts/org5.json
```

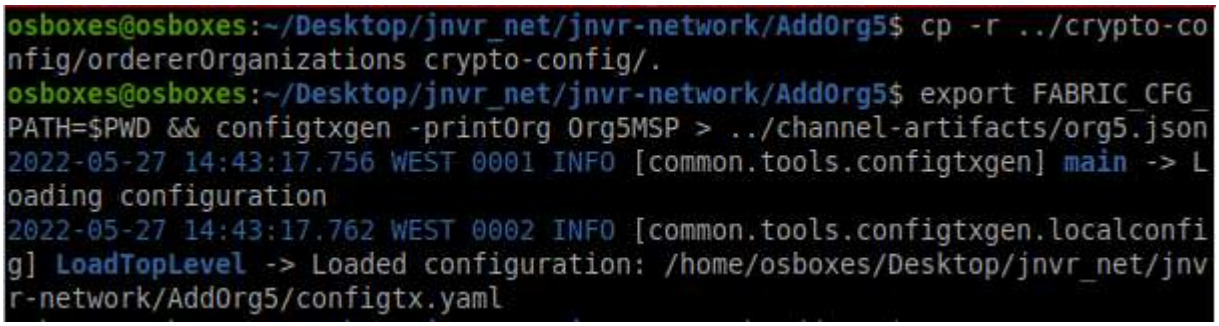


Figura 90 - Criação de org5.json

5) Nesta altura, entra-se no container cli (que se encontra a correr na Org1), com `docker exec -it cli /bin/sh` e correm-se os seguintes comandos.

```
export ORDERER_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tlsca.jnvr.com-cert.pem && export CHANNEL_NAME=channel1
```

```
peer channel fetch config config_block.pb -o orderer.jnvr.com:7050 -c $CHANNEL_NAME --tls --cafile $ORDERER_CA
```

Este processo irá criar o ficheiro *config_block.pb* que contém o último bloco de configuração do *blockchain*.

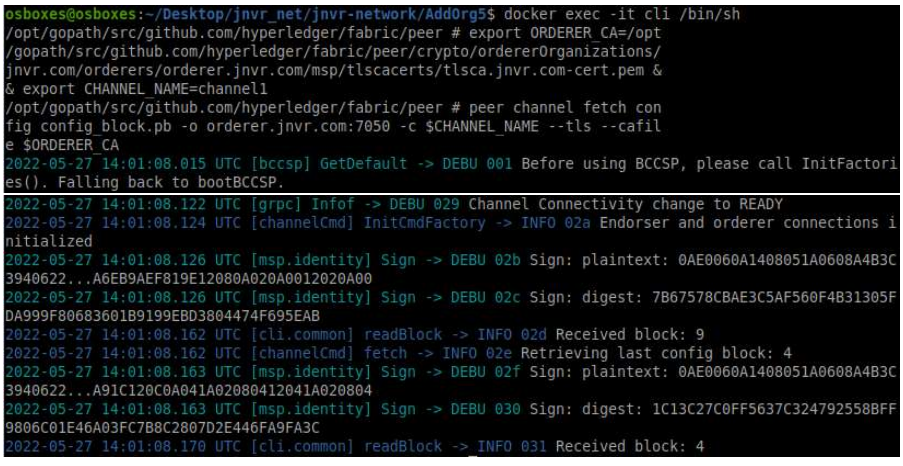


Figura 91 - Excertos da criação de config_block.pb

Como se observa na Figura 91 apresentada anteriormente, o último bloco de configuração é o número quatro. Isto deve-se a nesse momento existirem cinco blocos de configuração na rede, como se observa na seguinte figura (Figura 92).

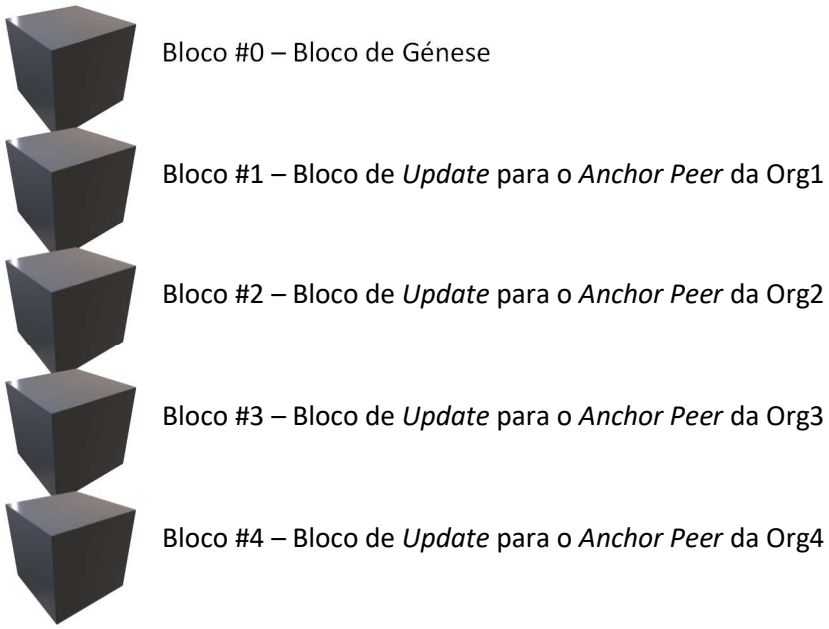


Figura 92 - Representação dos blocos presentes na rede

6) Seguidamente usa-se o seguinte comando, para extrair a porção de código relevante para o *update*, ou seja, a configuração de *ORG1MSP*, *ORG2MSP*, *ORG3MSP*, *ORG4MSP*, entre outros.

```
 configtxlator proto_decode --input config_block.pb --type common.Block
| jq .data.data[0].payload.data.config > config.json
```

7) Depois corre-se o comando apresentado para adicionar-se a definição de *ORG5MSP* (presente no *org5.json* criado anteriormente) antecedida do envelope definido no ficheiro *config.json*, criando o ficheiro *modified_config.json*.

```
jq -s '.[0] * {"channel_group":{"groups":{"Application":{"groups":
{"Org5MSP":.[1]}}}}}' config.json ./channel-artifacts/org5.json >
modified_config.json,
```

Figura 93 - Excerto de modified_config.json relativo à organização 5

8) Seguidamente usam-se os seguintes comandos, de modo a transformar os ficheiros *config.json* e *modified_config.json* para ficheiros do tipo *.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
```

9) Após ter os ficheiros da configuração inicial e da configuração com a organização cinco, usa-se o comando

```
configtxlator compute_update --channel_id $CHANNEL_NAME --original config.pb --updated modified_config.pb --output org5_update.pb,
```

Com o comando apresentado anteriormente, cria-se o ficheiro *org5_update.pb*, onde se computa a atualização da configuração, usando-se *modified_config.json*.

10) Usam-se os seguintes comandos para criar envelope a rodear *org5_update.pb*. Para isso transformou-se *org5_update.pb* em *JSON*, criou-se *org5_update_in_envelope.json* (já com o envelope) e transformou-se esse ficheiro em *.pb*.

```
configtxlator proto_decode --input org5_update.pb --type
common.ConfigUpdate | jq . > org5_update.json
```

```
echo '{"payload":{"header":{"channel_header":{"channel_id":"channel1",
"type":2}},"data":{"config_update":'$(cat org5_update.json)'}}}' | jq .
> org5_update_in_envelope.json
```

```
configtxlator proto_encode --input org5_update_in_envelope.json --type
common.Envelope --output org5_update_in_envelope.pb
```

11) Para terminar a ação da Org1, usa-se o comando `peer channel signconfigtx -f org5_update_in_envelope.pb`, que assina a modificação à rede, ou seja, declara que está de acordo com esta alteração.

```
/opt/gopath/src/github.com/hyperledger/fabric/peer # peer channel signconfigtx -f org5_update_in_envelope.pb
2022-05-27 16:55:56.920 UTC [bccsp] GetDefault -> DEBU 001 Before using BCCSP, please call InitFactories(). Falling ba
ck to bootBCCSP.
2022-05-27 16:55:56.922 UTC [bccsp] GetDefault -> DEBU 002 Before using BCCSP, please call InitFactories(). Falling ba
ck to bootBCCSP.
AiEA+6UxWokyURym+TpCftgggorZ2v3/MAteh3XLE9648usCIGQu06sNReBSrWUT
ZL2hoDbHYqjDUr/YJScsy8hx68ot
-----END CERTIFICATE-----
2022-05-27 16:55:56.948 UTC [msp] setupSigningIdentity -> DEBU 01a Signing identity expires at 2032-05-24 12:25:00 +00
00 UTC
2022-05-27 16:55:56.949 UTC [msp] GetDefaultSigningIdentity -> DEBU 01b Obtaining default signing identity
2022-05-27 16:55:56.949 UTC [channelCmd] InitCmdFactory -> INFO 01c Endorser and orderer connections initialized
2022-05-27 16:55:56.950 UTC [msp.identity] Sign -> DEBU 01d Sign: plaintext: 0AAA060A074F7267314D5350129E062D...726974
65727312002A0641646D696E73
2022-05-27 16:55:56.950 UTC [msp.identity] Sign -> DEBU 01e Sign: digest: 10B8064F12AC5E6D76BA7CB9430CB313997FB02EED34
9F1E368EC3FAF62C44B2
2022-05-27 16:55:56.951 UTC [msp.identity] Sign -> DEBU 01f Sign: plaintext: 0AE0060A1408021A06089C85C4940622...803896
BECAB4C14000191EAF74780711
2022-05-27 16:55:56.952 UTC [msp.identity] Sign -> DEBU 020 Sign: digest: CB89814C040BF52123ED608FC380C9909DD30E09C2EA
B0358CBE39BA0209A3CC
```

Figura 94 - Excertos da assinatura da modificação aplicada à rede

12) Devido à maneira de como foram definidas as políticas da rede, é necessário uma maioria das organizações autorizar uma alteração para esta entrar em vigor, como se observa na figura apresentada (Figura 95).

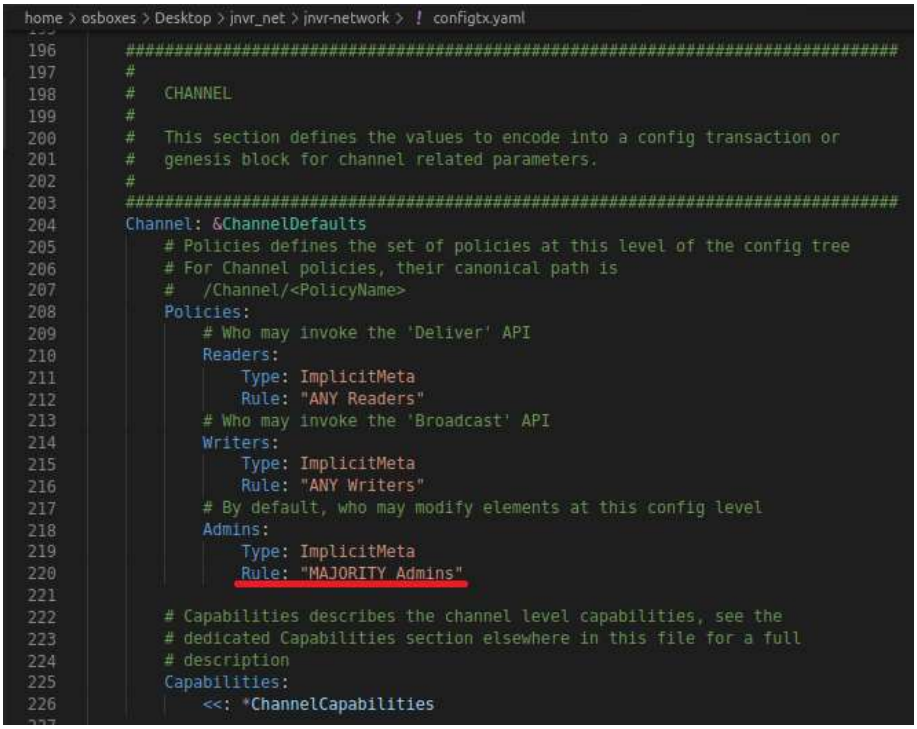


Figura 95 - Excerto da configuração da rede que dita ser necessário existir uma maioria para verificar alterações na rede

Como a rede a correr é constituída por quatro organizações é necessário a alteração ser assinada por pelo menos três organizações (para atingir a maioria). Para isso é necessário entrar em outras organizações e assinar a modificação, como se descreve de seguida:

a) Usam-se os comandos, para entrar no *container cli* através da Org2 e assinar a modificação

```
docker exec -e CORE_PEER_LOCALMSPID="Org2MSP" -e
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org2.jnvr.com/peers/peer0.org2.j
nvr.com/tls/ca.crt -e
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org2.jnvr.com/users/Admin@org2.jnvr.
com/msp/ -e CORE_PEER_ADDRESS=peer0.org2.jnvr.com:7051 -it cli
/bin/sh

peer channel signconfigtx -f org5_update_in_envelope.pb
```

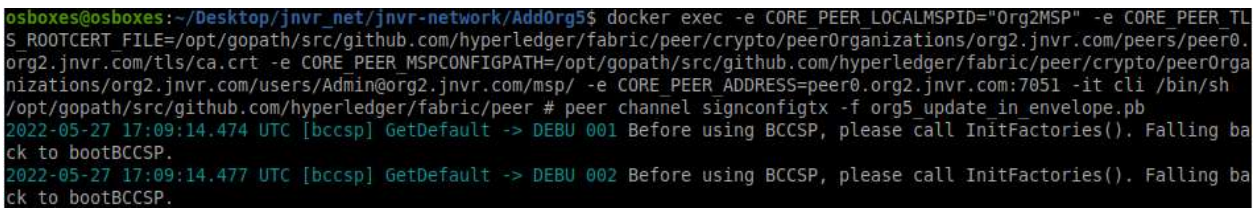


Figura 96 - Assinatura da modificação da rede por parte da organização 2

b) Finalmente utilizam-se os comandos apresentados de seguida, para entrar no container *cli* através da Org3 e fazer *update*, que automaticamente assina a modificação por parte da Org3.

```
docker exec -e CORE_PEER_LOCALMSPID="Org3MSP" -e
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org3.jnvr.com/peers/peer0.org3.j
nvr.com/tls/ca.crt -e
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org3.jnvr.com/users/Admin@org3.jnvr.
com/msp/ -e CORE_PEER_ADDRESS=peer0.org3.jnvr.com:7051 -it cli
/bin/sh
```

```
export
ORDERER_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto
/ordererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscace
rts/tlsca.jnvr.com-cert.pem && export CHANNEL_NAME=channel1
```

```
peer channel update -f org5_update_in_envelope.pb -c $CHANNEL_NAME -
o orderer.jnvr.com:7050 --tls --cafile $ORDERER_CA
```

