

# Programación en Blockchain II

## Ejercicios



Universidad  
del País Vasco

Euskal Herriko  
Unibertsitatea

Sergio Anguita Lorenzo @MrSergioAnguita

Ejercicio 1

# Entorno de desarrollo

# Entorno de desarrollo (docker)

Se te pide:

1. Instalar maquina virtual ubuntu
2. Instalar docker 19.X:

```
DOCKER_VERSION="5:19.03.4~3-0~ubuntu-xenial"  
sudo apt-get update  
sudo apt-get install -y apt-transport-https ca-certificates curl  
software-properties-common  
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  
sudo apt-get update  
sudo apt-key fingerprint 0EBFCD88  
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu  
xenial stable"  
sudo apt-get update  
sudo rm -rf /etc/docker/daemon.json  
sudo rm -rf /etc/default/docker  
sudo apt-get install -y docker-ce=$DOCKER_VERSION --reinstall --allow-downgrades
```

# Entorno de desarrollo (docker cont.)

Después de instalar Docker v19, verifica que está correctamente instalado con los siguientes comandos


```
sudo groupadd docker
sudo usermod -a -G docker $USER
echo "Setting docker to start on system boot..."
sudo systemctl enable docker
echo "Restarting docker service..."
sudo service docker restart
sudo newgrp docker
```

# Entorno de desarrollo (docker cont.)

Verifica que la instalación se ha completado correctamente con `docker version`

```
Client: Docker Engine - Community
Version:      20.10.6
API version:  1.40
Go version:   go1.13.15
Git commit:   370c289
Built:        Fri Apr  9 22:46:01 2021
OS/Arch:      linux/amd64
Context:      default
Experimental: true

Server: Docker Engine - Community
Engine:
Version:      19.03.4
API version:  1.40 (minimum version 1.12)
Go version:   go1.12.10
Git commit:   9013bf583a
Built:        Fri Oct 18 15:52:23 2019
OS/Arch:      linux/amd64
Experimental: false
containerd:
Version:      1.4.4
GitCommit:    05f951a3781f4f2c1911b05e61c160e9c30eaa8e
runc:
Version:      1.0.0-rc93
GitCommit:    12644e614e25b05da6fd08a38ffa0cfe1903fdec
docker-init:
Version:      0.18.0
GitCommit:    fec3683
```



# Entorno de desarrollo (docker cont.)

Haz un hello-world de prueba

```
docker run hello-world
```

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:f2266cbfc127c960fd30e76b7c792dc23b588c0db76233517e1891a4e357d519
Status: Downloaded newer image for hello-world:latest
```

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:  
\$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:  
<https://hub.docker.com/>

For more examples and ideas, visit:  
<https://docs.docker.com/get-started/>

# Entorno de desarrollo (docker-compose)

Instalar docker-compose: <https://docs.docker.com/compose/install/>

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.1/docker-compose-$(uname
-s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

Finalmente, verifica que docker-compose se ha instalado correctamente con el comando

```
docker-compose --version
```

```
docker-compose version 1.25.5, build 8a1c60f6
docker-py version: 4.1.0
CPython version: 3.7.5
OpenSSL version: OpenSSL 1.1.0l 10 Sep 2019
```

# Entorno de desarrollo (go)

Instalar go: <https://golang.org/doc/install>

```
wget https://golang.org/dl/go1.16.3.linux-amd64.tar.gz
sudo su
rm -rf /usr/local/go && tar -C /usr/local -xzf go1.16.3.linux-amd64.tar.gz
export PATH=$PATH:/usr/local/go/bin
```

Completa la instalación añadiendo a línea "export..." al fichero \$HOME/.profile  
Finalmente, verifica que go se ha instalado correctamente con el comando go version

