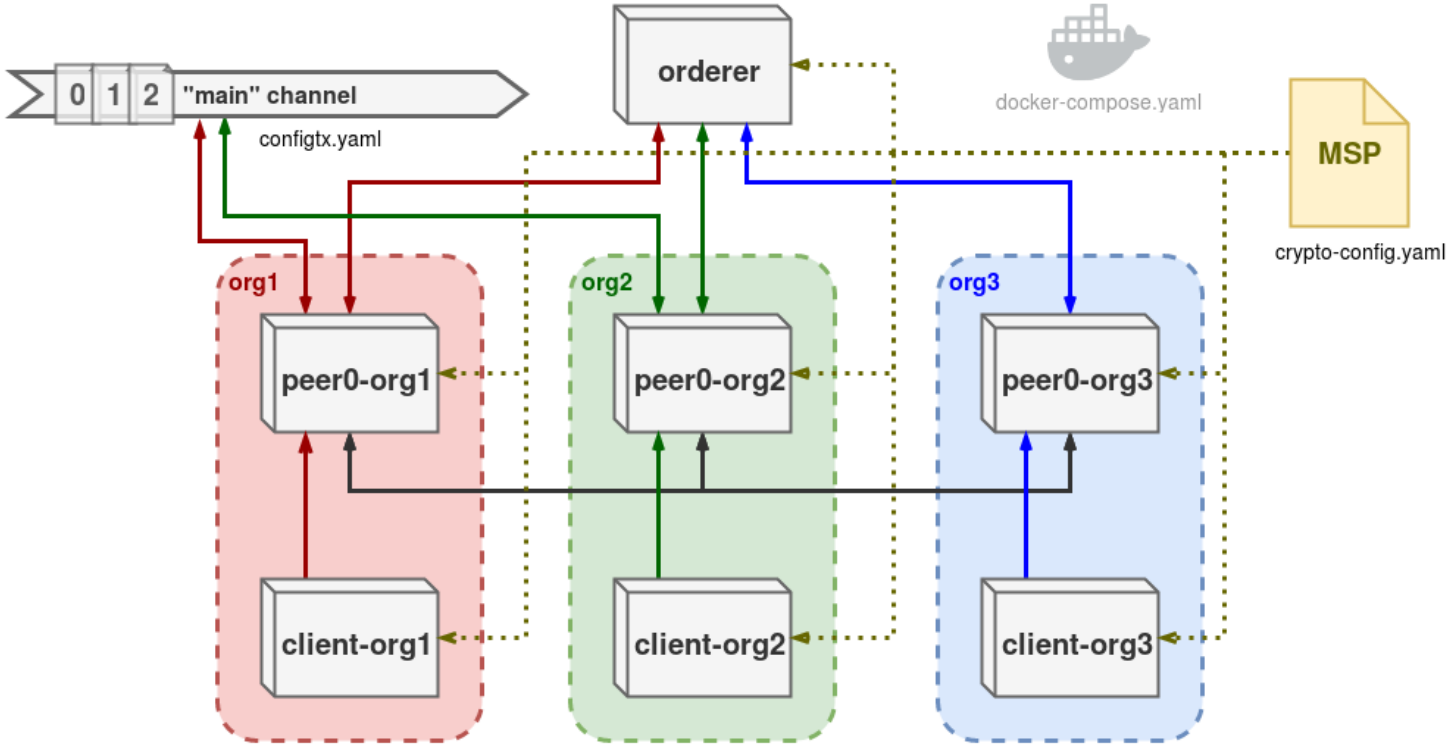
TENEMOS
MUCHO
QUE HACER
JUNTOS

2 - Análisis de un despliegue de una red privada Hyperledger Fabric

tecnalia Inspiring Business

Arquitectura de la red



Componentes

- 1 orderer node
  - Sin consenso
- 3 organizaciones
  - Componentes por organización:
    - 1 peer node
    - 1 cliente
- Sin CAs (Certification Authorities)
  - Las claves y certificados MSP y TLS se generan a partir de un fichero de configuración *crypto-config.yaml* mediante la herramienta cryptogen
- 1 canal
  - Compartido únicamente por las organizaciones 1 y 2 (véase el fichero *configtx.yaml*)