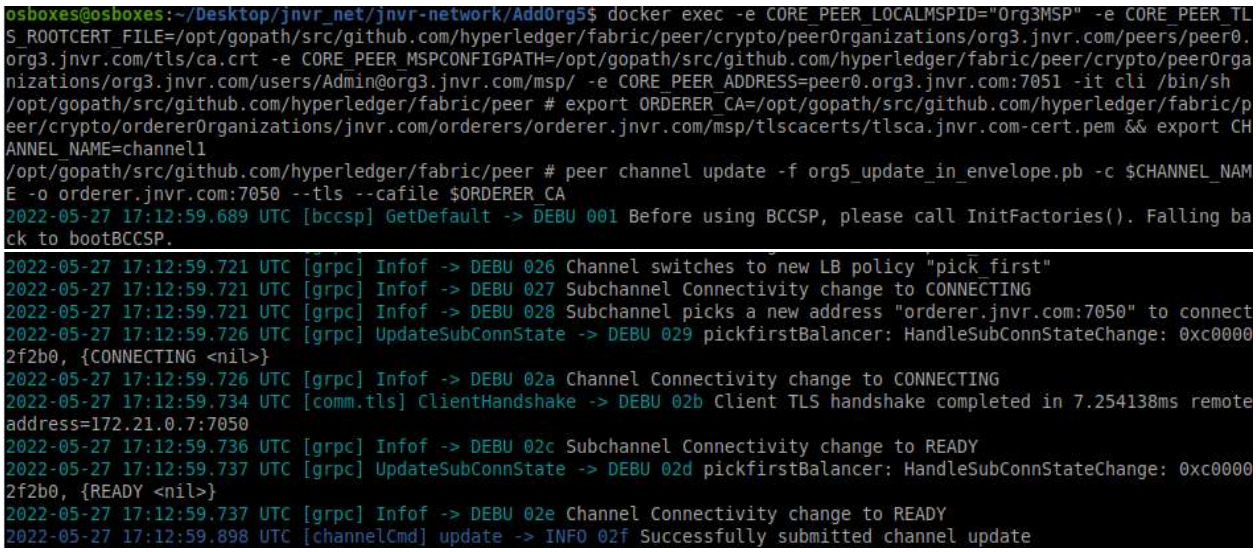
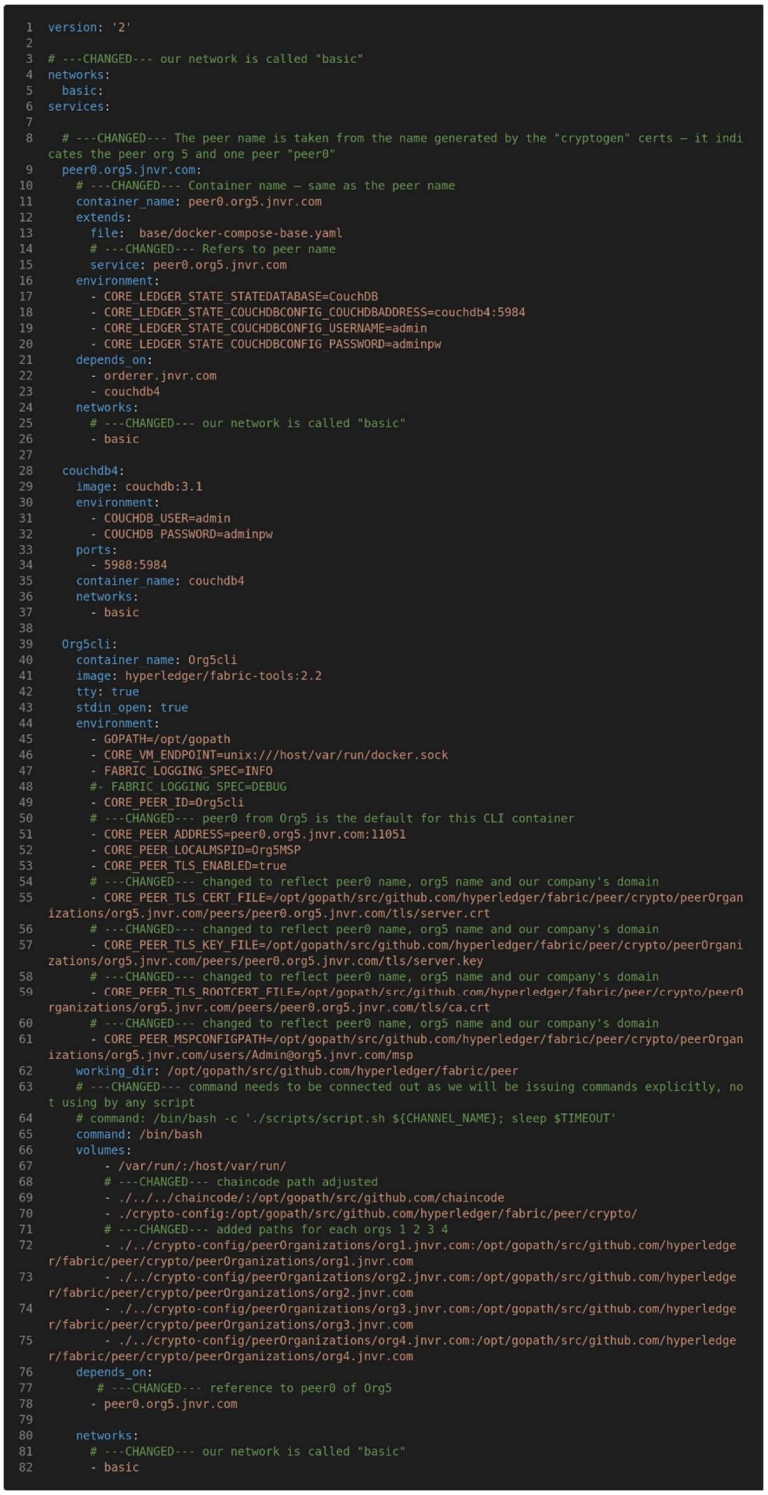


Figura 97 - Excertos da execução de update por parte da organização 3

13) Após ter a rede atualizada com a Org5 usa-se o comando `docker-compose -f docker-compose-couch-org5.yaml up -d` para inicializar um *peer* da organização, uma base de dados de *couchdb* e uma linha de comandos denominada Org5cli.



```yaml
version: '2'

# ---CHANGED--- our network is called "basic"
networks:
  basic:
services:

  # ---CHANGED--- The peer name is taken from the name generated by the "cryptogen" certs - it indicates the peer org 5 and one peer "peer0"
  peer0.org5.jnvr.com:
    # ---CHANGED--- Container name - same as the peer name
    container_name: peer0.org5.jnvr.com
    extends:
      file:  base/docker-compose-base.yaml
      # ---CHANGED--- Refers to peer name
      service: peer0.org5.jnvr.com
    environment:
      - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
      - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb4:5984
      - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
      - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
    depends_on:
      - orderer.jnvr.com
      - couchdb4
    networks:
      # ---CHANGED--- our network is called "basic"
      - basic

  couchdb4:
    image: couchdb:3.1
    environment:
      - COUCHDB_USER=admin
      - COUCHDB_PASSWORD=adminpw
    ports:
      - 5988:5984
    container_name: couchdb4
    networks:
      - basic

  Org5cli:
    container_name: Org5cli
    image: hyperledger/fabric-tools:2.2
    tty: true
    stdin_open: true
    environment:
      - GOPATH=/opt/gopath
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - FABRIC_LOGGING_SPEC=INFO
      #- FABRIC_LOGGING_SPEC=DEBUG
      - CORE_PEER_ID=Org5cli
      # ---CHANGED--- peer0 from Org5 is the default for this CLI container
      - CORE_PEER_ADDRESS=peer0.org5.jnvr.com:11051
      - CORE_PEER_LOCALMSPID=Org5MSP
      - CORE_PEER_TLS_ENABLED=true
      # ---CHANGED--- changed to reflect peer0 name, org5 name and our company's domain
      - CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org5.jnvr.com/peers/peer0.org5.jnvr.com/tls/server.crt
      # ---CHANGED--- changed to reflect peer0 name, org5 name and our company's domain
      - CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org5.jnvr.com/peers/peer0.org5.jnvr.com/tls/server.key
      # ---CHANGED--- changed to reflect peer0 name, org5 name and our company's domain
      - CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org5.jnvr.com/peers/peer0.org5.jnvr.com/tls/ca.crt
      # ---CHANGED--- changed to reflect peer0 name, org5 name and our company's domain
      - CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org5.jnvr.com/users/Admin@org5.jnvr.com/msp
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    # ---CHANGED--- command needs to be connected out as we will be issuing commands explicitly, not using by any script
    # command: /bin/bash -c './scripts/script.sh ${CHANNEL_NAME}; sleep $TIMEOUT'
    command: /bin/bash
    volumes:
        - /var/run/:/host/var/run/
        # ---CHANGED--- chaincode path adjusted
        - ./../../chaincode/:/opt/gopath/src/github.com/chaincode
        - ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
        # ---CHANGED--- added paths for each orgs 1 2 3 4
        - ./../crypto-config/peerOrganizations/org1.jnvr.com:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.jnvr.com
        - ./../crypto-config/peerOrganizations/org2.jnvr.com:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.jnvr.com
        - ./../crypto-config/peerOrganizations/org3.jnvr.com:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.jnvr.com
        - ./../crypto-config/peerOrganizations/org4.jnvr.com:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org4.jnvr.com
    depends_on:
       # ---CHANGED--- reference to peer0 of Org5
      - peer0.org5.jnvr.com

    networks:
      # ---CHANGED--- our network is called "basic"
      - basic
```

Figura 98 - Definição das configurações do novo peer, novo couchdb e org5cli

```
osboxes@osboxes:~/Desktop/jnvr_net/jnvr-network/AddOrg5$ docker-compose -f docker-compose-couch-org5.yaml up -d
[+] Running 3/3
 ⠿ Container couchdb4             Started
 ⠿ Container peer0.org5.jnvr.com  Started
 ⠿ Container Org5cli              Started
```

Figura 99 - Inicialização dos componentes da organização 5

14) Seguidamente usa-se `docker exec -it Org5cli /bin/sh` para entrar na linha de comandos acabada de criar e faz-se os seguintes *exports*.

```
export
ORDERER_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/or
dererOrganizations/jnvr.com/orderers/orderer.jnvr.com/msp/tlscacerts/tl
sca.jnvr.com-cert.pem && export CHANNEL_NAME=channel1
```

15) Depois usa-se o comando `peer channel fetch 0 channel1.block -o orderer.jnvr.com:7050 -c $CHANNEL_NAME --tls --cafile $ORDERER_CA` para ir trazer o bloco de génese do *blockchain*, ou seja, para ir buscar o bloco#0.

```
/opt/gopath/src/github.com/hyperledger/fabric/peer # peer channel fetch 0 channel1.block -o orderer.jnvr.com:7050 -c $CHANN
EL_NAME --tls --cafile $ORDERER_CA
2022-05-27 20:17:53.231 UTC [bccsp] GetDefault -> DEBU 001 Before using BCCSP, please call InitFactories(). Falling back to
 bootBCCSP.
2022-05-27 20:17:53.234 UTC [bccsp] GetDefault -> DEBU 002 Before using BCCSP, please call InitFactories(). Falling back to
 bootBCCSP.
2022-05-27 20:17:53.244 UTC [bccsp] GetDefault -> DEBU 003 Before using BCCSP, please call InitFactories(). Falling back to
 bootBCCSP.
2022-05-27 20:17:53.249 UTC [bccsp_sw] openKeyStore -> DEBU 004 KeyStore opened at [/opt/gopath/src/github.com/hyperledger/
fabric/peer/crypto/peerOrganizations/org1.jnvr.com/users/Admin@org1.jnvr.com/msp/keystore]...done
2022-05-27 20:17:53.249 UTC [msp] getPemMaterialFromDir -> DEBU 005 Reading directory /opt/gopath/src/github.com/hyperledge
r/fabric/peer/crypto/peerOrganizations/org1.jnvr.com/users/Admin@org1.jnvr.com/msp/signcerts
2022-05-27 20:17:53.254 UTC [grpc] Infof -> DEBU 020 ClientConn switching balancer to "pick_first"
2022-05-27 20:17:53.254 UTC [grpc] Infof -> DEBU 021 Channel switches to new LB policy "pick_first"
2022-05-27 20:17:53.255 UTC [grpc] Infof -> DEBU 022 Subchannel Connectivity change to CONNECTING
2022-05-27 20:17:53.255 UTC [grpc] Infof -> DEBU 023 Subchannel picks a new address "orderer.jnvr.com:7050" to connect
2022-05-27 20:17:53.255 UTC [grpc] UpdateSubConnState -> DEBU 024 pickfirstBalancer: HandleSubConnStateChange: 0xc000020fe0
, {CONNECTING <nil>}
2022-05-27 20:17:53.256 UTC [grpc] Infof -> DEBU 025 Channel Connectivity change to CONNECTING
2022-05-27 20:17:53.267 UTC [comm.tls] ClientHandshake -> DEBU 026 Client TLS handshake completed in 4.311385ms remoteaddre
ss=172.21.0.7:7050
2022-05-27 20:17:53.269 UTC [grpc] Infof -> DEBU 027 Subchannel Connectivity change to READY
2022-05-27 20:17:53.269 UTC [grpc] UpdateSubConnState -> DEBU 028 pickfirstBalancer: HandleSubConnStateChange: 0xc000020fe0
, {READY <nil>}
2022-05-27 20:17:53.269 UTC [grpc] Infof -> DEBU 029 Channel Connectivity change to READY
2022-05-27 20:17:53.270 UTC [channelCmd] InitCmdFactory -> INFO 02a Endorser and orderer connections initialized
2022-05-27 20:17:53.271 UTC [msp.identity] Sign -> DEBU 02b Sign: plaintext: 0AE0060A1408051A0608F1E3C4940622...E63F6D49802
912080A021A0012021A00
2022-05-27 20:17:53.272 UTC [msp.identity] Sign -> DEBU 02c Sign: digest: C05B2D0A48600E0643164983BAE10B94EB32AA0EA8F29F9B0
2C8E4653C44E0CE
2022-05-27 20:17:53.276 UTC [cli.common] readBlock -> INFO 02d Received block: 0
```

Figura 100 - Excertos do uso de peer channel fetch 0

16) Passando para o *localhost* correm-se os seguintes comandos para copiar channel1.block de *cli* para Org5cli.

```
docker cp
cli:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel1.block
./
```

```
docker cp channel1.block
Org5cli:/opt/gopath/src/github.com/hyperledger/fabric/peer/
```

17) No passo seguinte, deparou-se com um problema que não se conseguiu resolver devido ao *host peer0.prg5.jnvr.com* não reconhecido, ao correr o comando `peer channel join -b channel1.block`.

```
/opt/gopath/src/github.com/hyperledger/fabric/peer # peer channel join -b chann
el1.block
Error: error getting endorser client for channel: endorser client failed to con
nect to peer0.org5.jnvr.com:11051: failed to create new connection: connection
error: desc = "transport: error while dialing: dial tcp: lookup peer0.org5.jnvr
.com: no such host"
```

Figura 101 - Erro verificado ao tentar juntar o novo peer da organização 5 ao canal

Concluindo esta implementação, apesar de não ter sido possível completar a implementação e execução, é importante destacar que a operação de adição de uma organização a uma rede em funcionamento operação é algo que deve ser evitado. Isto deve-se à sua complexidade, implicando bastante *tinkering*. Esta sofisticação deve-se à natureza *permissioned* das redes em *hyperledger-fabric*, onde a entrada a um *channel* da rede é restringida.

5.2.3.6 Implementação de uma rede a correr em várias máquinas

Nesta implementação pretendeu-se criar uma rede com duas organizações, cada uma com dois *peers*, correndo cada *peer* numa máquina diferente. A implementação foi baseada no tutorial presente em [31]. Os passos utilizados para implementar uma rede com esta configuração são descritos de seguida:

1) Para criar a configuração utilizou-se o comando `multipass launch 18.04 --name vm<#> -d 7G`, quatro vezes, de modo a criar VMs de *Ubuntu 18.04* onde apenas se especifica o espaço de disco (sete *Gigabytes*) utilizado de modo a ter espaço suficiente para os documentos que irão ser utilizados no exemplo, sendo os restantes parâmetros colocados nos seus valores prefinidos, para os quatro *peers* presentes na *rede*.

2) De seguida criou-se uma diretoria local denominada *MultipassFolder*. Nesta diretoria estão os seguintes ficheiros, que podem ser obtidos com o comando `git clone https://github.com/jnvr/MultipassFolder.git` (não esquecer de dar permissões aos ficheiros a utilizar posteriormente). Neste caso apenas alguns irão ser utilizados entre os quais:

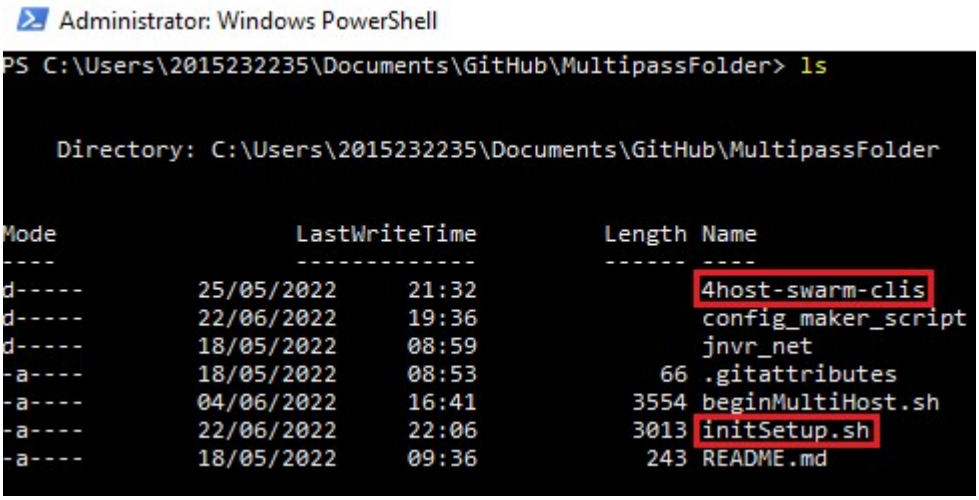


Figura 102 - Ficheiros presentes na diretoria MultipassFolder

- initSetup.sh: Usado na instalação de todos os pré-requisitos da implementação.

```
#!/bin/bash

echo " ============================================================================== "
echo "---Update and Upgrade---"
sudo apt update -y && sudo apt upgrade -y
echo " ============================================================================== "
echo "---Install Git---"
sudo apt-get install git
echo " ============================================================================== "
echo " ============================================================================== " >>versions.txt
echo "---Install Curl---"
sudo apt install curl
curl --version >versions.txt
echo " ============================================================================== " >>versions.txt
echo " ============================================================================== "
echo "---Installing Docker---"
sudo apt-get remove docker docker-engine docker.io
sudo apt install docker.io -y
sudo systemctl start docker
sudo systemctl enable docker
USER_NAME=$(whoami)
sudo usermod -a -G docker $USER_NAME
docker --version >>versions.txt
echo " ============================================================================== " >>versions.txt
echo " ============================================================================== "
echo "---Installing Docker-Compose---"
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)
-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo docker-compose --version >>versions.txt
echo " ============================================================================== " >>versions.txt
echo " ============================================================================== "
echo "---Install Fabric-Samples---"
sudo curl -sSL https://bit.ly/2ysbOFE | sudo bash -s
export PATH=./fabric-samples/bin:$PATH
echo " ============================================================================== "
echo "---Install NodeJS---"
wget http://nodejs.org/dist/v16.14.0/node-v16.14.0-linux-x64.tar.gz
sudo tar -C /usr/local --strip-components 1 -xzf node-v16.14.0-linux-x64.tar.gz
npm --version >>versions.txt
echo " ============================================================================== "
echo " ============================================================================== " >>versions.txt
echo "---Install Go---"
wget https://go.dev/dl/go1.18.1.linux-amd64.tar.gz
sudo rm -rf /usr/local/go && sudo tar -C /usr/local -xzf go1.18.1.linux-amd64.tar.gz
export PATH=$PATH:/usr/local/go/bin
go version >>versions.txt
echo " ============================================================================== "
echo "---Install Subversion---"
sudo apt-get install subversion -y
echo " ============================================================================== "
echo "---Install yq---"
sudo snap install yq

rm node-v16.14.0-linux-x64.tar.gz
rm go1.18.1.linux-amd64.tar.gz

export PATH=/home/ubuntu/fabric-samples/bin:$PATH
```

Figura 103 - Ficheiro initSetup.sh

- Diretoria 4host-swarm-clis: Onde estão os ficheiros utilizados para criar a rede em várias máquinas, que irão ser descritos futuramente. Caso não se pretenda fazer download da totalidade do repositório pode ser utilizado o comando `sudo svn checkout https://github.com/jnvr/MultipassFolder/trunk/4host-swarm-clis`. Seguido de `sudo chmod -R 755 4host-swarm-clis`, para dar permissões aos ficheiros.