```
go version
```

```
go version go1.16.3 linux/amd64
```



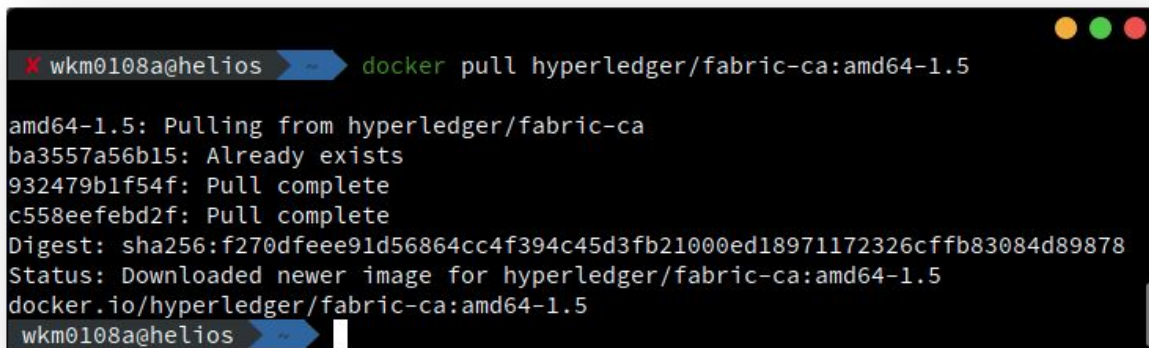
Ejercicio 2

# Creando una CA de fabric

# Crear una CA

Todas las CAs de Hyperledger fabric se basan en <https://github.com/hyperledger/fabric-ca>  
Para usar el código fuente, en vez de tener que descargarlo, compilarlo y demás, tenemos disponibles imágenes Docker de fabric-ca en [https://hub.docker.com/r/hyperledger/fabric-ca/tags?page=1&ordering=last\\_updated](https://hub.docker.com/r/hyperledger/fabric-ca/tags?page=1&ordering=last_updated)

```
docker pull hyperledger/fabric-ca:amd64-1.5
```



```
wkm0108a@helios ➜ docker pull hyperledger/fabric-ca:amd64-1.5

amd64-1.5: Pulling from hyperledger/fabric-ca
ba3557a56b15: Already exists
932479b1f54f: Pull complete
c558eefebd2f: Pull complete
Digest: sha256:f270dfeee91d56864cc4f394c45d3fb21000ed18971172326cffb83084d89878
Status: Downloaded newer image for hyperledger/fabric-ca:amd64-1.5
docker.io/hyperledger/fabric-ca:amd64-1.5
wkm0108a@helios ➜
```

# Crear una CA

Ahora que ya tenemos el código descargado lo podemos ejecutar en nuestro entorno de desarrollo.

```
docker run -it hyperledger/fabric-ca:amd64-1.5
```

```
wkm0108a@helios ➤ docker run -it hyperledger/fabric-ca:amd64-1.5
2021/04/19 07:49:16 [INFO] Created default configuration file at /etc/hyperledger/fabric-ca-server/fabric-ca-server-config.yaml
2021/04/19 07:49:16 [INFO] Starting server in home directory: /etc/hyperledger/fabric-ca-server
2021/04/19 07:49:16 [INFO] Server Version: 1.5.0
2021/04/19 07:49:16 [INFO] Server Levels: &{Identity:2 Affiliation:1 Certificate:1 Credential:1 RInfo:1 Nonce:1}
2021/04/19 07:49:16 [WARNING] &{69 The specified CA certificate file /etc/hyperledger/fabric-ca-server/ca-cert.pem does not exist}
2021/04/19 07:49:16 [INFO] generating key: &{A:ecdsa S:256}
2021/04/19 07:49:16 [INFO] encoded CSR
2021/04/19 07:49:16 [INFO] signed certificate with serial number 154062628788462411910499855170348390983364982685
2021/04/19 07:49:16 [INFO] The CA key and certificate were generated for CA
2021/04/19 07:49:16 [INFO] The key was stored by BCCSP provider 'SW'
2021/04/19 07:49:16 [INFO] The certificate is at: /etc/hyperledger/fabric-ca-server/ca-cert.pem
2021/04/19 07:49:16 [INFO] Initialized sqlite3 database at /etc/hyperledger/fabric-ca-server/fabric-ca-server.db
2021/04/19 07:49:17 [INFO] The issuer key was successfully stored. The public key is at: /etc/hyperledger/fabric-ca-server/IssuerPublicKey, secret key is
at: /etc/hyperledger/fabric-ca-server/msp/keystore/IssuerSecretKey
2021/04/19 07:49:17 [INFO] Idemix issuer revocation public and secret keys were generated for CA ''
2021/04/19 07:49:17 [INFO] The revocation key was successfully stored. The public key is at: /etc/hyperledger/fabric-ca-server/IssuerRevocationPublicKey,
private key is at: /etc/hyperledger/fabric-ca-server/msp/keystore/IssuerRevocationPrivateKey
2021/04/19 07:49:17 [INFO] Home directory for default CA: /etc/hyperledger/fabric-ca-server
2021/04/19 07:49:17 [INFO] Operation Server Listening on 127.0.0.1:9443
2021/04/19 07:49:17 [INFO] Listening on http://0.0.0.0:7054
```

# Crear una CA

Para poder acceder desde <http://127.0.0.1:7054>, es necesario añadir un parámetro extra

