keep coding

# Hyperledger Fabric

## Módulo “Certificación HLF (Administrator/Developer”

## Práctica:

**Víctor García Delgado**

## Contenido



# 1. Enunciado práctico

La empresa Merca-link, dedicada a la venta al por menor de alimentos, productos de limpieza e higiene personal y enseres, se encuentra implementando una red Blockchain para dotar de las más altas cotas de trazabilidad a los productos de su marca blanca: "Haciendo".

Los ingenieros de Merca-link optaron por una red basada en Hyperledger Fabric, a la que bautizaron como "Merca-chain", y han estado trabajando en la creación de algunos scripts y ficheros de despliegue de comunicación en vistas del próximo lanzamiento a nivel nacional. Sin embargo, a la hora de hacer las pruebas, parece que existen algunos problemas y cuestiones por perfeccionar que escapan a los conocimientos del personal de Merca-link.

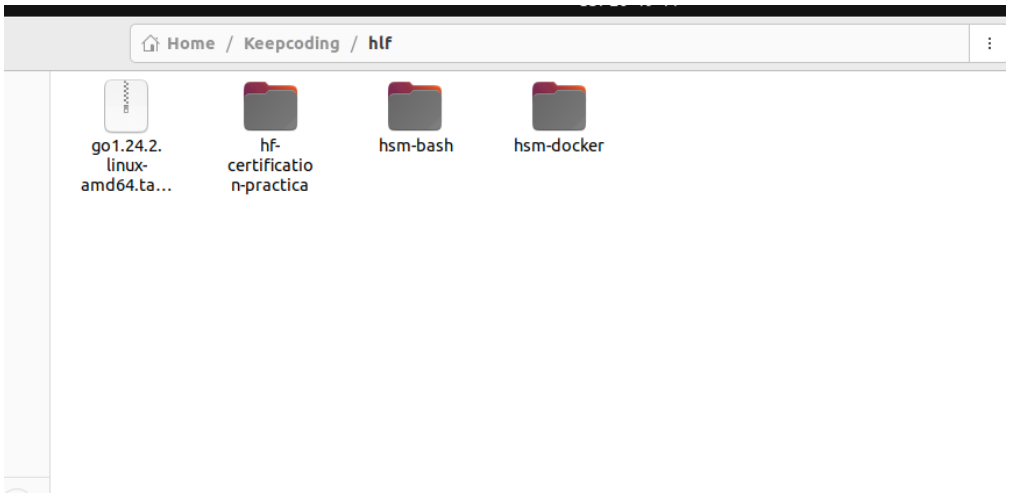
En esta tesitura, Merca-link te proporciona el código que han elaborado (instrucciones básicas de puesta en marcha en el Anexo 1):

https://github.com/KeepCodingBlockchain-I/hf-certification-practica

Además, te pide que le prestes una serie de servicios en calidad de experto en administración de redes Hyperledger Fabric:

# 2. Realización Ejercicios

En primer lugar, vamos a preparar el entorno. Dentro de nuestro entorno, nos creamos una nueva carpeta (hlf-certificacion-practica)



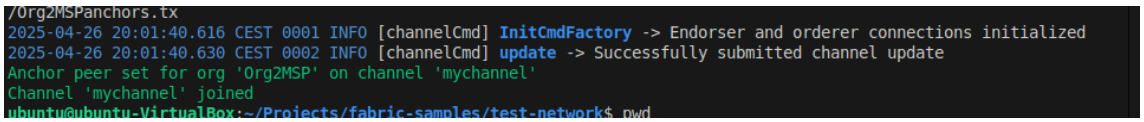

Nos traemos el repo de git y verificamos que la red funciona correctamente

(*)En esta MV ya estamos usando o tenemos instalado un fabric que tenemos sobre otra carpeta raiz /home/ubuntu/Projects/fabric-samples/test-network).

Actualmente, usando el comando:

```
./network.sh up createChannel -ca
```

Verificamos que la red funciona correctamente y levanta ok




Y los contendores también se levantan correctamente:


Paramos la red -> ok

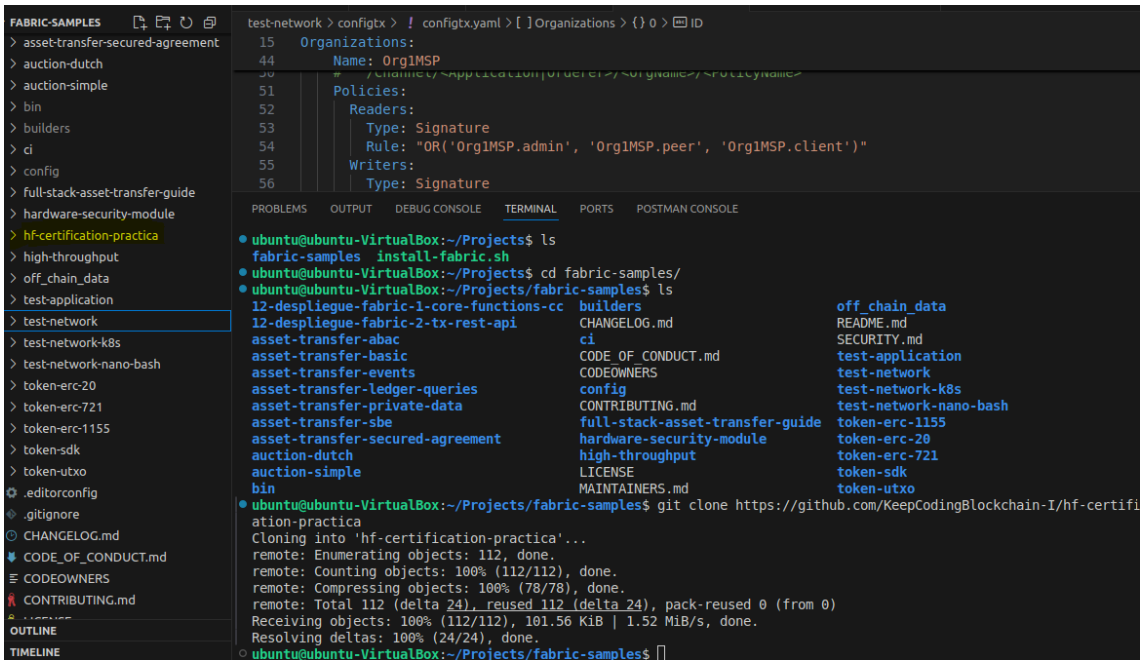
```
./network.sh down
```

Verificamos que todo se para correctamente ->ok

```
ubuntu@ubuntu-VirtualBox:~/Projects/fabric-samples/test-network$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Me vuelvo a traer la carpeta proporcionado a esta ruta:

```
Git clone https://github.com/KeepCodingBlockchain-I/hf-certification-practica
```

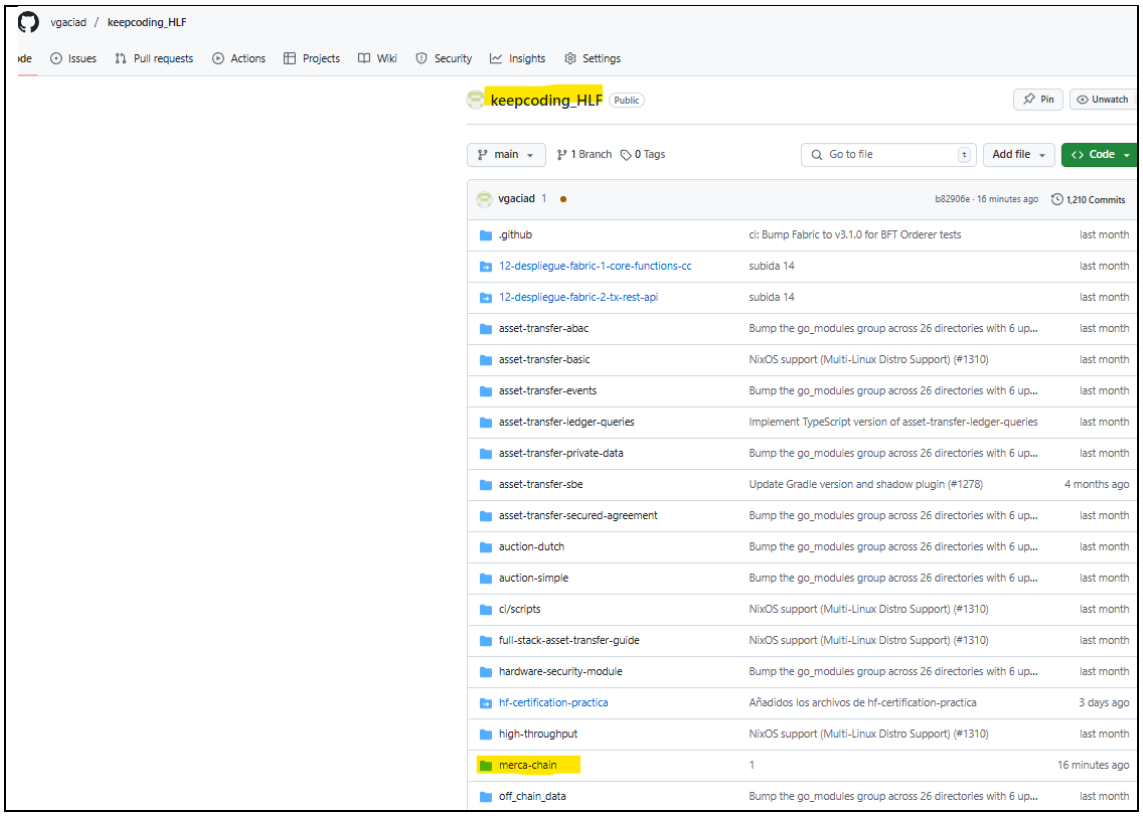


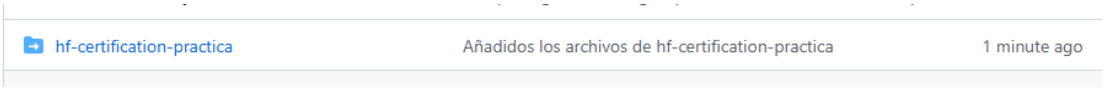

Y Ahora, vamos a subir toda la carpeta a .git para seguir trabajando y comiteando los cambios sobre esta ruta todos los cambios que realicemos a continuación de merca-chain sobre la red original (* fork de la red de fabric oficial https://github.com/hyperledger/fabric-samples):

