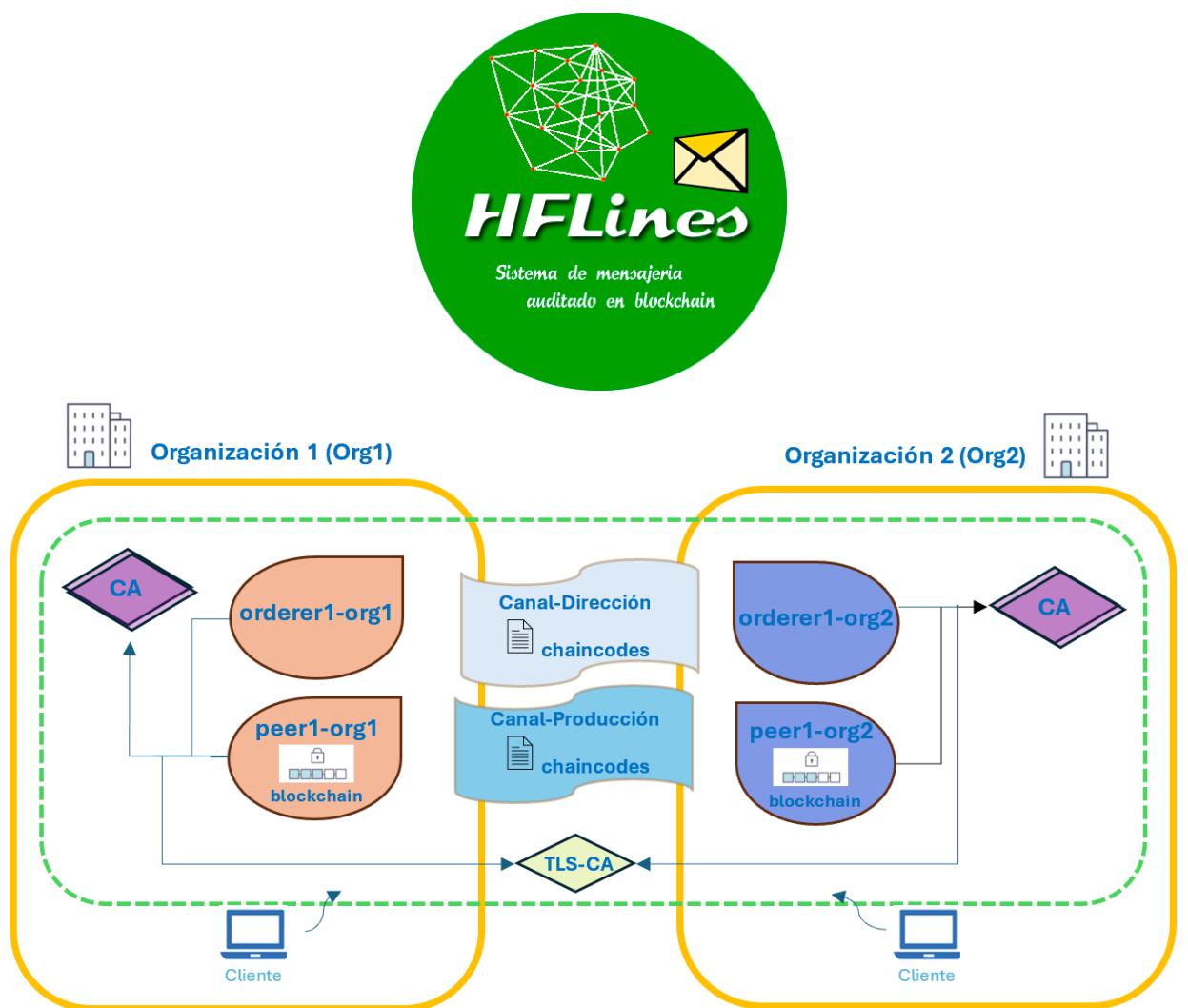


Proyecto de Red

Hyperledger Fabric

Sistema de auditoría de comunicaciones
para una UTE de empresas



Autor: **Quim de Dalmases Juanet**



Esta obra está sujeta a una licencia de
[Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

Índice de Contenido

01.	INTRODUCCIÓN	5
02.	CONTENIDO DEL DOCUMENTO	5
03.	PROPUESTA DE PROYECTO	5
04.	ESTRUCTURA DE DIRECTORIOS DEL PROYECTO	7
05.	FICHEROS docker-compose.yaml Y configtx.yaml	8
06.	METODOLOGÍA PARA LA CREACIÓN Y DESPLIEGUE DE LA RED HFLines .	11
07.	CREACIÓN DE LA RED HFLINES PASO A PASO	14
08.	DEFINICIÓN, INSTALACIÓN Y EJECUCIÓN DE UN CHAINCODE.....	26
09.	IMPLEMENTACIÓN Y EJECUCIÓN DE UN CHAINCODE EN HFLines	31
010.	API REST de conexión a la red para clientes de HFLines	40
011.	MONITORIZACIÓN DE LA RED HFLines: PROMETHEUS Y GRAFANA.....	42
012.	CONCLUSIONES FINALES	45
013.	AGRADECIMIENTOS.....	46

ÍNDICE DE FIGURAS

<i>Figura 1:</i> Estructura de directorios completa del proyecto HFLines	7
<i>Figura 2:</i> Archivo de configuración configtx.yaml en Hyperledger Fabric HFLines	9
<i>Figura 3:</i> Archivo Docker-compose.yaml de la red Hyperledger Fabric HFLines	10
<i>Figura 4:</i> Descripción de la instancia VM en Google Cloud	11
<i>Figura 5:</i> Descripción de las dos organizaciones existentes en HFLines	12
<i>Figura 6:</i> Listado de ejemplo de los contenedores Docker levantados en HFLines	12
<i>Figura 7:</i> Contenido del script 08_eliminar_hflines	14
<i>Figura 8:</i> Contenido del script 00_levantar_CAs.sh	15
<i>Figura 9:</i> Salida parcial del script 00_levantar_Cas.sh	16
<i>Figura 10:</i> Contenido parcial del script 01_certificar_Orgs.sh	18
<i>Figura 11:</i> Salida parcial del script 01_certificar_Orgs.sh	19
<i>Figura 12:</i> Contenido parcial del script 02_levantar_nodos_db.sh	20
<i>Figura 13:</i> Salida en terminal del script 02_levantar_nodos_db.sh	20
<i>Figura 14:</i> Código fuente del script 03_artifacts.sh	21
<i>Figura 15:</i> Salida en terminal del script 03_artifacts.sh	21
<i>Figura 16:</i> Código fuente del script 04_union_canal_orderer1-org1.sh	22
<i>Figura 17:</i> Salida en terminal del script 04_union_canal_orderer1-org1.sh	23
<i>Figura 18:</i> Código fuente del script 05_union_canal_orderer1-org2.sh	23
<i>Figura 19:</i> Salida en terminal del script 05_union_canal_orderer1-org2.sh	24
<i>Figura 20:</i> Ayuda del comando ‘ peer channel ’ para la ejecución de ‘ peer channel join ’ y agregar un nodo peer al canal	24
<i>Figura 21:</i> Código fuente del script 06_union_canal_peer1-org1.sh	25
<i>Figura 22:</i> Salida en terminal del script 06_union_canal_peer1-org1.sh	25
<i>Figura 23:</i> Código fuente del script 07_union_canal_peer1-org2.sh	25
<i>Figura 24:</i> Salida en terminal del script 07_union_canal_peer1-org2.sh	25
<i>Figura 25:</i> Código fuente del script 10a_chaincode_alta_org1.sh	26
<i>Figura 26:</i> Salida parcial del script 10a_chaincode_alta_org1.sh	27
<i>Figura 27:</i> Código fuente del script 10b_chaincode_alta_org1.sh	28
<i>Figura 28:</i> Salida en terminal del script 10b_chaincode_alta_org1.sh	28
<i>Figura 29:</i> Código fuente del script 11_chaincode_alta_org2.sh	29
<i>Figura 30:</i> Salida en terminal del script 11_chaincode_alta_org2.sh	29
<i>Figura 31:</i> Código fuente del script 12_chaincode_aprove_commit.sh	30
<i>Figura 32:</i> Salida en terminal del script 12_chaincode_aprove_commit.sh	30
<i>Figura 33:</i> Website de Spring Boot para la generación rápida de una Java application	32
<i>Figura 34:</i> Contenido del fichero build.gradle	33
<i>Figura 35:</i> Contenido del fichero Mensaje.java	34
<i>Figura 36:</i> Contenido del fichero MensajeTransfer.java	35
<i>Figura 37:</i> Contenido del script r.sh	36
<i>Figura 38:</i> Conjunto de clases del API Fabric-Chaincode-shim para Java	36
<i>Figura 39:</i> Ejemplo de ejecución para enviar un mensaje en HFLines	38
<i>Figura 40:</i> Ejemplo de ejecución del script 30_ejemplos_uso_20_app0x_org1.sh	39
<i>Figura 41:</i> Ejemplo de ejecución del script 30_ejemplos_uso_20_app0x_org2.sh	40
<i>Figura 42:</i> Código del proyecto Gateway a modo de backend API Rest	41
<i>Figura 43:</i> API Rest Gateway de acceso a la funcionalidad del chaincode de HFLines	41
<i>Figura 44:</i> Ejecución de la función ListarMensajes usando el API Rest del proyecto Gateway usando la librería Swagger	42
<i>Figura 45:</i> Ejemplos de consultas en Prometheus: (a) de métrica ‘ up ’ (b) versión de cadvisor activa	43
<i>Figura 46:</i> Ejemplos de ‘dashboards’ en tiempo real con la herramienta Grafana	44

01. INTRODUCCIÓN

Una vez realizado el **Máster en Blockchain Engineering en EBIS TechSchool** es necesario demostrar la utilidad del curso, y de la mejor manera posible, recordar y utilizar todo el conocimiento adquirido. Para poner en práctica todo lo aprendido se ha realizado el proyecto **HFLines**.

HFLines surge de la necesidad por parte de las empresas de disponer de una herramienta que permita el **registro auditado de las comunicaciones** entre todos los miembros de inicialmente dos empresas (que se podrían incrementar en caso de ser necesario). **HFLines** puede realizar el seguimiento de todas las decisiones propuestas, dialogadas y finalmente acordadas en el tiempo almacenándolas en una cadena de bloques, de manera que una vez registradas no puedan ser, ni modificadas ni manipuladas a gusto de alguna de las partes. Establece una **forma de confianza** en donde la información está distribuida en las dos empresas y se puede consultar fácilmente sin la necesidad de recibir la autorización de un tercera parte, tan solo utilizando la tecnología **Hyperledger Fabric**. A medida que las comunicaciones se van realizando quedan registradas en el tiempo almacenándolas como una secuencia de transacciones.

Se propone una aplicación en donde las empresas disponen para sus miembros de una red propia, de un espacio en el que pueden interaccionar de manera colaborativa siempre que tengan permiso para hacerlo, pero en donde al mismo tiempo, la información está protegida criptográficamente y la privacidad de las comunicaciones es una característica configurable en la red.

02. CONTENIDO DEL DOCUMENTO

Este documento contiene la propuesta inicial del proyecto, más la documentación de todo el trabajo realizado para:

- Definir e implementar la **arquitectura física** de la red Hyperledger Fabric **HFLines**.
- Definir todo el material criptográfico, en donde se han creado entidades certificadoras, se han registrado en ellas componentes de red y se han generado los **certificados digitales** necesarios para dotar de identidad e integridad a las comunicaciones.
- Definición de **los canales de comunicación** dentro de la red **HFLines**.
- Definición del empaquetado de ‘Smart contracts’ y sus políticas en forma de **chaincodes**.
- Definición de un **Backend** de tipo **API Rest**, para la disponibilidad de un servicio de la funcionalidad de la mensajería accesible mediante aplicaciones por parte de los clientes de la red.
- Definición de la arquitectura hardware y software **para la monitorización del funcionamiento de la red**. Herramienta **Prometheus** para la obtención de métricas y la herramienta **Grafana** para la elaboración de **visualizaciones gráficas** de estas métricas.
- Metodología para la puesta en marcha y uso de este proyecto.

03. PROPUESTA DE PROYECTO

HFLines se pensó como un proyecto que utiliza la tecnología blockchain de Hyperledger Fabric para implementar **UN SISTEMA DE AUDITORIA DE COMUNICACIONES** para empresas. Y a medida que se iba implementando se han ido viendo nuevas utilidades. En este apartado describimos la propuesta inicial que se presentó a **EBIS Techschool**.

Aplicación HFLines

HFLines se concibe como una aplicación de utilidad múltiple y diversa en todos los proyectos que decidan implementar conjuntamente varias empresas, o como sería el caso particular en que se establece una UTE (Unión temporal de Empresas) entre varias empresas.

Este sistema registrará todas las comunicaciones y las almacenará de manera transparente e inmutable en una cadena de bloques, permitiendo aplicar en cada comunicación la privacidad necesaria.

- Inicialmente se implementará un canal global para la comunicación entre las organizaciones pero una posible mejora futura podría incorporar un nuevo canal, con privacidad reservado sólo para el nivel directivo.
- Al usar la aplicación, se dispone al finalizar cada proyecto, de la secuencia de decisiones tomadas por el equipo directivo y de todas las operaciones realizadas conjuntamente así como su cronología, a modo de auditoría global almacenadas en el ledger distribuido de la cadena de bloques en forma de mensajes.

El uso de la tecnología blockchain permite garantizar a todas las organizaciones el acceso a los datos según su nivel de acceso y confiar en el mismo, por su capacidad de inmutabilidad del registro, siendo una herramienta de confianza para la resolución de conflictos, en la responsabilidad de cada toma de decisiones.

En la propuesta de proyecto se implementará cada uno de los aspectos inicialmente demandados para el proyecto del máster:

- Arquitectura de red : contenedores Docker, descripción de la red propuesta para el problema a solucionar.
- Implementación de chaincodes para el registro de las comunicaciones realizadas entre los trabajadores de las empresas.
- Desarrollo chaincode y publicación de las apis.
- Desarrollo frontal que se integre con las apis (opcional).
- Monitorización con Prometheus + Grafana.

Se intentará aplicar aspectos de originalidad/innovación, uso de frameworks/librerías, presentar una implementación de calidad del código y arquitectura de software, junto con la elaboración de repositorio Github bien documentado con instrucciones para levantar la red.

Una parte de esta propuesta se ha desarrollado en el estudio de viabilidad y se puede consultar hasta los últimos cambios, en la siguiente repositorio:

<https://github.com/EsbrinaChain/hflines>

Este último párrafo, hacía referencia a las pruebas que en ese momento se habían realizado a modo de prototipo, que actualmente se pueden consultar como versión v1.1 y que se basaban en la viabilidad del funcionamiento de la red y en la ejecución de chaincodes.

04. ESTRUCTURA DE DIRECTORIOS DEL PROYECTO

El proyecto **HFLines** está publicado en Github en la dirección <https://github.com/EsbrinaChain/hflines>

En las siguientes secciones se hará referencia a partes de la estructura de directorios descrita en este apartado y que se puede descargar cuando se necesite poner el marcha el proyecto clonando el proyecto.

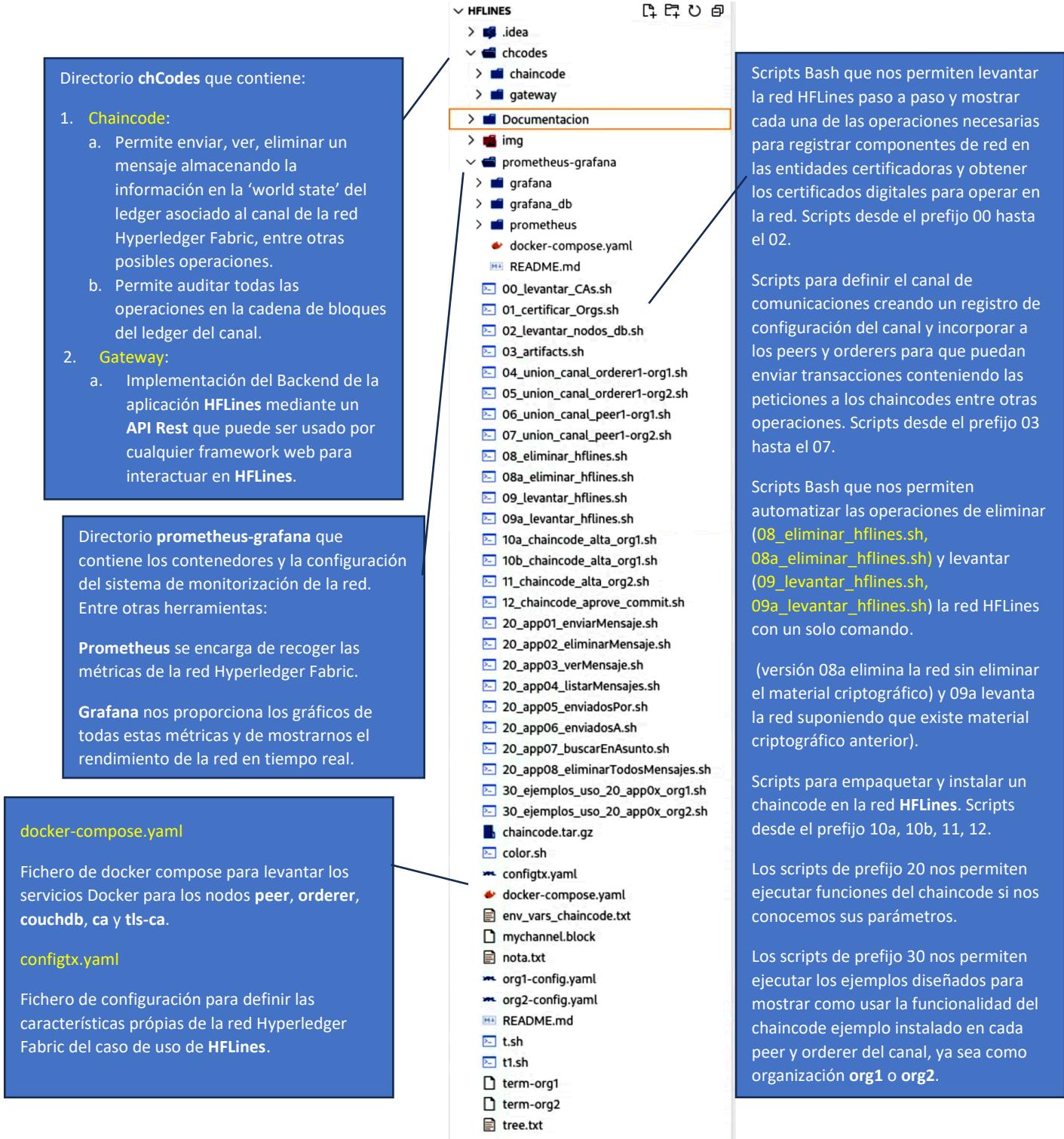


Figura 1: Estructura de directorios completa del proyecto HFLines.

05. FICHEROS docker-compose.yaml Y configtx.yaml

En este documento se describe la red **Hyperledger Fabric** creada para el proyecto **HFLines** (en adelante red **HFLines**). Debido a que se ha implementado utilizando contenedores Docker, el fichero por defecto que describe la arquitectura de la red y los servicios a levantar es **docker-compose.yaml** y para configurar la red, **Hyperledger Fabric** utiliza el fichero **configtx.yaml**.