```
docker run -it -p 7054:7054 hyperledger/fabric-ca:amd64-1.5
```

Internamente, el parámetro `-p 7054:7054`, está creando una regla de IPTABLES, para redirigir el tráfico TCP del contenedor docker a nuestra interfaz loopback. Dicho de otra forma, nos permite conectarnos al servicio (fabric-ca) que está funcionando en el puerto 7054 de forma fácil.

Abre el navegador y visita: <http://localhost:7054/api/v1/cainfo>

# Crear una CA (defaults)

El contenedor de fabric-ca incluye por defecto, la siguiente configuración:

```
ENV FABRIC_CA_HOME /etc/hyperledger/fabric-ca-server
```

Y se inicia con el superusuario admin y la contraseña adminpw.

Esta configuración se puede ver en el fichero Dockerfile  
<https://github.com/hyperledger/fabric-ca/blob/main/images/fabric-ca/Dockerfile>  
que sirve como base para la creación de la imagen que estamos usando.

# Crear una CA (superuser)

La forma más fácil de sobrescribir la configuración por defecto, es mediante un fichero `docker-compose.yml` que tenga toda nuestra configuración.

```
fabric-ca-server:  
  image: hyperledger/fabric-ca:amd64-1.5  
  container_name: fabric-ca-server  
  ports:  
    - "7054:7054"  
  environment:  
    - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server  
  volumes:  
    - "./fabric-ca-server:/etc/hyperledger/fabric-ca-server"  
  command: sh -c 'fabric-ca-server start -b my_admin:my_adminpw'
```

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

# Crear una CA 100% personalizada

Si con todo lo anterior no tienes suficiente, porque necesitas personalizar más parámetros de la CA, existe el fichero llamado `fabric-ca-server.yml` que contiene toda la configuración parametrizable por el usuario.

El orden de prioridad es el siguiente:

The Fabric CA provides 3 ways to configure settings on the Fabric CA server and client. The precedence order is:

1. CLI flags
2. Environment variables
3. Configuration file

# ¿Que ocurre cuando se inicia la CA?

Internamente la CA ejecuta los siguientes pasos:

1. Crea el usuario admin si no existe en base a la configuración dada y generate los ficheros ca-cert.pem y ca-cert.key.
2. Inicia el servicio en el puerto definido