```
git remote add origin https://github.com/vgaciad/keepcoding_HLF.git
```

Una vez subidos los cambios, este será la dirección sobre la que realizaremos la entrega de la práctica:

```
https://github.com/vgaciad/keepcoding_HLF
```

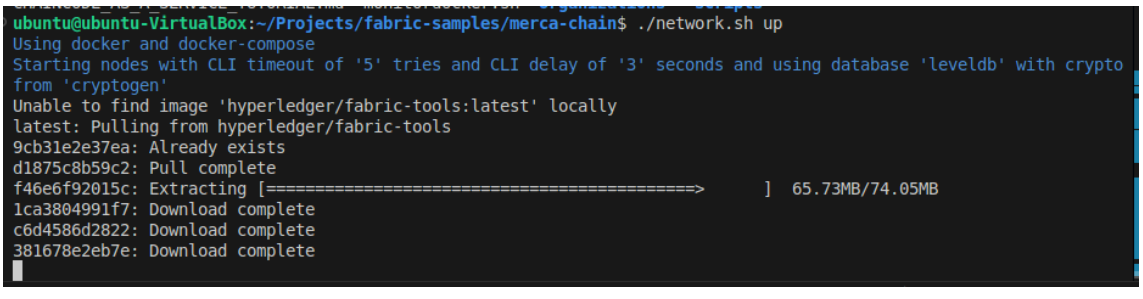



Ahora vamos a probar a ver si despliega la red:

Ubicamos la carpeta “merca-chain” en la ruta correcta (tenemos que subirla un nivel a la altura de la carpeta “test-network” para que funcione correctamente )y probamos con el comando:

```
./network.sh up
```



```
Generating CCP files for Org1 and Org2
./network.sh: line 253: ./organizations/ccp-generate.sh: Permission denied
Creating network "fabric_test" with the default driver
Creating volume "compose_orderer.example.com" with default driver
Creating volume "compose_peer0.org1.example.com" with default driver
Creating volume "compose_peer0.org2.example.com" with default driver
Creating orderer.example.com    ... done
Creating peer0.org1.example.com ... done
Creating peer0.org2.example.com ... done
Creating cli                    ... done
```

A priori, el despliegue parece correcto, pero vamos a verificar los contenedores:

```
docker ps -a
```

```
ubuntu@ubuntu-VirtualBox:~/Projects/fabric-samples/merca-chain$ docker ps -a
CONTAINER ID   IMAGE                               COMMAND             CREATED         STATUS                       P
ORTS                                                                                   NAMES
dd6181c67614   hyperledger/fabric-tools:latest     "/bin/bash"         2 minutes ago   Up 2 minutes
                                                                                       cli
3da41a21d407   hyperledger/fabric-peer:latest      "peer node start"   2 minutes ago   Up 2 minutes                 0
.0.0.0:9051->9051/tcp, :::9051->9051/tcp, 7051/tcp, 0.0.0.0:9445->9445/tcp, :::9445->9445/tcp   peer0.org2.example.
com
479a7103aa3f   hyperledger/fabric-orderer:latest   "orderer"           2 minutes ago   Exited (2) 2 minutes ago
                                                                                       orderer.example.com
270e4d949e74   hyperledger/fabric-peer:latest      "peer node start"   2 minutes ago   Exited (1) 2 minutes ago
                                                                                       peer0.org1.example.
com
```

Vemos que existen, 2 contenedores en estado “Exited”. Procedemos a pasar al ejercicio 1.

***1*. Despliega la red. Parece que existen algunos contenedores que no se levantan correctamente. Identifícalos y resuelve los problemas asociados.**

(*) Dado que llevamos un gran tiempo dedicado a la preparación del entorno y para poder continuar también con el resto de ejercicios a continuación vamos a listar los errores de los id de los contenedores que presentan errores para poder continuar con la realización de toda la práctica.

Verificamos y revisamos los logs de los contenedores:

ID 479a7103aa3f

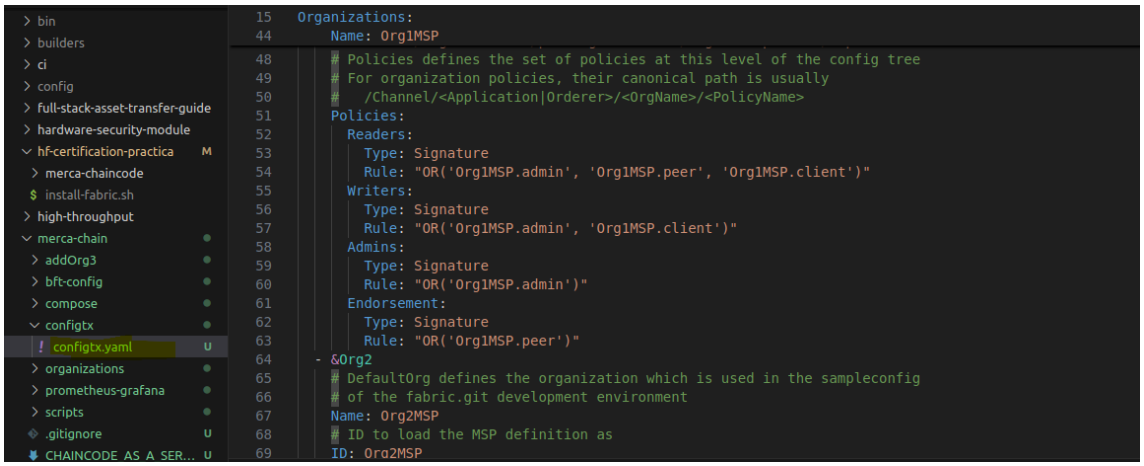
ID 270e4d949e74

```
docker logs 479a7103aa3f

Peer: Error 1
2025-04-26 19:19:22.636 UTC 000e PANI [orderer.common.server] Main -> failed to start admin server: open /var/hyperledger/orderer/tls/servre.crt: no such file or directory
panic: failed to start admin server: open /var/hyperledger/orderer/tls/servre.crt: no such file or directory

goroutine 1 [running]:
go.uber.org/zap/zapcore.CheckWriteAction.OnWrite(0x0?, 0x4254fc?, {0x0?, 0x0?, 0xc00028aae0?})
```

```
	/vendor/go.uber.org/zap/zapcore/entry.go:196 +0x54
go.uber.org/zap/zapcore.(*CheckedEntry).Write(0xc0002a20d0, {0x0, 0x0, 0x0})
	/vendor/go.uber.org/zap/zapcore/entry.go:262 +0x24e
go.uber.org/zap.(*SugaredLogger).log(0xc0001206b8, 0x4, {0x12abcc9?, 0x27?},
{0xc000051c10?, 0x1?, 0x1?}, {0x0, 0x0, 0x0})
	/vendor/go.uber.org/zap/sugar.go:355 +0xec
go.uber.org/zap.(*SugaredLogger).Panicf(...)
	/vendor/go.uber.org/zap/sugar.go:229
github.com/hyperledger/fabric/common/flogging.(*FabricLogger).Panicf(...)
	/common/flogging/zap.go:74
github.com/hyperledger/fabric/orderer/common/server.Main()
	/orderer/common/server/main.go:267 +0x165d
main.main()
	/cmd/orderer/main.go:15 +0xf
```

```
docker logs 270e4d949e74

Orderer

Error 2:
2025-04-26 19:19:22.345 UTC 0005 FATA [nodeCmd] serve -> Error loading secure
config for peer (error loading TLS key (open /etc/hyperledger/fabric/tls/server.key:
no such file or directory))
```

La depuración de los errores se deja para un análisis posterior.




* Revisar fichero configtx.yaml

**2. Despliega el chaincode “merca-chaincode”. Aunque el proceso de ciclo de vida del chaincode es satisfactorio, parece que este no se despliega correctamente. Identifica el/los problemas y resuélvelos. Documenta este proceso de investigación y resolución.**

(*) Dado que llevamos un gran tiempo dedicado a la preparación del entorno y para poder continuar también con el resto de ejercicios a continuación vamos a listar los errores que se presentan al intentar desplegar el contrato.

En primer lugar, verificamos que el contrato de la red original se despliega correctamente. Para ello, no ubicamos en la carpeta de la test-net.