3) Após a criação da diretoria *MultipassFolder* utilizou-se o seguinte comando para cada VM.

```
multipass mount <MultipassFolder directory location>\MultipassFolder\
vm<#>:/home/ubuntu/sharedFolder
```

No caso da implementação usa-se o seguinte comando, para criar uma diretoria partilhada entre a VM criada (localizada em `~/sharedFolder`) e a máquina local (localizada em `C:\Users\2015232235\Documents\Github\MultipassFolder\`).

```
multipass mount C:\Users\2015232235\Documents\Github\MultipassFolder\
vm<#>:/home/ubuntu/sharedFolder
```

Verifica-se que as VM estão a correr, a versão de ubuntu a ser usada e os seus *IPs* com o comando `multipass ls`.

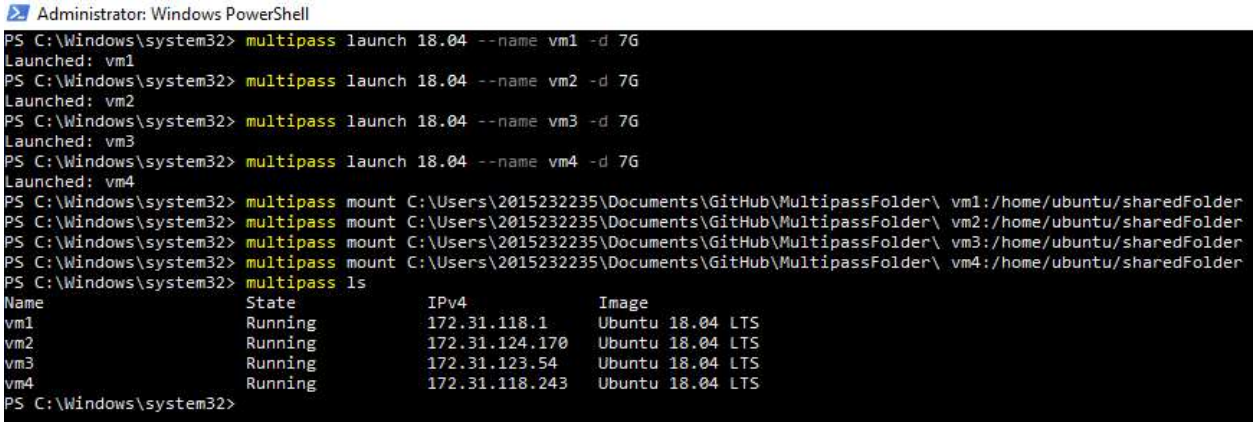


Figura 104 - Inicialização das VM, criação de diretorias partilhadas e listagem de máquinas existentes

Após a criação das VM e das suas ligações à máquina local é fulcral ter bastante cuidado na forma como as VM são utilizadas. Durante as diversas tentativas de uso, estas ficavam bloqueadas ao inicializar, ou os comandos ficavam bloqueados. Nestes casos inicialmente deve-se abrir e fechar o *shell*. Caso volte a bloquear o comando, deverá parar-se a VM, da maneira apresentada na Figura 105.

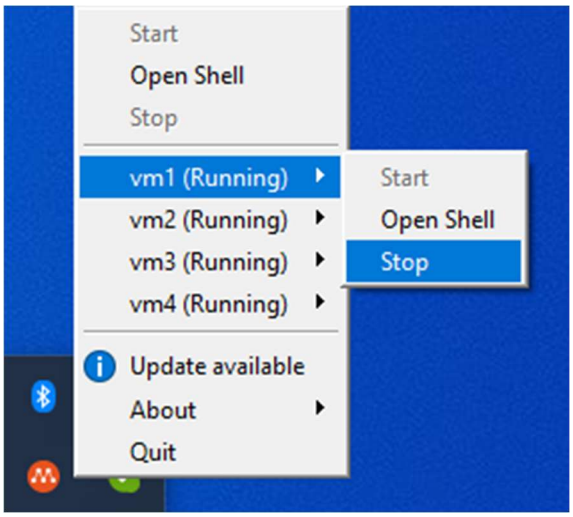


Figura 105 - Demonstração da paragem de uma máquina multipass

Adicionalmente, é de extrema importância não fechar o Shell enquanto este está a inicializar (como se observa na imagem). Isso poderá levar a que a VM fique corrompida não voltando a inicializar mais.

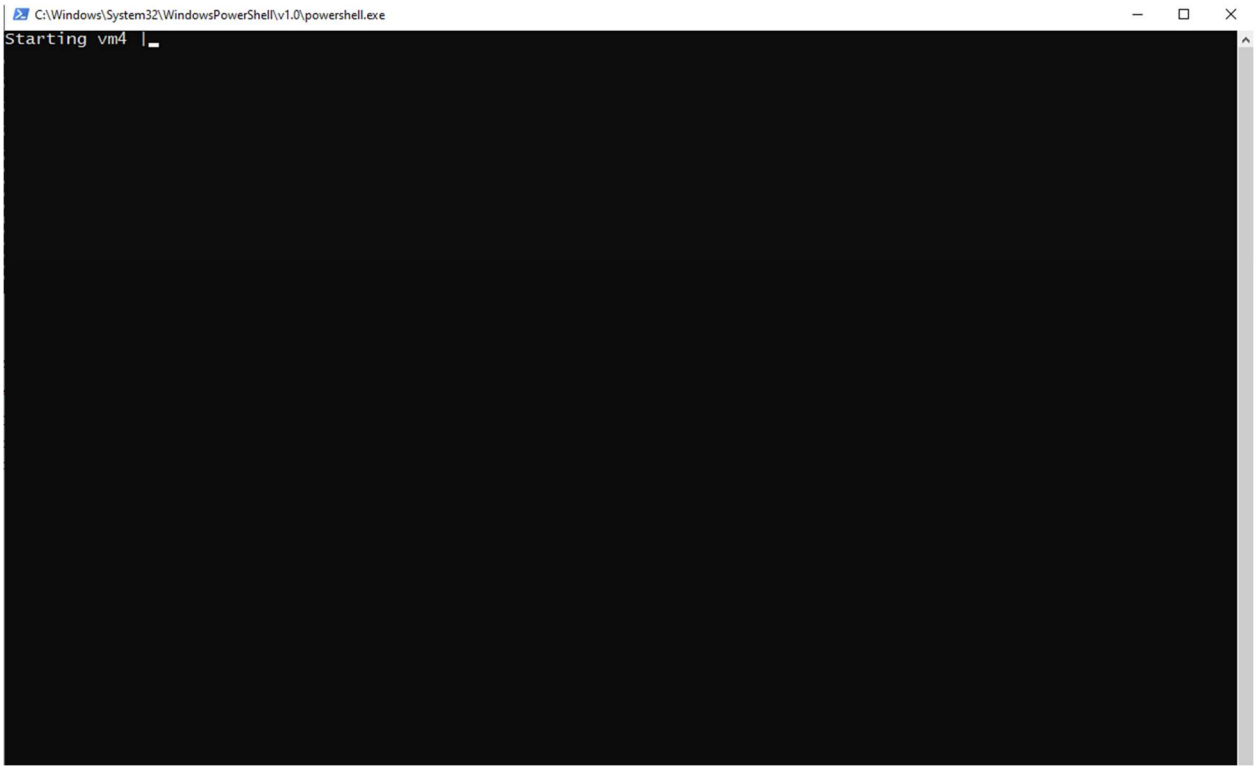


Figura 106 - Inicialização de uma máquina multipass

4) Após a criação das VMs e tendo em conta os cuidados ter, vai-se correr o *setup* inicial em cada VM. Para isso utiliza-se o seguinte comando, de modo a copiar o script de *setup* inicial, uma vez que é impossível mudar permissões e executar comandos na diretoria partilhada em *Multipass*.

```
cp ./sharedFolder/initSetup.sh ./
```

De seguida mudam-se as permissões do ficheiro com `chmod 755 initSetup.sh` e corre-se o script com `sudo ./initSetup.sh`.

Caso no decorrer da execução do script apareçam erros semelhantes ao apresentado de seguida, deve-se voltar a correr o script.

```
E: Could not get lock /var/lib/dpkg/lock-frontend - open (11: Resource temporarily unavailable)

E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), is another process using it?
```

Depois de executar *initSetup.sh* utiliza-se o comando `go version`. Caso o comando não seja reconhecido, corre-se novamente as linhas do script initSetup.sh relativas ao "---Install Go---" (ou seja, linhas 43-45).

A linguagem de programação *Go* estará bem instalada quando se utilizar `go version` e se obtiver o output: `go version go1.18.1 linux/amd64`

5) Após correr a configuração inicial em cada VM passa-se para a criação de um *overlay* de *Docker Swarm*, que permite a criação de uma rede distribuída em diversos containers de *Docker*, de modo a ser possível comunicar entre os *hosts*. Devido às redes de *overlay* encriptadas não funcionarem em *Windows*, para se ter uma solução mais flexível não se utilizou encriptação. Tendo isso em conta utilizou-se o comando `docker swarm init --advertise-addr <vm1 IP address>`.

Pode-se verificar o IP das VMs através do uso do comando `ifconfig` (dentro da VM) ou do uso de `multipass ls` (fora da VM, ou seja, na máquina local). Nesta implementação usa-se o comando.

```
sudo docker swarm init --advertise-addr 172.31.118.1
```

De modo a iniciar um *Docker Swarm* com o *host* da primeira VM. Seguido por `sudo docker swarm join-token manager` para obter o *token* para outros *hosts* entrarem na rede como *managers* (permite que cada nodo tenha permissões para gerir a rede). Neste exemplo obtém-se o seguinte comando.

```
docker swarm join --token SWMTKN-1-17mdrwru7onzq9vgx1jb1vd3nsis3iht8ocntj7m1gomwjumgq-e8cixyzvl8dbt9sz9kivoc4mf 172.31.118.1:2377
```

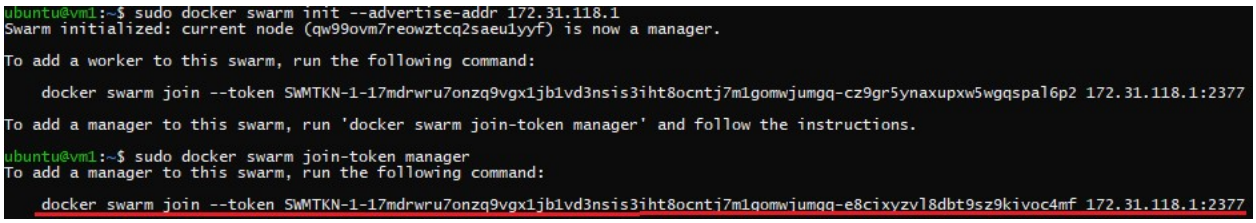


Figura 107 - Obtenção do comando para adicionar um manager à rede

Seguidamente utiliza-se o comando `sudo <output from join-token manager> --advertise-addr <VM# IP address>` para as outras três VMs. Especificamente, nesta circunstância, usa-se o seguinte comando para vm2.

```
sudo docker swarm join --token SWMTKN-1-
17mdrwru7onzq9vgx1jb1vd3nsis3iht8ocntj7m1gomwjumgq-
e8cixyzvl8dbt9sz9kivoc4mf 172.31.118.1:2377 --advertise-addr
172.31.124.170
```

O seguinte comando para vm3.

```
sudo docker swarm join --token SWMTKN-1-
17mdrwru7onzq9vgx1jb1vd3nsis3iht8ocntj7m1gomwjumgq-
e8cixyzvl8dbt9sz9kivoc4mf 172.31.118.1:2377 --advertise-addr
172.31.123.54
```

E o seguinte comando para vm4.

```
sudo docker swarm join --token SWMTKN-1-
17mdrwru7onzq9vgx1jb1vd3nsis3iht8ocntj7m1gomwjumgq-
e8cixyzvl8dbt9sz9kivoc4mf 172.31.118.1:2377 --advertise-addr
172.31.118.243
```

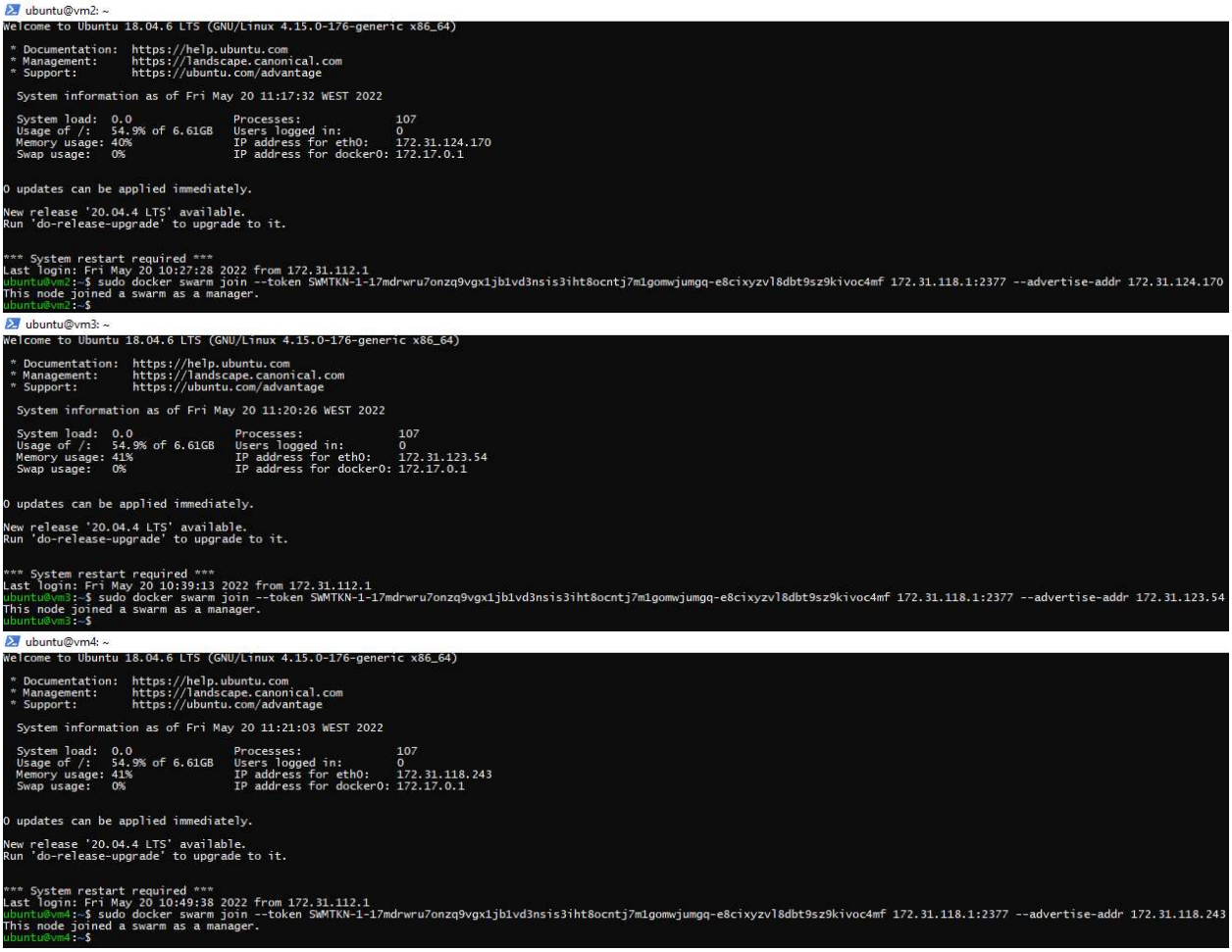


Figura 108 - Junção das VM 2,3,4 ao swarm criado pela máquina 1

Observa-se as máquinas ligadas ao nodo utilizando o comando `sudo docker node ls`. Este comando pode ser utilizado a partir de qualquer *host*. Para correr este comando é necessário uma maioria dos *hosts* do tipo *manager* estarem acessíveis.

```
ubuntu@vm1:~$ sudo docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
qw99ovm7reowztcq2saeu1yyf *   vm1        Ready     Active         Leader           20.10.7
xwtdclriynnkww1do0hvi66g0     vm2        Ready     Active         Reachable        20.10.7
bmy0xckw4a8201l63gpa1k8yg     vm3        Ready     Active         Reachable        20.10.7
wbxixbynv67npziaof005ufv4     vm4        Ready     Active         Reachable        20.10.7
ubuntu@vm1:~$
```

Figura 109 - Listagem de máquinas ligadas a nodo

Caso seja necessário modificar o leader utilizar `sudo docker node demote <VMs que não se quer como leader>` a partir da VM que se quer como leader. Seguido de `sudo docker node promote <as mesmas VMs introduzidas anteriormente>`.