```
wkm0108a@helios ➔ docker run -it hyperledger/fabric-ca:amd64-1.5
2021/04/19 07:49:16 [INFO] Created default configuration file at /etc/hyperledger/fabric-ca-server/fabric-ca-server-config.yaml
2021/04/19 07:49:16 [INFO] Starting server in home directory: /etc/hyperledger/fabric-ca-server
2021/04/19 07:49:16 [INFO] Server Version: 1.5.0
2021/04/19 07:49:16 [INFO] Server Levels: &{Identity:2 Affiliation:1 Certificate:1 Credential:1 RInfo:1 Nonce:1}
2021/04/19 07:49:16 [WARNING] &{69 The specified CA certificate file /etc/hyperledger/fabric-ca-server/ca-cert.pem does not exist}
2021/04/19 07:49:16 [INFO] generating key: &{A:ecdsa S:256}
2021/04/19 07:49:16 [INFO] encoded CSR
2021/04/19 07:49:16 [INFO] signed certificate with serial number 154062628788462411910499855170348390983364982685
2021/04/19 07:49:16 [INFO] The CA key and certificate were generated for CA
2021/04/19 07:49:16 [INFO] The key was stored by BCCSP provider 'SW'
2021/04/19 07:49:16 [INFO] The certificate is at: /etc/hyperledger/fabric-ca-server/ca-cert.pem
2021/04/19 07:49:16 [INFO] Initialized sqlite3 database at /etc/hyperledger/fabric-ca-server/fabric-ca-server.db
2021/04/19 07:49:17 [INFO] The issuer key was successfully stored. The public key is at: /etc/hyperledger/fabric-ca-server/IssuerPublicKey, secret key is
at: /etc/hyperledger/fabric-ca-server/msp/keystore/IssuerSecretKey
2021/04/19 07:49:17 [INFO] Idemix issuer revocation public and secret keys were generated for CA ''
2021/04/19 07:49:17 [INFO] The revocation key was successfully stored. The public key is at: /etc/hyperledger/fabric-ca-server/IssuerRevocationPublicKey,
private key is at: /etc/hyperledger/fabric-ca-server/msp/keystore/IssuerRevocationPrivateKey
2021/04/19 07:49:17 [INFO] Home directory for default CA: /etc/hyperledger/fabric-ca-server
2021/04/19 07:49:17 [INFO] Operation Server Listening on 127.0.0.1:9443
2021/04/19 07:49:17 [INFO] Listening on http://0.0.0.0:7054
```



# Ejercicio 3 User Enrollment

# CA User Enrollment

En Fabric, toda interacción con la red está permissionada en base a una identidad digital (certificado X509). Para obtenerlo, hay que interactuar con la CA.

1. Obtén la IP del contendor de la CA
2. Mediante el uso de la utilidad `fabric-ca-client`, ejecuta la operación `enroll`
3. Revisa que se te ha creado una clave privada correctamente.

## **docker-compose.yml**

```
fabric-ca-server:
  image: hyperledger/fabric-ca:amd64-1.5
  container_name: fabric-ca-server
  ports:
    - "7054:7054"
  environment:
    -
  FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
  command: sh -c 'fabric-ca-server start -b
  admin:adminpw'

fabric-ca-client:
  image: hyperledger/fabric-ca:amd64-1.5
  container_name: fabric-ca-client
  environment:
    -
  FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
```

# CA User Enrollment (solución)

En Fabric, toda interacción con la red está permitida en base a una identidad digital (certificado X509). Para obtenerlo, hay que interactuar con la CA.

1. Obtén la IP del contenedor de la CA

```
docker inspect fabric-ca-server | grep IPAddress
```

2. Mediante el uso de la utilidad fabric-ca-client, ejecuta la operación enroll

```
docker exec -it fabric-ca-client sh
```

```
fabric-ca-client enroll -u http://admin:adminpw@192.168.20.2:7054
```

3. Revisa que se te ha creado una clave privada correctamente.

```
cat /etc/hyperledger/fabric-ca-server/msp/keystore/*_sk
```

```
-----BEGIN PRIVATE KEY-----  
MIGHAgEAMBMGBYqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgnBLH8pHtB/NMmOF8  
X5R41R7r89Fh1U/uFp6igh/bmqhRANCAASvYryjto0cPwIKHAgtXSh9hjrtrs2  
Li/1kUUsCQaqK1sCsRXWayn52B86+0qiiFID8rICVSYavk3HP9UIST+1  
-----END PRIVATE KEY-----
```

Ejercicio 4

# User Registering

# CA Register New User

La CA de fabric, también nos permite registrar usuarios en función de las necesidades que tengamos.

Pasos:

1. Inicia sesión con el usuario establecido como superadmin
2. Crea un nuevo usuario
3. Inicia sesión con el nuevo usuario creado

**Llamamos 'iniciar sesión' al proceso de obtener la identidad digital de ese usuario en Fabric, es decir, su certificado digital y su clave privada. Recuerda que no existe el concepto de sesión como tal.**

# CA Register New User (solución)

La CA de fabric, también nos permite registrar usuarios en función de las necesidades que tengamos.

**Obligatorio haber realizado “enroll” con el usuario admin o con un usuario que tenga el rol de REGISTRAR, para poder registrar nuevos usuarios**