```
./network.sh up
```

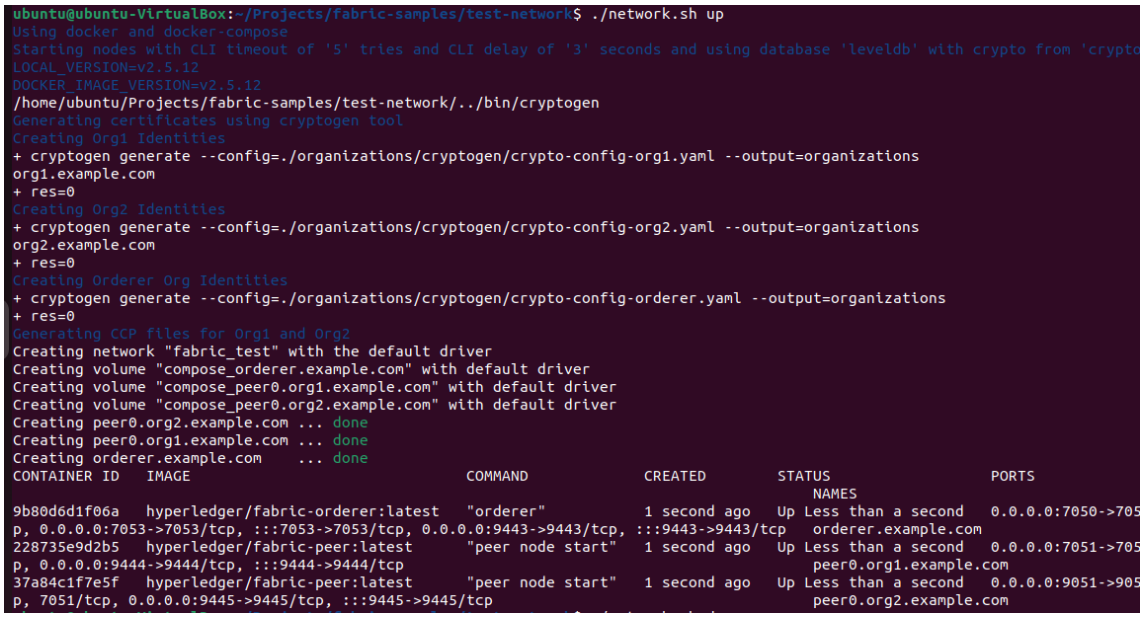



*Ilustración 1. Test-Network levantada*

Verificamos también que el comando, crea el canal correctamente -> ok

```

```


createChannel ok

Verificamos que podemos desplegar un chaincode sobre la red -> ok

./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go

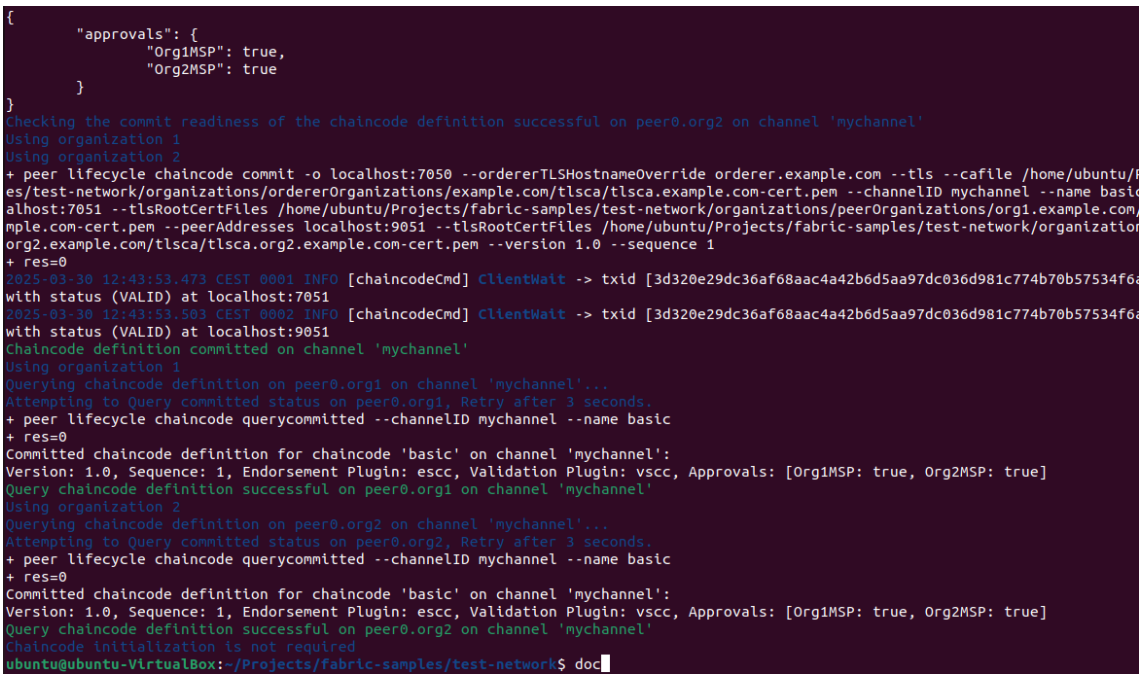



*Ilustración 2. Chaincode go -> ok*

Tiramos la red y listamos los volúmenes de Docker para verificar que no tenemos nada corriendo -> ok

./network.sh down -> ok
docker volume ls -> ok

Ahora vamos a probar el chain-code desde la red de “Merca-chain”.

Desde la carpeta de “Merca-chain” volvemos a levantar la red de “merca chain”

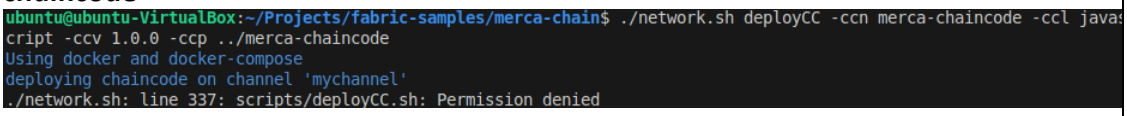
./network.sh up createChannel

Aparecen los siguientes errores cuando intentamos levantar la red:
Error:
sing docker and docker-compose
Creating channel 'mychannel'.
If network is not up, starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb
Bringing up network
LOCAL_VERSION=v2.5.12
DOCKER_IMAGE_VERSION=v2.5.12
Starting peer0.org1.example.com ...
Starting peer0.org1.example.com ... done
Starting orderer.example.com ... done
cli is up-to-date

./network.sh: line 331: scripts/createChannel.sh: Permission denied

Vamos a intentar de todas formas desplegar el contrato

***./network.sh deployCC -ccn merca-chaincode -ccl javascript -ccv 1.0.0 -ccp ../merca-chaincode***

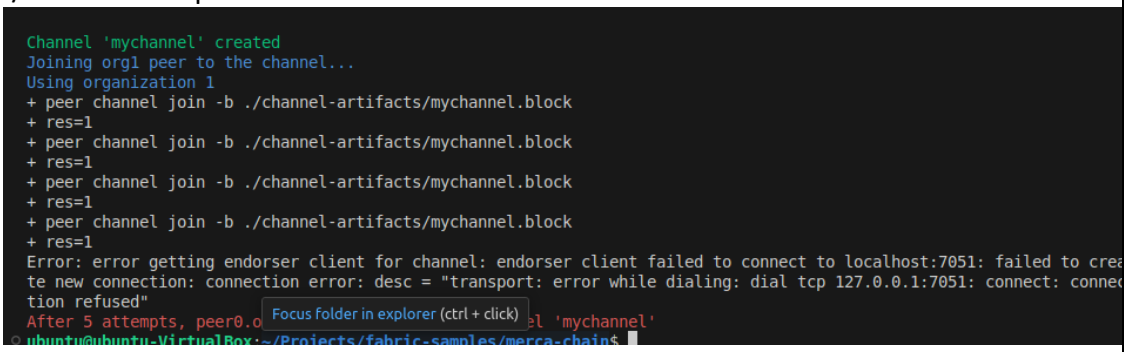



Aparece el siguiente error:
./network.sh deployCC -ccn merca-chaincode -ccl javascript -ccv 1.0.0 -ccp ../merca-chaincode
Using docker and docker-compose
deploying chaincode on channel 'mychannel'
./network.sh: line 337: scripts/deployCC.sh: Permission denied
Deploying chaincode failed

Como en todos es común el error de los permisos, vamos a probar a otorgar permisos a toda la carpeta (merca-chain, merca-chaincode).

Una vez otorgados los permisos, probamos de nuevo.

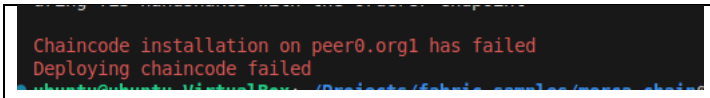
./network.sh up createChannel -ca




Error: error getting endorser client for channel: endorser client failed to connect to localhost:7051: failed to create new connection: connection error: desc = "transport: error while dialing: dial tcp 127.0.0.1:7051: connect: connection refused"

Ahora aparece el siguiente error, pero tiene mejor pinta. Vamos a probar sin el -ca. Sigue apareciendo el error…aún así, como el canal parece que se crea, vamos a probar a ver si nos deja deplegar el chaincode.

./network.sh deployCC -ccn merca-chaincode -ccl javascript -ccv 1.0.0 -ccp ../merca-chaincode



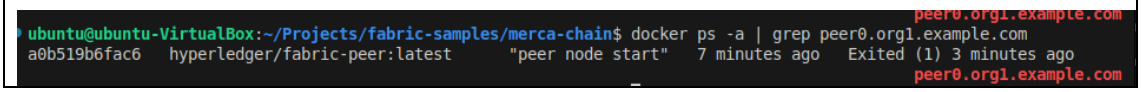