Finalmente passa-se para a criação da rede de *overlay* em si, usando `sudo docker network create --attachable --driver overlay first-network`, no primeiro *host*.

Com o comando `sudo docker network ls`, observa-se que foi criada a rede de *overlay* denominada *first-network* em cada *host*.

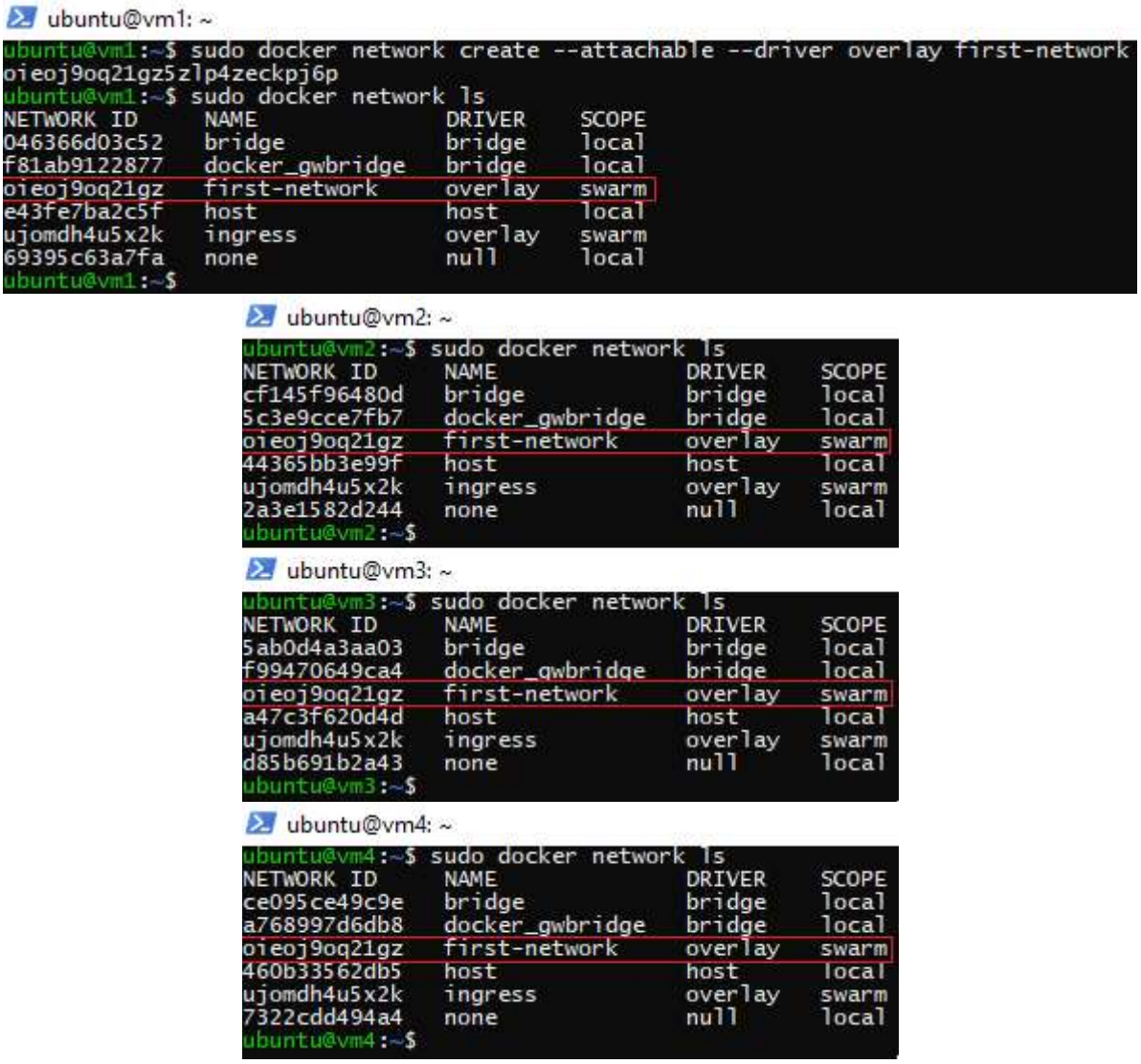


Figura 110 - Listagem de redes presentes a partir de cada máquina

6) Seguidamente, utiliza-se `cd fabric-samples/4host-swarm-clis/` para entrar na diretoria onde se encontram ficheiros de configuração da rede.

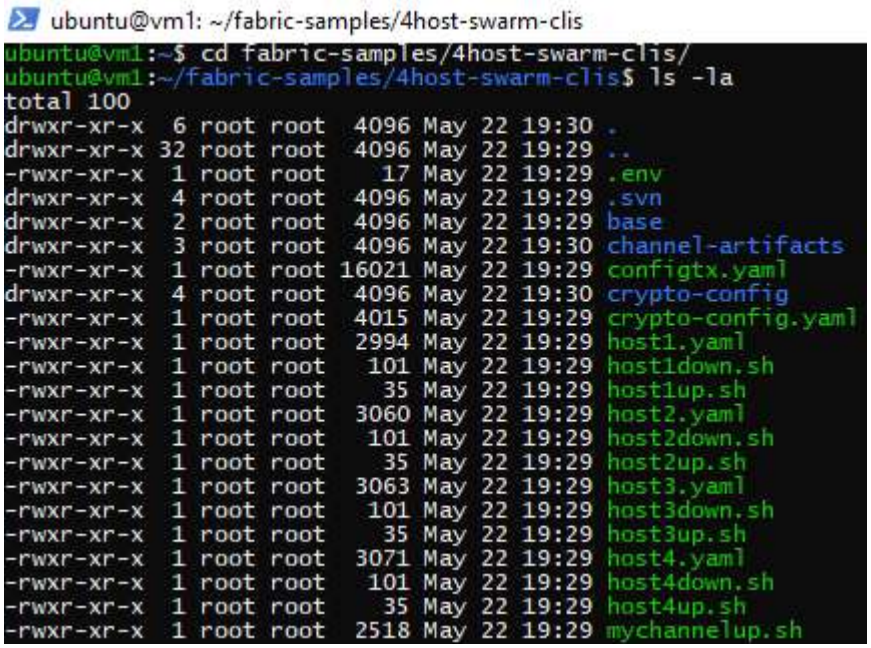


Figura 111 - Listagem dos ficheiros presentes na directoria 4host-swarm-clis

Os ficheiros denominados *host<#>up.yaml* iniciam os *containers* de *Docker* para cada *host* (ou seja, cada VM). Cada um destes documentos utiliza o *host<#>.yaml* correspondente, definindo a configuração de cada *host*, que se encontra caracterizado na Figura 112.

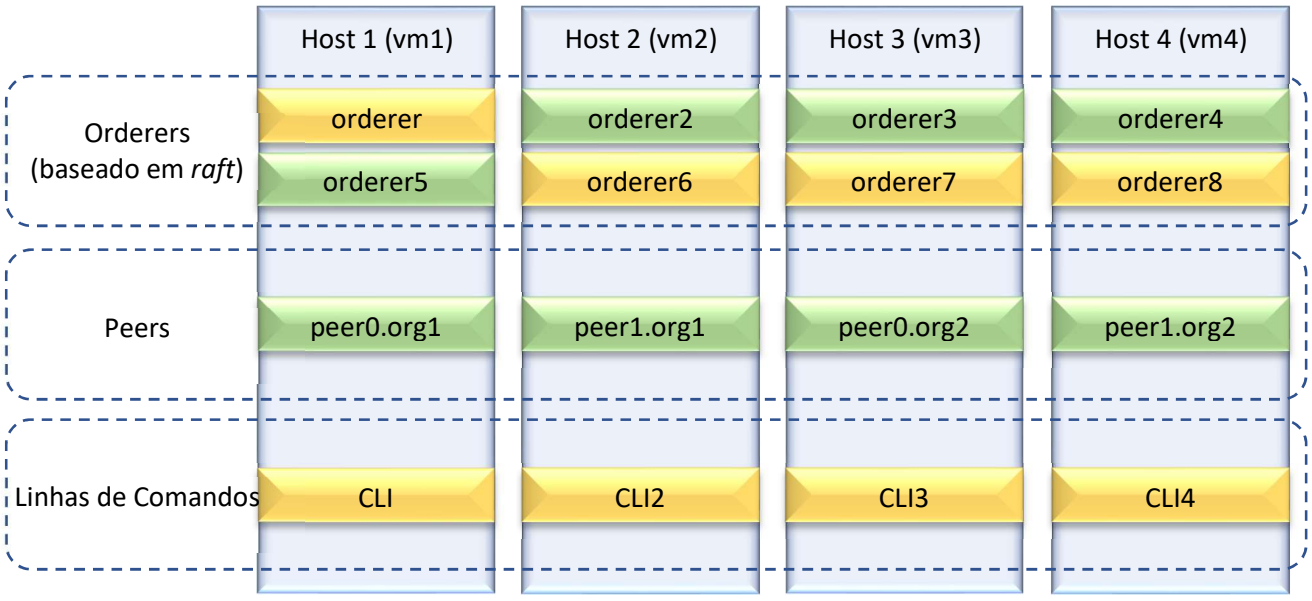


Figura 112 - Caracterização da configuração da rede

Cada VM tem então uma linha de comandos, um *peer* e dois *orderers* (um para o *peer* e outro para a linha de comandos) baseados em *raft* que significa que qualquer *orderer* pode ser acedido por qualquer *host*.

Caso seja necessário modificar a configuração, basta alterar os ficheiros *configtx.yaml*, *crypto-config.yaml*, *base/docker-compose-base.yaml* e *base/peer-base.yaml*, podendo modificar as organizações, número de *peers*, linhas de comandos, organizações, entre outros.

Após ter as novas configurações utilizam-se os comandos:

-----Geração material criptográfico-----

```
cryptogen generate --config=./crypto-config.yaml
```

```
export FABRIC_CFG_PATH=$PWD
```

-----Criação do bloco de génese-----

```
configtxgen -profile SampleMultiNodeEtcdRaft -channelID system-channel -outputBlock ./channel-artifacts/genesis.block
```

```
configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID mychannel
```

```
configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID mychannel -asOrg Org1MSP
```

```
configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID mychannel -asOrg Org2MSP
```

É de salientar que os comandos a serem corridos podem ser diferentes consoante a configuração. Pode-se desejar mais que dois *anchor peers* e ter o nome do *profiles* (presentes em *configtx.yaml*) diferentes.

7) Empregando-se o comando `./host<#>up.sh` para cada respetiva VM, são inicializados os *containers* de *Docker* em cada *host*.

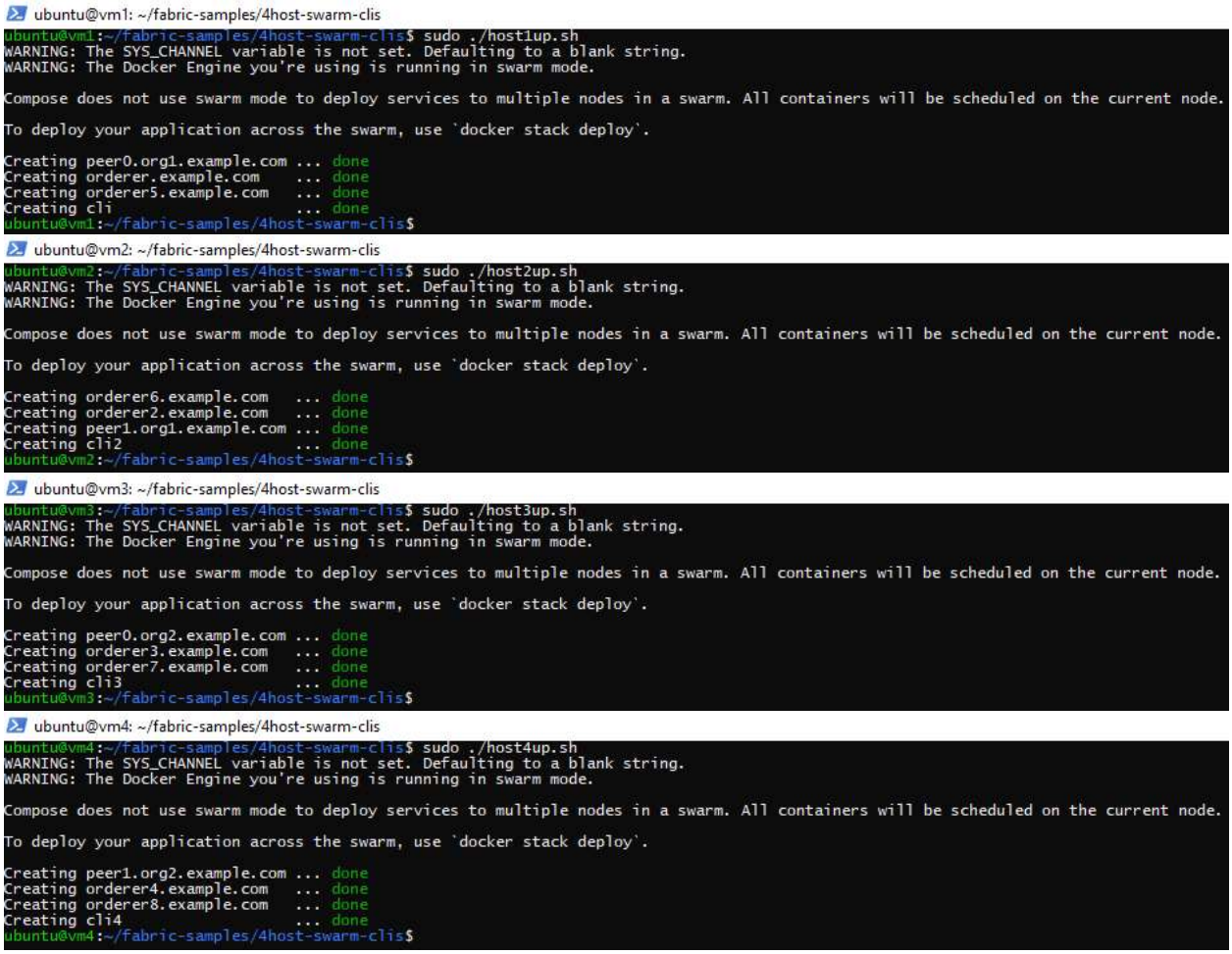


Figura 113 - Inicialização dos componentes em cada peer

Caso dê erro devido a um dos clis não conseguir ligar-se à rede com ID igual ao da *first-network* criada anteriormente, deverá parar-se e reinicializar a VM e tentar correr o script novamente.