```
fabric-ca-client register --id.name sergio --id.secret 123456789 --id.affiliation  
org1.department1 --id.attrs 'hf.Revoker=true,admin=true:ecert'
```

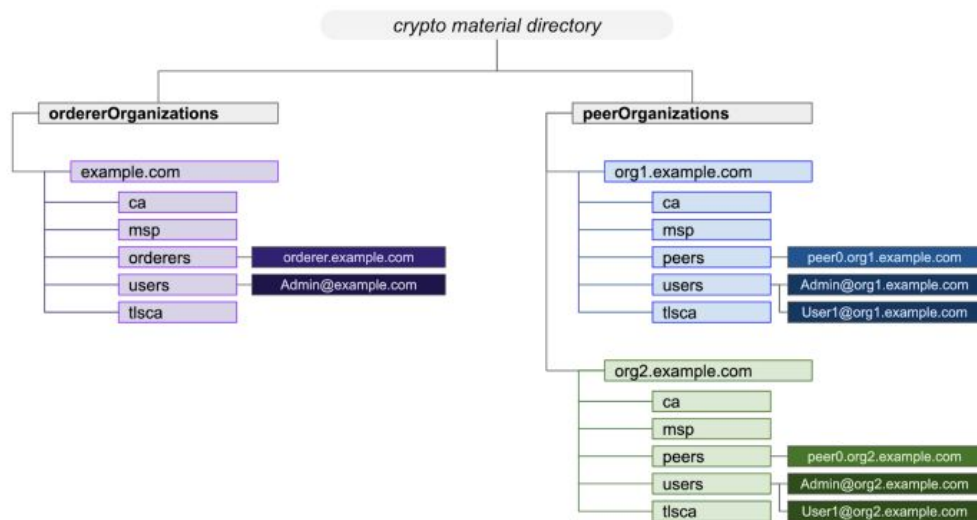
```
[INFO] Configuration file location: /etc/hyperledger/fabric-ca-server/fabric-ca-client-config.yaml  
Password: 123456789
```

```
fabric-ca-client enroll -u http://sergio:123456789@192.168.20.2:7054
```

Ejercicio 5

# Create the crypto-stuff

# Running cryptogen





# Running cryptogen

Fabric tiene algunas utilidades para hacernos la vida más fácil. Una de ellas es cryptogen. Una aplicación que nos genera en base a una configuración dada, el árbol completo de certificados y claves necesarias para desplegar una red usando el resultado de cryptogen como MSP (estático). La herramienta cryptogen la podéis encontrar en:

```
docker pull hyperledger/fabric-tools:1.4.11
docker run -it --entrypoint=sh -v $(pwd)/volumes/tmp:/tmp hyperledger/fabric-tools:amd64-1.4.11
cryptogen
```

```
usage: cryptogen [<flags>] <command> [<args> ...]

Utility for generating Hyperledger Fabric key material

Flags:
  --help Show context-sensitive help (also try --help-long and --help-man).

Commands:
  help [<command>...]
    Show help.
  generate [<flags>]
    Generate key material
  showtemplate
    Show the default configuration template
  version
    Show version information
  extend [<flags>]
    Extend existing network
```

# Running cryptogen

Si ejecutas cryptogen sin indicar ningún fichero, se creará la configuración de ejemplo

```
cryptogen generate --output="crypto-config"
```

```
org1.example.com  
org2.example.com
```

**¿Que contiene la carpeta crypto-config después de ejecutar el comando?**

Ahora vuelve a ejecutar cryptogen para que el resultado sea:

- 3 organizaciones que se llamen: bat.ehu.eus, bi.ehu.eus, hiru.ehu.eus
- 5 usuarios por defecto, en cada organización anterior

Nota: para ello tienes que usar un fichero config.yaml personalizado

```
cryptogen generate --output="crypto-config" --config=config.yaml
```

**Recuerda que el fichero config.yaml es el que define la plantilla de contenido a generar**

# Running cryptogen

¿Que arbol MSP obtienes como resultado?

```
echo "para mostrar el arbol de ficheros instalaremos tree"
apt install tree

echo "tree se usa asi"
tree ./crypto-config

echo "para borrar la carpeta crypto-config y generar una nueva usamos"
rm -rf ./crypto-config

echo "para mostrar los ficheros existentes en el directorio actual"
ls -alh

echo "para buscar todas las claves privadas"
find . -name "*_sk"

echo "para instalar bash"
apt install bash

echo "para entrar en bash"
bash
```

# Running cryptogen

Consejo: renombra los fichero \*\_sk a una constante, por ejemplo 'key'

```
echo "crea un script llamado update_keynames.sh"
touch update_keynames.sh
```