```
Using docker and docker-compose
deploying chaincode on channel 'mychannel'
executing with the following
- CHANNEL_NAME: mychannel
- CC_NAME: merca-chaincode
- CC_SRC_PATH: ../merca-chaincode
- CC_SRC_LANGUAGE: javascript
- CC_VERSION: 1.0.0
- CC_SEQUENCE: auto
- CC_END_POLICY: NA
- CC_COLL_CONFIG: NA
- CC_INIT_FCN: NA
- DELAY: 3
- MAX_RETRY: 5
- VERBOSE: false
executing with the following
- CC_NAME: merca-chaincode
- CC_SRC_PATH: ../merca-chaincode
- CC_SRC_LANGUAGE: javascript
- CC_VERSION: 1.0.0
+ '[' false = true ']'
+ peer lifecycle chaincode package merca-chaincode.tar.gz --path ../merca-chaincode
--lang node --label merca-chaincode_1.0.0
+ res=0
Chaincode is packaged
Installing chaincode on peer0.org1...
Using organization 1
+ peer lifecycle chaincode queryinstalled --output json
+ jq -r 'try (.installed_chaincodes[].package_id)'
+ grep '^merca-
chaincode_1.0.0:2fe5761cf062628f09fa31a0e7c29059db695faee2cfa5797153416ff9
70bdb3$'
Error: failed to retrieve endorser client for queryinstalled: endorser client failed to
connect to localhost:7051: failed to create new connection: connection error: desc =
"transport: error while dialing: dial tcp 127.0.0.1:7051: connect: connection refused"
Usage:
  peer lifecycle chaincode queryinstalled [flags]

Flags:
      --connectionProfile string       The fully qualified path to the connection profile
that provides the necessary connection information for the network. Note: currently
only supported for providing peer connection information
  -h, --help                           help for queryinstalled
  -O, --output string                  The output format for query results. Default is human-
readable plain-text. json is currently the only supported format.
```

```
    --peerAddresses stringArray      The addresses of the peers to connect to
    --targetPeer string              When using a connection profile, the name of the peer
to target for this action
    --tlsRootCertFiles stringArray   If TLS is enabled, the paths to the TLS root cert files
of the peers to connect to. The order and number of certs specified should match
the --peerAddresses flag

Global 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)
    --keyfile string                 Path to file containing PEM-encoded private key to use
for mutual TLS communication with the orderer endpoint
 -o, --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

+ test 1 -ne 0
+ peer lifecycle chaincode install merca-chaincode.tar.gz
+ res=1
Error: failed to retrieve endorser client for install: endorser client failed to connect to
localhost:7051: failed to create new connection: connection error: desc = "transport:
error while dialing: dial tcp 127.0.0.1:7051: connect: connection refused"
Usage:
  peer lifecycle chaincode install [flags]

Flags:
    --connectionProfile string       The fully qualified path to the connection profile
that provides the necessary connection information for the network. Note: currently
only supported for providing peer connection information
 -h, --help                          help for install
    --peerAddresses stringArray      The addresses of the peers to connect to
    --targetPeer string              When using a connection profile, the name of the peer
to target for this action
    --tlsRootCertFiles stringArray   If TLS is enabled, the paths to the TLS root cert files
of the peers to connect to. The order and number of certs specified should match
the --peerAddresses flag

Global 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)
      --keyfile string                 Path to file containing PEM-encoded private key to use
for mutual TLS communication with the orderer endpoint
  -o, --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

Chaincode installation on peer0.org1 has failed
Deploying chaincode failed
```

Analizando el error, parece que se trata del error común de que el peer0 de la organización1 no se ha levantado correctamente, que como efectivamente podemos verificar realizando un:

```
docker ps -a | grep peer0.org1.example.com
```




Por lo que habría que habría que seguir revisando el fichero configtx.yaml, como en el ejercicio 1, para poder levantar la red correctamente y poder desplegar el contrato.

3. **Desde Merca-link nos comentan que los logs de los peers no proporcionan la información deseada. Modifica el logging de la red para que se ajuste a los siguientes requerimientos:**

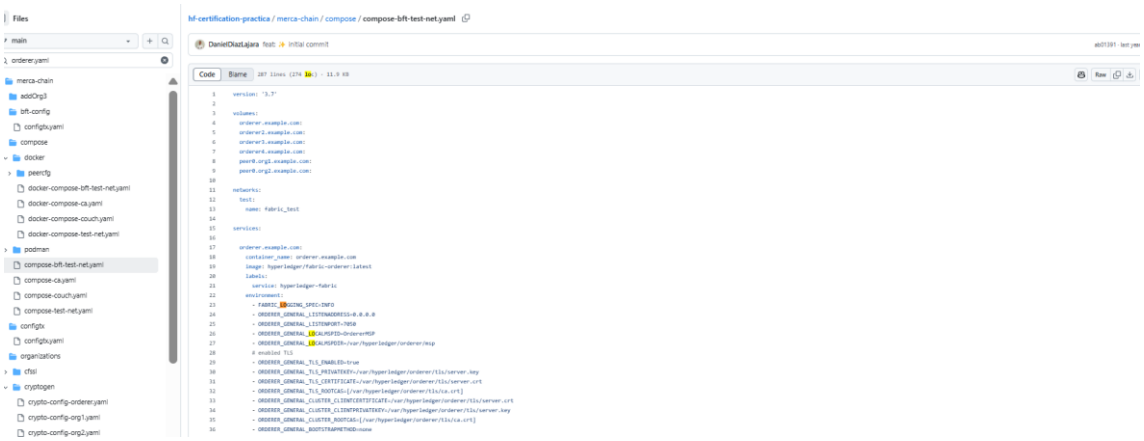
- **Los logs de los orderers deben estar en formato JSON.**
- **Los logs de los peers deben de mostrar la fecha y la hora al final.**
- **Dado que estamos en fase de pruebas, los logs de los peers deben de tener un nivel de logging de DEBUG por defecto. Además, los logs asociados al logger "gossip" tendrán un nivel por defecto de WARNING y los del logger de "chaincode" de INFO para no sobrecargar demasiado los logs.**

Para poder modificar los logs, se puede realizar de tres maneras:

1. Variables de entorno
2. Flags de configuración
3. Configuración del fichero de configuración .yaml

Nos vamos a decantar por cambiar la configuración del fichero .yaml, y para ello debemos situarnos en el archivo:

| merca-chain/compose/compose-bft-test-net.yaml |
|---|



Aquí se realizan las modificaciones oportunas (compose-bft-test-net.yaml)

**1. Los logs de los orderers deben estar en formato JSON.**

```
orderer.example.com:
  container_name: orderer.example.com
  image: hyperledger/fabric-orderer:latest
  labels:
    service: hyperledger-fabric
  environment:
    - FABRIC_LOGGING_SPEC=INFO
    - FABRIC_LOGGING_FORMAT = JSON
    - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
    - ORDERER_GENERAL_LISTENPORT=7050
    - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
    - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
    # enabled TLS
    - ORDERER_GENERAL_TLS_ENABLED=true
    - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
    - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
    - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
```

Orderer en formato json

**2. Los logs de los peers deben de mostrar la fecha y la hora al final.**