Vemos los dos ficheros utilizados para crear la red **HFLines**:

```
# SPDX-License-Identifier: Apache-2.0
# - This section defines the different organizational identities which will be referenced later in the configuration.
Organizations:
# SampleOrg defines an MSP using the sampleconfig. It should never be used in production but may be used as a template for other definitions
- &org1
  # DefaultOrg defines the organization which is used in the sampleconfig of the fabric.git development environment
  Name: org1MSP
  ID: org1MSP                               # ID to load the MSP definition as
  MSPDir: /tmp/hyperledger/org1/msp

  # Policies defines the set of policies at this level of the config tree. For organization policies, their canonical path is usually
  # /Channel/<Application>/Orderer/>/<OrgName>/<PolicyName>
  Policies:
    Readers:
      Type: Signature
      Rule: "OR('org1MSP.member')"           # Rule: "OR('org1MSP.admin', 'org1MSP.peer', 'org1MSP.client')"
    Writers:
      Type: Signature
      Rule: "OR('org1MSP.member')"
    Admins:
      Type: Signature
      Rule: "OR('org1MSP.admin')"
    Endorsement:
      Type: Signature
      Rule: "OR('org1MSP.peer')"
  OrdererEndpoints:
    - orderer1-org1:7050
  AnchorPeers:
    - Host: peer1-org1
      Port: 7051

- &org2
  Name: org2MSP
  ID: org2MSP
  MSPDir: /tmp/hyperledger/org2/msp
  Policies:
    Readers:
      Type: Signature
      Rule: "OR('org2MSP.admin', 'org2MSP.peer', 'org2MSP.client')"
      Rule: "OR('org2MSP.member')"
    Writers:
      Type: Signature
      Rule: "OR('org2MSP.member')"
    Admins:
      Type: Signature
      Rule: "OR('org2MSP.admin')"
    Endorsement:
      Type: Signature
      Rule: "OR('org2MSP.peer')"
  OrdererEndpoints:
    - orderer1-org2:7050
  AnchorPeers:
    - Host: peer1-org2
      Port: 9051

Capabilities:
  Channel: &ChannelCapabilities
  V2_0: true
  Orderer: &OrdererCapabilities
  V2_0: true
  Application: &ApplicationCapabilities
  V2_0: true
Application: &ApplicationDefaults
  Organizations:
    Policies:
      Readers:
        Type: ImplicitMeta
        Rule: "ANY Readers"
      Writers:
        Type: ImplicitMeta
        Rule: "ANY Writers"
      Admins:
        Type: ImplicitMeta
        Rule: "MAJORITY Admins"
    LifecycleEndorsement:
      Type: ImplicitMeta
      Rule: "ANY Endorsement"
    Endorsement:
      Type: ImplicitMeta
      Rule: "MAJORITY Endorsement"

  Capabilities:
    <<: *ApplicationCapabilities
```

```

Orderer: &OrdererDefaults
  OrdererType: etcraft
  EtcdRaft:
    Consenters:
      - Host: orderer1-org1
        Port: 7050
        ClientTLSCert: /tmp/hyperledger/org1/orderer1/tls-msp/signcerts/cert.pem
        ServerTLSCert: /tmp/hyperledger/org1/orderer1/tls-msp/signcerts/cert.pem
      - Host: orderer1-org2
        Port: 7050
        ClientTLSCert: /tmp/hyperledger/org2/orderer1/tls-msp/signcerts/cert.pem
        ServerTLSCert: /tmp/hyperledger/org2/orderer1/tls-msp/signcerts/cert.pem
  BatchTimeout: 2s
  BatchSize:
    MaxMessageCount: 10
    AbsoluteMaxBytes: 99 MB
    PreferredMaxBytes: 512 KB
  Organizations:
  Policies:
    Readers:
      Type: ImplicitMeta
      Rule: "ANY Readers"
    Writers:
      Type: ImplicitMeta
      Rule: "ANY Writers"
    Admins:
      Type: ImplicitMeta
      Rule: "MAJORITY Admins"
  BlockValidation:
    Type: ImplicitMeta
    Rule: "ANY Writers"
Channel: &ChannelDefaults
  Policies:
    Readers:
      Type: ImplicitMeta
      Rule: "ANY Readers"
    Writers:
      Type: ImplicitMeta
      Rule: "ANY Writers"
    Admins:
      Type: ImplicitMeta
      Rule: "MAJORITY Admins"
  Capabilities:
    <<; *ChannelCapabilities
  Profiles:
    OrgsOrdererGenesis:
      <<; *ChannelDefaults
      Orderer:
        <<; *OrdererDefaults
        Organizations:
          - *org1
          - *org2
      Capabilities:
        <<; *OrdererCapabilities
    Consortiums:
      SampleConsortium:
        Organizations:
          - *org1
          - *org2
  OrgsChannel:
    Consortium: SampleConsortium
    <<; *ChannelDefaults
    Application:
      <<; *ApplicationDefaults
      Organizations:
        - *org1
        - *org2
    Capabilities:
      <<; *ApplicationCapabilities
  SampleAppChannelEtcdRaft:
    <<; *ChannelDefaults
    Orderer:
      <<; *OrdererDefaults
      Organizations:
        - *org1
        - *org2
    Capabilities:
      <<; *OrdererCapabilities
    Application:
      <<; *ApplicationDefaults
      Organizations:
        - *org1
        - *org2
    Capabilities:
      <<; *ApplicationCapabilities

```

Figura 2: Archivo de configuración **configtx.yaml** en Hyperledger Fabric **HFLines**

```

# docker-compose.yaml
1 networks:
2   | hflines:
3
4 services:
5   ca-tls:
6     container_name: ca-tls
7     image: hyperledger/fabric-ca:latest
8     command: sh -c 'Fabric-ca-server start -d -b tls-ca-admin:tls-ca-adminpw --port 7052'
9     environment:
10    - FABRIC_CA_SERVER_HOME=/tmp/hyperledger/fabric-ca/crypto
11    - FABRIC_CA_SERVER_TLS_ENABLED=true
12    - FABRIC_CA_SERVER_CSR_CN=ls-ca
13    - FABRIC_CA_SERVER_CSR_HOSTS=0.0.0.0
14    - FABRIC_CA_SERVER_DEBUG=true
15     volumes:
16       - /tmp/hyperledger/tls-ca:/tmp/hyperledger/fabric-ca
17     networks:
18       - hflines
19     ports:
20       - 7052:7052
21
22 rca-org1:
23   container_name: rca-org1
24   image: hyperledger/fabric-ca:latest
25   command: sh -c 'Fabric-ca-server start -d -b rca-org1-admin:rca-org1-adminpw --port 7054'
26   environment:
27    - FABRIC_CA_SERVER_HOME=/tmp/hyperledger/fabric-ca/crypto
28    - FABRIC_CA_SERVER_TLS_ENABLED=true
29    - FABRIC_CA_SERVER_CSR_CN=rca-org1
30    - FABRIC_CA_SERVER_CSR_HOSTS=0.0.0.0
31    - FABRIC_CA_SERVER_DEBUG=true
32     volumes:
33       - /tmp/hyperledger/org1/ca:/tmp/hyperledger/fabric-ca
34     networks:
35       - hflines
36     ports:
37       - 7054:7054
38
39 rca-org2:
40   container_name: rca-org2
41   image: hyperledger/fabric-ca:latest
42   command: /bin/bash -c 'Fabric-ca-server start -d -b rca-org2-admin:rca-org2-adminpw --port 7055'
43   environment:
44    - FABRIC_CA_SERVER_HOME=/tmp/hyperledger/fabric-ca/crypto
45    - FABRIC_CA_SERVER_TLS_ENABLED=true
46    - FABRIC_CA_SERVER_CSR_CN=rca-org2
47    - FABRIC_CA_SERVER_CSR_HOSTS=0.0.0.0
48    - FABRIC_CA_SERVER_DEBUG=true
49     volumes:
50       - /tmp/hyperledger/org2/ca:/tmp/hyperledger/fabric-ca
51     networks:
52       - hflines
53     ports:
54       - 7055:7055
55
56 couchdb1:
57   container_name: couchdb1
58   image: couchdb:3.3.2
59   labels:
60     service: hyperledger-fabric
61   environment:
62    - COUCHDB_USER=admin
63    - COUCHDB_PASSWORD=adminpw
64   ports:
65    - 5984:5984
66   networks:
67     - hflines
68
69 peer1-org1:
70   container_name: peer1-org1
71   image: hyperledger/fabric-peer:latest
72   environment:
73    - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
74    - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb1:5984
75    - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
76    - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
77    - CORE_PEER_ID=peer1-org1
78    - CORE_PEER_ADDRESS=peer1-org1:7051
79    - CORE_PEER_LOCALMSPID=org1MSP
80    - CORE_PEER_LISTENADDRESS=0.0.0.0:7051
81    - CORE_PEER_MSPPORTIGPATH=/tmp/hyperledger/org1/peer1/msp
82    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
83    - CORE_PEER_DOCKER_HOSTCONFING_NETWORKMODE=hflines_hflines
84    - FABRIC_LOGGING_SPEC=info
85    - CORE_PEER_TLS_ENABLED=true
86    - CORE_PEER_TLS_CERT_FILE=/tmp/hyperledger/org1/peer1/tls-msp/signcerts/cert.pem
87    - CORE_PEER_TLS_KEY_FILE=/tmp/hyperledger/org1/peer1/tls-msp/keystore/key.pem
88    - CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org1/peer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
89    - CORE_PEER_GOSSIP_USELEADERELECTION=true
90    - CORE_PEER_GOSSIP_ORGLEADER=false
91    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1-org1:7051
92    - CORE_PEER_GOSSIP_SKIPHANDSHAKE=true
93 depends_on:
94   - couchdb1
95 working_dir: /opt/gopath/src/github.com/hyperledger/fabric/org1/peer1
96 volumes:
97   - /var/run:/host/var/run
98   - /tmp/hyperledger/org1/peer1:/tmp/hyperledger/org1/peer1
99 ports:
100  - 7051:7051
101  - 9444:9444
102 networks:
103   - hflines
104
105 couchdb2:
106   container_name: couchdb2
107   image: couchdb:3.3.2
108   labels:
109     service: hyperledger-fabric
110   environment:
111    - COUCHDB_USER=admin
112    - COUCHDB_PASSWORD=adminpw
113   ports:
114    - 7984:5984
115   networks:
116     - hflines
117
118 peer1-org2:
119   container_name: peer1-org2
120   image: hyperledger/fabric-peer:latest
121   environment:
122    - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
123    - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb2:5984
124    - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
125    - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
126    - CORE_PEER_ID=peer1-org2
127    - CORE_PEER_ADDRESS=peer1-org2:9051
128    - CORE_PEER_LOCALMSPID=org2MSP
129    - CORE_PEER_LISTENADDRESS=0.0.0.0:9051
130    - CORE_PEER_MSPPORTIGPATH=/tmp/hyperledger/org2/peer1/msp
131    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
132    - FABRIC_LOGGING_SPEC=info
133    - CORE_PEER_TLS_ENABLED=true
134    - CORE_PEER_TLS_CERT_FILE=/tmp/hyperledger/org2/peer1/tls-msp/signcerts/cert.pem
135    - CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org2/peer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
136    - CORE_PEER_GOSSIP_ORACLE=true
137    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1-org1:7051
138    - CORE_PEER_GOSSIP_ORGLEADER=false
139    - CORE_PEER_GOSSIP_SKIPHANDSHAKE=true
140
141 working_dir: /opt/gopath/src/github.com/hyperledger/fabric/org2/peer1
142 volumes:
143   - /var/run:/host/var/run
144   - /tmp/hyperledger/org2/peer1:/tmp/hyperledger/org2/peer1
145 ports:
146   - 9051:9051
147   - 9445:9445
148 networks:
149   - hflines
150
151 orderer1-org1:
152   container_name: orderer1-org1
153   image: hyperledger/fabric-orderer:latest
154   environment:
155    - ORDERER_HOME=/tmp/hyperledger/orderer
156    - ORDERER_HOST=orderer1-org1
157    - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
158    # - ORDERER_GENERAL_GENESISMETHOD=org1MSP
159    - ORDERER_GENERAL_GENESISFILE=/tmp/hyperledger/org1/orderer1/genesis.block
160    - ORDERER_GENERAL_LOCALMSPID=org1MSP
161    - ORDERER_GENERAL_LOCALMSPIR=/tmp/hyperledger/org1/orderer1/msp
162    - ORDERER_GENERAL_TLS_ENABLED=true
163    - ORDERER_GENERAL_TLS_CERTIFICATE=/tmp/hyperledger/org1/orderer1/tls-msp/signcerts/cert.pem
164    - ORDERER_GENERAL_TLS_PRIVATEKEY=/tmp/hyperledger/org1/orderer1/tls-msp/keystore/key.pem
165    - ORDERER_GENERAL_TLS_ROOTCERTFILE=/tmp/hyperledger/org1/orderer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
166    - ORDERER_GENERAL_LOGLEVEL=debug
167    - ORDERER_GENERAL_DEBUG_BROADCASTTRACEIDIR=data/logs
168    - ORDERER_GENERAL_CLUSTER_CLIENTCERTIFICATE=/tmp/hyperledger/org1/orderer1/tls-msp/signcerts/cert.pem
169    - ORDERER_GENERAL_CLUSTER_CLIENTPRIVATEKEY=/tmp/hyperledger/org1/orderer1/tls-msp/keystore/key.pem
170    - ORDERER_GENERAL_CLUSTER_ROOTCAS=/tmp/hyperledger/org1/orderer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
171    - ORDERER_GENERAL_BOOTSTRAPMETHOD=none
172    - ORDERER_ADMIN_LISTENADDRESS=0.0.0.0:7080
173    - ORDERER_ADMIN_TLS_ENABLED=true
174    - ORDERER_ADMIN_TLS_PRIVATEKEY=/tmp/hyperledger/org1/orderer1/tls-msp/keystore/key.pem
175    - ORDERER_ADMIN_TLS_CERTIFICATE=/tmp/hyperledger/org1/orderer1/tls-msp/signcerts/cert.pem
176    - ORDERER_ADMIN_TLS_CLIENTAUTOREQUIRED=true
177    - ORDERER_ADMIN_TLS_CLIENTROOTCAS=/tmp/hyperledger/org1/admin/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
178    - ORDERER_CHANNELPARTICIPATION_ENABLED=true
179 volumes:
180   - /tmp/hyperledger/org1/orderer1:/tmp/hyperledger/org1/orderer1/
181   - /tmp/hyperledger/org1/admin:/tmp/hyperledger/org1/admin/
182 ports:
183   - 7050:7050
184   - 7080:7080
185   - 9443:9443
186 networks:
187   - hflines
188
189 orderer1-org2:
190   container_name: orderer1-org2
191   image: hyperledger/fabric-orderer:latest
192   environment:
193    - ORDERER_HOME=/tmp/hyperledger/orderer
194    - ORDERER_HOST=orderer1-org2
195    - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
196    # - ORDERER_GENERAL_GENESISMETHOD=org2MSP
197    - ORDERER_GENERAL_GENESISFILE=/tmp/hyperledger/org2/orderer1/genesis.block
198    - ORDERER_GENERAL_LOCALMSPID=org2MSP
199    - ORDERER_GENERAL_LOCALMSPIR=/tmp/hyperledger/org2/orderer1/msp
200    - ORDERER_GENERAL_TLS_ENABLED=true
201    - ORDERER_GENERAL_TLS_CERTIFICATE=/tmp/hyperledger/org2/orderer1/tls-msp/signcerts/cert.pem
202    - ORDERER_GENERAL_TLS_PRIVATEKEY=/tmp/hyperledger/org2/orderer1/tls-msp/keystore/key.pem
203    - ORDERER_GENERAL_TLS_ROOTCAS=/tmp/hyperledger/org2/orderer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
204    - ORDERER_GENERAL_DEBUG_BROADCASTTRACEIDIR=data/logs
205    - ORDERER_GENERAL_CLUSTER_CLIENTCERTIFICATE=/tmp/hyperledger/org2/orderer1/tls-msp/signcerts/cert.pem
206    - ORDERER_GENERAL_CLUSTER_CLIENTPRIVATEKEY=/tmp/hyperledger/org2/orderer1/tls-msp/keystore/key.pem
207    - ORDERER_GENERAL_CLUSTER_ROOTCAS=/tmp/hyperledger/org2/orderer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
208    - ORDERER_GENERAL_BOOTSTRAPMETHOD=none
209    - ORDERER_ADMIN_LISTENADDRESS=0.0.0.0:7080
210    - ORDERER_ADMIN_TLS_ENABLED=true
211    - ORDERER_ADMIN_TLS_PRIVATEKEY=/tmp/hyperledger/org2/orderer1/tls-msp/keystore/key.pem
212    - ORDERER_ADMIN_TLS_CERTIFICATE=/tmp/hyperledger/org2/orderer1/tls-msp/signcerts/cert.pem
213    - ORDERER_ADMIN_TLS_CLIENTAUTOREQUIRED=true
214    - ORDERER_ADMIN_TLS_CLIENTROOTCAS=/tmp/hyperledger/org2/admin/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
215    - ORDERER_CHANNELPARTICIPATION_ENABLED=true
216 volumes:
217   - /tmp/hyperledger/org2/orderer1:/tmp/hyperledger/org2/orderer1/
218   - /tmp/hyperledger/org2/admin:/tmp/hyperledger/org2/admin/
219 ports:
220   - 8050:8050
221   - 8080:8080
222   - 9446:9446
223 networks:
224   - hflines
225

```

Figura 3: Archivo Docker-compose.yaml de la red Hyperledger Fabric HFLines

06. METODOLOGÍA PARA LA CREACIÓN Y DESPLIEGUE DE LA RED HFLINES

En este apartado se describe como realizar la puesta en marcha del proyecto **HFLines**.

Para utilizar **HFLINES 2.0** y realizar su puesta en funcionamiento en pocos pasos, ejecutaremos todas las operaciones descritas a continuación, después de haber clonado el proyecto publicado en GitHub en su máquina virtual.

En nuestro caso, los desarrollos se han realizado en una instancia VM en Google Cloud:

Tipo de máquina	n2-standard-4			
Plataforma de CPU	Intel Cascade Lake			
Plataforma de CPU mínima	Ninguna			
Arquitectura	x86-64			
Imagen	Tipo de interfaz	Tamaño (GB)	Tipo	Arquitectura
ubuntu-2204-jammy-v20240501	SCSI	50	Disco SSD persistente	x86-64

Figura 4: Descripción de la instancia VM en Google Cloud.

Para una adecuada instalación, necesitaremos cumplir una serie de requisitos, previos al despliegue de **HFLines** (las versiones son sólo orientativas, pero en nuestro caso, son las que no han presentado incompatibilidades):

- **Curl** (curl 7.81.0).
- **Git** (git version 2.34.1)
- **Docker** (Docker version 26.1.3, build b72abbb)
- **Docker compose** (Docker Compose version v2.27.0).
- **Npm** (8.5.1).
- **Java**
 - openjdk version "21.0.3" 2024-04-16
 - OpenJDK Runtime Environment (build 21.0.3+9-Ubuntu-1ubuntu122.04.1)
 - OpenJDK 64-Bit Server VM (build 21.0.3+9-Ubuntu-1ubuntu122.04.1, mixed mode, sharing)
 - [Si se programa Chaincodes en Java, los contenedores Docker disponen JDK 11]
- **Jq** (jq-1.6)
- **Fabric-samples** (v 2.x):


```
curl -sSL https://bit.ly/2ysbOFE | bash -s
```

Existe un script (**09_levantar_hflines.sh**) para crear la red **Hyperledger Fabric HFLines** que automatiza la disponibilidad de:

- Todo el **material criptográfico** para todos los componentes de red y usuarios.
- Levantar toda la arquitectura de contenedores para implementar la red **Hyperledger Fabric HFLines**. En este proyecto se ha decidido disponer de dos organizaciones **Org1** y **Org2** con los siguientes contenedores:

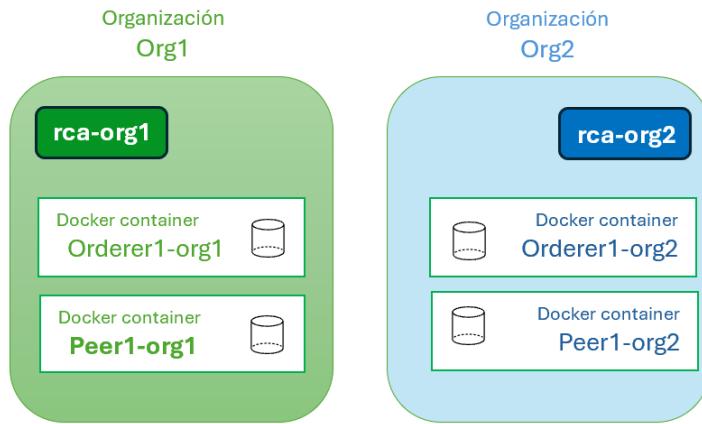


Figura 5: Descripción de las dos organizaciones existentes en HFLines.

- Se dispone de un canal en el cual se facilita a las dos organizaciones la posibilidad de ejecutar un **chaincode**.
- Se monitorizan las métricas de la red y su rendimiento con:
 - Prometheus (queries)
 - Grafana (Gráficos)
- Toda la puesta en marcha está automatizada en el **script 09_levantar_hflines.sh**.
- Para eliminar una red en funcionamiento, podemos ejecutar el script **08_eliminar_hflines.sh**.
- Para visualizar la arquitectura de contenedores de la red ejecutar:

```
docker ps --format 'table {{.Names}}\t{{.ID}}\t{{.State}}'
```

- Para una información más detallada:

```
docker ps --format 'table {{.Names}}\t{{.ID}}\t{{.State}}\t{{.Ports}}'
```

```
[esbrinachain@hfabric1:~/fabric-samples/hflines$ ./02_levantar_nodos_db.sh
[*] Running 9/9
✓ Container ca-tls      Running
✓ Container couchdb1    Started
✓ Container peer1-org2  Started
✓ Container rca-org2    Running
✓ Container couchdb2    Started
✓ Container orderer1-org2 Started
✓ Container orderer1-org1 Started
✓ Container rca-org1    Running
✓ Container peer1-org1  Started
Arquitectura de contenedores de HFLINES :
CONTAINER ID  NAME        STATUS     PORTS
6dfa1fb9d3ab6  ca-tls      Up 3 minutes  0.0.0.0:7052->7052/tcp, :::7052->7052/tcp, 7054/tcp
a585ac24a980   couchdb1   Up 5 seconds   4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp, :::5984->5984/tcp
6c83384c4db5   couchdb2   Up 5 seconds   4369/tcp, 9100/tcp, 0.0.0.0:7984->7984/tcp, :::7984->7984/tcp
f96da01d1f60   orderer1-org1 Up 5 seconds   0.0.0.0:7050->7050/tcp, :::7050->7050/tcp, 0.0.0.0:7080->7080/tcp, :::7080->7080/tcp, 0.0.0.0:9443->9443/tcp, :::9443->9443/tcp
9720aaec048d   orderer1-org2 Up 5 seconds   0.0.0.0:9446->9446/tcp, :::9446->9446/tcp, 0.0.0.0:8050->7050/tcp, :::8050->7050/tcp, 0.0.0.0:8080->7080/tcp, :::8080->7080/tcp
bf596c999c1e   peer1-org1  Up 5 seconds   0.0.0.0:7051->7051/tcp, :::7051->7051/tcp, 0.0.0.0:9444->9444/tcp, :::9444->9444/tcp
f118993d5d24   peer1-org2  Up 5 seconds   0.0.0.0:9051->9051/tcp, :::9051->9051/tcp, 7051/tcp, 0.0.0.0:9445->9445/tcp, :::9445->9445/tcp
680125ea0c7b   rca-org1   Up 3 minutes   0.0.0.0:7054->7054/tcp, :::7054->7054/tcp
44efc92e15fc   rca-org2   Up 3 minutes   7054/tcp, 0.0.0.0:7055->7055/tcp, :::7055->7055/tcp
[esbrinachain@hfabric1:~/fabric-samples/hflines$
```

Figura 6: Listado de ejemplo de los contenedores Docker levantados en HFLines

- Para el despliegue de un **chaincode** en el canal ‘**mychannel**’ disponible tanto para **org1** como **org2**, usaremos una secuencia de scripts:
 - **10a_chaincode_alta_org1.sh**

La ejecución seria:

- **./10a_chaincode_alta_org1.sh 1**

el parámetro 1 indica que se registrará el chaincode con el nombre chaincode1, si 2 chaincode2 etc...