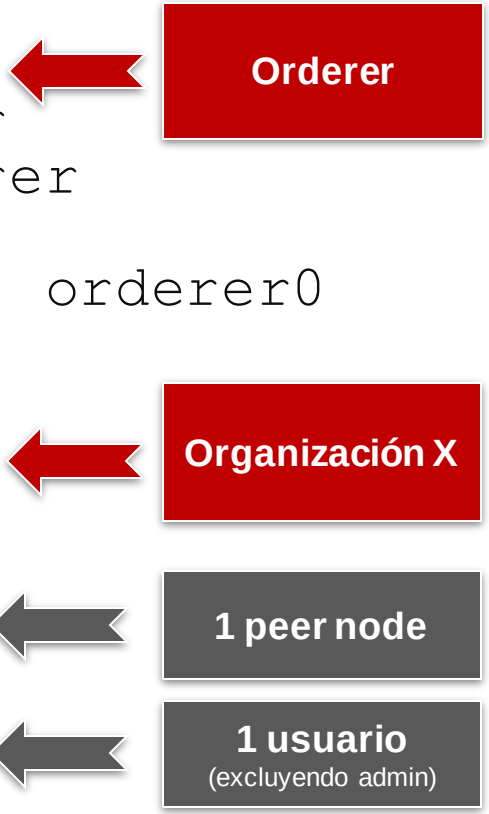
Generación de la red

- Herramienta **cryptogen**: Claves y certificados (MSP y TLS) basados en fichero de configuración **crypto-config.yaml**
- Herramienta **configtxgen**
  - Fichero de configuración **configtx.yaml**
  - Creación del orderer **genesis block** (genesis.block)
  - Creación de la **Transacción de Configuración del Canal** (main.tx)
- Despliegue basado en contenedores **docker**
  - Fichero **docker-compose.yaml**
  - Imágenes docker Hyperledger Fabric: https://hub.docker.com/u/hyperledger/

crypto-config.yaml

```
OrdererOrgs:
  - Name: orderer
    Domain: orderer
    Specs:
      - Hostname: orderer0

PeerOrgs:
  - Name: orgX
    Domain: orgX
    Template:
      Count: 1
    Users:
      Count: 1
```



configtx.yaml

```
Profiles:
  MainGenesis:
    Orderer:
      <<: *OrdererDefaults
      Organizations:
        - *orderer
    Consortiums:
      MainConsortium:
        Organizations:
          - *orgX

  MainChannel:
    Consortium: MainConsortium
    Application:
      <<: *ApplicationDefaults
      Organizations:
        - *orgX
```

Genesis block

Orderer

Consorcio (cjto de organizaciones)

Canal

Consorcio

Organizaciones

```
Organizations:
  - &orderer
    Name: ordererMSP
    ID: ordererMSP
    MSPDir: "../orderer0.orderer/msp"
  - &orgX
    Name: orgXMSP
    ID: orgXMSP
    MSPDir: "../peer0.orgX/msp"
    AnchorPeers:
      - Host: peer0-orgX
        Port: 7051
Orderer: &OrdererDefaults
  OrdererType: solo
  Addresses:
    - orderer:7050
  BatchTimeout: 2s
  BatchSize:
    MaxMessageCount: 10
    AbsoluteMaxBytes: 99 MB
    PreferredMaxBytes: 512 KB
```

Organizaciones (referenciadas en profiles)

MSP (material criptográfico de la org estructurado)

Anchor Peers (visibles por otras orgs)

Orderer (referenciado en profiles)

Datos sobre la constitución del bloque

docker-compose.yaml - Services I

```
orderer:
  restart: "no"
  image: hyperledger/fabric-orderer:x86_64-1.1.0
  environment:
    - ORDERER_GENERAL_TLS_ENABLED=false
    - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
    - ORDERER_GENERAL_GENESISMETHOD=file
    - ORDERER_GENERAL_GENESISFILE=.../genesis.block
    - ORDERER_GENERAL_LOCALMSPID=ordererMSP
    - ORDERER_GENERAL_LOCALMSPDIR=.../msp
  volumes:
        - .../genesis.block:.../genesis.block
        - .../msp:.../msp
  ports:
    - 7050:7050
  command: orderer
```