```
  peer0.org1.example.com:
    container_name: peer0.org1.example.com
    image: hyperledger/fabric-peer:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_CFG_PATH=/etc/hyperledger/peercfg
      - FABRIC_LOGGING_SPEC=INFO
      - FABRIC_LOGGING_FORMAT = "%{color}%{time:2006-01-02 15:04:05.000 MST} [%{module}] %{shortfunc} -> %
      #- FABRIC_LOGGING_SPEC=DEBUG
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_PROFILE_ENABLED=false
      - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
      - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
      # Peer specific variables
      - CORE_PEER_ID=peer0.org1.example.com
      - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
```

Peers en con la fecha y hora final.

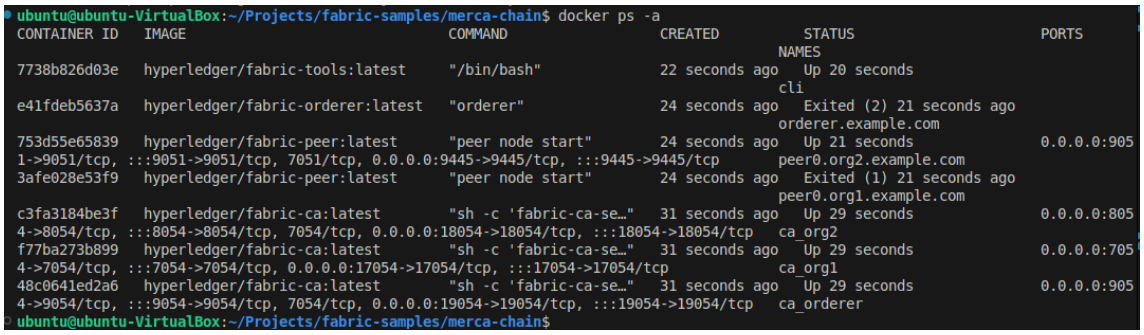
**3. Dado que estamos en fase de pruebas, los logs de los peers deben de tener un nivel de logging de DEBUG por defecto. Además, los logs asociados al logger “gossip” tendrán un nivel por defecto de WARNING y los del logger de “chaincode” de INFO para no sobrecargar demasiado los logs.**

```
  peer0.org2.example.com:
    container_name: peer0.org2.example.com
    image: hyperledger/fabric-peer:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_CFG_PATH=/etc/hyperledger/peercfg
      #- FABRIC_LOGGING_SPEC=INFO
      - FABRIC_LOGGING_FORMAT = "%{color}%{time:2006-01-02 15:04:05.000 MST} [%{module}]
      - FABRIC_LOGGING_SPEC=DEBUG
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_PROFILE_ENABLED=false
      - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
      - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
      # Peer specific variables
      - CORE_PEER_ID=peer0.org2.example.com
```

Comiteamos los cambios sobre git

```
ubuntu@ubuntu-VirtualBox:~/Projects/fabric-samples/merca-chain$ git add .
ubuntu@ubuntu-VirtualBox:~/Projects/fabric-samples/merca-chain$ git commit "3"
error: pathspec '3' did not match any file(s) known to git
ubuntu@ubuntu-VirtualBox:~/Projects/fabric-samples/merca-chain$ git commit -m  "3"
[main 6908cd0] 3
 1 file changed, 1 insertion(+), 1 deletion(-)
ubuntu@ubuntu-VirtualBox:~/Projects/fabric-samples/merca-chain$ git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 421 bytes | 421.00 KiB/s, done.
Total 5 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/vgaciad/keepcoding_HLF.git
   c0acc8e..6908cd0  main -> main
```

Ahora verificamos los logs de pej un orderer y un peer




**orderer**

```
docker logs e41806c1e37a

JSON
panic: failed to start admin server: open /var/hyperledger/orderer/tls/servre.crt: no such file or directory

goroutine 1 [running]:
go.uber.org/zap/zapcore.CheckWriteAction.OnWrite(0x0?, 0x4254fc?, {0x0?, 0x0?, 0xc000988b80?})
	/vendor/go.uber.org/zap/zapcore/entry.go:196 +0x54
go.uber.org/zap/zapcore.(*CheckedEntry).Write(0xc0004c6c30, {0x0, 0x0, 0x0})
	/vendor/go.uber.org/zap/zapcore/entry.go:262 +0x24e
go.uber.org/zap.(*SugaredLogger).log(0xc0002c22f8, 0x4, {0x12abcc9?, 0x27?}, {0xc000051c10?, 0x1?, 0x1?}, {0x0, 0x0, 0x0})
	/vendor/go.uber.org/zap/sugar.go:355 +0xec
go.uber.org/zap.(*SugaredLogger).Panicf(...)
	/vendor/go.uber.org/zap/sugar.go:229
github.com/hyperledger/fabric/common/flogging.(*FabricLogger).Panicf(...)
	/common/flogging/zap.go:74
github.com/hyperledger/fabric/orderer/common/server.Main()
	/orderer/common/server/main.go:267 +0x165d
```

**Peer:**

```
docker logs 753d55e65839
2025-04-29 21:44:40.337 UTC 0001 INFO [nodeCmd] serve -> Starting peer:
 Version: v2.5.12
 Commit SHA: af0b647
 Go version: go1.23.5
 OS/Arch: linux/amd64
 Chaincode:
  Base Docker Label: org.hyperledger.fabric
  Docker Namespace: hyperledger
2025-04-29 21:44:40.339 UTC 0002 INFO [nodeCmd] serve -> Peer config with combined core.yaml settings and environment variable overrides:
chaincode:
```

```
  builder: $(DOCKER_NS)/fabric-ccenv:$(TWO_DIGIT_VERSION)
  executetimeout: 300s
  externalbuilders:
  - name: ccaas_builder
    path: /opt/hyperledger/ccaas_builder
    propagateEnvironment:
    - CHAINCODE_AS_A_SERVICE_BUILDER_CONFIG
  golang:
    dynamiclink: false
    runtime: $(DOCKER_NS)/fabric-baseos:$(TWO_DIGIT_VERSION)
  installtimeout: 300s
  java:
    runtime: $(DOCKER_NS)/fabric-javaenv:$(TWO_DIGIT_VERSION)
  keepalive: 0
  logging:
    format: '%{color}%{time:2006-01-02 15:04:05.000 MST} [%{module}] %{shortfunc}
      -> %{level:.4s} %{id:03x}%{color:reset} %{message}'
    level: info
    shim: warning
  mode: net
  node:
    runtime: $(DOCKER_NS)/fabric-nodeenv:$(TWO_DIGIT_VERSION)
  pull: false
  startuptimeout: 300s
  system:
    _lifecycle: enable
    cscc: enable
    lscc: enable
    qscc: enable
ledger:
  history:
    enablehistorydatabase: true
  pvtdatastore:
    collelgprocdbbatchesinterval: 1000
    collelgprocmaxdbbatchsize: 5000
    deprioritizeddatareconcilerinterval: 60m
  snapshots:
    rootdir: /var/hyperledger/production/snapshots
  state:
    couchdbconfig:
      cachesize: 64
      couchdbaddress: 127.0.0.1:5984
      createglobalchangesdb: false
      internalquerylimit: 1000
      maxbatchupdatesize: 1000
      maxretries: 3
      maxretriesonstartup: 10
```

```
      requesttimeout: 35s
    statedatabase: goleveldb
    totalquerylimit: 100000
logging_level: ""
metrics:
  provider: prometheus
  statsd:
    address: 127.0.0.1:8125
    network: udp
    writeinterval: 10s
operations:
  listenaddress: peer0.org2.example.com:9445
  tls:
    clientauthrequired: false
    clientrootcas:
      files: []
    enabled: false
peer:
  address: peer0.org2.example.com:9051
  addressautodetect: false
  authentication:
    timewindow: 15m
  bccsp:
    default: SW
    sw:
      hash: SHA2
      security: 256
  client:
    conntimeout: 3s
  deliveryclient:
    blockgossipenabled: true
    conntimeout: 3s
    reconnectbackoffthreshold: 3600s
    reconnecttotaltimethreshold: 3600s
  discovery:
    authcacheenabled: true
    authcachemaxsize: 1000
    authcachepurgeretentionratio: 0.75
    enabled: true
    orgmembersallowedaccess: false
  filesystempath: /var/hyperledger/production
  gateway:
    dialtimeout: 2m
    enabled: true
    endorsementtimeout: 30s
  gossip:
    aliveexpirationtimeout: 25s
```

```
  alivetimeinterval: 5s
  bootstrap: peer0.org2.example.com:9051
  conntimeout: 2s
  dialtimeout: 3s
  digestwaittime: 1s
  election:
    leaderalivethreshold: 10s
    leaderelectionduration: 5s
    membershipsampleinterval: 1s
    startupgraceperiod: 15s
  externalendpoint: peer0.org2.example.com:9051
  maxblockcounttostore: 10
  maxconnectionattempts: 120
  maxpropagationburstlatency: 10ms
  maxpropagationburstsize: 10
  membershiptrackerinterval: 5s
  msgexpirationfactor: 20
  orgleader: true
  propagateiterations: 1
  propagatepeernum: 3
  publishcertperiod: 10s
  publishstateinfointerval: 4s
  pullinterval: 4s
  pullpeernum: 3
  pvtdata:
    btlpullmargin: 10
    implicitcollectiondisseminationpolicy:
      maxpeercount: 1
      requiredpeercount: 0
    pullretrythreshold: 60s
    pushacktimeout: 3s
    reconcilebatchsize: 10
    reconcilesleepinterval: 1m
    reconciliationenabled: true
    skippullinginvalidtransactionsduringcommit: false
    transientstoremaxblockretention: 1000
  reconnectinterval: 25s
  recvbuffsize: 20
  requeststateinfointerval: 4s
  requestwaittime: 1500ms
  responsewaittime: 2s
  sendbuffsize: 200
  skipblockverification: false
  state:
    batchsize: 10
    blockbuffersize: 20
    checkinterval: 10s
```