Internamente el script usa el chaincode existente en la carpeta de desarrollo:

./chcodes/chaincode/build/install/chaincode

Ejemplo de resultado:

chaincode1:c2066d1ea51ebe1c70db570860f9e56a064d13283729e43aae2eec04b33212a0

- Si queremos utilizar otro chaincode simplemente hay que especificar su ruta.

- **10b_chaincode_alta_org1.sh**

Este script, usa un parámetro de índice “1”, y como segundo parámetro el **chaincodeID** obtenido de la ejecución del script **10a_chaincode_alta_org1.sh**.

Siguiendo el ejemplo práctico actual ejecutaríamos (todo en la misma linea):

./10b_chaincode_alta_org1.sh 1 chaincode1:c2066d1ea51ebe1c70db570860f9e56a064d13283729e43aae2eec04b33212a0

Mediante este script realizamos el 'approveformyorg' del chaincodeID. en el **peer** de **org1** (peer1-org1) para el canal 'mychannel'.

- **11_chaincode_alta_org2.sh**

Realiza el alta ('install' y 'approveformyorg') del chaincode mediante su chaincodeID en el peer de org2 para el canal 'mychannel'.

Siguiendo el procedimiento del ejemplo actual seria:

./11_chaincode_alta_org2.sh 1 chaincode1:c2066d1ea51ebe1c70db570860f9e56a064d13283729e43aae2eec04b33212a0

A continuación se realiza para **org1** el 'checkcommitreadiness', 'commit' y comprobación del commit (querycommitted) para el chaincode instalado en el canal 'mychannel' para el uso por las organizaciones **org1** y **org2**, una vez esté realizado el 'commit'. Todo el proceso lo automatiza el script:

- **12_chaincode_aprove_commit.sh**

Siguiendo el ejemplo actual ejecutaríamos el comando (recordar que se sigue manteniendo el índice):

./12_chaincode_aprove_commit.sh 1

Una vez finalizada esta ejecución el paquete-id (**chaincode1**) ya está disponible.

Veremos un mensaje similar a:

Committed chaincode definition for chaincode 'chaincode1' on channel 'mychannel':

Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc,

Approvals: [org1MSP: true, org2MSP: true]

- Para ejecutar las funciones del chaincode, disponemos de los scripts con prefijo '**20_app0x**':

- Para aprender los parámetros que son necesarios para su correcta ejecución disponemos de los scripts de ejemplos:

- **30_ejemplos_uso_20_app0x_org1.sh**
- **30_ejemplos_uso_20_app0x_org2.sh**

Se recomienda revisar su código, para entender cómo se pueden usar individualmente cada uno de los scripts con prefijo '**20_app0x**'.

El primer script ejecuta los ejemplos como la organización **org1** y el segundo para la organización **org2**.

Los dos script necesitan obligatoriamente que el primer parámetro sea el índice numérico usado para etiquetar el nombre del chaincode.

Siguiendo el actual ejemplo, la instrucción a ejecutar si estamos en **org1** sería la siguiente:

```
./30_ejemplos_uso_20_app0x_org1.sh 1
```

Mientras que si estamos en **org2**:

```
./30_ejemplos_uso_20_app0x_org2.sh 1
```

El parámetro de índice variará según el que se haya usado al registrar el chaincode con **10a_chaincode_alta_org1.sh**.

07. CREACIÓN DE LA RED HFLINES PASO A PASO

En este apartado se realiza una descripción de detalle de la ejecución de cada script diseñado para crear la red Hyperledger Fabric **HFLines** con el objetivo de implementar un **sistema de auditoría de mensajería** entre una UTE de las organizaciones **org1** y **org2**.

Paso 1

Ejecutaremos el script **08_eliminar_hflines.sh** para asegurarnos de que no existen restos de una implementación anterior:

```
08_eliminar_hflines.sh
1
2
3 . color.sh
4
5 docker compose down
6 sudo rm -rf /tmp/hyperledger
7
8 echo
9 echo -----
10 echo -e $WHITE_L Red Hyperledger fabric $BLUE_HFLINES $WHITE_L eliminada.
11 echo -----
12 echo
```

Figura 7: Contenido del script **08_eliminar_hflines**.

La instrucción de la línea 3, ejecuta un script que define los colores del texto de los mensajes por pantalla. En la línea 5, eliminamos cualquier arquitectura de red montada con el fichero de configuración de Docker **docker-compose.yaml**, que se utiliza para levantar la red **HFLines**. Finalmente eliminamos

toda la estructura de certificados e información almacenada en forma de volúmenes de la red. Más adelante se explicará cómo levantar y eliminar la red sin eliminar el material criptográfico.

En caso de que también se haya levantado el sistema de monitorización almacenado en la carpeta **prometheus-grafana**, también debería ejecutar el comando

```
docker compose down -d
```

para eliminar cualquier arquitectura de contenedores. Para ello debe estar en este directorio.

Paso 2

Partiendo de cero y seguros de que no existen restos de un levantamiento de red anteriores empezamos el procedimiento de creación de la red **HFLines** descrito en el apartado anterior pero mostrando todo lo que va sucediendo al ejecutar cada script paso a paso. Este procedimiento es para describir al detalle el funcionamiento de la red **Hyperledger Fabric** pero como ya ha quedado claro anteriormente se puede realizar de manera automática ejecutando el script **09_levantar_hflines.sh**.

Ejecutamos el script **00_levantar_CAs.sh**:

```
00_levantar_CAs.sh
1 export PATH=$PATH:$HOME/fabric-samples/bin
2
3 # Levantando contenedores de las Entidades Certificadoras CA para Org1 y Org2:
4 docker compose up -d ca-tls rca-org1 rca-org2
5 sleep 5
6
7 docker compose ps --format 'table {{.ID}}\t{{.Name}}\t{{.Status}}\t{{.Ports}}'
8 USER=$(whoami); $(whoami)
9 sudo chown -R $USER /tmp/hyperledger/*
10
11 . color.sh
12 echo -e $WHITE_L Registrando en $BLUE TLS-CA $NORMAL...
13 sleep 2
14 export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/tls-ca/crypto/tls-cert.pem
15 export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/tls-ca/admin
16
17 fabric-ca-client enroll -d -u https://tls-ca-admin:tls-ca-adminpw@0.0.0.0:7052
18 sleep 5
19 echo -e $WHITE_L Registrando peer1-org1 $BLUE en TLS-CA $NORMAL...
20 sleep 2
21 fabric-ca-client register -d --id.name peer1-org1 --id.secret peer1PW --id.type peer -u https://0.0.0.0:7052
22 echo -e $WHITE_L Registrando peer1-org2 $BLUE en TLS-CA $NORMAL...
23 sleep 2
24 fabric-ca-client register -d --id.name peer1-org2 --id.secret peer1PW --id.type peer -u https://0.0.0.0:7052
25 echo -e $WHITE_L Registrando orderer1-org1 $BLUE en TLS-CA $NORMAL...
26 sleep 2
27 fabric-ca-client register -d --id.name orderer1-org1 --id.secret ordererPW --id.type orderer -u https://0.0.0.0:7052
28 echo -e $WHITE_L Registrando orderer1-org2 $BLUE en TLS-CA $NORMAL...
29 sleep 2
30 fabric-ca-client register -d --id.name orderer1-org2 --id.secret ordererPW --id.type orderer -u https://0.0.0.0:7052
31 echo -e $WHITE_L Registrando admin-org1 $BLUE en TLS-CA $NORMAL...
32 sleep 2
33 fabric-ca-client register -d --id.name admin-org1 --id.secret org1AdminPW --id.type admin -u https://0.0.0.0:7052
34 echo -e $WHITE_L Registrando admin-org1 $BLUE en TLS-CA $NORMAL...
35 sleep 2
36 fabric-ca-client register -d --id.name admin-org2 --id.secret org2AdminPW --id.type admin -u https://0.0.0.0:7052
37
38 echo -e $WHITE_L Registrando en $BLUE RCA-ORG1 $NORMAL...
39
40 export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/ca/crypto/ca-cert.pem
41 export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org1/ca/admin
42
43 fabric-ca-client enroll -d -u https://rca-org1-admin:rca-org1-adminpw@0.0.0.0:7054
44 sleep 5
45
46 echo -e $WHITE_L Registrando peer1-org1 $BLUE en RCA-ORG1 $NORMAL...
47 sleep 2
48 fabric-ca-client register -d --id.name peer1-org1 --id.secret peer1PW --id.type peer -u https://0.0.0.0:7054
49 echo -e $WHITE_L Registrando orderer1-org1 $BLUE en RCA-ORG1 $NORMAL...
50 sleep 2
51 fabric-ca-client register -d --id.name orderer1-org1 --id.secret ordererPW --id.type orderer -u https://0.0.0.0:7054
52 echo -e $WHITE_L Registrando admin-org1 $BLUE en RCA-ORG1 $NORMAL...
53 sleep 2
54 fabric-ca-client register -d --id.name admin-org1 --id.secret org1AdminPW --id.type admin -u https://0.0.0.0:7054
55
56 echo -e $WHITE_L Registrando en $BLUE RCA-ORG2 $NORMAL...
57 sleep 2
58 export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org2/ca/crypto/ca-cert.pem
59 export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org2/ca/admin
60
61 fabric-ca-client enroll -d -u https://rca-org2-admin:rca-org2-adminpw@0.0.0.0:7055
62 sleep 5
63
64 echo -e $WHITE_L Registrando peer1-org2 $BLUE en RCA-ORG2 $NORMAL...
65 sleep 2
66 fabric-ca-client register -d --id.name peer1-org2 --id.secret peer1PW --id.type peer -u https://0.0.0.0:7055
67 echo -e $WHITE_L Registrando orderer1-org2 $BLUE en RCA-ORG2 $NORMAL...
68 sleep 2
69 fabric-ca-client register -d --id.name orderer1-org2 --id.secret ordererPW --id.type orderer -u https://0.0.0.0:7055
70 echo -e $WHITE_L Registrando admin-org2 $BLUE en RCA-ORG2 $NORMAL...
71 sleep 2
72 fabric-ca-client register -d --id.name admin-org2 --id.secret org2AdminPW --id.type admin -u https://0.0.0.0:7055
73 echo
74 echo -e $WHITE_L Todos los registros para las entidades certificadoras CA en $GREEN HFLINES$BLUE realizados$NORMAL!
```

Figura 8: Contenido del script **00_levantar_CAs.sh**

Figura 9: Salida parcial del script **00_levantar_Cas.sh**

Analizando las figuras 8 y 9, podemos observar código y ejecución del script, en donde básicamente registramos componentes de red en entidades certificadoras:

Línea 1: acceso a las herramientas facilitadas por Hyperledger Fabric: **configtxlator**, **cryptogen**, **discover**, **fabric-ca-client**, **fabric-ca-server**, **ledgerutil**, **orderer**, **osnadmin**, **peer**.

Línea 4: levantar tres entidades certificadoras **rca-org1**, **rca-org2** y **ca-tls** (o lo que es lo mismo sus contenedores).

Líneas 8 y 9: obtención del usuario de consola y asignación de propiedad del directorio **/tmp/hyperledger** para almacenar todos los certificados digitales (formato X.509) e información ‘msp’ (members service provider) que se generen para cada componente de la red.

Líneas 14 i 15: definición de las variables de entorno necesarias para el buen funcionamiento de la herramienta **fabric-ca-client**. En este caso la ruta al certificado digital para acceder a la entidad certificadora **ca-tls** (para poder registrar en este nodo **fabric-ca-server**) y la carpeta donde almacenar el material criptográfico de los componentes registrados (nodos y usuarios) de org1 y org2 y el certificado digital de **tls-ca-admin**.

```
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/tls-ca/crypto/tls-cert.pem
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/tls-ca/admin
```

Línea 17: obtenemos el certificado digital de **tls-ca-admin** (operación enroll, posible porque **tls-ca-admin** ya se registró al dar de alta el contenedor).

Línea 21: registramos **peer1-org1** indicando sus características: **id.name**, **id.secret**, **id.type** y la **url** de la entidad certificadora **ca-tls**.

Línea 24, 27, 30: hacemos el registro en **ca-tls** para **peer1-org2**, **orderer1-org1**, **orderer1-org2**.

Línea 33, 36: hacemos el registro en **ca-tls** para los usuarios **admin-org1** y **admin-org2**.

En este punto, ya hemos registrado todos los componentes de **ca-tls**. Nos queda registrar los componentes de cada organización en su entidad certificadora.

Línea 40, 41: modificamos las variables de entorno para que contengan el valor de la ruta del certificado digital de la entidad certificadora **rca-org1** y la ruta de la carpeta donde almacenaremos el material criptográfico (publicKey, keystore serán las carpetas que almacenen la clave pública y privada) y ‘msp’ los miembros que pertenecen a esta entidad certificadora.

Línea 43: Obtenemos el certificado digital del usuario administrador de **rca-org1**.

Líneas 48, 51 y 54: registramos los nodos **peer1-org1**, **orderer1-org1** y el usuario **admin** en **rca-org1**.

Este proceso se repite para **rca-org2**, finalizando el script habiendo registrado cada componente de red donde necesitaba estar registrado para poder obtener su certificado digital en la ejecución del siguiente script **01_certificar_Orgs.sh**.

Para que el usuario de **HFLines** pueda seguir todo el proceso a medida que va realizándose, se han insertado mensajes que informan en la línea de comandos de cada registro. Las órdenes ‘sleep’ son muy necesarias pues los registros tienen sus tiempos de latencia que hay que respetar, al mismo tiempo que permitimos leer los resultados y mensajes por pantalla.

Paso 3

La siguiente operación finalizará el procedimiento iniciado en el paso anterior. Se obtendrá todo el material criptográfico para operar con seguridad en la red Hyperledger Fabric **HFLines**. Todo el proceso está automatizado en el script **01_certificar_Orgs.sh** y en segundo plano se va almacenado todo este material de manera ordenada en la estructura de directorios **/tmp/hyperledger**.

Como en el paso anterior mostramos el código y luego la salida por terminal en sus aspectos más importantes, puesto que son tareas repetitivas que entendido un caso, el resto son iguales.

```

1  01_certificar_Orgs.sh X
2  01_certificar_Orgs.sh
3
4  . color.sh
5
6  echo -e $WHITE_L Emisión de certificados para la organización $GREEN Org1$NORMAL!
7  echo -e $WHITE_L Emisión de certificado CA para $GREEN peer1-org1$NORMAL!
8  sleep 3
9
10 mkdir -p /tmp/hyperledger/org1/peer1/assets/ca
11 cp /tmp/hyperledger/org1/ca/admin/msp/cacerts/0-0-0-0-7054.pem /tmp/hyperledger/org1/peer1/assets/ca/org1-ca-cert.pem
12
13 mkdir -p /tmp/hyperledger/org1/peer1/assets/tls-ca
14 cp /tmp/hyperledger/tls-ca/admin/msp/cacerts/0-0-0-0-7052.pem /tmp/hyperledger/org1/peer1/assets/tls-ca/tls-ca-cert.pem
15
16 export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org1/peer1
17 export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/peer1/assets/ca/org1-ca-cert.pem
18 export FABRIC_CA_CLIENT_MSPDIR=msp
19
20 fabric-ca-client enroll -d -u https://peer1-org1:peer1PW@0.0.0.0:7054
21 sleep 5
22
23 echo -e $WHITE_L Emisión de certificado TLS para $GREEN peer1-org1$NORMAL!
24 sleep 2
25
26 export FABRIC_CA_CLIENT_MSPDIR=tls-msp
27 export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/peer1/assets/tls-ca/tls-ca-cert.pem
28
29 fabric-ca-client enroll -d -u https://peer1-org1:peer1PW@0.0.0.0:7052 --enrollment.profile tls --csr.hosts peer1-org1 --csr.hosts localhost
30 sleep 5
31
32 cp /tmp/hyperledger/org1/peer1/tls-msp/keystore/*_sk /tmp/hyperledger/org1/peer1/tls-msp/keystore/key.pem
33
34 echo -e $WHITE_L Emisión de certificado CA para $GREEN orderer1-org1$NORMAL!
35 sleep 2
36
37 mkdir -p /tmp/hyperledger/org1/orderer1/assets/ca
38 cp /tmp/hyperledger/org1/ca/admin/msp/cacerts/0-0-0-0-7054.pem /tmp/hyperledger/org1/orderer1/assets/ca/org1-ca-cert.pem
39
40 mkdir -p /tmp/hyperledger/org1/orderer1/assets/tls-ca
41 cp /tmp/hyperledger/tls-ca/admin/msp/cacerts/0-0-0-0-7052.pem /tmp/hyperledger/org1/orderer1/assets/tls-ca/tls-ca-cert.pem
42
43 export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org1/orderer1
44 export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/orderer1/assets/ca/org1-ca-cert.pem
45 export FABRIC_CA_CLIENT_MSPDIR=msp
46
47 fabric-ca-client enroll -d -u https://orderer1-org1:ordererpw@0.0.0.0:7054
48 sleep 5
49
50 echo -e $WHITE_L Emisión de certificado TLS para $GREEN orderer1-org1$NORMAL!
51 sleep 2
52
53
54 export FABRIC_CA_CLIENT_MSPDIR=tls-msp
55 export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/orderer1/assets/tls-ca/tls-ca-cert.pem
56
57 fabric-ca-client enroll -d -u https://orderer1-org1:ordererpw@0.0.0.0:7052 --enrollment.profile tls --csr.hosts orderer1-org1 --csr.hosts localhost
58 sleep 5
59
60 cp /tmp/hyperledger/org1/orderer1/tls-msp/keystore/*_sk /tmp/hyperledger/org1/orderer1/tls-msp/keystore/key.pem
61
62 echo -e $WHITE_L Emisión de certificado CA para $GREEN admin-org1$NORMAL!
63 sleep 2
64
65 export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org1/admin
66 export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/peer1/assets/ca/org1-ca-cert.pem
67 export FABRIC_CA_CLIENT_MSPDIR=msp
68
69 fabric-ca-client enroll -d -u https://admin-org1:org1AdminPW@0.0.0.0:7054
70
71 echo -e $WHITE_L Emisión de certificado TLS para $GREEN admin-org1$NORMAL!
72 sleep 2
73
74 export FABRIC_CA_CLIENT_MSPDIR=tls-msp
75 export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/peer1/assets/tls-ca/tls-ca-cert.pem
76
77 fabric-ca-client enroll -d -u https://admin-org1:org1AdminPW@0.0.0.0:7052 --enrollment.profile tls
78 sleep 5
79
80 cp /tmp/hyperledger/org1/admin/tls-msp/keystore/*_sk /tmp/hyperledger/org1/admin/tls-msp/keystore/key.pem
81
82 mkdir -p /tmp/hyperledger/org1/peer1/msp/admincerts
83 cp /tmp/hyperledger/org1/admin/msp/signcerts/cert.pem /tmp/hyperledger/org1/peer1/msp/admincerts/org1-admin-cert.pem
84
85 mkdir -p /tmp/hyperledger/org1/admin/msp/admincerts
86 cp /tmp/hyperledger/org1/admin/msp/signcerts/cert.pem /tmp/hyperledger/org1/admin/msp/admincerts/org1-admin-cert.pem
87
88 mkdir -p /tmp/hyperledger/org1/msp/{admincerts,cacerts,tlscacerts,users}
89 cp /tmp/hyperledger/org1/peer1/assets/ca/org1-ca-cert.pem /tmp/hyperledger/org1/msp/cacerts/
90 cp /tmp/hyperledger/org1/peer1/assets/tls-ca/tls-ca-cert.pem /tmp/hyperledger/org1/msp/tlsCACerts/
91 cp /tmp/hyperledger/org1/admin/msp/signcerts/cert.pem /tmp/hyperledger/org1/msp/admincerts/admin-org1-cert.pem
92 cp ./org1-config.yaml /tmp/hyperledger/org1/msp/config.yaml
93 cp ./org1-config.yaml /tmp/hyperledger/org1/orderer1/msp/config.yaml
94
95 echo -e $WHITE_L Emisión de certificados para la organización $GREEN Org2$NORMAL!
96 echo -e $WHITE_L Emisión de certificado CA para $GREEN peer1-org2$NORMAL!