8) Seguidamente usa-se o script mychannelup.sh para:

- Criar o canal denominado *mychannel*, fazendo uso do comando:

```
docker exec cli peer channel create -o orderer.example.com:7050 -c
mychannel -f ./channel-artifacts/channel.tx --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrg
anizations/example.com/orderers/orderer.example.com/msp/tlscacerts/t
lsca.example.com-cert.pem
```

- Juntar as organizações ao canal criado, que é realizado através dos seguintes comandos:

-----peer0.org1-----

```
docker exec cli peer channel join -b mychannel.block
```

-----peer1.org1-----

```
docker exec -e CORE_PEER_ADDRESS=peer1.org1.example.com:8051 -e
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org1.example.com/peers/peer1.org
1.example.com/tls/ca.crt cli peer channel join -b mychannel.block
```

-----peer0.org2-----

```
docker exec -e
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.ex
ample.com/msp -e CORE_PEER_ADDRESS=peer0.org2.example.com:9051 -e
CORE_PEER_LOCALMSPID="Org2MSP" -e
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org
2.example.com/tls/ca.crt cli peer channel join -b mychannel.block
```

-----peer1.org2-----

```
docker exec -e
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.ex
ample.com/msp -e CORE_PEER_ADDRESS=peer1.org2.example.com:10051 -e
CORE_PEER_LOCALMSPID="Org2MSP" -e
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org2.example.com/peers/peer1.org
2.example.com/tls/ca.crt cli peer channel join -b mychannel.block
```

- Definir os Anchor Peers, um para cada organização, com os comandos apresentados de seguida:

```
docker exec cli peer channel update -o orderer.example.com:7050 -c
mychannel -f ./channel-artifacts/Org1MSPanchors.tx --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrg
anizations/example.com/orderers/orderer.example.com/msp/tlscacerts/t
lsca.example.com-cert.pem
```

```
docker exec -e
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.ex
ample.com/msp -e CORE_PEER_ADDRESS=peer0.org2.example.com:9051 -e
CORE_PEER_LOCALMSPID="Org2MSP" -e
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org
2.example.com/tls/ca.crt cli peer channel update -o
```

```
orderer.example.com:7050 -c mychannel -f ./channel-
artifacts/Org2MSPanchors.tx --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrg
anizations/example.com/orderers/orderer.example.com/msp/tlscacerts/t
lsca.example.com-cert.pem
```

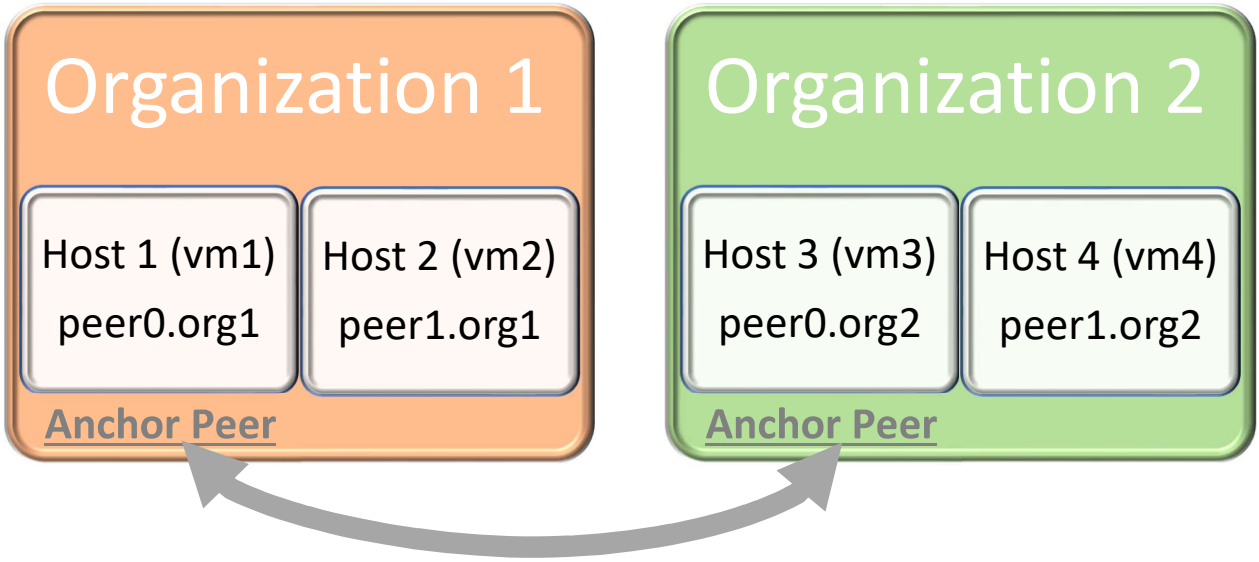


Figura 114 - Representação do funcionamento da rede

Utilizando então o comando `sudo ./mychannelup.sh` corre-se o *script* obtendo-se o seguinte resultado.

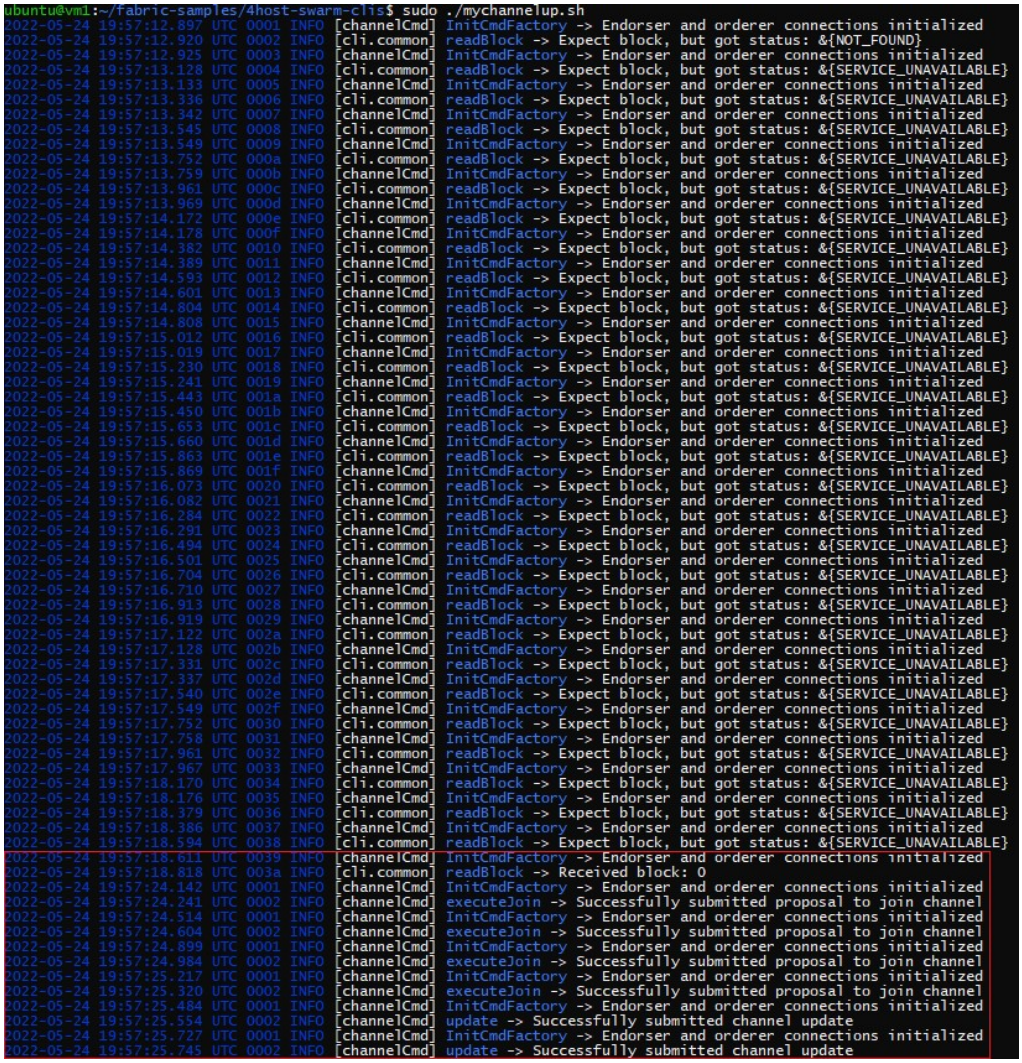


Figura 115 - Inicialização do canal

Como se observa na Figura 115, quando se correu o comando ocorreram algumas falhas de conexão, mas acabou por se realizar o pretendido. Para confirmar a inicialização do canal usa-se `sudo docker exec peer<#>.org<#>.example.com peer channel getinfo -c mychannel`, em cada *host*, consoante o *peer* presente.

Este comando deverá retornar a mesma *height* de *Blockchain* e o mesmo *BlockHash* em todos os *hosts*. Tem-se *height* igual a três devido ao primeiro bloco ser o bloco de génese, o segundo reter o *update* realizado para adicionar o *anchor peer* da organização 1 e o terceiro ter sido criado com o *update* realizado para adicionar o *anchor peer* da organização 2.

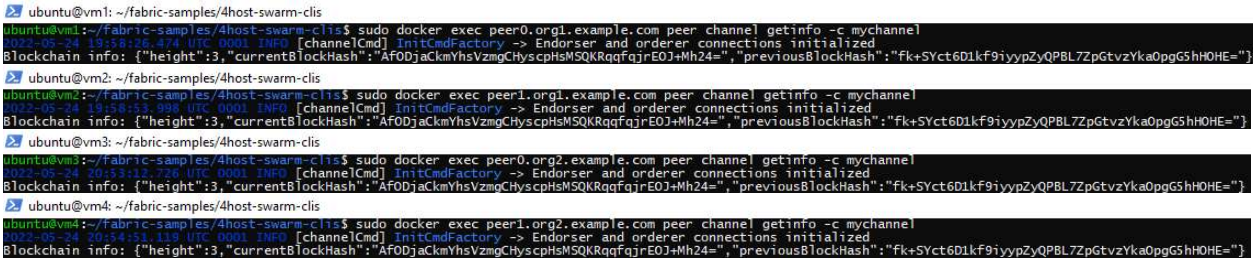


Figura 116 - Verificação das informações do blockchain a correr em cada máquina

9) Passando para o *chaincode*, utiliza-se *fabcar* que vem com *fabric-samples*. Caso se quisesse um *chaincode* diferente, utilizava-se os passos referentes ao chaincode realizados à implementação anterior da secção 5.2.3.2.

No *host* 1 usa-se `sudo chmod -R 777 ./../chaincode/fabcar/go`, a partir da diretoria *fabric-samples/4host-swarm-clis*, para evitar problemas de acesso.

Com os seguintes comandos ainda no *host* 1.

```
pushd ../chaincode/fabcar/go

GO111MODULE=on go mod vendor

popd
```

Muda-se a diretoria para a localização do *chaincode*, coloca-se a variável de estado *GO111MODULE* em *on* (para poder modificar a maneira como o *go* importa pacotes), usa-se `go mod vendor` (para criar uma diretoria localizada onde *go* será invocado, que contem cópias dos *packages* necessários para a *build* e testa-os) e volta-se para a diretoria inicial (neste caso *fabric-samples/4host-swarm-clis*).

Caso o comando go volte a não ser reconhecido utilizar novamente os comandos referidos no final do quarto passo desta implementação.

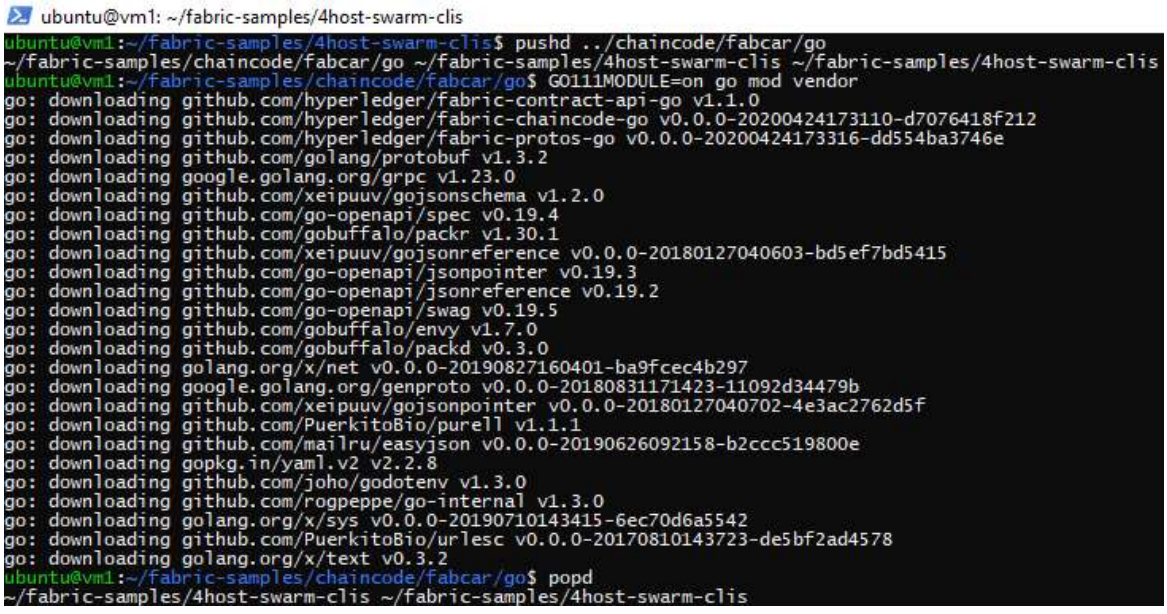


Figura 117 - Geração de packages necessários para a execução do chaincode

Estando a preparações em *go* terminadas passa-se para a criação do pacote do *chaincode*, utilizando o seguinte comando no primeiro *host*.

```
sudo docker exec cli peer lifecycle chaincode package fabcar.tar.gz --path /opt/gopath/src/github.com/chaincode/fabcar/go --label fabcar_1
```

Utilizando o pacote criado instala-se o *chaincode* em todas as VMs através do primeiro *host*, com os comandos.

-----peer0.org1-----

```
sudo docker exec cli peer lifecycle chaincode install fabcar.tar.gz
```

-----peer1.org1-----

```
sudo docker exec -e CORE_PEER_ADDRESS=peer1.org1.example.com:8051 -e CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/tls/ca.crt cli peer lifecycle chaincode install fabcar.tar.gz
```

-----peer0.org2-----

```
sudo docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp -e CORE_PEER_ADDRESS=peer0.org2.example.com:9051 -e CORE_PEER_LOCALMSPID="Org2MSP" -e CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt cli peer lifecycle chaincode install fabcar.tar.gz
```

-----peer1.org2-----

```
sudo docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp -e CORE_PEER_ADDRESS=peer1.org2.example.com:10051 -e CORE_PEER_LOCALMSPID="Org2MSP" -e CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer1.org2.example.com/tls/ca.crt cli peer lifecycle chaincode install fabcar.tar.gz
```

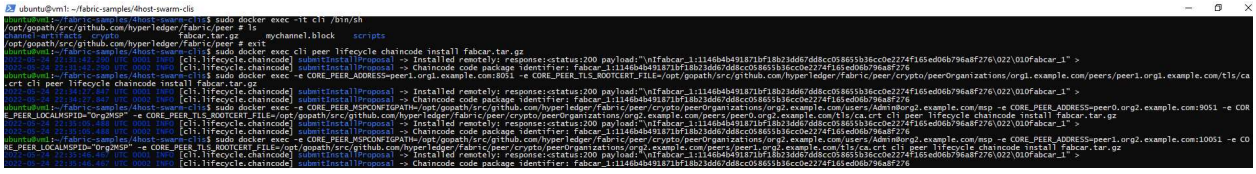


Figura 118 - Instalação do chaincode em cada peer

Seguidamente aprova-se o *chaincode* em ambas as organizações, utilizando os seguintes comandos ainda na *vm1*.

```
sudo docker exec cli peer lifecycle chaincode approveformyorg --tls
--cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrg
anizations/example.com/orderers/orderer.example.com/msp/tlscacerts/t
lsca.example.com-cert.pem --channelID mychannel --name fabcar --
version 1 --sequence 1 --waitForEvent --package-id <package-id
obtido nos comandos anteriores>
```

```
sudo docker exec -e
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.ex
ample.com/msp -e CORE_PEER_ADDRESS=peer0.org2.example.com:9051 -e
CORE_PEER_LOCALMSPID="Org2MSP" -e
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org
2.example.com/tls/ca.crt cli peer lifecycle chaincode
approveformyorg --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrg
anizations/example.com/orderers/orderer.example.com/msp/tlscacerts/t
lsca.example.com-cert.pem --channelID mychannel --name fabcar --
version 1 --sequence 1 --waitForEvent --package-id  <package-id
obtido nos comandos anteriores>
```

Nesta implementação o *package-id* obtido foi:

```
fabcar_1:1146b4b491871bf18b23dd67dd8cc058655b36cc0e2274f165ed06b796a
8f276
```

Usando-se então os comandos apresentados.

```
sudo docker exec cli peer lifecycle chaincode approveformyorg --tls
--cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrg
anizations/example.com/orderers/orderer.example.com/msp/tlscacerts/t
lsca.example.com-cert.pem --channelID mychannel --name fabcar --
version 1 --sequence 1 --waitForEvent --package-id
fabcar_1:1146b4b491871bf18b23dd67dd8cc058655b36cc0e2274f165ed06b796a
8f276
```

```
sudo docker exec -e
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.ex
ample.com/msp -e CORE_PEER_ADDRESS=peer0.org2.example.com:9051 -e
CORE_PEER_LOCALMSPID="Org2MSP" -e
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/f
abric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org
2.example.com/tls/ca.crt cli peer lifecycle chaincode
approveformyorg --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrg
anizations/example.com/orderers/orderer.example.com/msp/tlscacerts/t
lsca.example.com-cert.pem --channelID mychannel --name fabcar --
```

```
version 1 --sequence 1 --waitForEvent --package-id
fabcar_1:1146b4b491871bf18b23dd67dd8cc058655b36cc0e2274f165ed06b796a
8f276
```

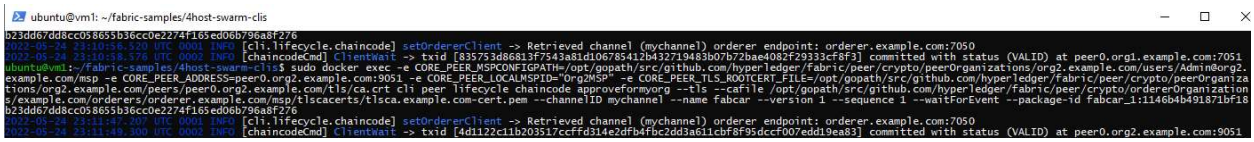


Figura 119 – Aprovação do chaincode em ambas as organizações

Para verificar que as organizações aprovaram o *chaincode*, corre-se o comando.

```
sudo docker exec cli peer lifecycle chaincode checkcommitreadiness -
--channelID mychannel --name fabcar --version 1 --sequence 1
```