```
    enabled: false
    maxretries: 3
    responsetimeout: 3s
  useleaderelection: false
handlers:
  authfilters:
  - name: DefaultAuth
  - name: ExpirationCheck
  decorators:
  - name: DefaultDecorator
  endorsers:
    escc:
      name: DefaultEndorsement
  validators:
    vscc:
      name: DefaultValidation
id: peer0.org2.example.com
keepalive:
  client:
    interval: 60s
    timeout: 20s
  deliveryclient:
    interval: 60s
    timeout: 20s
  interval: 7200s
  mininterval: 60s
  timeout: 20s
limits:
  concurrency:
    deliverservice: 2500
    endorserservice: 2500
listenaddress: 0.0.0.0:9051
localmspid: Org2MSP
localmsptype: bccsp
maxrecvmsgsize: 104857600
maxsendmsgsize: 104857600
mspconfigpath: /etc/hyperledger/fabric/msp
networkid: dev
profile:
  enabled: "false"
  listenaddress: 0.0.0.0:6060
tls:
  cert:
    file: /etc/hyperledger/fabric/tls/server.crt
  clientauthrequired: false
  clientrootcas:
    files:
```

```
    - tls/ca.crt
   enabled: "true"
   key:
    file: /etc/hyperledger/fabric/tls/server.key
   rootcert:
    file: /etc/hyperledger/fabric/tls/ca.crt
vm:
 docker:
   attachstdout: false
   hostconfig:
    logconfig:
     config:
      max-file: "5"
      max-size: 50m
     type: json-file
    memory: 2147483648
    networkmode: fabric_test
   tls:
    ca:
     file: docker/ca.crt
    cert:
     file: docker/tls.crt
    enabled: false
    key:
     file: docker/tls.key
 endpoint: unix:///host/var/run/docker.sock
2025-04-29 21:44:40.340 UTC 0003 INFO [peer] getLocalAddress -> Auto-detected
peer address: 172.18.0.7:9051
2025-04-29 21:44:40.340 UTC 0004 INFO [peer] getLocalAddress -> Returning
peer0.org2.example.com:9051
2025-04-29 21:44:40.353 UTC 0005 INFO [nodeCmd] initGrpcSemaphores ->
concurrency limit for endorser service is 2500
2025-04-29 21:44:40.359 UTC 0006 INFO [nodeCmd] initGrpcSemaphores ->
concurrency limit for deliver service is 2500
2025-04-29 21:44:40.359 UTC 0007 INFO [nodeCmd] serve -> Starting peer with TLS
enabled
2025-04-29 21:44:40.471 UTC 0008 INFO [certmonitor] trackCertExpiration -> The
enrollment certificate will expire on 2026-04-29 21:45:00 +0000 UTC
2025-04-29 21:44:40.471 UTC 0009 INFO [certmonitor] trackCertExpiration -> The
server TLS certificate will expire on 2026-04-29 21:45:00 +0000 UTC
2025-04-29 21:44:40.472 UTC 000a INFO [ledgermgmt] NewLedgerMgr -> Initializing
LedgerMgr
2025-04-29 21:44:40.718 UTC 000b INFO [ledgermgmt] NewLedgerMgr -> Initialized
LedgerMgr
2025-04-29 21:44:40.719 UTC 000c INFO [gossip.service] New -> Initialize gossip with
endpoint peer0.org2.example.com:9051
```

```
2025-04-29 21:44:40.720 UTC 000d INFO [gossip.gossip] New -> Creating gossip service with self membership of Endpoint: peer0.org2.example.com:9051, InternalEndpoint: peer0.org2.example.com:9051, PKI-ID: 7ec41501e222280b9df0eb91e8dafaaf4ef60e55799dfc95a28dbd747a3577b6, Metadata:
2025-04-29 21:44:40.721 UTC 000e INFO [gossip.gossip] start -> Gossip instance peer0.org2.example.com:9051 started
2025-04-29 21:44:40.721 UTC 000f INFO [lifecycle] InitializeLocalChaincodes -> Initialized lifecycle cache with 0 already installed chaincodes
2025-04-29 21:44:40.721 UTC 0010 INFO [nodeCmd] computeChaincodeEndpoint -> Entering computeChaincodeEndpoint with peerHostname: peer0.org2.example.com
2025-04-29 21:44:40.721 UTC 0011 INFO [nodeCmd] computeChaincodeEndpoint -> Exit with ccEndpoint: peer0.org2.example.com:9052
2025-04-29 21:44:40.722 UTC 0012 INFO [sccapi] DeploySysCC -> deploying system chaincode 'lscc'
2025-04-29 21:44:40.723 UTC 0013 INFO [sccapi] DeploySysCC -> deploying system chaincode 'cscc'
2025-04-29 21:44:40.723 UTC 0014 INFO [sccapi] DeploySysCC -> deploying system chaincode 'qscc'
2025-04-29 21:44:40.723 UTC 0015 INFO [sccapi] DeploySysCC -> deploying system chaincode '_lifecycle'
2025-04-29 21:44:40.723 UTC 0016 INFO [nodeCmd] serve -> Deployed system chaincodes
2025-04-29 21:44:40.723 UTC 0017 INFO [discovery] NewService -> Created with config TLS: true, authCacheMaxSize: 1000, authCachePurgeRatio: 0.750000
2025-04-29 21:44:40.723 UTC 0018 INFO [nodeCmd] serve -> Discovery service activated
2025-04-29 21:44:40.723 UTC 0019 INFO [nodeCmd] serve -> Starting peer with Gateway enabled
2025-04-29 21:44:40.723 UTC 001a INFO [nodeCmd] serve -> Starting peer with ID=[peer0.org2.example.com], network ID=[dev], address=[peer0.org2.example.com:9051]
2025-04-29 21:44:40.726 UTC 001b INFO [nodeCmd] serve -> Started peer with ID=[peer0.org2.example.com], network ID=[dev], address=[peer0.org2.example.com:9051]
2025-04-29 21:44:40.726 UTC 001c INFO [kvledger] LoadPreResetHeight -> Loading prereset height from path [/var/hyperledger/production/ledgersData/chains]
2025-04-29 21:44:40.726 UTC 001d INFO [blkstorage] preResetHtFiles -> No active channels passed
```

(*) Revisando la memoria, he debido de copiar mal el log, porque creo que si que he visto logs en modo “DEBUG”

**4. Los técnicos de Merca-link nos muestran su preocupación acerca de las claves criptográficas, que se almacenan en texto plano dentro de los directorios de la red. Como una primera medida de securización de las claves y de las operaciones**

**criptográficas sugieren el acoplamiento de un HSM a las CAs, utilizándose softhsm2 para esta fase de pruebas y configurando un token diferente para cada CA.**

Para las claves criptográficas seguimos el procedimiento visto en clase ( Doc referencia Modulo Certificación HF - Presentación)

- Paso 1: reconstrucción de imágenes de Docker de HF
- Paso 2: instalación de nuestro Soft HSM
- Paso 3: configuración de nodos para usar HSM
- Paso 4: almacenamiento de claves en HSM con Fabric CA

**5. Así mismo, explícales cómo configurar y utilizar el fabric-ca-client con el softHSM configurado. Registra, a modo de prueba, un usuario de nombre "merca-admin" y contraseña "merka-12345", que tenga capacidad para registrar usuarios clientes y nuevos peers.**

Configuración y Uso del fabric-ca-client con softHSM

1. Instalar y Configurar el softHSM:
2. Configurar el fabric-ca-client para usar PKCS#11 (softHSM):
3. Utilizar el fabric-ca-client:

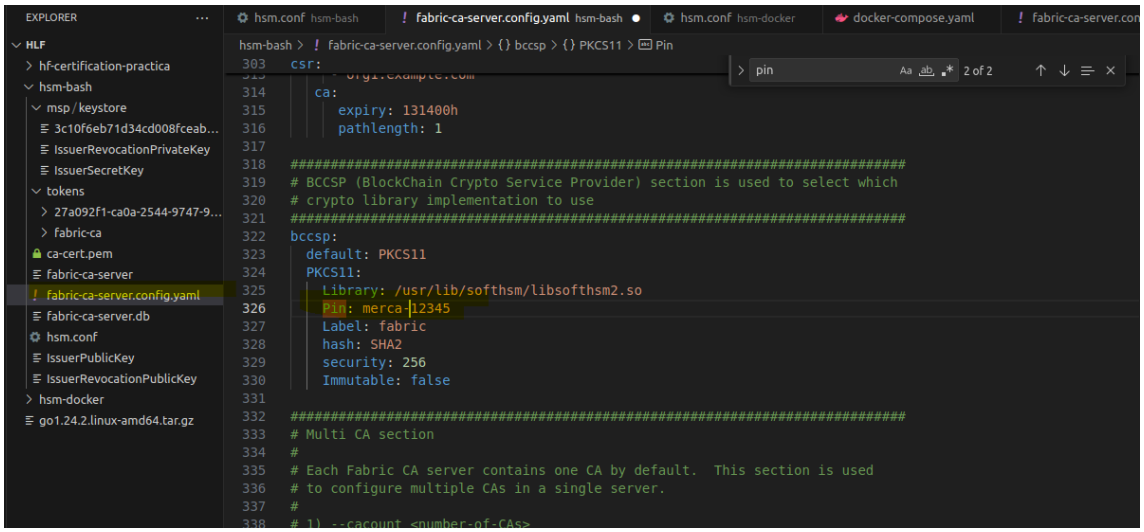
Registro del Usuario "merca-admin"

Luego, vamos a registrar el usuario "merca-admin" con la capacidad de registrar usuarios clientes y nuevos peers. Asumimos que ya tienes un fabric-ca-server en funcionamiento

1. **Ejecutar el comando de registro.** Utilizando el comando fabric-ca-client register para registrar al nuevo usuario. Necesitaremos una identidad existente con los permisos necesarios para registrar nuevos usuarios (generalmente la identidad del administrador de la CA).