```

Figura 10: Contenido parcial del script 01_certificar_Orgs.sh

Figura 11: Salida parcial del script 01_certificar_Orgs.sh

En el script **01_certificar_Orgs.sh**, se realiza el minucioso trabajo de almacenamiento de los certificados de cada organización en su entidad certificadora y los de cada componente de red (de cada organización) en la entidad certificadora de su organización. Al utilizarse el protocolo de transporte TLS, además de usarse un certificado digital que prueba que está emitido por su entidad certificadora, se tiene que almacenar también el certificado digital que le facilita **ca-tls**, esencial para firmar cada transacción en la red **HFLines**.

Comprobamos estos aspectos en el código del script:

Línea 8, 9: se almacena el certificado de la organización **org1** para **peer1-org1**.

Línea 11, 12: se almacena el certificado TLS de **peer1-org1**.

Líneas 14, 15, 16: se da valor a las variables de entorno:

```
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org1/peer1
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/peer1/assets/ca/org1-ca-cert.pem
export FABRIC_CA_CLIENT_MSPDIR=msp
```

Para la primera variable se indica la ruta de almacenamiento del componente del cual obtenemos el material criptográfico. Para la segunda, el certificado de la entidad certificadora a la que nos dirigimos.

Y en la tercera la carpeta el nombre de la carpeta ‘member service provider’.

Línea 18: obtenemos el certificado digital y todo el material criptográfico para **peer1-org1**.

Línea 24, 25: se modifican los valores de las variables para obtener ahora el certificado y material criptográfico del protocolo de transporte TLS facilitado por **ca-tls**, que ya tiene a este componente registrado del paso anterior.

Línea 27: se obtiene el certificado digital y material criptográfico realizando una operación de ‘enroll’.

El resto del script repite este proceso para el resto de los componentes de **org1** en **rca-org1** i **ca-tls**, y para los componentes de **org2** en **rca-org2** i **ca-tls**. Para la carpeta del msp del protocolo de transporte TLS utilizamos la notación **msp-tls**.

Paso 4

Una vez generado el material criptográfico, podemos levantar el resto de los contenedores Docker de la red **HFLines**, proceso que realiza el script **02_levantar_nodos_db.sh**.

```
ls 02_levantar_nodos_db.sh
1 # Levantar peers y clientes
2 docker compose up -d
3 sleep 5
4 echo -e $WHITE_L Arquitectura de contenedores de$BLUE HFLINES $NORMAL:
5 docker compose ps --format 'table {{.ID}}\t{{.Name}}\t{{.Status}}\t{{.Ports}}'
```

Figura 12: Contenido parcial del script **02_levantar_nodos_db.sh**

```
esbrinachain@hfabric1:~/fabric-samples/hflines$ ./02_levantar_nodos_db.sh
[*] Running 9/9
✓ Container peer1-org2      Started
✓ Container couchdb1        Started
✓ Container rca-org2        Running
✓ Container couchdb2        Started
✓ Container rca-org1        Running
✓ Container orderer1-org1   Started
✓ Container orderer1-org2   Started
✓ Container ca-tls          Running
✓ Container peer1-org1      Started
Apilamiento de contenedores en HFLINES :
CONTAINER ID        IMAGE               STATUS            PORTS
76ea33aabb3c        ca-tls              Up 3 hours        0.0.0.0:7052->7052/tcp, :::7052->7052/tcp, 7054/tcp
be3084aa8a8a        couchdb1           Up 5 seconds       4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp, :::5984->5984/tcp
87af6c8cbe5f        couchdb2             Up 5 seconds       4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp, :::5984->5984/tcp
e9c58b678b32        orderer1-org1      Up 5 seconds       0.0.0.0:7050->7050/tcp, :::7050->7050/tcp, 0.0.0.0:7080->7080/tcp, :::7080->7080/tcp, 0.0.0.0:9443->9443/tcp, :::9443->9443/tcp
344adb24444         orderer1-org2      Up 5 seconds       0.0.0.0:9446->9446/tcp, 0.0.0.0:8050->7050/tcp, :::8050->7050/tcp, 0.0.0.0:8080->7080/tcp, :::8080->7080/tcp
f7aca82cc347        peer1-org1         Up 5 seconds       0.0.0.0:7051->7051/tcp, :::7051->7051/tcp, 0.0.0.0:9444->9444/tcp, :::9444->9444/tcp
b366e9673bf6        peer1-org2         Up 5 seconds       0.0.0.0:9051->9051/tcp, :::9051->9051/tcp, 7051/tcp, 0.0.0.0:9445->9445/tcp, :::9445->9445/tcp
88eb990a91dd        rca-org1           Up 3 hours        0.0.0.0:7054->7054/tcp, :::7054->7054/tcp
ff44d03a9d68        rca-org2           Up 3 hours        7054/tcp, 0.0.0.0:7055->7055/tcp, :::7055->7055/tcp
esbrinachain@hfabric1:~/fabric-samples/hflines$
```

Figura 13: Salida en terminal del script **02_levantar_nodos_db.sh**

Paso 5

En este momento ya tenemos en funcionamiento la red del proyecto **HFLines**. Pero no tenemos manera de ejecutar chaincodes ni de comunicar **org1** con **org2**. Para ello, necesitamos **definir un canal e incorporar uno más nodos de cada organización de tipo peer y orderer**. En **HFLines** definimos un canal con **dos nodos peer y dos nodos orderer**.

Las decisiones de diseño se tomaron teniendo en cuenta que se debía empezar por una estructura de red sencilla, y cuando esta funcionara ir incrementándola según necesidades operativas.

En cuanto al diseño de la red, aporta un elemento innovador diferente a los modelos existentes, pues se ha diseñado un esquema en el que no existe una organización destinada a mantener nodos **orderer**, sino que se ha optado por un diseño en el que hay un nodo **orderer** por organización.

Este esquema nos parece que proporciona al proyecto unas capacidades apropiadas para nuestro caso de uso, en el que cada empresa representada por una organización puede usar el proyecto de manera interna o colaborar con más organizaciones.

Para crear un **registro de configuración de canal**, y generar el **bloque génesis** para el canal ejecutaremos el script **03_artifacts.sh**.

Vemos en las figuras 14 y 15 el código del script y su salida en el terminal respectivamente:

```
03_artifacts.sh
22 configtxgen -profile SampleAppChannelEtcdrAft \ 
23 | -configPath ${PWD} \
24 | -outputBlock mychannel.block \
25 | -channelID mychannel
26
27 echo -e $WHITE_L Consultando el status del canal$NORMAL ...
28
29 osnadmin channel list -o localhost:7080 \
30 | --ca-file $ORDERER_CA \
31 | --client-cert /tmp/hyperledger/org1/admin/tls-msp/signcerts/cert.pem \
32 | --client-key /tmp/hyperledger/org1/admin/tls-msp/keystore/key.pem
33
```

Figura 14: Código fuente del script **03_artifacts.sh**

Si ejecutamos el script **03_artifacts.sh**, desde la línea de comandos necesitaremos pasarle 2 parámetros: (dentro del script hemos indicado algunos parámetros fijos que veremos más adelante)

1. La ruta donde queremos definir el directorio de configuración y que será donde se almacenará el fichero del bloque génesis, con la notación '**<nombre_canal>.block**', (en nuestro caso usamos la ruta raíz de HFLines): **/home/esbrinachain/fabric-samples/hflines**
2. Como el proceso lo ejecuta una organización usamos **org1**, y como hace falta la referencia de un nodo orderer, usamos el **orderer1-org1** y la ruta de su certificado se pasa como segundo parámetro:

/tmp/hyperledger/org1/orderer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem

```
esbrinachain@hfabric1:~/fabric-samples/hflines$ export ORDERER_CA=/tmp/hyperledger/org1/orderer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
esbrinachain@hfabric1:~/fabric-samples/hflines$ ./03_artifacts.sh /home/esbrinachain/fabric-samples/hflines
Creando bloque génesis para el canal ...
2024-06-17 21:12:34.578 UTC 0001 INFO [common.tools.configtxgen] main -> Loading configuration
2024-06-17 21:12:34.588 UTC 0002 INFO [common.tools.configtxgen.localconfig] completeInitialization -> orderer type: etcdrft
2024-06-17 21:12:34.588 UTC 0003 INFO [common.tools.configtxgen.localconfig] completeInitialization -> Orderer.Etcdrft.Options unset, setting to tick_interval:"500ms" election_ticks:10 heartbeat_tick:1 max_inflight_blocks:5 snapshot_interval_size:16777216
2024-06-17 21:12:34.589 UTC 0004 INFO [common.tools.configtxgen.localconfig] Load -> Loaded configuration: /home/esbrinachain/fabric-samples/hflines/configtx.yaml
2024-06-17 21:12:34.590 UTC 0005 INFO [common.tools.configtxgen] doOutputBlock -> Generating genesis block
2024-06-17 21:12:34.591 UTC 0006 INFO [common.tools.configtxgen] doOutputBlock -> Creating application channel genesis block
2024-06-17 21:12:34.591 UTC 0007 INFO [common.tools.configtxgen] doOutputBlock -> Writing genesis block
Consultando el status del canal ...
Status: 200
{
    "systemChannel": null,
    "channels": null
}
```

Figura 15: Salida en terminal del script **03_artifacts.sh**

Utilizando estos parámetros generamos el **bloque génesis** y conseguimos definir un canal para la red.

Paso 6

Para poder comunicar por el canal nos falta añadirle los **nodos peer** de cada organización y en nuestro caso añadir los **nodos orderer**. Este paso lo automatiza el script **04_union_canal_orderer1-org1.sh**.

Como se puede observar en la figura 16 necesitaremos actuar como la organización **org1** y por ello hay que definir todas las variables de entorno, que le indican a Hyperledger Fabric los valores usados en los comandos y que describen a **org1** prioritariamente a lo definido en el fichero **configtx.yaml**:

```
04_union_canal_orderer1-org1.sh
1
2 export PATH=$PATH:${HOME}/fabric-samples/bin
3 export FABRIC_CFG_PATH=${HOME}/fabric-samples/config
4 export ORDERER_CA=/tmp/hyperledger/org1/orderer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
5
6 export CORE_PEER_TLS_ENABLED=true
7 export CORE_PEER_LOCALMSPID="org1MSP"
8 export CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org1/peer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
9 export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org1/admin/msp
10 export CORE_PEER_ADDRESS=localhost:7051
11
12 echo -e $WHITE_L Estado actual del canal respecto Org1$NORMAL ...
13 sleep 2
14 osnadmin channel list -o localhost:7080 \
15           --ca-file $ORDERER_CA \
16           --client-cert /tmp/hyperledger/org1/admin/tls-msp/signcerts/cert.pem \
17           --client-key /tmp/hyperledger/org1/admin/tls-msp/keystore/key.pem
18
19 echo -e $WHITE_L Uniendo al canal a orderer1-org1$NORMAL ...
20 sleep 2
21 osnadmin channel join --channelID mychannel \
22           --config-block mychannel.block \
23           -o localhost:7080 \
24           --ca-file $ORDERER_CA \
25           --client-cert /tmp/hyperledger/org1/admin/tls-msp/signcerts/cert.pem \
26           --client-key /tmp/hyperledger/org1/admin/tls-msp/keystore/key.pem
27
28
29 echo -e $WHITE_L Estado final del canal una vez añadido orderer1-org1$NORMAL ...
30 sleep 2
31 osnadmin channel list -o localhost:7080 \
32           --ca-file $ORDERER_CA \
33           --client-cert /tmp/hyperledger/org1/admin/tls-msp/signcerts/cert.pem \
34           --client-key /tmp/hyperledger/org1/admin/tls-msp/keystore/key.pem
```

Figura 16: Código fuente del script **04_union_canal_orderer1-org1.sh**

Observando el código podemos comprobar que:

Líneas 2, 3, 4, 6, 7, 8, 9: definen las variables de entorno que describen en la red **HFLines** a **org1**.

Líneas 14 a 17: consultamos el estado del canal, utilizando el material criptográfico del usuario **admin** de **org1** además de facilitar máquina y puerto de comunicaciones.

Líneas 21 a 26: añadimos el nodo **orderer** de **org1**, **orderer-org1** al canal creado en el paso 5, pues utilizamos los parámetros anteriores de **orderer1-org1** y para el canal su identificador y fichero de **bloque génesis**.

Líneas 31 a 34: Repetimos la consulta para obtener si el canal sigue en estado correcto y funcionando sin errores.

```

esbrinachain@hfabric1:~/fabric-samples/hflines$ ./04_union_canal_orderer1-org1.sh
Estado actual del canal respecto Org1 ...
Status: 200
{
    "systemChannel": null,
    "channels": null
}

Uniendo al canal a orderer1-org1 ...
Status: 201
{
    "name": "mychannel",
    "url": "/participation/v1/channels/mychannel",
    "consensusRelation": "conseenter",
    "status": "active",
    "height": 1
}

Estado final del canal una vez añadido orderer1-org1 ...
Status: 200
{
    "systemChannel": null,
    "channels": [
        {
            "name": "mychannel",
            "url": "/participation/v1/channels/mychannel"
        }
    ]
}

```

Figura 17: Salida en terminal del script **04_union_canal_orderer1-org1.sh**

En el siguiente paso repetiremos el mismo proceso pero añadiremos el nodo **orderer1-org2** actuando como la organización **org2**.

Paso 7

Mostramos el código fuente y la salida del terminal del script **05_union_canal_orderer1-org2.sh** pero el procedimiento es el mismo del paso anterior y solo cambiamos las variables de entorno para que representen a la organización **org2**.

```

05_union_canal_orderer1-org2.sh
1  export PATH=$PATH:${HOME}/fabric-samples/bin
2  export FABRIC_CFG_PATH=${HOME}/fabric-samples/config/
3  export ORDERER_CA=/tmp/hyperledger/org2/orderer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
4
5  export CORE_PEER_TLS_ENABLED=true
6  export CORE_PEER_LOCALMSPID="org2MSP"
7  export CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org2/peer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
8  export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org2/admin/msp
9  export CORE_PEER_ADDRESS=localhost:9051
10
11
12 echo -e $WHITE_L Estado actual del canal respecto Org1$NORMAL ...
13 sleep 2
14 osnadmin channel list -o localhost:8080 \
15   --ca-file $ORDERER_CA \
16   --client-cert /tmp/hyperledger/org2/admin/tls-msp/signcerts/cert.pem \
17   --client-key /tmp/hyperledger/org2/admin/tls-msp/keystore/key.pem
18
19 echo -e $WHITE_L Uniendo al canal a orderer1-org2$NORMAL ...
20 sleep 2
21 osnadmin channel join --channelID mychannel \
22   --config-block mychannel.block \
23   -o localhost:8080 \
24   --ca-file $ORDERER_CA \
25   --client-cert /tmp/hyperledger/org2/admin/tls-msp/signcerts/cert.pem \
26   --client-key /tmp/hyperledger/org2/admin/tls-msp/keystore/key.pem
27
28 echo -e $WHITE_L Estado final del canal una vez añadido orderer1-org2$NORMAL ...
29 sleep 2
30 osnadmin channel list -o localhost:8080 \
31   --ca-file $ORDERER_CA \
32   --client-cert /tmp/hyperledger/org2/admin/tls-msp/signcerts/cert.pem \
33   --client-key /tmp/hyperledger/org2/admin/tls-msp/keystore/key.pem

```

Figura 18: Código fuente del script **05_union_canal_orderer1-org2.sh**

```

esbrinachain@hfabric1:~/fabric-samples/hflines$ ./05_union_canal_orderer1-org2.sh
Estado actual del canal respecto Org1 ...
Status: 200
{
    "systemChannel": null,
    "channels": null
}

Uniendo al canal a orderer1-org2 ...
Status: 201
{
    "name": "mychannel",
    "url": "/participation/v1/channels/mychannel",
    "consensusRelation": "consenter",
    "status": "active",
    "height": 1
}

Estado final del canal una vez añadido orderer1-org2 ...
Status: 200
{
    "systemChannel": null,
    "channels": [
        {
            "name": "mychannel",
            "url": "/participation/v1/channels/mychannel"
        }
    ]
}

esbrinachain@hfabric1:~/fabric-samples/hflines$
```

Figura 19: Salida en terminal del script 05_union_canal_orderer1-org2.sh

Al igual que se han añadido los **nodos orderer** en el siguiente paso incluiremos los **nodos peer** ejecutando los scripts: **06_union_canal_peer1-org1.sh** y **07_union_canal_peer1-org2.sh**

Paso 8

Para añadir nodos peer utilizaremos la herramienta “**peer cli**” pero usándola con el parámetro **“channel”**. El comando **peer channel** permite a los administradores realizar operaciones relacionadas con el canal en un nodo peer (en castellano, par), como por ejemplo:

1. Unirse a un canal
2. Enumerar los canales a los que está unido un peer.

```

peer channel
Operate a channel: create|fetch|join|joinbysnapshot|joinbysnapshotstatus|list|update|signconfigtx|getinfo.

Usage:
  peer channel [command]

Available Commands:
  create           Create a channel
  fetch            Fetch a block
  getinfo          Get blockchain information of a specified channel.
  join             Joins the peer to a channel.
  joinbysnapshot  Joins the peer to a channel by the specified snapshot
  joinbysnapshotstatus Query if joinbysnapshot is running for any channel
  list             List of channels peer has joined.
  signconfigtx    Signs a configtx update.
  update           Send a configtx update.

Flags:
  --cafile string      Path to file containing PEM-encoded trusted certificate(s) for the ordering endpoint
  --certfile string    Path to file containing PEM-encoded X509 public key to use for mutual TLS communication with the orderer endpoint
  --clientauth         Use mutual TLS when communicating with the orderer endpoint
  --connTimeout duration Timeout for client to connect (default 3s)
  -h, --help           help for channel
  --keyfile string    Path to file containing PEM-encoded private key to use for mutual TLS communication with the orderer endpoint
  --orderer string     Ordering service endpoint
  --ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer
  --tls               Use TLS when communicating with the orderer endpoint
  --tlsHandshakeTimeShift duration The amount of time to shift backwards for certificate expiration checks during TLS handshakes with the orderer endpoint
```

Figura 20: Ayuda del comando ‘peer channel’ para la ejecución de ‘peer channel join’ y agregar un nodo peer al canal.

Actuando como organización **org1** uniremos al canal a **peer1-org1**:

El código fuente del script **06_union_canal_peer1-org1.sh** es el mostrado en la figura 21:

```

[-] 06_union_canal_peer1-org1.sh
1 export PATH=$PATH:${HOME}/fabric-samples/bin
2 export FABRIC_CFG_PATH=${HOME}/fabric-samples/config/
3 export ORDERER_CA=/tmp/hyperledger/org1/orderer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
4
5 export CORE_PEER_TLS_ENABLED=true
6 export CORE_PEER_LOCALMSPID="org1MSP"
7 export CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org1/peer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
8 export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org1/admin/msp
9 export CORE_PEER_ADDRESS=localhost:7051
10
11 echo -e $WHITE_L Uniendo al canal a peer1-org1$NORMAL ...
12 peer channel join -b mychannel.block

```

Figura 21: Código fuente del script **06_union_canal_peer1-org1.sh**

Una vez configuradas las variables de entorno para referenciar a la organización **org1**, agregamos al canal su nodo **peer1-org1** utilizando el comando:

peer channel join -b mychannel.block

La salida del script **06_union_canal_peer1-org1.sh** en el terminal es:

```

esbrinachain@hfabric1:~/fabric-samples/hflines$ ./06_union_canal_peer1-org1.sh
Uniendo al canal a peer1-org1 ...
2024-06-17 23:17:37.545 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2024-06-17 23:17:37.665 UTC 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel
esbrinachain@hfabric1:~/fabric-samples/hflines$ 

```

Figura 22: Salida en terminal del script **06_union_canal_peer1-org1.sh**

Como organización **org2** repetimos el proceso para añadir al canal el nodo **peer1-org2**:

El código fuente del script **07_union_canal_peer1-org2.sh** es el mostrado en la figura 23:

```

[-] 07_union_canal_peer1-org2.sh
1 export PATH=$PATH:${HOME}/fabric-samples/bin
2 export FABRIC_CFG_PATH=${HOME}/fabric-samples/config/
3 export ORDERER_CA=/tmp/hyperledger/org2/orderer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
4
5 export CORE_PEER_TLS_ENABLED=true
6 export CORE_PEER_LOCALMSPID="org2MSP"
7 export CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org2/peer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
8 export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org2/admin/msp
9 export CORE_PEER_ADDRESS=localhost:9051
10
11 echo -e $WHITE_L Uniendo al canal a peer2-org1$NORMAL ...
12 peer channel join -b mychannel.block

```

Figura 23: Código fuente del script **07_union_canal_peer1-org2.sh**

La salida del script **07_union_canal_peer1-org2.sh** en el terminal es:

```

esbrinachain@hfabric1:~/fabric-samples/hflines$ ./07_union_canal_peer1-org2.sh
Uniendo al canal a peer2-org1 ...
2024-06-17 23:21:03.466 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2024-06-17 23:21:03.580 UTC 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel
esbrinachain@hfabric1:~/fabric-samples/hflines$ 

```

Figura 24: Salida en terminal del script **07_union_canal_peer1-org2.sh**

Una vez añadidos todos los nodos al canal ya habremos finalizado el proceso de creación de la red **HFLines** y veremos en el terminal un mensaje nos informa de ello:

“Red Hyperledger Fabric HFLINES levantada”

Todo el proceso que se ha descrito queda automatizado y se puede ejecutar en un solo comando mediante el script **09_levantar_hflines.sh**.