```
echo "añade el siguiente contenido"
```

```
echo "#!/bin/bash
```

```
echo "converting filenames to 'key'"
```

```
files=$(find . -name "*_sk")
```

```
for f in $files
```

```
do
```

```
echo $f
```

```
base=$(dirname $f)
```

```
mv $f ${base}/key
```

```
done" > update_keynames.sh
```

```
echo "dale permisos de RWX"
```

```
sudo chmod +rwx ./update_keynames.sh
```

```
echo "ejecutalo"
```

```
sudo ./update_keynames.sh
```

## Ejercicio 6

# Enable TLS on CA

# Enable TLS on CA

Partiendo del ejercicio anterior donde se despliega una CA

- Aplica los cambios de configuración necesarios para habilitar conexiones seguras con TLS en el fichero `docker-compose.yml` y relanza el contenedor.



# Enable TLS on CA (solución)

Partiendo del ejercicio anterior donde se despliega una CA

- aplica los cambios de configuración necesarios para habilitar conexiones seguras con TLS.

## **docker-compose.yml**

```
fabric-ca-server:  
  image: hyperledger/fabric-ca:amd64-1.5  
  environment:  
    - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server  
    - FABRIC_CA_SERVER_CA_NAME=fabric-ca-server  
    - FABRIC_CA_SERVER_TLS_ENABLED=true  
    - FABRIC_CA_SERVER_PORT=7054  
  ports:  
    - "7054:7054"  
  command: sh -c 'fabric-ca-server start -b admin:adminpw --tls.enabled -d'
```

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

# Enable TLS on CA (solución)

En caso de querer usar certificados personalizados, la configuración varía ligeramente debido a que hay que indicar el certificado y la clave a usar

## **docker-compose.yml**

```
ca.org1.example.com:
  image: hyperledger/fabric-ca:amd64-1.5
  environment:
    - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
    - FABRIC_CA_SERVER_CA_NAME=ca.org1.example.com
    - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-ca-server-config/ca.org1.example.com-cert.pem
    - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-config/key
  ports:
    - "7054:7054"
  command: sh -c 'fabric-ca-server start -b admin:adminpw --tls.enabled -d'
  volumes:
    - ./crypto-config/peerOrganizations/org1.example.com/ca/:/etc/hyperledger/fabric-ca-server-config
  container_name: ca.org1.example.com
```

**Recuerda que tienes que ejecutar `cryptogen` para generar la carpeta `crypto-config` y todo su contenido**



# My First Basic Network

```
peer chaincode instantiate --name "example" --version 1 --channelID  
mychannel --ctor "{\"Function\":\"init\", \"Args\": [\"a\", \"2\",  
\"b\", \"3\"]}"
```

Ejercicio 7

# Booting an orderer in solo mode

# Ordering Service | Orderer (solución)

Primero, se descargan las imagenes si no existen ya

```
docker pull hyperledger/fabric-orderer:amd64-1.4.11
```

```
docker pull hyperledger/fabric-orderer:amd64-2.4
```

Para ejecutar el ordering service en su forma más simple basta con:

```
docker run -it hyperledger/fabric-orderer:amd64-1.4.11
```

# Ordering Service | Orderer (solución)

Pero si queremos usar una configuración personalizada (99% de las veces), tenemos que configurar un `docker-compose.yml` para ello.

## **docker-compose.yml**

```
orderer.example.com:
  container_name: orderer.example.com
  image: hyperledger/fabric-orderer:amd64-1.4.11
  environment:
    - FABRIC_LOGGING_SPEC=info
    - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
    - ORDERER_GENERAL_GENESISMETHOD=file
    - ORDERER_GENERAL_GENESISFILE=/etc/hyperledger/configtx/genesis.block
    - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
    - ORDERER_GENERAL_LOCALMSPDIR=/etc/hyperledger/msp/orderer/msp
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/orderer
  command: orderer
  ports:
    - 7050:7050
  volumes:
    - ./config/:/etc/hyperledger/configtx
    - ./crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/:/etc/hyperledger/msp/orderer
```

**Recuerda que necesitas tener el bloque génesis para poder desplegar el orderer en versiones 2.X**

Ejercicio 8

# Create a peer

# Creating the peers