```
fabric-ca-client register --id.name merca-admin --id.secret merka-12345 --id.type client --role client --attrs 'hf.Registrar.Roles=client,peer;hf.Registrar.DelegateRoles=client,peer' --config fabric-ca-client-config.yaml
```

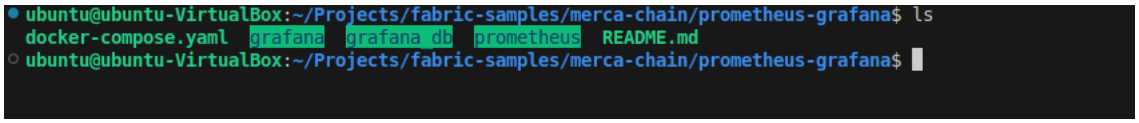
Una vez configurado, actualizamos y verificamos tambié el fichero *fabric-ca-server.config.yaml.*



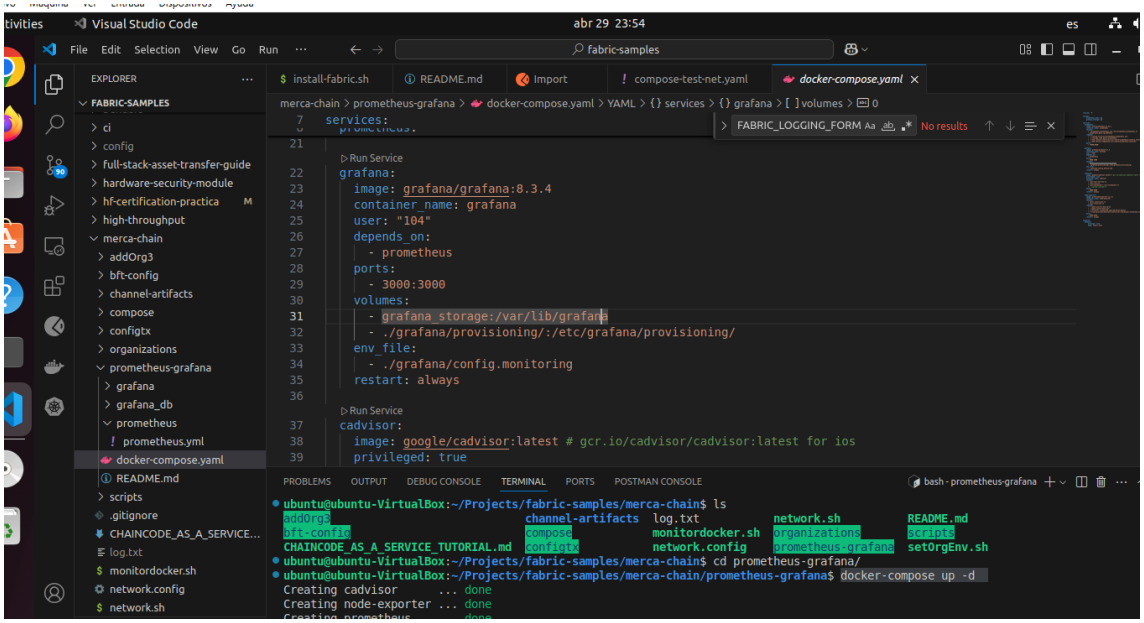

**6. Los merca-ingenieros de Merca-link consideran que tener herramientas de monitorización a su disposición agilizaría estas tareas de administración en futuras ocasiones. Despliega sendas instancias de Prometheus y Grafana para satisfacer estas necesidades.**

Ahora vamos a probar a monitorizar la red de Merca-link. Para poder desplegar Prometheus y Grafana.

En primer lugar, nos situamos en la carpeta:




Una vez configurados los puertos de grafana y prometheus sobre el fichero Docker-compose.yaml



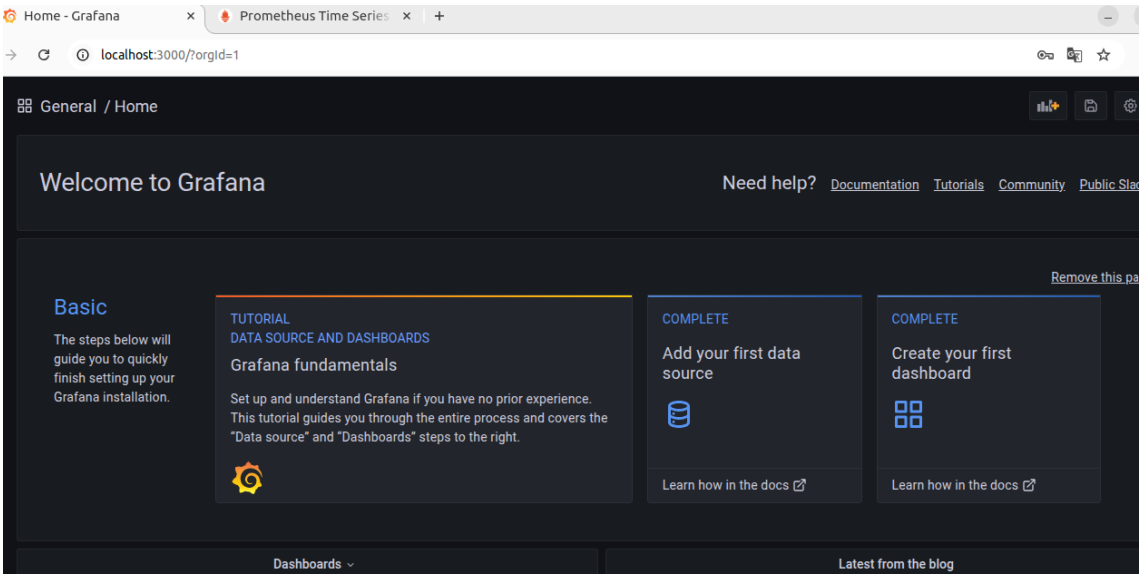
Ejecutamos el comando