Ahora, **HFLines** está levantada y tiene configurado un **canal de comunicaciones de nombre “mychannel”** preparado para recibir transacciones (tx). El canal de comunicaciones registrará todas las

tx en su **libro mayor** (en inglés **ledger**). Para acceder al **ledger** los usuarios o miembros de **org1** y **org2** deben ejecutar la funcionalidad programada en algún chaincode.

08. DEFINICIÓN, INSTALACIÓN Y EJECUCIÓN DE UN CHAINCODE

El ejemplo de ejecución de un chaincode implica realizar varias tareas:

1. Desarrollar un código en alguno de los siguientes lenguajes de programación: Go, Node.js o **Java** utilizando e integrando el API facilitado por **Hyperledger Fabric** para el desarrollo de Smart contracts, que en **Hyperledger Fabric** se empaquetan en **chaincodes**. Las organizaciones que quieran **validar transacciones** o **consultar el libro mayor** (en inglés **ledger**) de un canal deben instalar un **chaincode** en sus **peers** adscritos al canal.
2. Usar la aplicación **peer cli** para ejecutar el ciclo de vida del **chaincode**:
 - Empaquetar el **chaincode** (en inglés **package**).
 - Instalar el paquete **chaincode** (en inglés **install**).
 - Aprobar la definición del **chaincode** (en inglés **approveformyorg**).
 - Enviar la definición del chaincode al canal (en inglés **commit**).
 - Invocar al **chaincode** (en inglés **invoke**) para ejecutar su funcionalidad.

Este último paso lo podemos realizar con **peer chaincode invoke** o usar una aplicación programada para utilizar e integrar el API **Fabric Gateway Client** de **Hyperledger Fabric** que se conecte al canal y envíe transacciones para que ejecuten la funcionalidad de ese **chaincode**.

En esta sección describiremos como se puede realizar todo el **ciclo de vida del chaincode**, y ejecutar su funcionalidad para **consultar** (operación **lectura** del libro mayor) o modificar (**escritura**) en el **libro mayor**. Para este proceso suponemos que tenemos ya programado un chaincode.

Para realizar el empaquetado e instalación del chaincode en el peer de la organización **org1**, utilizaremos el script **10a_chaincode_alta_org1.sh**.

```

10a_chaincode_alta_org1.sh
1  # Muestra del parametro $1
2  # $1=../chcodes/hf1/build/install/hf1
3
4  # Ruta chaincode code compiled
5  ruta_install=../chcodes/chaincode/build/install/chaincode
6
7  source ./term-org1
8
9  peer lifecycle chaincode package chaincode.tar.gz \
10   |--path $ruta_install \
11   |--lang java \
12   |--label chaincode$1
13
14  peer lifecycle chaincode install chaincode.tar.gz \
15   |--peerAddresses $CORE_PEER_ADDRESS_ORG1 \
16   |--tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE
17
18  peer lifecycle chaincode queryinstalled \
19   |--peerAddresses $CORE_PEER_ADDRESS_ORG1 \
20   |--tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE
21

```

Figura 25: Código fuente del script **10a_chaincode_alta_org1.sh**

En la figura 25, analizando el código, observamos que:

El script recibe un parámetro de ejecución **\$1**. Este parámetro se utiliza para poder ejecutar el script modificando dinámicamente el nombre del chaincode añadiendo un índice numérico a la palabra

chaincode (ej: chaincode1, chaincode2, ...). Para nuestro caso consideraremos **\$1=1**. El script utiliza la herramienta **peer cli** con el comando **lifecycle chaincode**.

Línea 5: Definimos la variable **ruta_install** asignándole el valor de la ruta donde se encuentra nuestro chaincode (su carpeta de proyecto). En el proyecto **HFLines**, todas las carpetas de chaincodes se encuentran localizadas en la carpeta **chCodes**.

Línea 7: Definimos las variables de Hyperledger Fabric para operar en el canal como la organización **org1** y que están almacenadas en el fichero **term-org1**. El comando **source** ejecuta todas las asignaciones definidas en **term-org1** (se pueden recordar consultando la figura 14).

Líneas 9 a 12: empaquetamos el chaincode localizado en **ruta_install** en un fichero llamado '**chaincode.tar.gz**' y lo etiquetamos en la red **HFLines** con el valor "**chaincode"+\$1**". Suponiendo una ejecución con **\$1=1**, etiquetamos el chaincode como **chaincode1**.

Líneas 14 a 16: instalamos el paquete '**chaincode.tar.gz**' en **peer-org1**.

Líneas 18 a 20: consultamos el resultado del comando de instalación (install).

En la figura 24 vemos la salida de la ejecución de este código en el terminal:

```
esbrinachain@hfabric1:~/fabric-samples/hflines$ ./10a_chaincode_alta_org1.sh
2024-06-18 10:50:31.648 UTC 0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:<status:200 payload:"\nKchaincode1:e64e4222158d5aa7f036416166e2fbce5baa32f1115af0861bba128817e0b3d6\022nchaincode1" >
2024-06-18 10:50:31.649 UTC 0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: chaincode1:e64e4222158d5aa7f036416166e2fbce5baa32f1115af0861bba128817e0b3d6
Installed chaincodes on peer:
Package ID: chaincode1:e64e4222158d5aa7f036416166e2fbce5baa32f1115af0861bba128817e0b3d6, Label: chaincode1
esbrinachain@hfabric1:~/fabric-samples/hflines$
```

Figura 26: Salida parcial del script **10a_chaincode_alta_org1.sh**

La petición de instalación ha funcionado bien: **submitInstallProposal -> Installed remotely**.

La consulta de los chaincodes instalados en peer1-org1 nos responde el package ID de nuestro chaincode con etiqueta (label) **chaincode1**.

El parámetro indicado en el terminal **Package ID** lo utilizaremos como parámetro de entrada en los próximos scripts a ejecutar, y deberemos copiarlo/pegarlo en la terminal, en nuestro ejemplo es:

chaincode1:e64e4222158d5aa7f036416166e2fbce5baa32f1115af0861bba128817e0b3d6

y la ejecución de nuestro próximo script será:

./10b_chaincode_alta_org1.sh 1 chaincode1:e64e4222158d5aa7f036416166e2fbce5baa32f1115af0861bba128817e0b3d6

Siguiendo con el procedimiento de ejecutar el ciclo de vida del chaincode chaincode1, necesitamos **aprobarlo** una vez instalado en el peer de **org1**. Este paso lo automatiza el script **10b_chaincode_alta_org1.sh**.

Una vez más, analizamos su código fuente y su salida por el terminal una vez ejecutado (figuras 27 y 28).

```

10b_chaincode_alta_org1.sh
1 source ./term-org1
2
3 peer lifecycle chaincode approveformyorg \
4   -o $ORDERER_ORG1_ADDRESS \
5   --ordererTLSHostnameOverride orderer1-org1 \
6   --tls --cafile $ORDERER_CA \
7   --channelID mychannel \
8   --name chaincode$1 \
9   --version 1.0 \
10  --package-id $2 \
11  --sequence 1

```

Figura 27: Código fuente del script 10b_chaincode_alta_org1.sh

El script mantiene los parámetros que arrastramos durante todo el proceso:

\$1: el índice usado para el etiquetado del chaincode instalado (1, para generar chaincode1).

\$2: el package ID del chaincode:

chaincode1:e64e4222158d5aaaf7036416166e2fbce5baa32f1115af0861bba128817e0b3d6

Como parámetros fijados para facilitar los datos del comando:

1. Indicamos la máquina y puerto de **orderer1-org1** (flag: -o)
2. Nombre TLS del nodo orderer: **orderer-org1** (flag: --ordererTLSHostnameOverride)
3. Su material criptográfico (flag: tls) (flag: --cafile)
4. El nombre del canal (flag: --channelID)
5. Versión del chaincode: 1.0
6. Secuencia del chaincode: 1

En la figura 28 observamos el resultado de su ejecución en el terminal de la orden:

./10b_chaincode_alta_org1.sh 1 chaincode1:e64e4222158d5aaaf7036416166e2fbce5baa32f1115af0861bba128817e0b3d6

```

esbrinachain@hfabric1:~/fabric-samples/hflines$ ./10b_chaincode_alta_org1.sh 1 chaincode1:e64e4222158d5aaaf7036416166e2fbce5baa32f1115af0861bba128817e0b3d6
2024-06-18 20:55:36.318 UTC 0001 INFO [chaincodeCmd] ClientWait -> txid [23dd862b3be85f2237efe3d00dfd3f2d0ae9f1e87ffde43fd35140b7bee2fcac] committed with status (VALID) at localhos
t:7051
esbrinachain@hfabric1:~/fabric-samples/hflines$
```

Figura 28: Salida en terminal del script 10b_chaincode_alta_org1.sh

Observamos de la figura 28 que:

El chaincode1 queda respaldado en el canal (operación de **commit**), con estado **VALIDO** en el **peer1-org1**. Y se nos proporciona la **tx** de la cadena de bloques del canal que ha realizado la operación. Si nos interesaría conocer más sobre los procesos, siempre podemos ejecutar la instrucción:

docker logs id_contenedor_docker

Que nos muestra todo lo que vamos ejecutando en cada contenedor. En nuestro caso le indicaríamos el ID del contenedor **peer-org1**.

Hasta ahora hemos instalado el **chaincode** en el peer de la organización **org1 (peer1-org1)**, vamos a hacer lo mismo en la organización **org2 (peer1-org2)**. El proceso es idéntico pero usando las variables de entorno para describir a la organización **org2**.

Mostramos el código fuente y el resultado de su salida por el terminal:

```

11_chaincode_alta_org2.sh
1  # La última ejecución de peer en '10_chaincode_alta_org1.sh', genera el parametro $1
2  # (ejemplo que varia por ser id de chaincode generado)
3  # ejemplo exacto en formato:
4  #          chaincode1:8029661264f62c396e5c8086ff41d46ae9704cdeace463e1alaebbf3df4c39e6
5  # Si no se pasa este parámetro se genera un error.
6
7  source ./term-org2
8
9  peer lifecycle chaincode install chaincode.tar.gz \
10 |   --peerAddresses $CORE_PEER_ADDRESS \
11 |   --tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE
12
13  peer lifecycle chaincode queryinstalled --peerAddresses $CORE_PEER_ADDRESS \
14 |   --tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE
15
16  peer lifecycle chaincode approveformyorg \
17 |   -o $ORDERER_ORG2_ADDRESS \
18 |   --ordererTLSHostnameOverride orderer1-org2 \
19 |   --tls --cafile $ORDERER_CA \
20 |   --channelID mychannel \
21 |   --name chaincode$1 \
22 |   --version 1.0 \
23 |   --package-id $2 \
24 |   --sequence 1

```

Figura 29: Código fuente del script **11_chaincode_alta_org2.sh**

La instrucción de la línea 7, ‘source ./term-org2’ es la que establece las variables de entorno a los valores correctos para describir a la organización **org2**. El resto del script es igual al caso de la organización **org1**.

```

esbrinachain@hfabric1:~/fabric-samples/hflines$ ./11_chaincode_alta_org2.sh 1 chaincode1:e64e4222158d5aaf7036416166e2fbce5baa32f1115af0861bba128817e0b3d6
2024-06-18 21:12:01.378 UTC 0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:<status:200 payload:"\nKchaincode1:e64e4222158d5aaf7036416166e2fbce5baa32f1115af0861bba128817e0b3d6>
2024-06-18 21:12:01.378 UTC 0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: chaincode1:e64e4222158d5aaf7036416166e2fbce5baa32f1115af0861bba128817e0b3d6
Installed chaincodes on peer:
Package ID: chaincode1:e64e4222158d5aaf7036416166e2fbce5baa32f1115af0861bba128817e0b3d6, Label: chaincode1
2024-06-18 21:12:03.625 UTC 0001 INFO [chaincodeCmd] ClientWait -> txid [e4031edb9946b06f22092e4ed357cc25ef798df30cb99926dc1ec7932ea8cff] committed with status (VALID) at localhost:9051
esbrinachain@hfabric1:~/fabric-samples/hflines$ 

```

Figura 30: Salida en terminal del script **11_chaincode_alta_org2.sh**

Observamos de la figura 30 que:

El chaincode1 pasa por el mismo ciclo de vida (instalación, aprobación y commit), pero ahora en la organización **org2** en el **peer1-org2 (localhost:9051)**. Finalmente queda aprobado en el canal (operación de **approveformyorg**), con estado **VALIDO** en el **peer1-org2**. Y se nos proporciona la tx de la cadena de bloques del canal que ha realizado la operación.

El último paso para completar el ciclo de vida del chaincode consiste en realizar un commit del mismo.

Mientras que hasta ahora hemos instalado y validado en los peers del canal el chaincode, nos queda **instanciar el chaincode en el canal** (operación de **commit**) para que pueda ser ejecutado por los posibles usuarios de las aplicaciones conectadas al canal. Con esta operación podemos afirmar que hemos desplegado (en inglés **deploy**) el chaincode en el canal utilizado.

Esta parte del proceso queda automatizada en el script **12_chaincode_aprove_commit.sh** (figura 31).

```

12_chaincode_aprove_commit.sh
1  # Parametro $1
2  # chaincode:711f71f0a7d3fa0262aa2c3572e92bec4b4994aa7b190a38eb56cc83f493963
3
4  source ./term-org1
5
6  peer lifecycle chaincode checkcommitreadiness \
7      --channelID mychannel \
8      --name chaincode$1 \
9      --version 1.0 \
10     --sequence 1 \
11     --output json
12
13  peer lifecycle chaincode commit -o $ORDERER_ORG1_ADDRESS \
14      --ordererTLSHostnameOverride orderer1-org1 \
15      --tls $CORE_PEER_TLS_ENABLED \
16      --cafile $ORDERER_CA \
17      --channelID mychannel \
18      --name chaincode$1 \
19      --peerAddresses $CORE_PEER_ADDRESS_ORG1 \
20      --tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE_ORG1 \
21      --peerAddresses $CORE_PEER_ADDRESS_ORG2 \
22      --tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE_ORG2 \
23      --version 1.0 \
24      --sequence 1
25
26  peer lifecycle chaincode querycommitted --channelID mychannel --name chaincode$1

```

Figura 31: Código fuente del script 12_chaincode_aprove_commit.sh

En el código fuente del script **12_chaincode_aprove_commit.sh** podemos observar:

Línea 4: Activamos los valores de las variables de entorno a los valores de la organización **org1**, que es la organización que instanciará el chaincode en el canal.

Líneas 6 a 11: Comprobamos la factibilidad de poder realizar el **commit** del chaincode.

Líneas 13 a 24: Se realiza el commit del chaincode y el chaincode queda desplegado.

Línea 26: Comprobamos el resultado de la operación anterior, consultando los chaincodes desplegados en el canal.

En la figura 32, visualizamos la salida en el terminal al ejecutar el script **12_chaincode_aprove_commit.sh**

```

esbrinachain@hfabric1:~/fabric-samples/hflines$ ./12_chaincode_aprove_commit.sh
{
    "approvals": [
        "org1MSP": true,
        "org2MSP": true
    ]
}
2024-06-18 22:05:36.084 UTC 0001 INFO [chaincodeCmd] ClientWait -> txid [0e3297ca3951649ce56d364e427524f402df2de1007f0a8318563dbc271f0e10] committed with status (VALID) at localhos
t:7051
2024-06-18 22:05:36.125 UTC 0002 INFO [chaincodeCmd] ClientWait -> txid [0e3297ca3951649ce56d364e427524f402df2de1007f0a8318563dbc271f0e10] committed with status (VALID) at localhos
t:9051
Committed chaincode definition for chaincode 'chaincode1' on channel 'mychannel':
Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [org1MSP: true, org2MSP: true]
esbrinachain@hfabric1:~/fabric-samples/hflines$ 

```

Figura 32: Salida en terminal del script 12_chaincode_aprove_commit.sh

Al ejecutar

```

peer lifecycle chaincode checkcommitreadiness \
--channelID mychannel \
--name chaincode$1 \
--version 1.0 \
--sequence 1 \
--output json

```

obtenemos que tanto org1MSP como org2MSP han aprobado el chaincode (“approvals”) y por tanto se puede realizar el commit.

De la ejecución de

```
peer lifecycle chaincode commit -o $ORDERER_ORG1_ADDRESS \
    --ordererTLSHostnameOverride orderer1-org1 \
    --tls $CORE_PEER_TLS_ENABLED \
    --cafile $ORDERER_CA \
    --channelID mychannel \
    --name chaincode$1 \
    --peerAddresses $CORE_PEER_ADDRESS_ORG1 \
    --tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE_ORG1 \
    --peerAddresses $CORE_PEER_ADDRESS_ORG2 \
    --tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE_ORG2 \
    --version 1.0 \
    --sequence 1
```

Obtenemos los dos '**commits**' tanto para el **peer1-org1**, como **peer1-org2**, y las transacciones donde ha quedado registrado. Finalmente al consultar que chaincodes están en estado de '**commit**' en el canal,

```
peer lifecycle chaincode querycommitted --channelID mychannel --name chaincode$1
```

en la salida se visualiza el mensaje de que tanto **org1** como **org2** ahora tienen el código del chancode1 desplegado en el canal **mychannel** en su versión 1.0 y su secuencia 1.

Committed chaincode definition for chaincode 'chaincode1' on channel 'mychannel':

Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [org1MSP: true, org2MSP: true]

En este momento ya se puede afirmar que el **chaincode** está instanciado al canal **mychannel** y lo podemos usar.

Para ejecutar el chaincode desde el terminal utilizaremos el comando **chaincode invoke** de la herramienta **peer cli**. Antes de hacerlo vamos a describir en que consiste el chaincode '**chaincode1**', y luego con detalle ejecutaremos su funcionalidad.

09. IMPLEMENTACIÓN Y EJECUCIÓN DE UN CHAINCODE EN HFLINES

Para describir como implementar y ejecutar un chaincode en la red **HFLines**, necesitaremos tomar decisiones previas:

1. Framework de desarrollo.
2. Lenguaje de alto nivel usado.

Para la red **HFLines** se ha utilizado **IntelliJ IDEA Community Edition** y **Java**. Todos los ejemplos se mostrarán en este contexto de aplicaciones.

En **HFLines** se pretende implementar una red **Hyperledger Fabric** que dote a sus usuarios de un sistema de mensajería auditado por blockchain que permita a las dos organizaciones tener un registro de todas las decisiones propuestas, realizadas o no y operaciones ejecutadas de manera que ninguna de las empresas sea la que controle el registro. La cadena de bloques nos permite disponer de un registro transparente de las transacciones realizadas, al mismo tiempo que da opción a la definición de canales privados de comunicación, en dónde solo los usuarios habilitados tendrán acceso a la información.

El chaincode diseñado se compone de un activo '**Mensaje**' y de un conjunto de operaciones realizadas con ese activo: **enviarMensaje**, **verMensaje**, **listarMensajes**, **eliminarMensaje**, **enviadoA**, **enviadoPor**, **buscarEnAsunto**. Cada vez que se usa la funcionalidad del chaincode accedemos al 'world

state' del canal ejecutando la operación requerida y la cadena de bloques registra todas las transacciones realizadas en la operación.

Spring Boot nos permite programar en Java y nos facilita parte de las tareas a realizar mediante código para ejecutar la aplicación. Nosotros sólo nos concentraremos en definir activos, e implementar su funcionalidad.