VARIABLES DE ENTORNO prevalecen sobre el fichero de configuración */etc/hyperledger/fabric/core.yaml*

```
peer0-orgX:
  restart: "no"
  image: hyperledger/fabric-peer:x86_64-1.1.0
  environment:
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=example_default
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_PEER_TLS_ENABLED=false
    - CORE_PEER_ID=peer0-orgX
    - CORE_PEER_LOCALMSPID=orgXMSP
    - CORE_PEER_ADDRESS=peer0-orgX
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0-orgX:7051
    - CORE_PEER_MSPCONFIGPATH=.../msp
    - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
  volumes:
    - /var/run/:/host/var/run/
    - .../msp:.../msp
  ports:
    - 7051:7051
    - 7053:7053
  command: peer node start
```

PROTOCOLO GOSSIP
Elección dinámica de LÍDER
· Conexión con orderer
· Diseminación de bloques

docker-compose.yaml - Services II

```
client-orgX:
  image: hyperledger/fabric-tools:x86_64-1.1.0
  tty: true
  environment:
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_PEER_TLS_ENABLED=false
    - CORE_PEER_ID=client-orgX
    - CORE_PEER_LOCALMSPID=orgXMSP
    - CORE_PEER_ADDRESS=peer0-orgX:7051
    - CORE_PEER_COMMITTER_LEDGER_ORDERER=orderer:7050
    - CORE_PEER_MSPCONFIGPATH=.../msp
  volumes:
    - /var/run/:/host/var/run/
    - .../channel-artifacts:.../channel-artifacts
    - .../peerOrganizations/orgX/users/Admin@orgX/msp:.../msp
  working_dir: /root
  command: bash
```

Scripts de apoyo

- *requirements.sh* - Instalación de requisitos
  - docker: Virtualización de contenedores a nivel de sistema operativo
  - docker-compose: Definición y arranque de aplicaciones multi-contenedor
  - Go language: Lenguaje de programación concurrente y compilado en el que se basan las herramientas de Hyperledger Fabric
  - Herramientas Hyperledger Fabric:
    - cryptogen: Generación de material criptográfico (MSP y TLS) en ausencia de CAs
    - configtxgen: Generación de artefactos de canal (genesis block, configuración del canal)
- *deploy.sh* - Despliegue de nueva red
  - Generación de material criptográfico (MSP y TLS)
  - Generación del orderer genesis block
  - Creación de la Transacción de Configuración del Canal
  - Lanzamiento de la red de contenedores docker
- *clean.sh* - Finalización de la red y limpieza

Manejo básico de contenedores

- Listar los contenedores:
  - **`docker ps`**
- Consultar los logs de un contenedor:
  - **`docker logs`** `orderer`
- Acceder vía terminal a un contenedor:
  - **`docker exec -it`** `client-org1` **`bash`**
- Ayuda:
  - **`docker --help`**

Interacción administrador-peer: Comando *peer*

- Información acerca del comando:

```
peer --help
```

- Crear un canal y unirse al mismo:

```
peer channel create -c main -f channel-artifacts/main.tx -o
$CORE_PEER_COMMITTER_LEDGER_ORDERER

peer channel join -b main.block
```

- Obtener el bloque de configuración de un canal existente y unirse al mismo:

```
peer channel fetch config -c main -o $CORE_PEER_COMMITTER_LEDGER_ORDERER

peer channel join -b main_config.block
```

Ejercicio

1. Crea el canal *main* desde el contenedor *client-org1* y une el peer correspondiente al canal.
2. Obtén el bloque de configuración del canal *main* desde el contenedor *client-org2* y une el peer correspondiente al canal.
3. Intenta obtener el bloque de configuración del canal *main* desde el contenedor *client-org3* y unir el peer correspondiente al canal. ¿Qué ocurre? ¿Por qué?

Reconfiguración: *configtxlator*

- Herramienta para reconfigurar la red Hyperledger Fabric:
  - Genesis block
  - Organizaciones
  - Canales
  - …
- https://hyperledger-fabric.readthedocs.io/en/latest/commands/configtxlator.html
- Ejecutable que inicia una REST API
- Pasos de una actualización
  1. Decodificar la configuración a JSON usando *configtxlator*
  2. Extraer la sección de configuración
  3. Crear la nueva configuración a partir de la original
  4. Codificar la configuración original y la nueva
  5. Computar la delta de actualización de la configuración a partir de ambas configuraciones
  6. Decodificar la delta de actualización de la configuración y empaquetarla
  7. Crear una nueva transacción de configuración
  8. Procesar la transacción para actualizar
- https://www.ibm.com/developerworks/cloud/library/cl-add-an-organization-to-your-hyperledger-fabric-blockchain/index.html