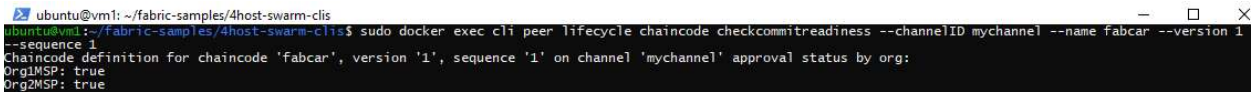


Figura 120 - Verificação da aprovação do chaincode

Finalmente falta apenas fazer *commit* do *chaincode*, que é feito com o comando, no host 1.

```
sudo docker exec cli peer lifecycle chaincode commit -o
orderer.example.com:7050 --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrgani
zations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.e
xample.com-cert.pem --peerAddresses peer0.org1.example.com:7051 --
tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizat
ions/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --
peerAddresses peer0.org2.example.com:9051 --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizat
ions/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt --
channelID mychannel --name fabcar --version 1 --sequence 1
```

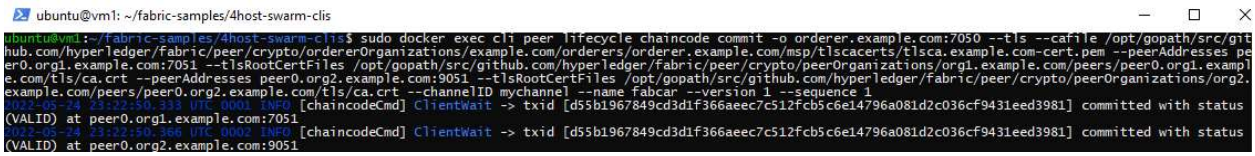


Figura 121 - Aprovação do chaincode

10) Passando à avaliação da rede, começa-se por utilizar a função do *chaincode* denominada *initLedger*, utilizada para popular o *ledger*, com o comando apresentado, no primeiro *host*.

```
sudo docker exec cli peer chaincode invoke -o
orderer3.example.com:9050 --tls true --cafile
```

```
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrg
anizations/example.com/orderers/orderer3.example.com/msp/tlscacerts/
tlsca.example.com-cert.pem -C mychannel -n fabcar --peerAddresses
peer0.org1.example.com:7051 --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrgani
zations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --
peerAddresses peer0.org2.example.com:9051 --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrgani
zations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt -c
'{"Args":["initLedger"]}'
```

Seguido pelo comando a utilizar a função *queryCar*, usada na procura de *items* no *ledger*, retornando a informação desse item, no *host* 1.

```
sudo docker exec cli peer chaincode query -n fabcar -C mychannel -c
'{"Args":["queryCar","CAR0"]}'
```

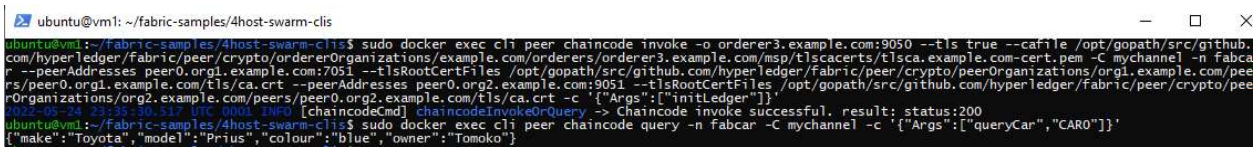


Figura 122 - Inicialização de dados no ledger e invocação da função queryCar

Para verificar que as linhas de comando definidas na configuração funcionam, corre-se o seguinte comando, em vm2.

```
sudo docker exec cli2 peer chaincode query -n fabcar -C mychannel -c
'{"Args":["queryCar","CAR0"]}'
```

O seguinte comando em vm3.

```
sudo docker exec cli3 peer chaincode query -n fabcar -C mychannel -c
'{"Args":["queryCar","CAR0"]}'
```

O seguinte comando em vm4.

```
sudo docker exec cli4 peer chaincode query -n fabcar -C mychannel -c
'{"Args":["queryCar","CAR0"]}'
```

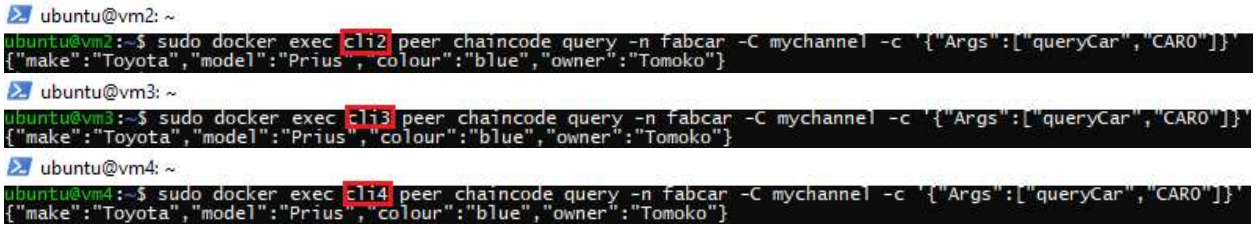


Figura 123 - Testagem da funcionalidade da rede em todos os hosts

Onde observa-se que o valor invocado pelo *host* 1 está presente em todos os outros *hosts*, usando as linhas de comando de cada organização.

5.2.3.7 Criação de um script para gerar redes consoante o número de Peers e Organizações são definidos pelo utilizador

Nesta implementação pretendeu-se criar um script que cria documentos de configuração de acordo com o número de organizações e p*eers* definidos pelo utilizador.

Para simplificar a execução da implementação, é escolhida a configuração com duas organizações e dois *peers* em cada organização tal como foi usado na prévia implementação (secção 5.2.3.6). Os passos utilizados na implementação são descritos de seguida:

0) Antes da criação da implementação é necessário, tal como na secção 5.2.3.6, ter os documentos da directoria *MultipassFolder* prontos. Para tal usa-se o comando `git clone https://github.com/jnvr/MultipassFolder.git`. Após o download do repositório mudar as permissões de acesso na directoria *./MultipassFolder/config_maker_script* com o comando `sudo chmod -R 777 ./MultipassFolder/config_maker_script`.

1) A criação dos documentos de configuração, irá ocorrer com a execução do script presente na diretoria *config_maker_script* denominado makeConfigFiles.sh, script este que utiliza uma mistura de comandos `yq` (para editar ficheiros .yaml e adicionar estruturas) e `sed` (para editar valores nos ficheiros .yaml). O script encontra-se dividido em várias secções, que são descritas de seguida:
    - Inicialização dos valores de peer e organizações: nesta secção é pedido ao utilizador para declarar o número de organizações e *peers* por organização
    - Configuração de *configtx.yaml*: inicialmente faz-se *reset* de *configtx.yaml*. Depois introduzem-se estruturas *template* por cada organização (presente em *OrgStructureConftx.yaml*) e editam-se os valores do *template* consoante a organização a ser declarada. Adicionam-se as novas organizações ao *profile TwoOrgsChannel*. Depois adiciona-se a estrutura *template* de *orderer* (presente em *OrdererStructureConftx.yaml*) para cada *peer* e editam-se os valores tendo em conta os *peers*. Seguidamente criam-se novos *orderers* no profile *SampleMultiNodeEtcdRaft.Orderer* e *SampleMultiNodeEtcdRaft.EtcdRaft* para cada *peer* e para *cli* (linha de

Figura 124 - Script makeConfigFiles.sh, inicialização dos valores de peer e organizações

```
1  #!/bin/sh
2
3      printf "\v $(tput setaf 7)$(tput setab 4)$(tput bold) All configurations are meant to be multi-h
   ost and will run a peer in each machine $(tput sgr 0)\n\n"
4      echo "Introduce number of Organizations"
5      read -r OrgNUM
6      echo "Introduce number of Peers per Organization"
7      read -r PeerNUM
8
9      lastPeerjter=$((PeerNUM-1))
```

comandos). Finalmente adicionam-se organizações ao *profile* *SampleMultiNodeEtcdRaft*.

```
###############################################################################################
################################### Configuration of configtx.yaml ############################
###############################################################################################

cat configtx1.yaml > configtx.yaml # Reset modifications in configtx.yaml

iter=1
ordererPort=7050
orgPort=7051
ordererNumber=2
while [ "$iter" -le "$OrgNUM" ];do
    printf "\n$(tput setaf 5)$(tput bold)Adding Org$iter to Organizations in configtx.yaml $(tput sgr 0)"
    item=$(cat OrgStructureConftx.yaml) yq -ei '.Organizations += env(item)' configtx.yaml # Add new Org structure t
o configtx.yaml
    sed  -i 's/!!merge //g' configtx.yaml # To remove an unexpected result of the previous command

    # Change values of the new Organization structure
    sed -i s/"OrgTitle"/"Org$iter"/ configtx.yaml
    sed -i s/"OrgName"/"Org$iter\MSP"/ configtx.yaml
    sed -i s/"MSPDirVar"/"crypto-config\/peerOrganizations\/org$iter\.example.com\/msp"/ configtx.yaml
    sed -i s/"readRule"/"\"OR('Org$iter\MSP.admin', 'Org$iter\MSP.peer', 'Org$iter\MSP.client')\""/ configtx.yaml
    sed -i s/"writeRule"/"\"OR('Org$iter\MSP.admin', 'Org$iter\MSP.client')\""/ configtx.yaml
    sed -i s/"adminRule"/"\"OR('Org$iter\MSP.admin')\""/ configtx.yaml
    sed -i s/"endorseRule"/"\"OR('Org$iter\MSP.peer')\""/ configtx.yaml
    sed -i s/"anchorPeer"/"peer0.org$iter\.example.com"/ configtx.yaml
    sed -i s/"portNum"/"$orgPort"/ configtx.yaml


    # Add new Org to TwoOrgsChannel profile
    printf "\n$(tput setaf 5)$(tput bold)Adding Org$iter to TwoOrgsChannel profile in configtx.yaml $(tput sgr 0)"
    m="*Org$iter" yq -i '.Profiles.TwoOrgsChannel.Application.Organizations += strenv(m)' configtx.yaml
    sed -i s/"'\*Org$iter'"/"*Org$iter"/ configtx.yaml # Remove quotes created from the previous command
    sed -i '/- Textsample/d' configtx.yaml

    jter=0
    while [ "$jter" -lt "$PeerNUM" ]; do
        if [ "$iter" -eq 1 ] && [ "$jter" -eq 0 ]; then
            printf "\n$(tput setaf 5)$(tput bold)Adding new orderer for Peer0.Org0 to SampleMultiNodeEtcdRaft.EtcdRa
ft profile in configtx.yaml $(tput sgr 0)"
            item=$(cat OrdererStructureConftx.yaml) yq -ei '.Profiles.SampleMultiNodeEtcdRaft.Orderer.EtcdRaft.Conse
nters += env(item)' configtx.yaml # Add new Orderer structure to configtx.yaml
            sed -i s/"hostOrderer"/"orderer.example.com"/ configtx.yaml
            sed -i s/"ordPort"/"$ordererPort"/ configtx.yaml
            sed -i s/"tlsOrderer"/"crypto-config\/ordererOrganizations\/example.com\/orderers\/orderer.example.com\/
tls\/server.crt"/ configtx.yaml
            sed -i '/- insertText/d' configtx.yaml

            # Add an orderer to a peer in SampleMultiNodeEtcdRaft.Orderer profile
            printf "\n$(tput setaf 5)$(tput bold)Adding new orderer for Peer0.Org0 to SampleMultiNodeEtcdRaft.Ordere
r profile in configtx.yaml $(tput sgr 0)"
            m="orderer.example.com:$ordererPort" yq -i '.Profiles.SampleMultiNodeEtcdRaft.Orderer.Addresses += stren
v(m)' configtx.yaml
            sed -i '/- Textinsert/d' configtx.yaml
            ordererPort=$((ordererPort+1000))

        elif [ "$iter" -eq "$OrgNUM" ] && [ "$jter" -eq "$lastPeerjter" ]; then

            # Add an orderer to a peer in SampleMultiNodeEtcdRaft.EtcdRaft profile
            printf "\n$(tput setaf 5)$(tput bold)Adding new orderer for Peer$jter.Org$iter to SampleMultiNodeEtcdRaf
t.EtcdRaft profile in configtx.yaml $(tput sgr 0)"
            item=$(cat OrdererStructureConftx.yaml) yq -ei '.Profiles.SampleMultiNodeEtcdRaft.Orderer.EtcdRaft.Conse
nters += env(item)' configtx.yaml # Add new Orderer structure to configtx.yaml
            sed -i s/"hostOrderer"/"orderer$ordererNumber.example.com"/ configtx.yaml
            sed -i s/"ordPort"/"$ordererPort"/ configtx.yaml
            sed -i s/"tlsOrderer"/"crypto-config\/ordererOrganizations\/example.com\/orderers\/orderer$ordererNumber
.example.com\/tls\/server.crt"/ configtx.yaml

            # Add an orderer to a peer in SampleMultiNodeEtcdRaft.Orderer profile
            printf "\n$(tput setaf 5)$(tput bold)Adding new orderer for Peer$jter.Org$iter to SampleMultiNodeEtcdRaf
t.Orderer profile in configtx.yaml $(tput sgr 0)"
            m="orderer$ordererNumber.example.com:$ordererPort" yq -i '.Profiles.SampleMultiNodeEtcdRaft.Orderer.Addr
esses += strenv(m)' configtx.yaml
            ordererNumber=$((ordererNumber+1))
            ordererPort=$((ordererPort+1000))

            # Add an orderer for cli to SampleMultiNodeEtcdRaft.EtcdRaft profile
            printf "\n$(tput setaf 5)$(tput bold)Adding new orderer for cli to SampleMultiNodeEtcdRaft.EtcdRaft prof
ile in configtx.yaml $(tput sgr 0)"
            item=$(cat OrdererStructureConftx.yaml) yq -ei '.Profiles.SampleMultiNodeEtcdRaft.Orderer.EtcdRaft.Conse
nters += env(item)' configtx.yaml # Add new Orderer structure to configtx.yaml
            sed -i s/"hostOrderer"/"orderer$ordererNumber\.example.com"/ configtx.yaml
            sed -i s/"ordPort"/"$ordererPort"/ configtx.yaml
            sed -i s/"tlsOrderer"/"crypto-config\/ordererOrganizations\/example.com\/orderers\/orderer$ordererNumber
.example.com\/tls\/server.crt"/ configtx.yaml

            # Add an orderer for cli to SampleMultiNodeEtcdRaft.Orderer profile
            printf "\n$(tput setaf 5)$(tput bold)Adding new orderer for cli to SampleMultiNodeEtcdRaft.Orderer profi
le in configtx.yaml $(tput sgr 0)"
            m="orderer$ordererNumber.example.com:$ordererPort" yq -i '.Profiles.SampleMultiNodeEtcdRaft.Orderer.Addr
esses += strenv(m)' configtx.yaml
            ordererNumber=$((ordererNumber+1))
            ordererPort=$((ordererPort+1000))

        else
            # Add an orderer to a peer in SampleMultiNodeEtcdRaft.EtcdRaft profile
            printf "\n$(tput setaf 5)$(tput bold)Adding new orderer for Peer$jter.Org$iter to SampleMultiNodeEtcdRaf
t.EtcdRaft profile in configtx.yaml $(tput sgr 0)"
            item=$(cat OrdererStructureConftx.yaml) yq -ei '.Profiles.SampleMultiNodeEtcdRaft.Orderer.EtcdRaft.Conse
nters += env(item)' configtx.yaml # Add new Orderer structure to configtx.yaml
            sed -i s/"hostOrderer"/"orderer$ordererNumber\.example.com"/ configtx.yaml
            sed -i s/"ordPort"/"$ordererPort"/ configtx.yaml
            sed -i s/"tlsOrderer"/"crypto-config\/ordererOrganizations\/example.com\/orderers\/orderer$ordererNumber
.example.com\/tls\/server.crt"/ configtx.yaml

            # Add an orderer to a peer in SampleMultiNodeEtcdRaft.Orderer profile
            printf "\n$(tput setaf 5)$(tput bold)Adding new orderer for Peer$jter.Org$iter to SampleMultiNodeEtcdRaf
t.Orderer profile in configtx.yaml $(tput sgr 0)"
            m="orderer$ordererNumber.example.com:$ordererPort" yq -i '.Profiles.SampleMultiNodeEtcdRaft.Orderer.Addr
esses += strenv(m)' configtx.yaml
            ordererNumber=$((ordererNumber+1))
            ordererPort=$((ordererPort+1000))
        fi
        jter=$((jter+1))
    done

    # Add new Org to SampleMultiNodeEtcdRaft profile
    m="*Org$iter" yq -i '.Profiles.SampleMultiNodeEtcdRaft.Consortiums.SampleConsortium.Organizations += strenv(m)'
 configtx.yaml
    sed -i s/"'\*Org$iter'"/"*Org$iter"/ configtx.yaml # Remove quotes created from the previous command
    sed -i '/- sampleText/d' configtx.yaml

    orgPort=$((orgPort+PeerNUM*1000))
    iter=$((iter+1))
done
sed  -i 's/!!merge //g' configtx.yaml # To remove unexpected results from the previous while loop
```