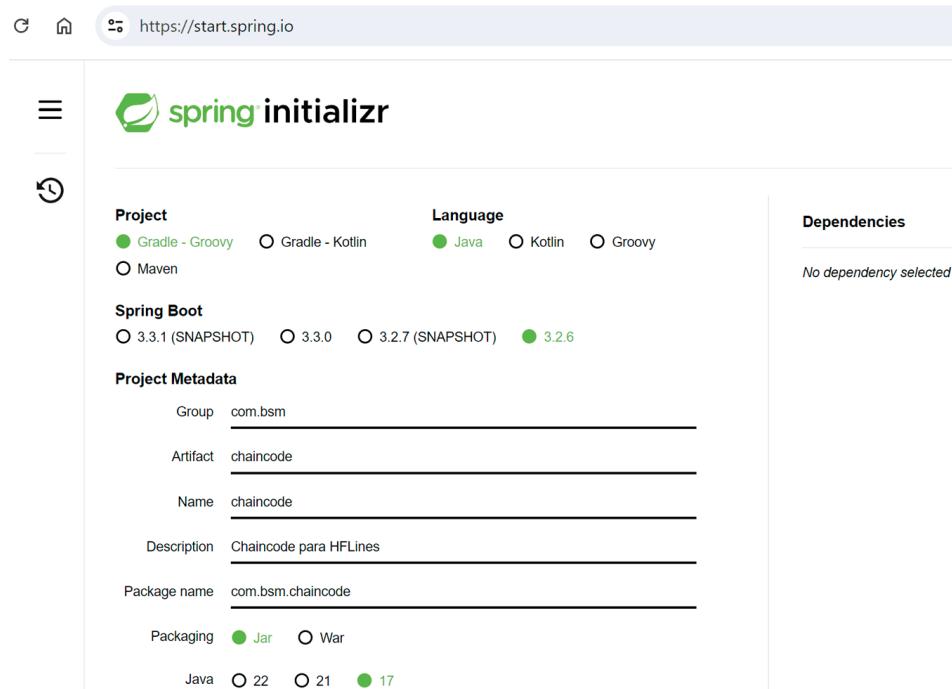


Figura 33: Website de Spring Boot para la generación rápida de una Java application

Como se puede ver en la figura 33, se ha elegido Gradle-Groovy como controlador de dependencias y versiones pero Maven sería igualmente válido. En la figura 33, se muestra marcada la versión 17 de Java, pero la documentación de Hyperledger Fabric recomienda por compatibilidad general usar la versión 11 del JDK.

Sobre todo para el desarrollo de chaincodes se debe configurar en la estructura del proyecto el uso de un JDK 11 (a no ser que el usuario avanzado consiga actualizar la versión JDK 11 del contenedor levantado por cada **peer** y que da el servicio de ejecución de chaincodes, a una versión superior sin problemas).

Para que el objeto de la clase **Mensaje y su funcionalidad** implementada en el chaincode sea considerada un **contrato** y pueda ejecutarse como un **Smart Contract** en **Hyperledger Fabric** usaremos en el proyecto **Java** de **Spring Boot**, las librerías facilitadas para la implementación de **chaincodes**.

```
// https://mvnrepository.com/artifact/org.hyperledger.fabric-chaincode-java/fabric-chaincode-shim
implementation 'org.hyperledger.fabric-chaincode-java:fabric-chaincode-shim:2.5.0'
```

En la figura 34, podemos observar todas las dependencias usadas en el proyecto de java 'Chaincode' descriptas en el fichero **build.gradle**.

Destacamos el uso de **com.owlike:genson:1.5** para la conversión de string a notación JSON en las dos direcciones.

Otro aspecto importante, es el uso en el apartado de **plug-ins** del **plug-in id ‘application’** que nos permite usar **gradle** y compilar el proyecto sin necesidad de tener **gradle** instalado.

```

1 plugins {
2     id 'java'
3     id 'org.springframework.boot' version '2.7.17'
4     id 'io.spring.dependency-management' version '1.0.15.RELEASE'
5     id 'application'
6 }
7
8 group = 'com.bsm'
9 version = '0.0.1-SNAPSHOT'
10
11 java {
12     sourceCompatibility = '11'
13 }
14
15 repositories {
16     mavenCentral()
17     maven {
18         url 'https://jitpack.io'
19     }
20 }
21
22
23 dependencies {
24     implementation 'org.springframework.boot:spring-boot-starter'
25     testImplementation 'org.springframework.boot:spring-boot-starter-test'
26     // https://mvnrepository.com/artifact/org.hyperledger.fabric-chaincode-java/fabric-chaincode-shim
27     implementation 'org.hyperledger.fabric-chaincode-java:fabric-chaincode-shim:2.5.0'
28     // https://mvnrepository.com/artifact/com.owllike/genson
29     implementation 'com.owllike:genson:1.5'
30     //implementation 'com.google.code.gson:gson:2.11.0'
31 }
32
33 tasks.named('test') {
34     useJUnitPlatform()
35 }

```

Figura 34: Contenido del fichero **build.gradle**

A continuación mostramos el código fuente de los ficheros **Mensaje.java**, **MensajeTransfer.java**, que contienen la programación de la clase Mensaje para poder instanciar objetos de tipo Mensaje y la clase **MensajeTransfer**, que describe de que funcionalidad dotamos a los mensajes enviados.

En la figura 35 se puede observar el código java de la clase **Mensaje.java** y en la figura 36 el código fuente del fichero **MensajeTransfer.java**.

```

1 package com.bsm.chaincode;
2
3 import com.owllike.genson.annotation.JsonProperty;
4 import org.hyperledger.fabric.contract.annotation.DataType;
5 import org.hyperledger.fabric.contract.annotation.Property;
6
7 import java.util.List;
8 import java.util.Objects;
9
10 @DataType() 22 usages
11 public final class Mensaje {
12
13     @Property() 3 usages
14     private final String id;
15     @Property() 3 usages
16     private final String emisor;
17     @Property() 3 usages
18     private final String asunto;
19     @Property() 3 usages
20     private final String receptor;
21     @Property() 3 usages
22     private final String texto;
23     @Property() 3 usages
24     private final String enCopia;
25
26     public Mensaje(@JsonProperty("id") final String id, 1usage
27                     @JsonProperty("emisor") final String emisor,
28                     @JsonProperty("asunto") final String asunto,
29                     @JsonProperty("receptor") final String receptor,
30                     @JsonProperty("texto") final String texto,
31                     @JsonProperty("enCopia") final String enCopia) {
32         this.id = id;
33         this.emisor = emisor;
34         this.asunto = asunto;
35         this.receptor = receptor;
36         this.texto = texto;
37         this.enCopia = enCopia;
38     }
39
40     public String getId() { 4 usages
41         return id;
42     }
43
44     public String getEmisor() { 4 usages
45         return emisor;
46     }
47
48     public StringgetAsunto() { 4 usages
49         return asunto;
50     }
51
52     public String getReceptor() { 4 usages
53         return receptor;
54     }
55
56     public String getTexto() { 3 usages
57         return texto;
58     }
59
60     public String getEnCopia() { 3 usages
61         return enCopia;
62     }
63
64     @Override
65     public boolean equals(final Object obj) {
66         if (this == obj) {
67             return true;
68         }
69
70         if ((obj == null) || (getClass() != obj.getClass())) {
71             return false;
72         }
73
74         Mensaje other = (Mensaje) obj;
75
76         return Objects.deepEquals(
77             new String[] {getId(), getEmisor(), getAsunto(), getReceptor(), getTexto(), getEnCopia()},
78             new String[] {other.getId(), other.getEmisor(), other.getAsunto(), other.getReceptor(), other.getTexto(), other.getEnCopia()});
79     }
80
81     @Override
82     public int hashCode() {
83         return Objects.hash(getId(), getEmisor(), getAsunto(), getReceptor(), getTexto(), getEnCopia());
84     }
85
86     @Override
87     public String toString() {
88         return this.getClass().getSimpleName() + "@" + Integer.toHexString(hashCode()) + " [id=" + id + ", emisor="
89                         + emisor + ", asunto=" + asunto + ", receptor=" + receptor + ", texto=" + texto + ", enCopia=" + enCopia + "]";
90     }
91 }

```

Figura 35: Contenido del fichero Mensaje.java

```

1 MensajeTransfer.java 2 Mensaje.java 3 ChaincodeApplication.java
1 package com.bsm.chaincode;
2
3 import com.owllike.genson.Genson;
4 import org.hyperledger.fabric.contract.Context;
5 import org.hyperledger.fabric.contract.ContractInterface;
6 import org.hyperledger.fabric.contract.annotation.Contract;
7 import org.hyperledger.fabric.contract.annotation.Default;
8 import org.hyperledger.fabric.contract.annotation.Info;
9 import org.hyperledger.fabric.contract.annotation.Transaction;
10 import org.hyperledger.fabric.shim.ChaincodeException;
11 import org.hyperledger.fabric.shim.ChaincodeStub;
12 import org.hyperledger.fabric.shim.ledger.KeyValue;
13 import org.hyperledger.fabric.shim.ledger.QueryResultsIterator;
14
15 import java.util.ArrayList;
16 import java.util.List;
17
18 @Contract( no usages
19     name = "MensajeTransfer",
20     info = @Info(
21         title = "MensajeTransfer contract",
22         description = "Chaincode para la transacción de Mensajes.",
23         version = "0.0.1"))
24 @Default
25 public final class MensajeTransfer implements ContractInterface {
26
27     private final Genson genson = new Genson(); 12 usages
28
29     private enum MensajeTransferErrors { 3 usages
30         MENSAGE_NOT_FOUND, 2 usages
31         MENSAGE_ALREADY_EXISTS 1 usage
32     }
33
34     @Transaction(intent = Transaction.TYPE.SUBMIT)  no usages
35     public Mensaje enviarMensaje(@final Context ctx, @final String id, @final String emisor,
36                                 @final String asunto, @final String receptor, @final String texto,
37                                 @final String enCopia) {
38
39         ChaincodeStub stub = ctx.getStub();
40
41         String state = stub.getStringState(id);
42
43         if (!state.isEmpty()) {
44             String errorMessage = String.format("Mensaje %s ya registrado", id);
45             System.out.println(errorMessage);
46             throw new ChaincodeException(errorMessage, MensajeTransferErrors.MENSAJE_ALREADY_EXISTS.toString());
47         }
48
49         Mensaje mensaje = new Mensaje(id, emisor, asunto, receptor, texto, enCopia);
50
51         String newState = genson.serialize(mensaje);
52
53         stub.putStringState(id, newState);
54         return mensaje;
55     }
56
57     @Transaction(intent = Transaction.TYPE.EVALUATE)  no usages
58     public Mensaje verMensaje(@final Context ctx, @final String id) {
59         ChaincodeStub stub = ctx.getStub();
60         String state = stub.getStringState(id);
61
62         if (state.isEmpty() || state == null) {
63             String errorMessage = String.format("Mensaje %s no existe", id);
64             System.out.println(errorMessage);
65             throw new ChaincodeException(errorMessage, MensajeTransferErrors.MENSAJE_NOT_FOUND.toString());
66         }
67
68         Mensaje mensaje = genson.deserialize(state, Mensaje.class);
69         return mensaje;
70     }
71
72     @Transaction(intent = Transaction.TYPE.SUBMIT)  no usages
73     public void eliminarMensaje(@final Context ctx, @final String id) {
74         ChaincodeStub stub = ctx.getStub();
75
76         String state = stub.getStringState(id);
77
78         if (state.isEmpty() || state == null) {
79             String errorMessage = String.format("Mensaje %s no existe", id);
80             System.out.println(errorMessage);
81             throw new ChaincodeException(errorMessage, MensajeTransferErrors.MENSAJE_NOT_FOUND.toString());
82         }
83
84         stub.delState(id);
85     }
86
87     @Transaction(intent = Transaction.TYPE.EVALUATE)  no usages
88     public String listarMensajes(@final Context ctx) {
89         ChaincodeStub stub = ctx.getStub();
90
91         List<Mensaje> queryResults = new ArrayList<>();
92         QueryResultsIterator<KeyValue> results = stub.getStateByRange("", "");
93
94         for (KeyValue result: results) {
95             Mensaje asset = genson.deserialize(result.getStringValue(), Mensaje.class);
96             System.out.println(asset);
97             queryResults.add(asset);
98         }
99
100        final String response = genson.serialize(queryResults);
101
102        return response;
103    }
104
105    @Transaction(intent = Transaction.TYPE.EVALUATE)  no usages
106    public String enviadosPor(@final Context ctx, String emisor) {
107        ChaincodeStub stub = ctx.getStub();
108
109        List<Mensaje> queryResults = new ArrayList<>();
110        QueryResultsIterator<KeyValue> results = stub.getStateByRange("", "");
111
112        for (KeyValue result: results) {
113            Mensaje asset = genson.deserialize(result.getStringValue(), Mensaje.class);
114            String emitidoPor=asset.getEmisor();
115            if (emisor.equals(emitidoPor)) {
116                System.out.println(asset);
117                queryResults.add(asset);
118            }
119        }
120
121        final String response = genson.serialize(queryResults);
122        return response;
123    }
124
125    @Transaction(intent = Transaction.TYPE.EVALUATE)  no usages
126    public String enviadosA(@final Context ctx, String receptor) {
127        ChaincodeStub stub = ctx.getStub();
128
129        List<Mensaje> queryResults = new ArrayList<>();
130        QueryResultsIterator<KeyValue> results = stub.getStateByRange("", "");
131
132        for (KeyValue result: results) {
133            Mensaje asset = genson.deserialize(result.getStringValue(), Mensaje.class);
134            String enviadoA = asset.getReceptor();
135            if (receptor.equals(enviadoA)) {
136                System.out.println(asset);
137                queryResults.add(asset);
138            }
139        }
140
141        final String response = genson.serialize(queryResults);
142        return response;
143    }
144
145    @Transaction(intent = Transaction.TYPE.EVALUATE)  no usages
146    public String buscarEnAsunto(@final Context ctx, String txt) {
147        ChaincodeStub stub = ctx.getStub();
148
149        List<Mensaje> queryResults = new ArrayList<>();
150        QueryResultsIterator<KeyValue> results = stub.getStateByRange("", "");
151
152        for (KeyValue result: results) {
153            Mensaje asset = genson.deserialize(result.getStringValue(), Mensaje.class);
154            String textoAsunto = asset.getAsunto();
155            if (textoAsunto.toLowerCase().contains(txt.toLowerCase())) {
156                System.out.println(asset);
157                queryResults.add(asset);
158            }
159        }
160
161        final String response = genson.serialize(queryResults);
162        return response;
163    }
164
165    @Transaction(intent = Transaction.TYPE.SUBMIT)  no usages
166    public String eliminarTodosMensajes(@final Context ctx) {
167
168        ChaincodeStub stub = ctx.getStub();
169
170        List<String> eliminados = new ArrayList<>();
171        QueryResultsIterator<KeyValue> results = stub.getStateByRange("", "");
172
173        for (KeyValue result: results) {
174            Mensaje asset = genson.deserialize(result.getStringValue(), Mensaje.class);
175            String MensajeId = asset.getId();
176            stub.delState(MensajeId);
177            eliminados.add(MensajeId);
178        }
179
180        final String response = genson.serialize(eliminados);
181        return response;
182    }

```

Figura 36: Contenido del fichero MensajeTransfer.java

El proceso de compilación del código Java actualizado se realiza con **gradle**, y se ha automatizado en el script **r.sh**:

```
chcodes > chaincode > r.sh
1 rm -rf build
2 ./gradlew installDist
3 ls -l
4 rm -rf ../../chaincode.tar.gz
5
```

Figura 37: Contenido del script **r.sh**

Una vez ejecutado en el directorio raíz del proyecto creado para el chaincode se genera una nueva versión compilada del chaincode, lista para ser usada por el script **10a_chaincode_alta_org1.sh**, que lo empaqueta antes de instalarlo en los nodos **peer** con una políticas de aprobación (en inglés **endorsement**) que restringen quien puede ejecutarlo. En nuestro caso, en el canal **mychannel** las organizaciones **org1** y **org2** pueden ejecutar el chaincode (como se ha describió al principio del apartado 07).

Queda fuera del alcance de este documento la descripción de las librerías para la programación del acceso y modificación del **libro mayor** del canal (en inglés **ledger**) que utiliza el chaincode para almacenar los mensajes enviados y registrar toda la información de interés para las auditorias. Sirva el código de la figura 36, para averiguar cómo se almacena y recupera la información del ‘world state’ del **ledger**, declarando transacciones de tipo **EVALUATE** cuando no se modifica el **ledger** y de tipo **SUBMIT** cuando se escribe en el ‘world state’. **Hyperledger Fabric** se encarga de la interacción con el **ledger** y nos facilita una variable **Context** que nos permite el uso de toda una funcionalidad existente para tal efecto.

En la figura 38, se puede observar la URL dónde se puede consultar la documentación de esta estructura de clases y obtener un listado de todas las que están disponibles.

The screenshot shows a list of Java classes from the Hyperledger Fabric-Chaincode-shim API. The classes listed include:

- Chaincode
- Chaincode.Response
- Chaincode.Response.Status
- ChaincodeException
- ChaincodeStub
- ClientIdentity
- CompositeKey
- Contact
- Context
- Contract
- ContractInterface
- DataType
- Default
- Info
- KeyModification
- KeyValue
- License
- Logger
- Logging
- Metrics
- MetricsProvider
- Property
- QueryResultsIterator
- QueryResultsIteratorWithMetadata
- ResponseUtils
- Serializer
- Serializer.TARGET
- StateBasedEndorsement
- StateBasedEndorsement.RoleType
- TaskMetricsCollector
- Transaction
- Transaction.TYPE

Figura 38: Conjunto de clases del **API Fabric-Chaincode-shim** para Java.

Ahora que conocemos la estructura y funcionalidad del chaincode **chaincode1**, podemos invocarlo actuando como **org1** o actuando como **org2**.

El proyecto **HFLines** contiene toda una serie de scripts para la ejecución individual de cada funcionalidad del **chancode1** y está implementada en los ficheros de script con prefijo “**20_app0x**”. En total son 7 scripts que conociendo los parámetros de ejecución permiten enviar un mensaje, visualizar un mensaje, eliminar un mensaje, conocer los mensajes enviados por un emisor, o los recibidos por un receptor, buscar un texto específico en un asunto de un mensaje entre otras.

Debido a que conocer los parámetros puede no ser una tarea fácil, se han implementado dos scripts más de nombre **30_ejemplos_uso_20_app0x_org1.sh** y **30_ejemplos_uso_20_app0x_org2.sh** en donde se puede encontrar patrones de ejecución (para cada función implementada en el chaincode), listos para ser ejecutados y visualizar por pantalla los resultados.

Para utilizar estos scripts es requisito indispensable:

1. Tener levantada la red **HFLines**.
2. Tener definido un canal.
3. Tener el chaincode instalado (install) , aprobado (approve) e instanciado al canal (commit).

Veamos cómo usar la herramienta **peer cli** con el comando **chaincode invoke** para ejecutar un chaincode.

Utilizamos el script **20_app01_enviarMensaje.sh** que recibe como parámetro el índice de chaincode usado (en nuestro caso será 1 porque el chaincode está etiquetado como **chaincode1**).

En la figura 39 se puede visualizar el código fuente de la secuencia de scripts que se ejecutan y el resultado obtenido en el terminal.

La figura 39 se compone de tres partes, en la parte superior, se puede observar el código fuente del script **20_app01_enviarMensaje.sh**. En la segunda parte, se puede observar la ejecución de este script en el script **30_ejemplos_uso_20_app0x_org1.sh** (que aquí se muestra parcialmente sólo en el caso de uso que utiliza la función **enviarMensaje** del chaincode **chaincode1**) y finalmente en la tercera parte, la salida obtenida en el terminal que nos confirma que el chaincode se ha invocado satisfactoriamente, retornando el mensaje incluido en el **ledger** del canal **mychannel**:

(Mensaje de Salida por el terminal)

```
2024-06-19 20:50:37.416 UTC 0001
INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful.
result: status:200
payload:"{"texto":"Reunion Lunes 22/09/2024 a las 11h, oficinas TELCO","receptor":"Maria Castillo","id":"1","enCopia":"Antonio Maria Rebeca","asunto":"Reunion urgente","emisor":"Juan Solana"}"
```

En este momento, si queremos consultar el ledger del canal para ver este mensaje, podemos utilizar la función “**verMensaje**” del chaincode1 si disponemos de la información de su **id**. O bien podríamos utilizar la función “**enviadosPor**” si sabemos quién ha enviado el mensaje, o bien la función “**enviadosA**” si conocemos el destinatario, o finalmente la función “**listarMensajes**” si no tenemos ninguna información, y tendríamos que buscarlo entre toda la lista facilitada.