Del ejemplo a producción

Ejemplo	Producción

• Generación de certificados
  • Herramienta *cryptogen*
  • *crypto-config.yaml*
	• CAs
	  • 1 CA por organización
	  • Fabric CA
	  • Register & enroll

• 1 orderer ⇒ Sin consenso	• Varios orderers (1 por organización) ⇒ Consenso
	  • Kafka/ZooKeeper
	  • PBFT (futuro)

• 1 peer por organización	• Varios peers por organización
	  • Endorsers
	  • Committers

• docker-compose up	• Cluster
	  • docker-swarm
	  • Kubernetes
	• Hyperledger Cello

• Volúmenes: Mapeo de archivos host-contenedor	• Volúmenes: Persistencia de ficheros • docker cp: Intercambio de ficheros host-contenedor

• No TLS	• TLS
	  • *cryptogen*
	  • *docker-compose.yaml*
	  • *CORE_PEER_TLS_**

MSP ↔ CAs (I)

- ~~cryptogen → crypto-config.yaml~~
- **Certification Authorities (CAs)**
  - Fabric CA: https://hyperledger-fabric-ca.readthedocs.io/en/latest/
  - Otras CAs (pre-existentes...)
- Configuración organizaciones:
  - 1 organization - 1 MSP
    - 1 division – 1 OU
      - R/W policies
      - Endorsement policies
  - 1 organization - N MSPs
    - 1 division - 1 MSP
      - Privacidad a nivel de división
  - N organizations (consortium) - 1 MSP
    - 1 organization – 1 OU

MSP ↔ CAs (II)

- Creación estructura de directorios MSP
- CA certs

```
export FABRIC_CA_CLIENT_HOME=$HOME/fabric-ca/msp
fabric-ca-client getcacert -u http://localhost:7054
```

- Register & enroll:
  1. Registro de identidad peer/app/user

```
export FABRIC_CA_CLIENT_HOME=$HOME/fabric-ca/clients/user1
fabric-ca-client register --id.type user --id.name user1 --id.secret
password1 --id.affiliation org1.department1
```

  2. Enroll: Generación de claves/certificado para la identidad

```
export FABRIC_CA_CLIENT_HOME=$HOME/fabric-ca/clients/user1
fabric-ca-client enroll -u http://user1:password1@localhost:7054
```

- *config.yaml*

```
OrganizationalUnitIdentifiers:
  - Certificate: "cacerts/cacert1.pem"
    OrganizationalUnitIdentifier: "commercial"
  - Certificate: "cacerts/cacert2.pem"
    OrganizationalUnitIdentifier: "administrators"
```