Figura 125 - Script makeConfigFiles.sh, configuração de configtx.yaml

- Configuração de *crypto-config.yaml*: inicialmente faz-se *reset* de *crypto-config.yaml*. Depois adicionam-se os valores de *orderer* consoante o número de *peers*. Finalmente, adiciona-se a estrutura *template* (presente em *createOrgCryCon.yaml*) para cada organização e editam-se os valores do *template* conforme a organização.

```
116 ###############################################################################################
117 ################################### Configuration of crypto-config.yaml #######################
118 ###############################################################################################
119
120 cat crypto-config1.yaml > crypto-config.yaml # Reset modifications in crypto-config.yaml
121
122 lastPortOrderer=$((ordererPort-1000))
123 ordererNumber=$((ordererNumber-1))
124 iter=2
125 while [ "$iter" -le "$ordererNumber" ]; do
126     printf "\n$(tput setaf 6)$(tput bold)Adding orderer$iter to crypto-config.yaml $(tput sgr 0)"
127     m="Hostname: orderer$iter" yq -i '.OrdererOrgs[0].Specs += strenv(m)' crypto-config.yaml
128     sed -i s/"'\Hostname: orderer$iter'"/"Hostname: orderer$iter"/ crypto-config.yaml # Remove quotes created
     from the previous command
129     iter=$((iter+1))
130 done
131
132 printf "\n$(tput setaf 6)$(tput bold)Adding Org1 to crypto-config.yaml $(tput sgr 0)"
133 yq -i ".PeerOrgs[0].Template.Count = $PeerNUM" crypto-config.yaml # Edit peer number of Org1
134 yq -i ".PeerOrgs[0].Users.Count = $lastPeerjter" crypto-config.yaml # Edit user number of Org1
135
136 iter=1
137 while [ "$iter" -lt "$OrgNUM" ];do
138     i=$((iter+1))
139     printf "\n$(tput setaf 6)$(tput bold)Adding Org$i to crypto-config.yaml $(tput sgr 0)"
140     item=$(cat createOrgCryCon.yaml) yq -ei '.PeerOrgs += env(item)' crypto-config.yaml # Add new Org structur
    e to crypto-config.yaml "https://stackoverflow.com/questions/68321476/insert-multiple-lines-of-one-attribute-i
    n-yaml-file-with-yq-v4-x"
141     yq -ei ".PeerOrgs[${iter}].Template.Count = ${PeerNUM}" crypto-config.yaml # Edit peer number of Org<#>
142     yq -ei ".PeerOrgs[${iter}].Users.Count = ${lastPeerjter}" crypto-config.yaml # Edit user number of Org<#>
143
144     # First try where there were problems passing strings: "Error: parsing expression: Lexer error: could not
     match text"
145     # yq -i ".PeerOrgs[${iter}].Domain = ""org$i.example.com""" crypto-config.yaml #
146     # yq -ei ".PeerOrgs[${iter}].Name |= Org$i" crypto-config.yaml
147
148     sed -i s/"nameInput"/"Org$i"/ crypto-config.yaml # Changing organization Name
149     sed -i s/"domainInput"/"org$i.example.com"/ crypto-config.yaml # Changing organization Domain
150     iter=$((iter+1))
151 done
```

Figura 126 - Script makeConfigFiles.sh, configuração de crypto-config.yaml

- Configuração de *docker-compose-base.yaml*: inicialmente apagam-se as modificações realizadas previamente. Depois adiciona-se a estrutura *template* (presente em *peerStructureDocker.yaml*) todos *peers* na rede e editam-se os valores para cada *peer*.

```
154 ##########################################################################################################
155 ############################ Configuration of base/docker-compose-base.yaml ##############################
156 ##########################################################################################################
157
158 sed -i '22,$d' docker-compose-base.yaml # Reset docker-compose-base.yaml file to its original form
159
160 iter=1
161 peerPort=7051
162 initpeerPort=7051
163 while [ "$iter" -le "$OrgNUM" ]; do
164     bootOrgIter=$((iter-1))
165     jter=0
166     while [ "$jter" -lt "$PeerNUM" ]; do
167         # Add peers to the Organization
168         printf "\n$(tput setaf 3)$(tput bold)Adding peer$jter.Org$iter to docker-compose-base.yaml $(tput sgr 0)"
169         item=$(cat peerStructureDocker.yaml) yq -ei '.services += env(item)' docker-compose-base.yaml
170         sed -i s/"peerName"/"peer$jter"/ docker-compose-base.yaml
171         sed -i s/"orgName"/"org$iter.example.com"/ docker-compose-base.yaml
172         sed -i s/"peerPort"/"$peerPort"/g docker-compose-base.yaml #In this command it is necessary to use 'g' mod
    ifier due to sed only replacing the first instance in a line by default
173         sed -i s/"peerchainPort"/"$((peerPort+1))"/ docker-compose-base.yaml
174
175         # Define CORE_PEER_GOSSIP_BOOTSTRAP as every other peer in the Organization
176         bootIter=0
177         bootPeerPort=$((initpeerPort+1000*PeerNUM*bootOrgIter))
178         while [ "$bootIter" -lt "$PeerNUM" ]; do
179             if [ "$bootIter" -eq "$jter" ]; then
180                 bootIter=$((bootIter+1))
181                 bootPeerPort=$((bootPeerPort+1000))
182             else
183                 sed -i s/"peerBoot"/"peer$bootIter"/ docker-compose-base.yaml
184                 sed -i s/"orgBoot"/"org$iter.example.com"/ docker-compose-base.yaml
185                 sed -i s/"portBoot"/"$bootPeerPort peerBoot.orgBoot:portBoot"/ docker-compose-base.yaml
186                 bootIter=$((bootIter+1))
187                 bootPeerPort=$((bootPeerPort+1000))
188             fi
189         done
190         sed -i 's/ peerBoot.orgBoot:portBoot//' docker-compose-base.yaml
191
192         sed -i s/"orgMSP"/"Org$iter\MSP"/ docker-compose-base.yaml
193         sed -i s/"mspVolume"/"..\/crypto-config\/peerOrganizations\/org$iter.example.com\/peers\/peer$jter.org$ite
    r.example.com\/msp:\/etc\/hyperledger\/fabric\/msp"/ docker-compose-base.yaml
194         sed -i s/"tlsVolume"/"..\/crypto-config\/peerOrganizations\/org$iter.example.com\/peers\/peer$jter.org$ite
    r.example.com\/tls:\/etc\/hyperledger\/fabric\/tls"/ docker-compose-base.yaml
195         peerPort=$((peerPort+1000))
196         jter=$((jter+1))
197     done
198     iter=$((iter+1))
199 done
```

Figura 127 - Script makeConfigFiles.sh, configuração de docker-compose-base.yaml

- Criação dos *host<#>.yaml* de cada *peer*: inicialmente faz-se *reset* dos documentos de *host<#>.yaml* previamente criados. Depois utiliza-se a estrutura presente em *host.yaml* para criar cada um do documentos e editam-se os valores consoante o *host*.

```
202 ##############################################################################################
203 ############################## Creation of host<#>.yaml for each peer ########################
204 ##############################################################################################
205
206 cat host1Structure.yaml > host1.yaml # Reset host1.yaml config
207 rm -r ./configDocs/host*.yaml 2> /dev/null # Reset the rest of host<#>.yaml, the error output is being ignored
    as it is irrelevant here
208
209 iter=1
210 number=2
211 numPort=8050
212 while [ "$iter" -le "$OrgNUM" ]; do
213     jter=0
214     while [ "$jter" -lt "$PeerNUM" ]; do
215         if [ "$iter" -eq 1 ] && [ "$jter" -eq 0 ]; then
216             # Create host1.yaml
217             printf "\n$(tput setaf 1)$(tput bold)Configuring host1.yaml $(tput sgr 0)"
218             sed -i s/"lastOrderer"/"orderer$ordererNumber.example.com"/ host1.yaml
219             sed -i s/"lastPortOrderer"/"$lastPortOrderer"/g host1.yaml
220             mv ./host1.yaml ./configDocs
221             jter=$((jter+1))
222
223         else
224             # Create the rest of the host<#>.yaml
225             printf "\n$(tput setaf 1)$(tput bold)Configuring host$number.yaml $(tput sgr 0)"
226             cat host.yaml > host$number.yaml
227             chmod 777 host$number.yaml
228             sed -i s/"ordererName"/"orderer$number.example.com"/ host$number.yaml
229             sed -i s/"peerName"/"peer$jter"/ host$number.yaml
230             sed -i s/"orgName"/"org$iter.example.com"/ host$number.yaml
231             sed -i s/"portOrderer"/"$numPort"/g host$number.yaml
232             mv ./host$number.yaml ./configDocs
233             number=$((number+1))
234             numPort=$((numPort+1000))
235             jter=$((jter+1))
236         fi
237     done
238     iter=$((iter+1))
239 done
```

Figura 128 - Script makeConfigFiles.sh, criação dos ficheiros host<#>.yaml

- Criação de *mychannel.sh*: inicialmente apagam-se as modificações realizadas previamente. Depois adicionam-se os comandos de juntar todos os *peers* ao canal, conforme os *peers* existentes e os comandos de adicionar *anchor peers*, um por organização.

```
242 ##############################################################################################
243 ################################### Creation of mychannel.sh #################################
244 ##############################################################################################
245
246 sed -i '5,$d' mychannel.sh # Reset mychannel.sh to its inicial form
247
248 # Join all peers to channel
249 printf "\n$(tput setaf 4)$(tput bold)Configuring mychannel.sh $(tput sgr 0)"
250 iter=2
251 ordPort=9051
252 while [ "$iter" -le "$OrgNUM" ]; do
253     jter=0
254     while [ "$jter" -lt "$PeerNUM" ]; do
255         if [ "$iter" -eq 1 ]; then
256             echo "docker exec -e CORE_PEER_ADDRESS=peer$jter.org$iter.example.com:$ordPort -e CORE_PEER_TLS_RO
    OTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org$iter.example.com/p
    eers/peer$jter.org$iter.example.com/tls/ca.crt cli peer channel join -b mychannel.block" >> mychannel.sh
257             ordPort=$((ordPort+1000))
258             jter=$((jter+1))
259         else
260             echo "docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/cr
    ypto/peerOrganizations/org$iter.example.com/users/Admin@org$iter.example.com/msp -e CORE_PEER_ADDRESS=peer$jte
    r.org$iter.example.com:$ordPort -e CORE_PEER_LOCALMSPID="\"Org$iter\MSP\"" -e CORE_PEER_TLS_ROOTCERT_FILE=/op
    t/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org$iter.example.com/peers/peer$jter.
    org$iter.example.com/tls/ca.crt cli peer channel join -b mychannel.block" >> mychannel.sh
261             ordPort=$((ordPort+1000))
262             jter=$((jter+1))
263         fi
264     done
265     iter=$((iter+1))
266 done
267
268 echo "docker exec cli peer channel update -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/Org1
    MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/ex
    ample.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" >> mychannel.sh
269 iter=2
270 corePeerOrg=7051
271 while [ "$iter" -le "$OrgNUM" ]; do
272     corePeerOrg=$((corePeerOrg+PeerNUM*1000))
273     echo "docker exec -e CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/pee
    rOrganizations/org$iter.example.com/users/Admin@org$iter.example.com/msp -e CORE_PEER_ADDRESS=peer0.org2.examp
    le.com:$corePeerOrg -e CORE_PEER_LOCALMSPID="\"Org${iter}MSP\"" -e CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/sr
    c/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org$iter.example.com/peers/peer0.org$iter.exampl
    e.com/tls/ca.crt cli peer channel update -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/Org$i
    ter MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganization
    s/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" >> mychannel.sh
274     iter=$((iter+1))
275 done
```

Figura 129 - Script makeConfigFiles.sh, criação de mychannel.sh

- Criação de *host<#>up.sh e host<#>down.sh* de cada *peer*: inicialmente eliminam-se os documentos previamente criados. Seguidamente criam-se e movem-se os scripts para a directoria *./MultipassFolder/config_maker_script/configDocs*.

```
278 ###############################################################################################
279 ############################## Creation of host<#>up.sh & host<#>down.sh  ##############################
280 ###############################################################################################
281
282 rm -r ./configDocs/host*.sh 2> /dev/null # Reset host<#>up.sh & host<#>down.sh files created previously, the e
    rror output is being ignored as it is irrelevant here
283
284 hostNumber=1
285 iter=1
286 while [ "$iter" -le "$OrgNUM" ]; do
287     jter=0
288     while [ "$jter" -lt "$PeerNUM" ]; do
289         # Creation of host<#>up.sh
290         printf "\n$(tput setaf 0)$(tput bold)Creating host${hostNumber}up.sh $(tput sgr 0)"
291         echo "docker-compose -f host$hostNumber.yaml up -d" > host${hostNumber}up.sh
292
293         # Creation of host<#>down.sh
294         printf "\n$(tput setaf 0)$(tput bold)Creating host${hostNumber}down.sh $(tput sgr 0)"
295         sed "1i docker-compose -f host$hostNumber.yaml down -v" hostDownStructure.sh > host${hostNumber}down.s
    h
296
297         mv ./host${hostNumber}up.sh ./configDocs && mv ./host${hostNumber}down.sh ./configDocs
298         hostNumber=$((hostNumber+1))
299         jter=$((jter+1))
300     done
301     iter=$((iter+1))
302 done
```

Figura 130 - Script makeConfigFiles.sh, criação dos ficheiros host<#>up.sh e host<#>down.sh

- Envio de documentos de configuração modificados para *configDocs*: finalmente enviam-se os ficheiros criados que restam para a diretoria *configDocs*.

```
304 echo    # To add a new line at the end
305 cp ./configtx.yaml ./configDocs/configtx.yaml
306 cp ./crypto-config.yaml ./configDocs/crypto-config.yaml
307 cp ./docker-compose-base.yaml ./configDocs/docker-compose-base.yaml
308 cp ./mychannel.sh ./configDocs/mychannel.sh
```

Figura 131 - Script makeConfigFiles.sh, envio de documentos de configuração modificados para configDocs

2) Passa-se então para a execução em si do script makeConfigFiles.sh.

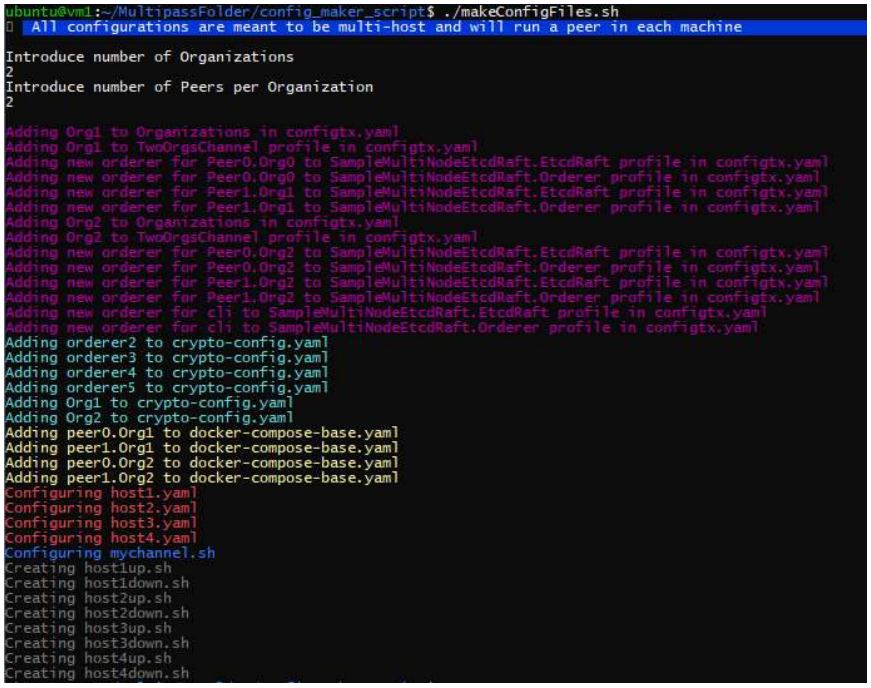


Figura 132 - Execução do script makeConfigFiles.sh

Criando os ficheiros obtém-se a configuração de *hosts* presente na Figura 133.