```
peer0.org1.example.com:
  container_name: peer0.org1.example.com
  image: hyperledger/fabric-peer
  environment:
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_PEER_ID=peer0.org1.example.com
    - FABRIC_LOGGING_SPEC=info
    - CORE_CHAINCODE_LOGGING_LEVEL=info
    - CORE_PEER_LOCALMSPID=Org1MSP
    - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/peer/
    - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_basic
    - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
    #- CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb:5984
    #- CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=
    #- CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric
  command: peer node start
  # command: peer node start --peer-chaincodedev=true
  ports:
    - 7051:7051
    - 7053:7053
  volumes:
    - /var/run:/host/var/run/
    - ./crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp:/etc/hyperledger/msp/peer
    - ./crypto-config/peerOrganizations/org1.example.com/users:/etc/hyperledger/msp/users
    - ./config:/etc/hyperledger/configtx
  depends_on:
    - orderer.example.com
    - couchdb
  networks:
    - basic
```

Ejercicio 9

# My first basic network

# My First Basic Network

1. Toma como punto el zip compartido 'basic-network.zip' y descargalo.
2. Extraelo y abre el README.md y sigue las instrucciones para desplegar la red de ejemplo
3. Si todo ha ido bien deberías tener funcionando:
  - a. 1 CA, 1 Orderer y 1 Peer
4. Si todo ha ido bien, ahora podrás ejecutar `run_cli.sh` para ejecutar un contenedor que actúe como cliente y empezar a lanzar comandos.
5. Y puedes empezar a lanzar comandos
6. `peer node status`
7. `peer channel list`
8. `peer channel getinfo -c mychannel`
9. `peer channel fetch oldest -c mychannel`
10. `peer chaincode list -C mychannel --installed`
11. `peer chaincode list -C mychannel --instantiated`
12. `peer chaincode install --help`
13. `peer chaincode install --name "example" --version 1 --path "github.com"`
14. `peer chaincode instantiate --name "example" --version 1 --channelID mychannel --ctor "{\"Function\":\"init\", \"Args\": []}"`



# Ejercicio 10

# Chaincode APIs

# Chaincode APIs

Dependiendo de la versión de Fabric que se use, se podrán usar unas APIs u otras

Fabric < 2.2	Fabric >= 2.2 (July 9, 2020)
Chaincode Stub Interface	Fabric Chaincode API
<a href="https://hyperledger-fabric.readthedocs.io/en/release-2.0/chaincode4ade.html#chaincode-api">https://hyperledger-fabric.readthedocs.io/en/release-2.0/chaincode4ade.html#chaincode-api</a>	<a href="https://hyperledger-fabric.readthedocs.io/en/release-2.2/chaincode4ade.html#fabric-contract-api">https://hyperledger-fabric.readthedocs.io/en/release-2.2/chaincode4ade.html#fabric-contract-api</a>
Métodos básicos para interactuar con la red	Capa adicional que da cierta abstracción sobre los métodos básicos.

# Chaincode Example 1

Descarga el siguiente chaincode de ejemplo y ábrelo con el editor de código favorito que tengas

<https://github.com/hyperledger-archives/education/blob/master/LFS171X/fabric-material/chaincode/sample-chaincode.go>

- ¿Qué hace este smart contract?
- ¿Que APIs esta usando este smart contract?
- ¿Para que version de fabric está orientado?
- ¿Cuántas operaciones declara este smart contract? ¿Como se llaman?
- ¿Cómo se debería invocar a cada una de las funciones de este smart contract?

# Chaincode Example 2

Descarga el siguiente chaincode de ejemplo y ábrelo con el editor de código favorito que tengas

<https://github.com/hyperledger-archives/education/blob/master/LFS171X/fabric-material/chaincode/tuna-app/tuna-chaincode.go>

- ¿Qué hace este smart contract?
- ¿Que APIs esta usando este smart contract?
- ¿Para que version de fabric está orientado?
- ¿Cuántas operaciones declara este smart contract? ¿Como se llaman?
- ¿Cómo se debería invocar a cada una de las funciones de este smart contract?

Ejercicio 11

# Calling the chaincode

# Calling the chaincode

Tomando como base el ejercicio ‘My First Network’ usaremos el smart contract de ese ejemplo para hacer operaciones de lectura y escritura.

Para ello:

1. Arrancaremos la red ‘basic-network’ si no la tenemos ya arrancada.
2. Instalaremos el smart contract, si no está ya instalado.

Una vez hecho eso, podremos hacer operaciones de lectura (query) y operaciones de escritura (invoke)

Las lecturas, se harán con el comando `peer chaincode query`

# Calling the chaincode (solución)

## query operation example