Consenso: Kafka/ZooKeeper

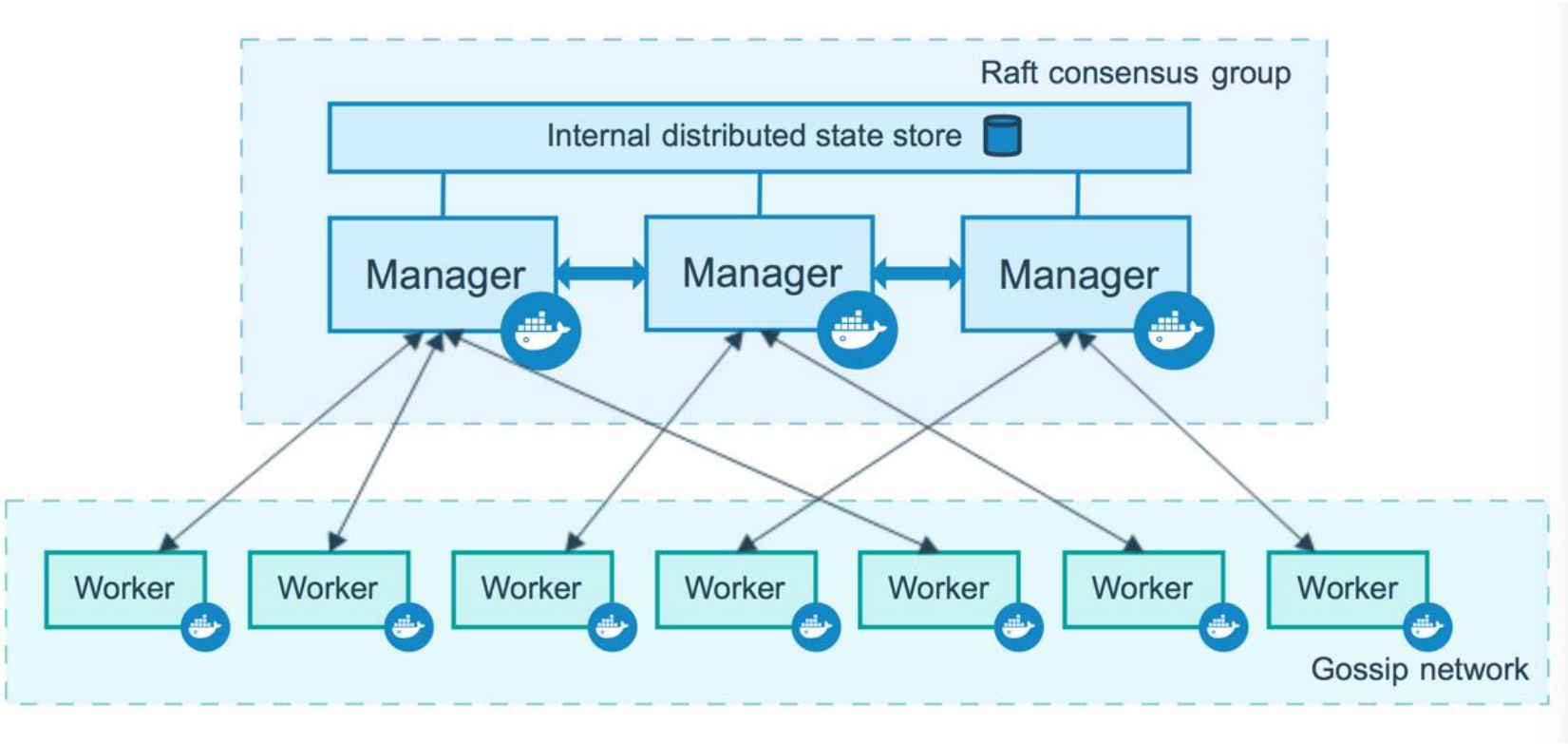
- **Kafka**: Sistema de almacenamiento publicador/subscriptor distribuido, particionado y replicado.
- **ZooKeeper**: Servicio de coordinación de aplicaciones distribuidas de alto rendimiento.
  - **ZAB** (Zookeeper Atomic Broadcast protocol).
- Imágenes docker:
  - Kafka: *hyperledger/fabric-kafka*
  - ZooKeeper: *hyperledger/fabric-zookeeper*
- Nº de nodos/contenedores:
  - Kafka: k ≥ 4
    - Crash fault tolerance
  - ZooKeeper: z = 3,5,7
    - Más de uno para evitar puntos únicos de fallo
    - Impar para evitar split-brain
- Ejemplo *docker-compose.yaml*: https://github.com/hyperledger/fabric-test/blob/master/feature/docker-compose/docker-compose-kafka.yml
- *configtx.yaml*

```
Orderer: &OrdererDefaults
    OrdererType: kafka
    # …
    Kafka:
        Brokers:
            - kafka0:9092
            - kafka1:9092
            - kafka2:9092
            - kafka3:9092
```

1 máquina - 1 nodo

- **No recomendado**
- 2 opciones:
  - Un contenedor por máquina
    - Un fichero *docker-compose.yaml* con un único servicio por máquina
    - Redirección de puertos (*ports*) ⇒ El resto de máquinas acceden a través de IP/dominio:puerto externos (*CORE_PEER_ADDRESS*, *CORE_PEER_COMMITTER_LEDGER_ORDERER*...)
  - Sin docker
    - Más complejo, no es la manera en que se distribuye Hyperledger Fabric
    - Particularmente complejo portarlo a algunas plataformas (Windows, etc.)