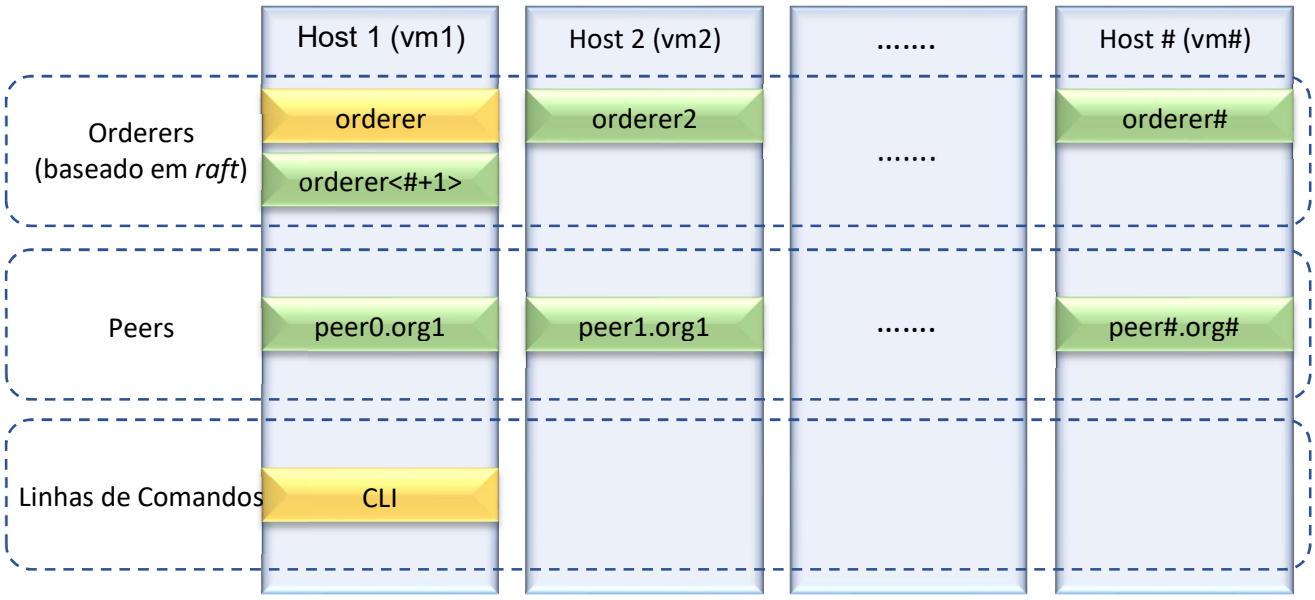


Figura 133 - Configuração de hosts obtida utilizando makeConfigFiles.sh

3) Finalmente, em todas as máquinas, substituem-se os ficheiros em *configDocs* na diretoria *4host-swarm-clis* mencionada na secção 5.2.3.6, exceto o documento *docker-compose-base.yaml* que irá substituir para a directoria *4host-swarm-clis/base*.

4) Para fazer a testagem basta voltar a percorrer os passos da secção 5.2.3.6, obtendo-se os mesmos resultados.

5.3 Ethereum-Geth

A opção do uso de Ethereum foi apenas apresentada e reconhecida numa fase bastante tardia no processo da realização da dissertação, pelo que não foi possível fazer uma implementação aprofundada da ideia descrita na secção 4.2 deste documento.

A implementação realizada baseia-se na realização o tutorial presente em [32], de seguida ir-se-á descrever o sistema operativo utilizado, o pré-requisito e a implementação.

5.3.1 Sistema operativo

5.3.1.1 Multipass - Ubuntu

Nesta implementação, tal como na secção 5.2.1.2, utiliza-se *multipass* que corre Ubuntu 20.04. Neste caso são criadas duas máquinas virtuais, *machine1* e *machine2*.

```
PS C:\Users\Develop> multipass launch -n machine1
Launched: machine1
PS C:\Users\Develop> multipass launch -n machine2
Launched: machine2
```

Figura 134 - Criação de máquinas virtuais

5.3.2 Pré-requisito

Na implementação é adquirido que se está a utilizar um sistema atualizado. Para isso corre-se o comando `sudo apt update && sudo apt upgrade`.

5.3.1.1 Geth

Tal como foi mencionado anteriormente na secção 2.2.6, *Geth* é um *standalone client* empregado para poder usar a implementação em linguagem *go* do protocolo *Ethereum*. Para instalar existem diversas possibilidades presentes em [33]. Neste caso usaram-se os seguintes comandos.

```
sudo add-apt-repository -y ppa:ethereum/Ethereum

sudo apt-get update

sudo apt-get install Ethereum
```

5.3.3 2.1.1 Implementação de uma rede privada com dois peers em máquinas diferentes

Nesta implementação pretendeu-se criar uma rede privada com dois *peers*, cada um numa VM. De seguida são descritos os passos necessários.

1) Inicialmente, na VM *machine1* ir-se-á criar a diretoria */tmp/eth/60* onde irá correr o comando apresentado de seguida.

```
geth --datadir="/tmp/eth/60/01" -verbosity 6 --ipcdisable --port
30301 --http.port 8101 console 2>> /tmp/eth/60/01.log
```

Com este comando inicia-se a consola de JavaScript do Geth, no comando descreve-se onde a instância corre, os portos a serem utilizados, entre outros. No final os erros são enviados para a diretoria */tmp/eth/60/01.log*.

```
ubuntu@machine1:~$ mkdir -p /tmp/eth/60
ubuntu@machine1:~$ geth --datadir="/tmp/eth/60/01" -verbosity 6 --ipcdisable --port 30301 --http.port 8101 console 2>>
 /tmp/eth/60/01.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.10.20-stable-8f2416a8/linux-amd64/go1.18.1
at block: 0 (Thu Jan 01 1970 01:00:00 GMT+0100 (CET))
 datadir: /tmp/eth/60/01
 modules: admin:1.0 debug:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
>
```

Figura 135 - Inicialização da consola de geth na VM machine1

2) Seguidamente utiliza-se o comando apresentado de seguida para obter o *enode*, que identifica o nodo de *Ethereum* onde se está presentemente.

   `admin.nodeInfo.enode`

   No caso desta implementação obtém-se o seguinte valor.

   `"enode://a0f8152465752aa0b5d3b2b8de5b45c94ae057291684b8a25452de82cc4033f35ad505f3466cdedba51401c04dac815759dadf3c73335db104073505f13e5d7d@127.0.0.1:30301"`

```
> admin.nodeInfo.enode
"enode://a0f8152465752aa0b5d3b2b8de5b45c94ae057291684b8a25452de82cc4033f35ad505f3466cdedba51401c04dac815759dadf3c73335
db104073505f13e5d7d@127.0.0.1:30301"
```

Figura 136 - Execução do comando admin.nodeInfo.enode

3) Posteriormente repete-se o passo 1 na VM machine2.

```
ubuntu@machine2:~$ mkdir -p /tmp/eth/60
ubuntu@machine2:~$ geth --datadir="/tmp/eth/60/02" --verbosity 6 --ipcdisable --port 30302 --http.port 8102 console 2>>
/tmp/eth/60/02.log

Welcome to the Geth JavaScript console!

instance: Geth/v1.10.20-stable-8f2416a8/linux-amd64/go1.18.1
at block: 0 (Thu Jan 01 1970 01:00:00 GMT+0100 (CET))
 datadir: /tmp/eth/60/02
 modules: admin:1.0 debug:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
>
```

Figura 137 - Inicialização da consola geth na VM machine2

4) Ainda na VM machine2, corre-se o comando apresentado de seguida para criar a rede privada constituída por *machine1* e *machine2*. Neste comando vai-se utilizar o enode obtido no passo 2, substituindo-se o valor de *localhost* (127.0.0.1) com o IP da VM *machine1*. Se ambas as máquinas estiverem na mesma rede de internet utiliza-se o IP, caso contrário utiliza-se IP externo (pode ser obtido através de pesquisa na internet). Estando a ser utilizadas VM de *multipass*, o IP é obtido com o comando `multipass ls`, executado fora das VM.

```
PS C:\Users\Develop> multipass ls
Name                    State             IPv4             Image
machine1                Running           172.20.250.152   Ubuntu 20.04 LTS
machine2                Running           172.20.242.31    Ubuntu 20.04 LTS
```

Figura 138 - Execução do comando multipass ls

`admin.addPeer(<enode obtido no passo 2 modificado >)`

No caso específico da implementação foi usado o seguinte comando.

```
admin.addPeer("enode://a0f8152465752aa0b5d3b2b8de5b45c94ae057291684b
8a25452de82cc4033f35ad505f3466cdedba51401c04dac815759dadf3c73335db10
4073505f13e5d7d@172.20.250.152:30301")
```

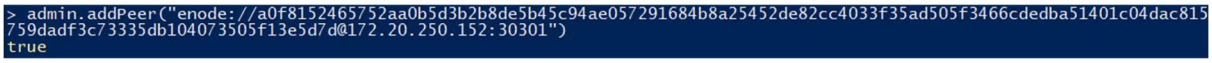


Figura 139 - Execução do comando admin.addPeer

5) Finalmente, observa-se que se criou uma rede com ambas as VM utilizando o comando `net.peerCount` na VM *machine2*, onde se observa a presença de dois *peers*.

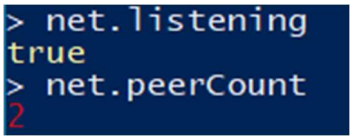


Figura 140 - Execução do comando net.peerCount

5.4 Avaliações

Comparando as implementações realizadas usando HF e a pequena implementação realizada em *Ethereum*, verifica-se que ambas as plataformas têm características interessantes e características desfavoráveis.

Enquanto HF é uma opção bastante interessante devido à segurança inerente de uma rede de natureza *permissioned* e por se programar os *chaincodes* com linguagens de programação bastantes conhecidas (*java*, *javascript*, *go*). Esta plataforma também tem alguns problemas relativos à sua falta de flexibilidade (para mudar a configuração da rede é necessários bastantes passos) e falta de recursos existentes derivado da sua relativa obscuridade.

No caso do *Ethereum* tem-se mais flexibilidade (uma vez que foi essa a intenção na sua criação, onde se podem criar diferentes *chaincodes* e *dApps*) e, mais importantemente, a grande facilidade de acesso a recursos, de modo a ultrapassar problemas encontrados, devido à enorme comunidade de *developers* e utilizadores. Porém, para programar em Ethereum tem de se aprender uma nova linguagem de programação, denominada *Solidity* e é necessário criar nodos dedicados a estabelecer a segurança de uma rede.

Concluindo apesar de ambas as plataformas serem alternativas interessantes, *Ethereum* seria a melhor opção, muito por causa da abundância de recursos disponíveis.

.

6 Trabalho

6.1 Trabalho realizado

Durante a Dissertação este trabalho focou-se maioritariamente na criação das diferentes implementações usando *Hyperledger Fabric*, seguido por uma breve familiarização e implementação com *Ethereum*, durante todo este período o documento da dissertação foi sendo escrito.

	Março			Abril				Maio				Junho					Julho	
Semanas:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Tarefas de Implementação em Hyperledger Fabric																		
Familiarização e implementação com Ethereum																		
Escrita da Dissertação																		

Figura 141 - Tabela do trabalho realizado

6.2 Trabalho futuro

No futuro seria interessante realizar a implementação de *Ethereum* mencionada na secção 4.2 da dissertação. Adicionalmente penso que seria uma boa ideia acompanhar a evolução de *Hyperledger Fabric* com novas implementações e configurações ainda mais detalhadas.

Referências

[1] Rivadeneira-Muñoz, Jorge Eduardo [et al.]; "A Privacy-Aware Framework integration into a Human-in-the-Loop IoT System", ACM Surveys (submitted), June 2021.

[2] Plotly," Dash". Disponível em: https://plotly.com/dash/. Acedido em 30-junho-2021.

[3] Xorum.io, "Cross-platform mobile apps development in 2021: Xamarin vs React Native vs Flutter vs Kotlin Multiplatform". Disponível em: https://medium.com/xorum-io/cross-platform-mobile-apps-development-in-2021-xamarin-vs-react-native-vs-flutter-vs-kotlin-ca8ea1f5a3e0. Acedido em 30-junho-2021.

[4] Microsoft." What is the Xamarin.Forms technology stack?". Disponível em: https://www.microsoft.com/pt-pt/videoplayer/embed/RE4LC7e?postJsllMsg=true&autoCaptions=pt-pt. Acedido em 30-junho-2021.

[5] Wikipedia, "Bitcoin". Disponível em: https://en.wikipedia.org/wiki/Bitcoin. Acedido em 30-junho-2021.

[6] Hyperledger Fabric, "How Fabric networks are structured" Disponível em: https://hyperledger-fabric.readthedocs.io/en/latest/network/network.html. Acedido em 30-junho-2021.

[7] Canonical." Multipass". Disponível em: https://multipass.run/. Acedido em 23-abril-2022

[8] J. Fernandes, D. Raposo, N. Armando, S. Sinche, J. Sá Silva, A. Rodrigues, V. Pereira, H. Oliveira, L. Macedo and F. Boavida, "ISABELA – A Socially-Aware Human-in-the-Loop Advisor System", Journal Online Social Networks and Media, Elsevier, ISSN: 2468-6964, Jan. 2020.

[9] Opencast, "Features". Disponível em: https://opencast.org/features/. Acedido em 30-junho-2021.

[10] Opencast, "Integrations". Disponível em: https://opencast.org/integrations/. Acedido em 30-junho-2021.

[11] Matomo, "What Is Matomo?". Disponível em: https://matomo.org/faq/new-to-piwik/faq_13/. Acedido em 30-junho-2021.

[12] Matomo, "BigBlueButton". Disponível em: https://docs.bigbluebutton.org/. Acedido em 30-junho-2021.

[13] ALEKS, "About ALEKS". Disponível em: https://www.aleks.com/about_aleks. Acedido em 30-junho-2021.

[14] Kelly Parke, "Designing AI Solutions". Disponível em: https://www.kellyeparke.com/can-you-really-find-friends-on-the-web/. Acedido em 30-junho-2021.

[15] STHEMBrasil, "Quem somos?". Disponível em: https://www.sthembrasil.com/quem-somos/. Acedido em 30-junho-2021.

[16] STHEMBrasil, "Projeto-piloto do STHEM Brasil para o uso da ALEKS é aprovado e elogiado por instituições consorciadas". Disponível em: https://www.sthembrasil.com/projeto-piloto-do-sthem-brasil-para-o-uso-da-aleks-e-aprovado-e-elogiado-por-instituicoes-consorciadas/. Acedido em 30-junho-2021.

[17] Microsoft," Criar aplicações móveis com o Xamarin.Forms". Disponível em: https://docs.microsoft.com/pt-pt/learn/paths/build-mobile-apps-with-xamarin-forms/. Acedido em 30-junho-2021.

[18] OsBoxes, "Linux Mint". Disponível em: https://www.osboxes.org/linux-mint/#linux-mint-20-3-vmware. Acedido em 10-março-2022.

[19] Canonical, "Multipass". Disponível em: https://multipass.run/. Acedido em 23-março-2022.

[20] Hyperledger Fabric, "Install Samples, Binaries, and Docker Images". Disponível em: https://hyperledger-fabric.readthedocs.io/en/release-2.2/install.html. Acedido em 23-março-2022.

[21] Go, "Download and install". Disponível em: https://go.dev/doc/install. Acedido em 23-março-2022.

[22] Karl Broman, "Start a new git repository". Disponível em: https://kbroman.org/github_tutorial/pages/init.html. Acedido em 23-abril-2022.

[23] GitHub, "GitHub Desktop". Disponível em: https://desktop.github.com/. Acedido em 23-abril-2022.

[24] Github Docs, “Gerar uma nova chave SSH e adicioná-la ao ssh-agent”. Disponível em: https://docs.github.com/pt/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent. Acedido em 23-abril-2022.

[25] Github Docs, “Adicionar uma nova chave SSH à sua conta do GitHub”. Disponível em: https://docs.github.com/pt/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account. Acedido em 23-abril-2022.

[26] Hyperledger-Fabric, “cryptogen”. Disponível em: https://hyperledger-fabric.readthedocs.io/en/latest/commands/cryptogen.html. Acedido em 30-junho-2021.

[27] Hyperledger-Fabric, “configtxgen”. Disponível em: https://hyperledger-fabric.readthedocs.io/en/latest/commands/configtxgen.html. Acedido em 30-junho-2021.

[28] Hyperledger-Fabric, “peer”. Disponível em: https://hyperledger-fabric.readthedocs.io/en/latest/commands/peercommand.html. Acedido em 30-junho-2021.

[29] Hyperledger-Fabric, “Membership Service Provider (MSP)”. Disponível em: https://hyperledger-fabric.readthedocs.io/en/release-2.2/membership/membership.html. Acedido em 10-junho-2022.

[30] KC Tam, “Add a New Organization on Existing Hyperledger Fabric Network”. Disponível em: https://kctheservant.medium.com/add-a-new-organization-on-existing-hyperledger-fabric-network-2c9e303955b2. Acedido em 15-abril-2022.

[31] KC Tam, “Multi-Host Deployment for First Network (Hyperledger Fabric v2)”. Disponível em: https://kctheservant.medium.com/multi-host-deployment-for-first-network-hyperledger-fabric-v2-273b794ff3d. Acedido em 28-abril-2022.

[32] Go Ethereum, “Private Network Tutorial”. Disponível em: https://geth.ethereum.org/docs/getting-started/private-net. Acedido em 2-julho-2022.

[33] Go Ethereum, “Installing Geth”. Disponível em: https://geth.ethereum.org/docs/install-and-build/installing-geth. Acedido em 2-julho-2022.