```
peer chaincode query -C mychannel --name example --ctor "{\"Function\":\"query\",  
\"Args\": [\"a\"]}"
```

## invoke operation example

```
peer chaincode invoke -C mychannel --name example --ctor "{\"Function\":\"invoke\",  
\"Args\": [\"a\", \"b\", \"1\"]}"
```

**Comprobad que el resultado del chaincode es correcto, leyendo de nuevo los valores de 'a' y 'b'**

Ejercicio 12

# Chaincode Designing



# Chaincode Designing

Sobre el chaincode de ejemplo del otro día, en este ejercicio se pide:

- Objetivo: crear un chaincode que tenga 2 operaciones: una de lectura y otra de escritura.
  - La operación de lectura, recibe un parámetro que será el ID del contenido a leer y lo devolverá.
  - La operación de escritura, recibe dos parámetros que serán, el ID del contenido y el contenido en sí.
- En este ejemplo, no se quiere realizar ninguna operación cuando se instancia el smart contract, que tendríamos que modificar?
- ¿Que cambios hay que realizar en el chaincode para llegar a este resultado?

Ejercicio 13

# Hyperledger Explorer

# Hyperledger Explorer

Hyperledger Explorer is a user-friendly Web application tool used to view, invoke, deploy or query blocks, transactions and associated data, network information (name, status, list of nodes), chain codes and transaction families, as well as any other relevant information stored in the ledger.

<https://github.com/hyperledger/blockchain-explorer>

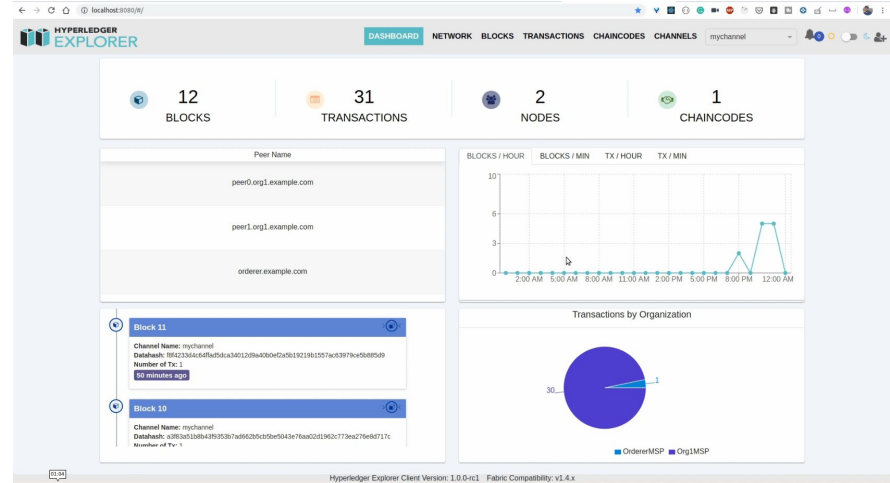
The screenshot shows the GitHub repository page for `hyperledger/blockchain-explorer`. The repository has 102 watchers, 1k stars, and 746 forks. It is currently on the `main` branch, with 3 other branches and 28 tags. The repository is licensed under Apache-2.0. The commit history shows 596 commits, with the latest commit being `bc5f3f5` from 7 days ago. The repository contains several directories and files, including `app`, `cli`, `client`, `devenv`, `docs`, `examples/net1`, `release_notes`, `scripts`, `ssl-certs`, `.dockerignore`, `.editorconfig`, `.eslintignore`, `.eslintrc.json`, `.gitignore`, `.mocharc.json`, `.nycrc.json`, `.sonarcloud.properties`, and `CHANGELOG.md`. The right sidebar shows the repository's README, license, releases, packages, contributors, and languages.

# Hyperledger Explorer

## Pasos del ejercicio

1. Lee el readme del proyecto.
2. Configúralo para que sea el explorador de la red de ejemplo que usamos en ejercicios anteriores

**Ejemplo de configuración:**  
<https://github.com/hyperledger/blockchain-explorer/blob/main/docker-compose.yaml>



this page intentionally left blank