Conociendo la metodología y sus parámetros se puede usar los scripts de prefijo ‘**20_app0x**’ directamente. Pero mientras no tenemos esa práctica, se recomienda ver como se pueden ejecutar consultando los ficheros de ejemplo con prefijo ‘**30_ejemplos**’.

```

20_app01_enviarMensaje.sh
1 # Muestra para la ejecución de funciones del chaincode.
2 # App01: ./20_app_enviar_mensaje.sh $1 $2 $3 $4 $5 $6 $7
3
4 # enviarMensaje
5 #
6 #           idMensaje $2="2"
7 #           Emisor $3="Emisor"
8 #           Asunto $4="Asunto"
9 #           Destinatario $5="destinatario",
10 #           Contenido $6="Contenido"
11 #           enCopia $7="enCopias"
12
13 . color.sh
14 source ./term-org1
15
16 echo -e $WHITE_L Enviando de Mensaje por $BLUE HFLINES $NORMAL
17 echo ""
18
19 printf -v param1 '{"Args": ["enviarMensaje", "%s", "%s", "%s", "%s", "%s", "%s"]}' "$2" "$3" "$4" "$5" "$6" "$7"
20 #echo $param1
21
22 op="peer chaincode invoke \
23   -o $ORDERER_ORG1_ADDRESS \
24   --ordererTLSHostnameOverride orderer1-org1 \
25   --tls $CORE_PEER_TLS_ENABLED \
26   --cafile $ORDERER_CA \
27   -C mychannel \
28   -n chaincode$1 \
29   --peerAddresses $CORE_PEER_ADDRESS_ORG1 \
30   --tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE_ORG1 \
31   --peerAddresses $CORE_PEER_ADDRESS_ORG2 \
32   --tlsRootCertFiles $CORE_PEER_TLS_ROOTCERT_FILE_ORG2 \
33   -c '${param1}'
34
35 op="${op}''"
36 #printf -v op $op
37
38 #echo $op
39
40 echo $op | bash
41 echo ""
42

```



```

30_ejemplos_uso_20_app0x_org1.sh
1
2 # Ejemplos de ejecución de las app que facilitan el uso
3 # de las funciones incluidas en el chaincode.
4
5 clear
6 . color.sh
7 source ./term-org1
8
9 echo -e $WHITE_L Ejemplo 1 sobre HFLINES $NORMAL
10 echo ""
11 echo -e $WHITE_L Envio Mensaje 1 : Emisor: Juan Solana, destinataria: Maria Castillo $BLUE RED HFLINES $NORMAL
12 ./20_app01_enviarMensaje.sh $1 "1" "Juan Solana" "Reunion urgente" "Maria Castillo" "Reunion Lunes 22/09/2024 a las 11h, oficinas TELCO" "Antonio Maria Rebeca"
13
14 sleep 3

```



Ejemplo 1 sobre HFLINES

```

Envio Mensaje 1 : Emisor: Juan Solana, destinataria: Maria Castillo RED HFLINES
Envio de Mensaje por HFLINES

```

```

2024-06-19 20:50:37.416 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"{\\"texto\\":\\"Reunion Lunes 22/09/2024 a las 11h, oficinas TELCO\\",\\"receptor\\":\\"Maria Castillo\\",\\"id\\":\\"1\\",\\"enCopia\\":\\"Antonio Maria Rebeca\\",\\"asunto\\":\\"Reunion urgente\\",\\"emisor\\":\\"Juan Solana\\"}"
esbrinachain@hfabric1:~/fabric-samples/hflines$
```

Figura 39: Ejemplo de ejecución para enviar un mensaje en HFLINES.

Los scripts **30_ejemplos_uso_20_app0x_org1.sh** y **30_ejemplos_uso_20_app0x_org2.sh** son equivalentes y nos permiten comprobar que el chaincode1 se puede ejecutar desde las dos organizaciones **org1** y **org2**.

En la figura 40, se muestra la ejecución completa del script **30_ejemplos_uso_20_app0x_org1.sh**, en donde se prueban todas las funciones implementadas en el **chaincode1**.

```

SSH en el navegador
SUBIR ARCHIVO DESCARGAR ARCHIVO
Ejemplo 1 sobre HFLINES
Envio Mensaje 1 : Emisor: Juan Solana, destinataria: Maria Castillo RED HFLINES
Envio de Mensaje por HFLINES
2024-06-19 21:39:41.812 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"{\"texto\":\"Reunion Lunes 22/09/2024 a las 11h, oficinas TELCO\",\"receptor\":\"Maria Castillo\",\"id\":\"1\",\"enCopia\":\"Antonio Maria Rebeca\",\"asunto\":\"Reunion urgente\",\"emisor\":\"Juan Solana\"}"
Ejemplo 2 Envio de tres mensajes por HFLINES
Envio Mensaje 2 : Emisor: Maria Castillo, destinatario: Juan Solana RED HFLINES
Envio de Mensaje por HFLINES
2024-06-19 21:39:44.912 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"{\"texto\":\"Reunion Lunes - OK. Nos desplazamos 4 personas\",\"receptor\":\"Juan Solana\",\"id\":\"2\",\"enCopia\":\"antonio maria rebeca\",\"asunto\":\"Reunion aceptada\",\"emisor\":\"Maria Castillo\"}"
Envio Mensaje 3 : Emisor: Maria Castillo, destinatario: Anna Gracia RED HFLINES
Envio de Mensaje por HFLINES
2024-06-19 21:39:45.009 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"{\"texto\":\"Deberias asistir a la reunion del lunes, oficinas TELCO a las 11h.\",\"receptor\":\"Anna Gracia\",\"id\":\"3\",\"enCopia\":\"Antonio Maria Rebeca\",\"asunto\":\"Reunion Lunes TELCO\",\"emisor\":\"Maria Castillo\"}"
Envio Mensaje 4 : Emisor: Maria Castillo, destinatario: Anna Gracia RED HFLINES
Envio de Mensaje por HFLINES
2024-06-19 21:39:45.109 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"{\"texto\":\"Finalmente, no podre asistir a la reunion del lunes. Por favor, encargate tu de substituirme.\",\"receptor\":\"Anna Gracia\",\"id\":\"4\",\"enCopia\":\"Antonio Maria Rebeca\",\"asunto\":\"Reunion de ultima hora...\",\"emisor\":\"Maria Castillo\"}"
Ejemplo 3 - Ver todos los mensajes en HFLINES
Listar todos Mensajes en HFLINES
2024-06-19 21:39:48.258 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"[{\\"asunto\":\"Reunion urgente\",\"emisor\":\"Juan Solana\",\"enCopia\":\"Antonio Maria Rebeca\",\"id\":\"1\",\"receptor\":\"Maria Castillo\",\"texto\":\"Reunion Lunes 22/09/2024 a las 11h, oficinas TELCO\"}, {\\"asunto\":\"Reunion Lunes - OK. Nos desplazamos 4 personas\",\"receptor\":\"Juan Solana\",\"enCopia\":\"antonio maria rebeca\",\"id\":\"2\",\"receptor\":\"Maria Castillo\",\"texto\":\"Reunion Lunes - OK. Nos desplazamos 4 pers onas\"}, {\\"asunto\":\"Reunion Lunes TELCO\",\"emisor\":\"Maria Castillo\",\"enCopia\":\"Antonio Maria Rebeca\",\"id\":\"3\",\"receptor\":\"Anna Gracia\",\"texto\":\"Deberias asistir a la reunion del lunes, oficinas TELCO a las 11h.\",\"receptor\":\"Anna Gracia\",\"enCopia\":\"Antonio Maria Rebeca\",\"id\":\"4\",\"receptor\":\"Anna Gracia\"}]"
Ejemplo 4 - Eliminar mensaje - World state, no del ledger-blockchain - sobre HFLINES
Eliminando Mensaje 1 : Emisor: Juan Solana, destinataria: Maria Castillo RED HFLINES
Eliminando Mensaje en HFLINES
2024-06-19 21:39:51.357 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200
receptor:\"Anna Gracia\", \"texto\":\"Finalmente, no podre asistir a la reunion del lunes. Por favor, encargate tu de substituirme.\")"
Ejemplo 4 - Eliminar mensaje - World state, no del ledger-blockchain - sobre HFLINES
Eliminando Mensaje 1 : Emisor: Juan Solana, destinataria: Maria Castillo RED HFLINES
Eliminando Mensaje en HFLINES
2024-06-19 21:39:51.357 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200
Ejemplo 5 - Ver mensaje por _id_ en HFLINES
Visualizando Mensaje 2 _existente_ : Emisor: Maria Castillo, destinatario: Juan Solana RED HFLINES
Ver Mensaje por [id] en HFLINES
2024-06-19 21:39:54.464 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"{\"texto\":\"Reunion Lunes - OK. Nos desplazamos 4 personas\",\"receptor\":\"Juan Solana\",\"id\":\"2\",\"enCopia\":\"antonio maria rebeca\",\"asunto\":\"Reunion aceptada\",\"emisor\":\"Maria Castillo\"}"
Ejemplo 6 - Intento de _Ver mensaje_ eliminado previamente
Visualizando Mensaje 1 : Emisor: Juan Solana, destinataria: Maria Castillo RED HFLINES
Ver Mensaje por [id] en HFLINES
Error: endorsement failure during invoke. response: status:500 message:"Mensaje 1 no existe" payload:"MENSAJE_NOT_FOUND"
Ejemplo 7 - Dado un emisor, mostrar todos los mensajes
Búsqueda de Mensajes enviados por Maria Castillo en HFLINES
2024-06-19 21:40:00.655 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"[{\\"asunto\":\"Reunion aceptada\",\"emisor\":\"Maria Castillo\",\"enCopia\":\"antonio maria rebeca\",\"id\":\"1\",\"receptor\":\"Juan Solana\",\"texto\":\"Reunion Lunes - OK. Nos desplazamos 4 personas\"}, {\\"asunto\":\"Reunion de ultima hora...\",\"emisor\":\"Maria Castillo\",\"enCopia\":\"Antonio Maria Rebeca\",\"id\":\"4\",\"receptor\":\"Anna Gracia\"}]"
Ejemplo 8 - Dado un receptor, mostrar todos los mensajes que se le han enviado
Búsqueda de Mensajes enviados a Anna Gracia en HFLINES
2024-06-19 21:40:03.750 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"[{\\"asunto\":\"Reunion Lunes TELCO\",\"emisor\":\"Maria Castillo\",\"enCopia\":\"Antonio Maria Rebeca\",\"id\":\"3\",\"receptor\":\"Anna Gracia\"}, {\\"asunto\":\"Reunion Lunes - OK. Nos desplazamos 4 personas\"}, {\\"asunto\":\"Reunion de ultima hora...\",\"emisor\":\"Maria Castillo\",\"enCopia\":\"Antonio Maria Rebeca\",\"id\":\"4\",\"receptor\":\"Anna Gracia\"}]"
Ejemplo 9 - Mostrar todos los mensajes con el texto indicado en el _Asunto_ del mensaje
Búsqueda de Mensajes enviados en HFLINES con el texto reunion en el campo ASUNTO
2024-06-19 21:40:06.848 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"[{\\"asunto\":\"Reunion aceptada\",\"emisor\":\"Maria Castillo\",\"enCopia\":\"antonio maria rebeca\",\"id\":\"2\",\"receptor\":\"Juan Solana\",\"receptor\":\"Juan Solana\",\"enCopia\":\"Antonio Maria Rebeca\",\"id\":\"3\",\"receptor\":\"Anna Gracia\",\"enCopia\":\"Antonio Maria Rebeca\",\"id\":\"4\",\"receptor\":\"Anna Gracia\"}]"
esbrinachain@hfabric1:~/fabric-samples/hflines$
```

Figura 40: Ejemplo de ejecución del script **30_ejemplos_uso_20_app0x_org1.sh**

Podemos comprobar que todas las funciones han funcionado bien. Si ejecutamos el script **30_ejemplos_uso_20_app0x_org2.sh** comprobaremos que desde la organización **org2** también se puede ejecutar el chaincode **chaincode1**. El resultado de esta ejecución lo podemos observar en la figura 41:

```

SSH en el navegador
SUBIR ARCHIVO DESCARGAR ARCHIVO
SSH en el navegador
SUBIR ARCHIVO DESCARGAR ARCHIVO

Ejemplo 10 sobre HFLINES

Enviendo Mensaje 5 : Emisor: Julio Solana, destinataria: Maria Garcia RED HFLINES
Enviendo de Mensaje por HFLINES

2024-06-19 21:54:59.327 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"{\"texto\":\"Entrega Martes 23/09/2024 a las 11h, oficinas TELCO.\",\"receptor\":\"Maria Garcia\",\"id\":\"5\",\"enCopia\":\"Antonio Maria Rebeca\",\"asunto\":\"Entrega urgente\",\"emisor\":\"Julio Solana\"}"

Ejemplo 11 Envio de tres mensajes por HFLINES

Enviendo Mensaje 6 : Emisor: Maria Garcia, destinatario: Julio Solana RED HFLINES
Enviendo de Mensaje por HFLINES

2024-06-19 21:55:01.424 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"{\"texto\":\"Entrega Martes - OK. 12 copias.\",\"receptor\":\"Julio Solana\",\"id\":\"6\",\"enCopia\":\"antonio maria rebeca\",\"asunto\":\"Entrega aceptada\",\"emisor\":\"Maria Garcia\"}"

Enviendo Mensaje 7 : Emisor: Maria Garcia, destinatario: Soledad Tuset RED HFLINES
Enviendo de Mensaje por HFLINES

2024-06-19 21:55:01.517 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"{\"texto\":\"Tenemos que realizar la Entrega del proyecto el martes 23/09/2024, oficinas TELCO a las 11h.\",\"receptor\":\"Soledad Tuset\",\"id\":\"7\",\"enCopia\":\"Antonio Maria Rebeca\",\"asunto\":\"Entrega Lunes TELCO\",\"emisor\":\"Maria Garcia\"}"

Enviendo Mensaje 8 : Emisor: Maria Garcia, destinatario: Soledad Tuset RED HFLINES
Enviendo de Mensaje por HFLINES

2024-06-19 21:55:01.615 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"{\"texto\":\"Finalmente, no podre participar de la Entrega del martes. Por favor, encargate tu de todo.\",\"receptor\":\"Soledad Tuset\",\"id\":\"8\",\"enCopia\":\"Antonio Maria Rebeca\",\"asunto\":\"Cambios de ultima hora...\",\"emisor\":\"Maria Garcia\"}"

Ejemplo 12 - Ver todos los mensajes en HFLINES

Listar todos Mensajes en HFLINES

2024-06-19 21:55:04.711 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"[{\\"asunto\":\"Reunion aceptada\",\\\"emisor\":\"Maria Castillo\",\\\"enCopia\":\"antonio maria rebeca\",\\\"id\":\"2\",\\\"receptor\":\"Juan Solana\",\\\"texto\":\"Reunion Lunes - OK. Nos desplazamos 4 personas.\",\\\"asunto\":\"Reuni\\n\\o Lunes TELCO.\",\\\"emisor\":\"Maria Castillo\",\\\"enCopia\":\"Antonio Maria Rebeca\",\\\"id\":\"3\",\\\"receptor\":\"Anna Gracia\",\\\"texto\":\"Deberias asistir a la reunion del lunes, o f\\ificinas TELCO a las 11h.\",\\\"asunto\":\"Reunion de ultima hora...\",\\\"emisor\":\"Maria Castillo\",\\\"enCopia\":\"Antonio Maria Rebeca\",\\\"id\":\"4\",\\\"receptor\":\"Anna Gracia\",\\\"texto\":\"Finalmente, no podre asistir a la reunion del lunes. Por favor, encargate tu de substituirme.\",\\\"asunto\":\"Entrega urgente\",\\\"emisor\":\"Julio Solana\",\\\"enCopia\":\"Antonio Maria Rebeca\",\\\"id\":\"5\",\\\"receptor\":\"Maria Garcia\",\\\"texto\":\"Entrega Martes 23/09/2024 a las 11h, oficinas TELCO.\",\\\"asunto\":\"Entrega aceptada\",\\\"emisor\":\"Maria Garcia\",\\\"enCopia\":\"antonio maria rebeca\",\\\"id\":\"6\",\\\"receptor\":\"Julio Solana\",\\\"texto\":\"Entrega Martes - OK. 12 copias.\",\\\"asunto\":\"Entrega Lunes TELCO\",\\\"emisor\":\"Maria Garcia\",\\\"enCopia\":\"Antonio Maria Rebeca\",\\\"id\":\"7\",\\\"receptor\":\"Soledad Tuset\",\\\"texto\":\"Tenemos que realizar la Entrega del proyecto el martes 23/09/2024, oficinas TELCO a las 11h.\",\\\"asunto\":\"Cambios de ultima hora...\",\\\"emisor\":\"Maria Garcia\",\\\"enCopia\":\"Antonio Maria Rebeca\",\\\"id\":\"8\",\\\"receptor\":\"Soledad Tuset\",\\\"texto\":\"Finalmente, no podre participar de la Entrega del martes. Por favor, encargate tu de todo.\")]"}

Ejemplo 13 - Eliminar mensaje - World state, no del ledger-blockchain - sobre HFLINES

Eliminando Mensaje 5 : Emisor: Julio Solana, destinataria: Maria Garcia RED HFLINES
Eliminando Mensaje en HFLINES

2024-06-19 21:55:07.806 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200

Ejemplo 14 - Ver mensaje por _id_ en HFLINES

Visualizando Mensaje 6 _existente_ : Emisor: Maria Garcia, destinatario: Julio Solana RED HFLINES
Ver Mensaje por [id] en HFLINES

2024-06-19 21:55:10.910 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"{\"texto\":\"Entrega Martes - OK. 12 copias.\",\"receptor\":\"Julio Solana\",\\\"id\":\"6\",\\\"enCopia\":\"antonio maria rebeca\",\\\"asunto\":\"Entrega aceptada\",\\\"emisor\":\"Maria Garcia\"}"

Ejemplo 15 - Intento de _Ver mensaje_ eliminado previamente

Visualizando Mensaje 5 : Emisor: Julio Solana, destinataria: Maria Garcia RED HFLINES
Ver Mensaje por [id] en HFLINES
Error: endorsement failure during invoke. response: status:500 message:"Mensaje 5 no existe" payload:"MENSAJE_NOT_FOUND"

Ejemplo 16 - Dado un emisor, mostrar todos sus mensajes

Búsqueda de Mensajes enviados por Maria Garcia en HFLINES

2024-06-19 21:55:11.096 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"[{\\"asunto\":\"Entrega aceptada\",\\\"emisor\":\"Maria Garcia\",\\\"enCopia\":\"antonio maria rebeca\",\\\"id\":\"6\",\\\"receptor\":\"Julio Solana\",\\\"texto\":\"Entrega Martes - OK. 12 copias.\",\\\"asunto\":\"Entrega Lunes TELCO\",\\\"emisor\":\"Maria Garcia\",\\\"enCopia\":\"Antonio Maria Rebeca\",\\\"id\":\"7\",\\\"receptor\":\"Soledad Tuset\",\\\"texto\":\"Tenemos que realizar la Entrega del proyecto el martes 23/09/2024, oficinas TELCO a las 11h.\",\\\"asunto\":\"Cambios de ultima hora...\",\\\"emisor\":\"Maria Garcia\",\\\"enCopia\":\"Antonio Maria Rebeca\",\\\"id\":\"8\",\\\"receptor\":\"Soledad Tuset\",\\\"texto\":\"Finalmente, no podre participar de la Entrega del martes. Por favor, encargate tu de todo.\")]"}

Ejemplo 17 - Dado un receptor, mostrar todos los mensajes que se le han enviado

Búsqueda de Mensajes enviados a Soledad Tuset en HFLINES

2024-06-19 21:55:12.187 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"[{\\"asunto\":\"Entrega Lunes TELCO\",\\\"emisor\":\"Maria Garcia\",\\\"enCopia\":\"Antonio Maria Rebeca\",\\\"id\":\"7\",\\\"receptor\":\"Soledad Tuset\",\\\"texto\":\"Tenemos que realizar la Entrega del proyecto el martes 23/09/2024, oficinas TELCO a las 11h.\",\\\"asunto\":\"Cambios de ultima hora...\",\\\"emisor\":\"Maria Garcia\",\\\"enCopia\":\"Antonio Maria Rebeca\",\\\"id\":\"8\",\\\"receptor\":\"Soledad Tuset\",\\\"texto\":\"Finalmente, no podre participar de la Entrega del martes. Por favor, encargate tu de todo.\")]"}

Ejemplo 18 - Mostrar todos los mensajes con el texto indicado en el _Asunto_ del mensaje

Búsqueda de Mensajes enviados en HFLINES con el texto Entrega en el campo ASUNTO

2024-06-19 21:55:12.280 UTC 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:"[{\\"asunto\":\"Entrega aceptada\",\\\"emisor\":\"Maria Garcia\",\\\"enCopia\":\"antonio maria rebeca\",\\\"id\":\"6\",\\\"receptor\":\"Julio Solana\",\\\"texto\":\"Entrega Martes - OK. 12 copias.\",\\\"asunto\":\"Entrega Lunes TELCO\",\\\"emisor\":\"Maria Garcia\",\\\"enCopia\":\"Antonio Maria Rebeca\",\\\"id\":\"7\",\\\"receptor\":\"Soledad Tuset\",\\\"texto\":\"Tenemos que realizar la Entrega del proyecto el martes 23/09/2024, oficinas TELCO a las 11h.\",\\\"asunto\":\"Cambios de ultima hora...\",\\\"emisor\":\"Maria Garcia\",\\\"enCopia\":\"Antonio Maria Rebeca\",\\\"id\":\"8\",\\\"receptor\":\"Soledad Tuset\",\\\"texto\":\"Finalmente, no podre participar de la Entrega del martes. Por favor, encargate tu de todo.\")]"}

esbrinachain@hfabric1:~/fabric-samples/hflines$
```

Figura 41: Ejemplo de ejecución del script 30_ejemplos_uso_20_app0x_org2.sh

010. API REST de conexión a la red para clientes de HFLINES

El proyecto **HFLINES** también proporciona un **API REST** de conexión a la red **HFLINES**. El proyecto de desarrollo se encuentra en la carpeta del proyecto **chCodes** con el nombre de **Gateway**.