```
docker-compose up -d
```

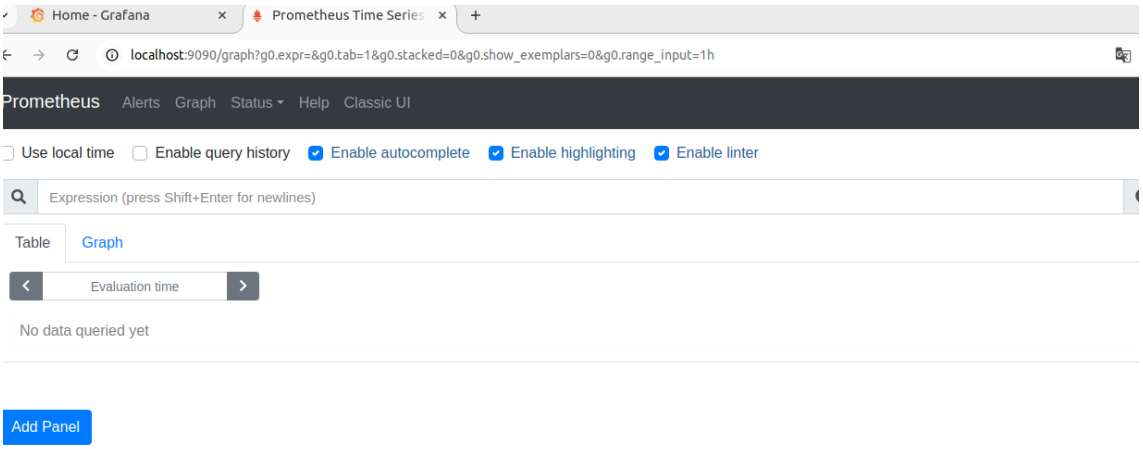
```
ubuntu@ubuntu-VirtualBox:~/Projects/fabric-samples/merca-chain/prometheus-grafana$ docker-compose up -d
Creating cadvisor      ... done
Creating node-exporter ... done
Creating prometheus    ... done
Creating grafana       ... done
ubuntu@ubuntu-VirtualBox:~/Projects/fabric-samples/merca-chain/prometheus-grafana$
```

Verificamos sobre el localhost si visualizamos el cuadro de mando de:

Grafana -> ok



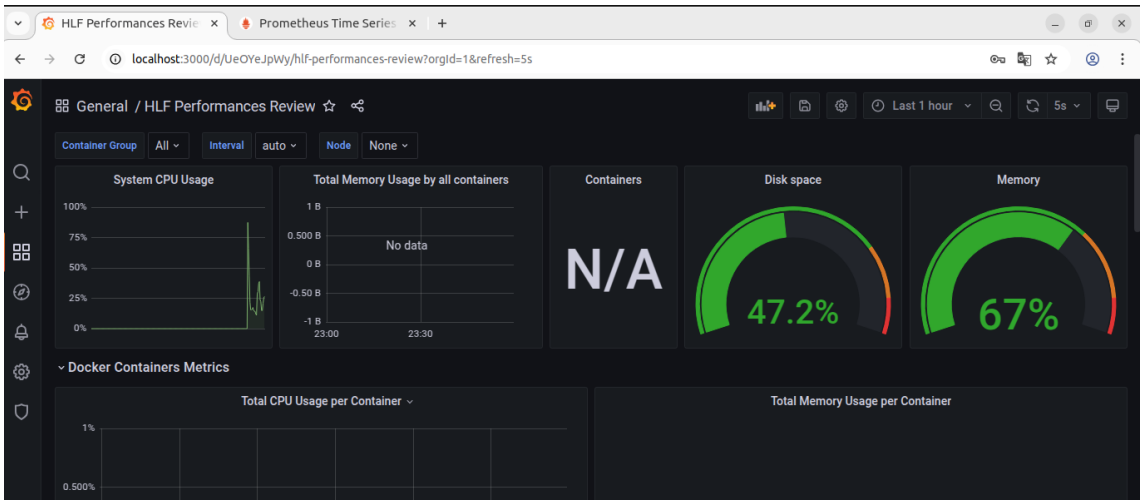

Prometeus -> ok




Ambas herramientas de monitorización se muestran correctamente.

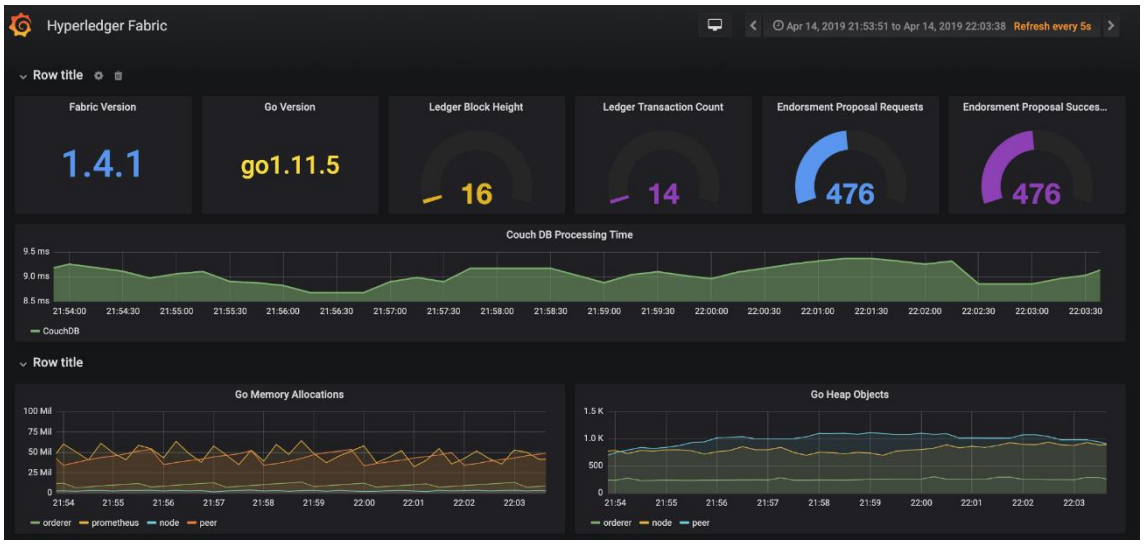
Investigamos un poco y nos movemos sobre las herramientas.

Pej, este es el dashboard que viene por defecto:

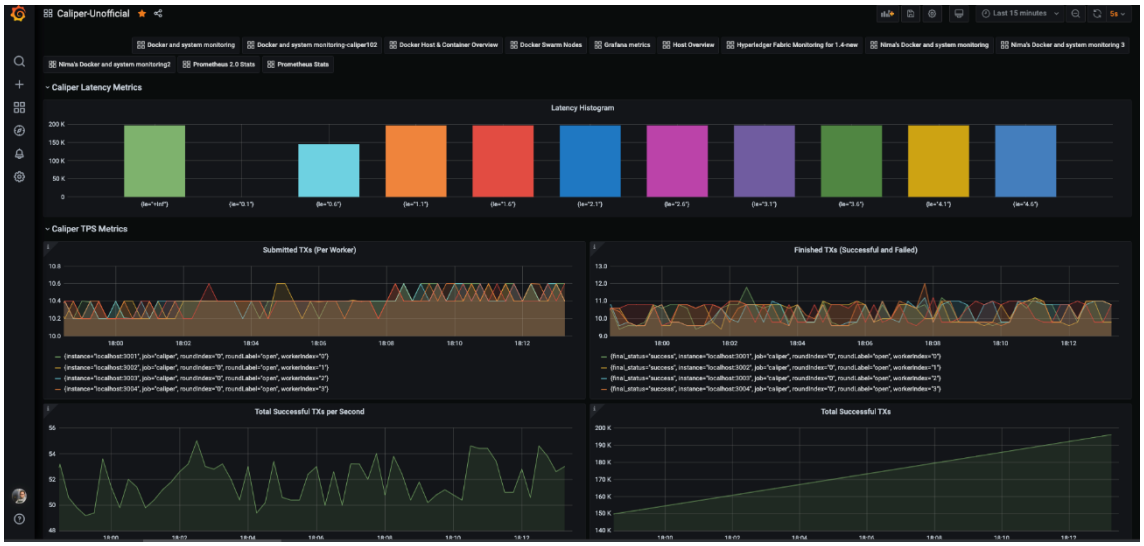



O también podríamos utilizar estos dos dashboard realizados por la comunidad y que se pueden importar en la herramienta:

https://grafana.com/grafana/dashboards/10716-hyperledger-fabric/




https://grafana.com/grafana/dashboards/13405-caliper-unofficial/




**7. Investiga sobre los despliegues distribuidos (esto es, en distintos nodos) de Hyperledger Fabric, y resume en unas pocas líneas tus recomendaciones y los principales puntos a tener en cuenta para que Merka-link despliegue su red de forma distribuida. No olvides referencias los artículos/blogs/páginas web que has consultado.**

(*) Ayuda con Gemini y ChatGPT. Respuesta revisada, editada e incluyendo cambios personales

**Recomendaciones principales:**

1. Desplegar orderers y peers en múltiples nodos (idealmente en distintas ubicaciones) para alta disponibilidad.
2. Utilizar una red de comunicación robusta y segura (VPNs/conexiones dedicadas) con TLS habilitado.
3. Planificar la gestión de identidades (CA) para un entorno distribuido.
4. Seleccionar un mecanismo de consenso tolerante a fallos (Raft)
5. Implementar medidas de seguridad perimetral en cada nodo.
6. Considerar el uso de Kubernetes para la orquestación y escalabilidad.

**Puntos clave a tener en cuenta:**

1. La configuración y gestión serán más complejas.
2. Costes de infraestructura
3. Asegurar la consistencia y robustez de los datos

**Referencias:**

- **Hyperledger Fabric Documentation on Deployment:** https://hyperledger-fabric.readthedocs.io/en/latest/deployment_guide_overview.html
- **IBM Blockchain Platform Documentation on Multi-Org Networks:** (conceptos aplicables) https://www.google.com/search?q=https://cloud.ibm.com/docs/blockchain%3Ftopic%3Dblockchain-ibp-multiorg
- **Artículos y blogs técnicos sobre despliegues distribuidos de Fabric**
- **Gemini**

# 3. Conclusiones

Con la realización de esta práctica, he podido aprender y repasar casi todos los conceptos que hemos visto en clase. Me ha parecido muy interesante, aunque por falta de tiempo no he podido profundizar todo lo que me hubiera gustado en algunos puntos. Sin embargo, considero que el análisis que se ha realizado es bastante completo y gracias a estos ejercicios he podido seguir ampliando mis conocimientos sobre el manejo de fabric.