Docker Swarm I



Docker Swarm II

- Cluster de contenedores
- Inicialización del cluster:
  - `docker swarm init --advertise-addr X.X.X.X`
  - `docker swarm join-token worker/manager`
  - `docker swarm join --token SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c X.X.X.X:2377`
- Configuración de una red externa:
  - `docker network create --opt encrypted -d overlay --attachable mynetwork`
- Depliegue:
  - `docker stack deploy -c conf/docker-compose.yaml -c conf/docker-compose.prod.yaml example`
- docker-compose.yaml:

```
networks:
  default:
    external:
      name: mynetwork
```

En cada service:

```
deploy:
  mode: replicated
  replicas: 1
  restart_policy:
    condition: any
  placement:
    constraints: [node.hostname==node-1]
```

Docker Swarm III

- **Docker Swarm no persiste los datos tras un reinicio**
  - Por defecto, vuelve al estado original de la imagen docker
  - Persistimos datos mediante volúmenes docker
    - Directorios:
      - /etc/hyperledger
      - /var/hyperledger
      - …
    - El servicio no tiene por qué arrancar las réplicas de los contenedores en las mismas máquinas (salvo que se especifique en las placement constraints) ⇒ Sistema de archivos compartido o externo:
      - NFS
      - GlusterFS
      - File servers
- Otros clusters de contenedores:
  - Kubernetes

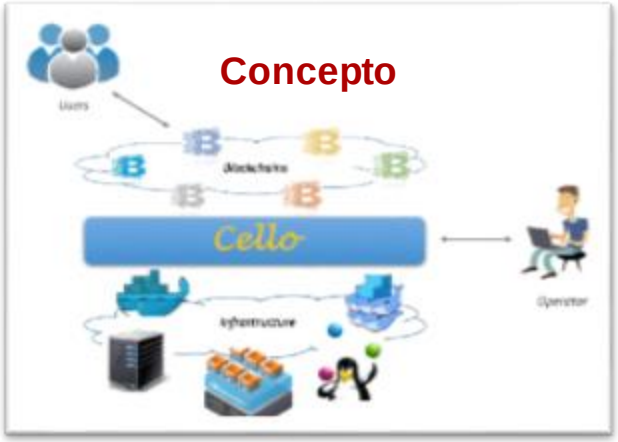
HYPERLEDGER CELLO

- Herramienta de despliegue **Blockchain as a Service (BaaS)**
- https://github.com/hyperledger/cello
- Estado: Incubation (v0.8.0)
- Impulsada por IBM, con la esponsorización de Soramitsu, Huawei e Intel
- Propósito:
  - Constitución de una plataforma BaaS rápidamente desde cero
  - Provisionamiento de redes Blockchain al instante (Hyperledger Fabric, etc.)
  - Mantener un pool de redes Blockchain sobre infraestructuras virtual cloud o clusters de contenedores (Docker, Swarm, Kubernetes)
  - Verificar el estado de las redes, escalar recursos, etc. a través de dashboards
- Características principales:
  - Gestión automática del ciclo de vida de las Blockchains (create/start/stop/delete/keep)
  - Soporte y personalización de distintas redes Blockchain (actualmente centrada en Hyperledger Fabric)
  - Gestión de SmartContracts
  - Soporte de diferentes worker nodes: Docker, Swarm, Kubernetes…
  - Soporte de diferentes arquitecturas: X86, POWER y Z; tanto físicos como virtuales
  - Extensible con otras herramientas Hyperledger: Monitorización, logs, health, analytics

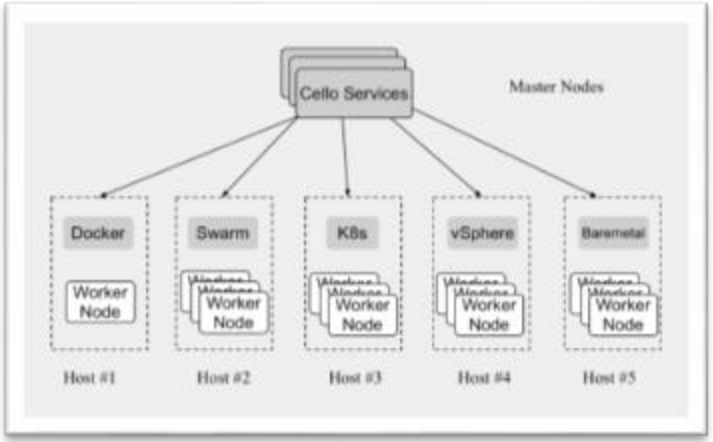
HYPERLEDGER CELLO

tecnalia Inspiring Business

Concepto



Arquitectura Master-Worker



Herramienta web