En la figura 42, se muestra algunos de los diferentes módulos implementados para la ejecución de la funcionalidad del chaincode instanciado en la red: **MensajeriaService.java**, **MensajeController.java**, **FabricGateway.java** y **EnviarMensajeDTO.java**.

Una vez ejecutado el proyecto Gateway pone a disposición un API Rest en el puerto 8085 que puede ser testeado en la URL localhost:

<http://localhost:8085/swagger-ui/>

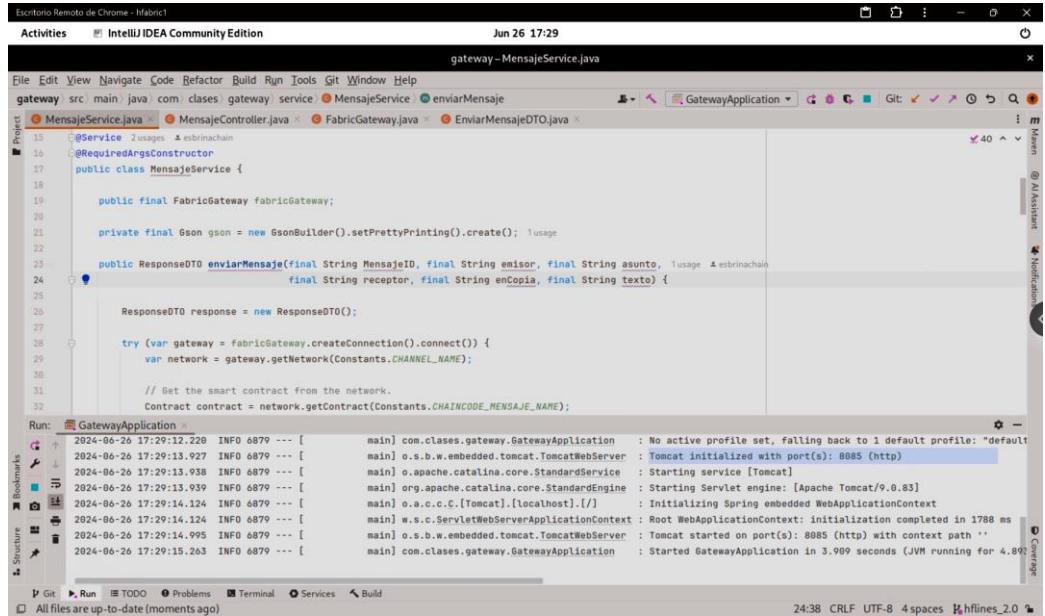


Figura 42: Código del proyecto Gateway a modo de backend API Rest

En la figura 43, se puede observar todos la funcionalidad del chaincode disponible en **HFLINES**.

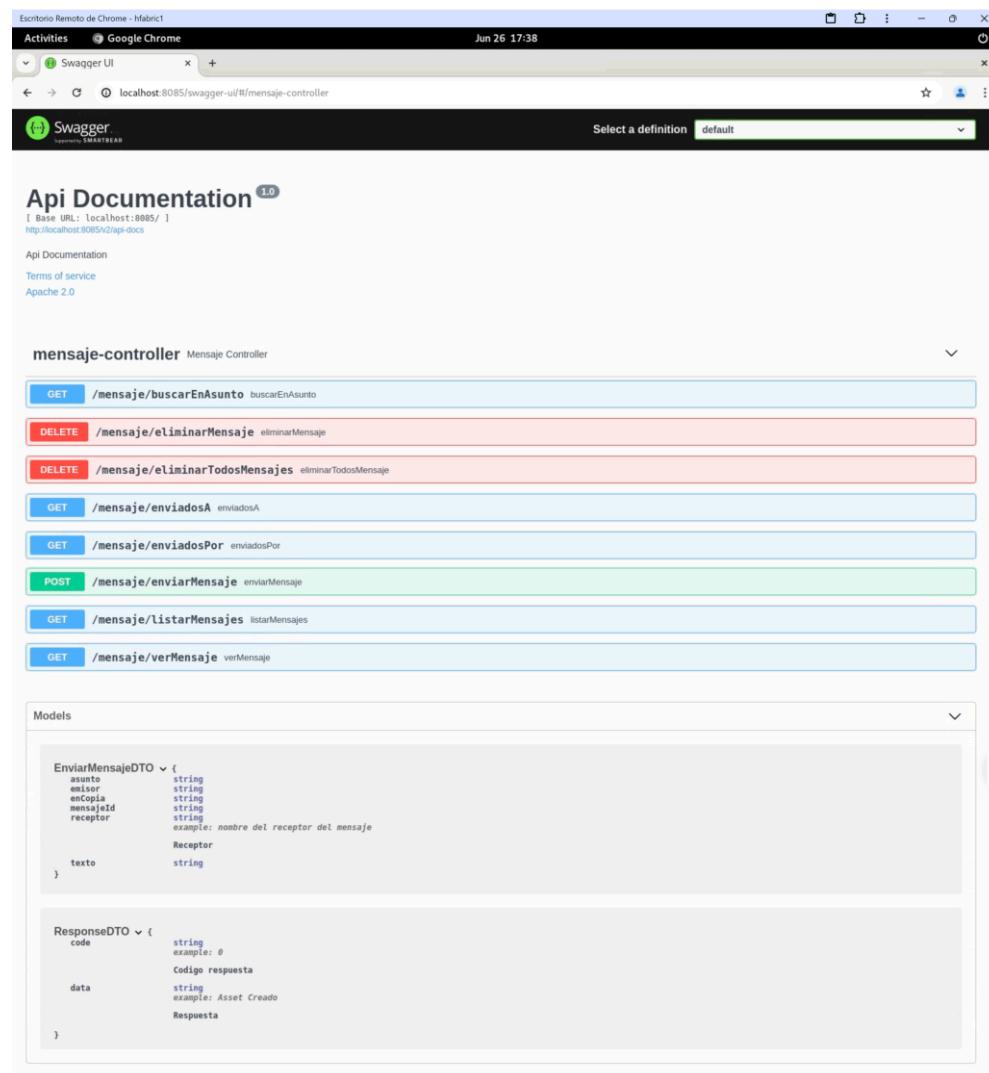


Figura 43: API Rest Gateway de acceso a la funcionalidad del chaincode de HFLines

En la figura 44, se muestra la ejecución de la función **ListarMensajes** una vez ejecutados los ejemplos del script **30_ejemplos_uso_20_app0x_org1.sh**.

```

curl -X GET "http://localhost:8085/mensaje/listarMensajes" -H "accept: */*"

```

```

{
  "code": 0,
  "data": "\n    \\"asunto\\": \"Reunion aceptada\",\\n    \\"emisor\\": \"Maria Castillo\",\\n    \\"enCopia\\\": \"antonio maria rebeca\",\\n    \\"id\\\": \"2\",\\n    \\"receptor\\\": \"Juan Solana\",\\n    \\"texto\\\": \"Reunion Lunes - OK. Nos desplazamos 4 personas\\n    \"),\\n    \\"asunto\\\": \"Reunion Lunes TELCO\",\\n    \\"emisor\\\": \"Maria Castillo\",\\n    \\"enCopia\\\": \"Antonio Maria Rebeca\",\\n    \\"id\\\": \"3\",\\n    \\"receptor\\\": \"Anna Gracia\",\\n    \\"texto\\\": \"Deberias asistir a la reunion del lunes, oficinas TELCO a las 11h.\\n    \"),\\n    \\"asunto\\\": \"Reunion de ultima hora..\",\\n    \\"emisor\\\": \"Maria Castillo\",\\n    \\"enCopia\\\": \"Antonio Maria Rebeca\",\\n    \\"id\\\": \"4\",\\n    \\"receptor\\\": \"Anna Gracia\",\\n    \\"texto\\\": \"Finalmente, no podre asistir a la reunion del lunes. Por favor, encargate tu de substituirme.\"\\n  \")\\n"
}

```

Code	Description
200	OK
401	Unauthorized
403	Forbidden
404	Not Found

Figura 44: Ejecución de la función **ListarMensajes** usando el API Rest del proyecto **Gateway** usando la librería **Swagger**

En la figura 44, se puede comprobar como se puede programar un cliente web que podría utilizar el **API Rest** del **backend** implementado en el **proyecto Gateway** para ejecutar la funcionalidad de nuestro **chaincode** sobre la cadena de bloques de **HFLines**.

011. MONITORIZACIÓN DE LA RED HFLINES: PROMETHEUS Y GRAFANA

Es esencial medir el rendimiento de la red **Hyperledger Fabric HFLines**. Para obtener las métricas de funcionamiento disponemos de la herramienta **Prometheus** y para generar gráficos de estas métricas o de la ejecución de los contenedores en tiempo real disponemos de la herramienta **Grafana**.

El website de **Prometheus** está activo en el puerto 9090 de nuestra máquina virtual. En el caso de uso que nos ocupa, en el momento de realizar el documento, la instancia virtual de Google utiliza la IP externa: 34.175.201.73 y podemos consultar las métricas de **Prometheus** en la Url <http://34.175.201.73:9090/>. En la figura 45, se muestra un ejemplo de dos consultas (en inglés query). Se consulta los contenedores levantados en este momento en la red **HFLines** y la versión de **cadvisor** (facilita métricas a Prometheus):

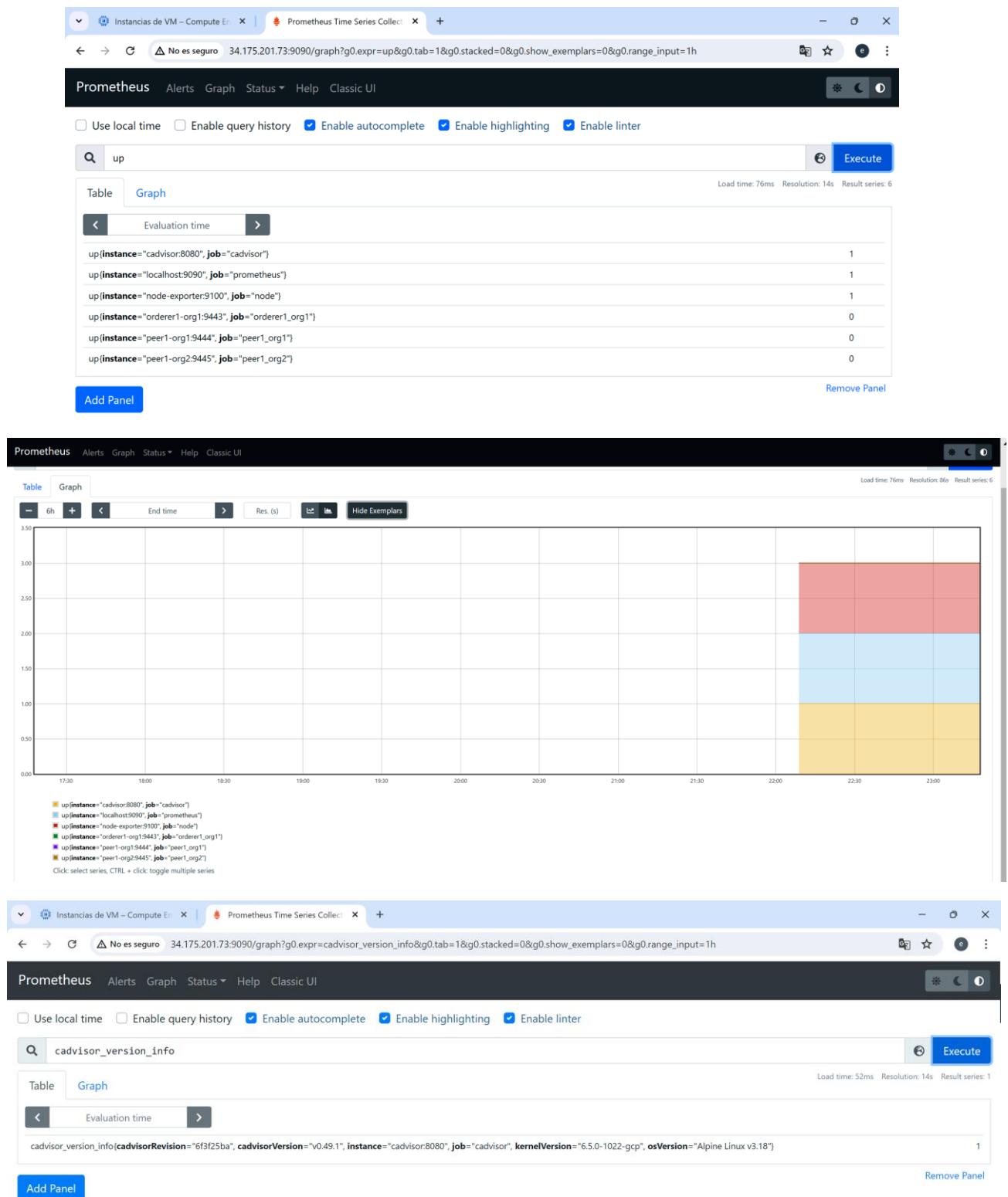


Figura 45: Ejemplos de consultas en Prometheus: (a) de métrica 'up' (b) versión de cAdvisor activa.

Para obtener **gráficas en tiempo real**, la herramienta **Grafana** nos proporciona un generador de paneles para ver el comportamiento de nuestra red. En la figura 46, podemos observar un ejemplo de panel (en inglés dashboard) obtenido con **Grafana**:

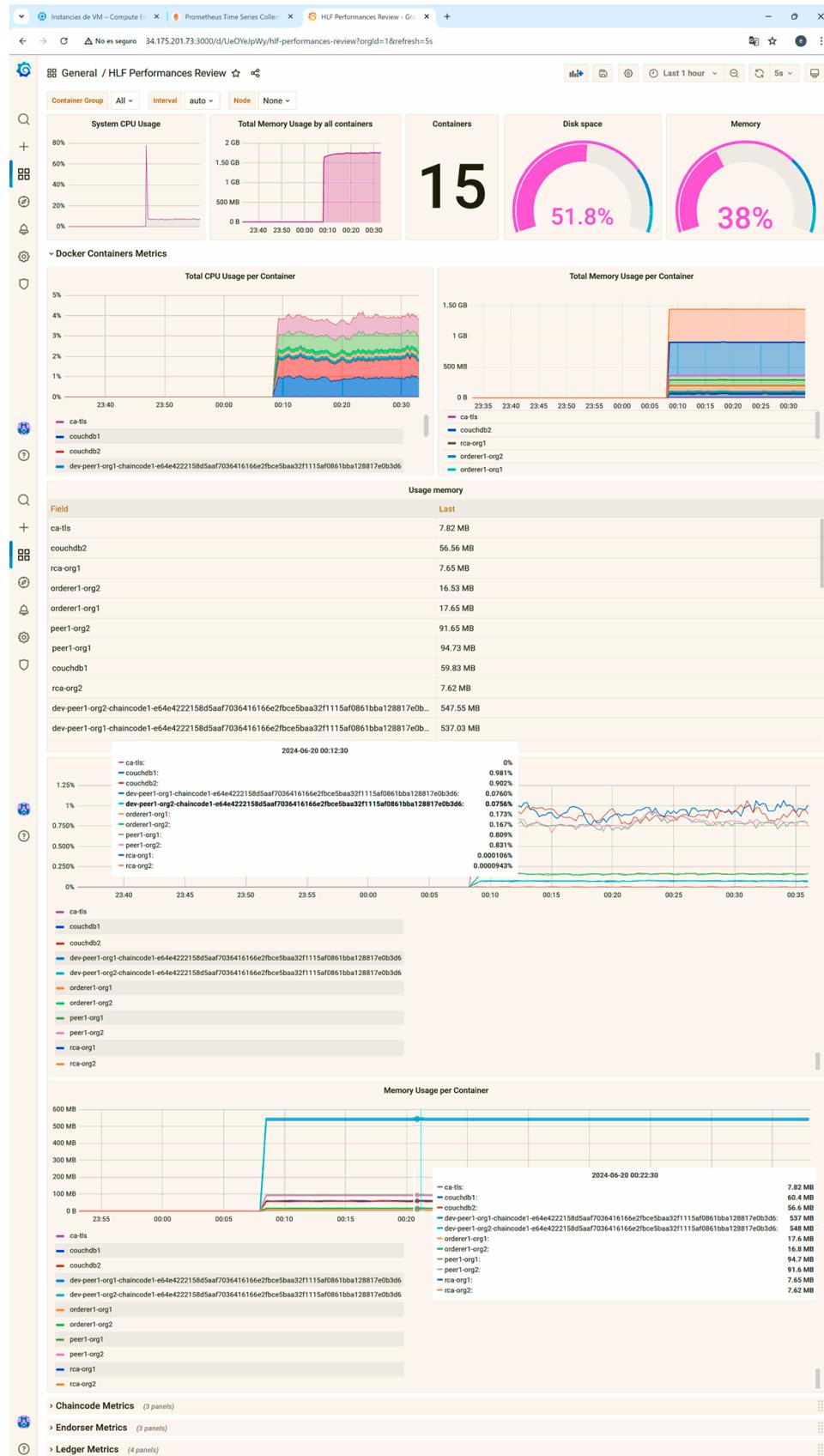


Figura 46: Ejemplos de ‘dashboards’ en tiempo real con la herramienta Grafana.

012. CONCLUSIONES FINALES

La red **Hyperledger Fabric HFLines** es un proyecto pensado para poner en práctica los conocimientos adquiridos en el curso e intenta ser un proyecto cercano a lo que sería un proyecto en producción real. Aunque en algunos aspectos lo es, en otros se ha optado por la estrategia de montar una red simple que funcione a modo de prototipo, para después ir ampliando la red en base a un núcleo inicial sólido.

En este sentido, el diseño inicial de la red para cubrir el objetivo de implementar un sistema de comunicaciones para el envío y recepción de mensajes no necesitaba de una gran cantidad de nodos **peer** ni nodos **orderer**, sobre todo si tenemos en cuenta que se ha pensado para una UTE de empresas de tipo PYME. Es por ello por lo que al no requerir una gran cantidad de nodos se ha optado por adquirir experiencia y ver como **configurar un diseño red innovador y diferente a los esquemas tradicionales** que dedican una organización de la red a los nodos **orderer**. Para el sistema de mensajería **HFLines**, cada organización tiene un nodo **orderer**, y el sistema puede funcionar dando cobertura a una sola empresa internamente y colaborar con otra si esta existe.

Respecto a la implementación de la red, ha costado bastante al principio, se ha dedicado mucho tiempo en generar el material criptográfico y disponerlo en una estructura de directorios accesible para su uso. El volumen de carpetas con pocos nodos ya es de una dimensión considerable.

En cuanto al número de **nodos peer** se podría haber incluido más peer por organización, pero el número de errores al principio y la necesidad de que funcionara algo básico diferente de la test-network usada en las clases ha mantenido el diseño inicial, para conseguir un **docker-compose.yaml** y un **configtx.yaml** funcional.

Las configuraciones del canal y las políticas de aprobación (en inglés endorsement) también han necesitado de realizar muchas pruebas hasta conseguir configurar el canal superando infinidad de errores que no siempre disponían de solución documentada. La configuración de los perfiles (en inglés profiles) también es otro apartado en que la red **Hyperledger Fabric** no es sencilla de sintonizar. Se han buscado alternativas a la propuesta actual (*SampleAppChannelEtcdrRaft*), pero no han funcionado.

La implementación mediante **contenedores** es un punto a favor de este tipo de redes pues una vez configurados permanecen sólidos y fallan poco, siendo relativamente sencillo realizar cambios o levantar un tipo de configuración u otra. La adaptación de Prometheus-Grafana parecía complicada y al final no lo ha sido.

La facilidad de implementar **chaincodes** con lenguajes de alto nivel también es una de las ventajas muy destacable, y queda para un futuro cercano, investigar la fiabilidad y rendimiento de incorporar desarrollos en **Solidity** existentes mediante iniciativas de otros desarrollos del equipo de **Fabric** como **FAB3** consultable en (<https://github.com/hyperledger/fabric-chaincode-evm>).

En general el proyecto **ha servido y mucho** para adquirir experiencia práctica de aquellos aspectos teóricos aprendidos, y que desde la teoría quizás no se llegan a entender del todo. Este proyecto nos ha permitido entender la versatilidad de **Hyperledger Fabric** en generar redes de colaboración entre organizaciones utilizando multitud de canales de comunicación para dar solución con seguridad y privacidad a las necesidades en cada proyecto.

A partir de la experiencia adquirida se nos ocurren multitud de sistemas que pueden ser implementados con **Hyperledger Fabric** y en los cuáles aprovechar las ventajas del intercambio seguro de activos de valor más complejos.

013. AGRADECIMIENTOS

Dar las gracias a los profesores de este módulo de máster **Eloy Valdera Carrasco, Alberto García Castro y Domingo Martínez Ródenas**, que han hecho posible la elaboración de este proyecto con su dedicación y esfuerzo en todas las clases.

21/06/2024