DESARROLLO DE UNA APLICACION CON HYPERLEDGER FABRIC Y COMPOSER

Marta Sebastián Rodriguez
Tecnologías de Telecomunicación

Felix Freitag

09/06/2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	DESARROLLO DE UNA APLICACION CON HYPERLEDGER FABRIC Y COMPOSER
Nombre del autor:	Marta Sebastián Rodriguez
Nombre del consultor:	Felix Freitag
Fecha de entrega (mm/aaaa):	06/2019
Área del Trabajo Final:	APLICACIONES Y SISTEMAS DISTRIBUIDOS
Titulación:	*TECNOLOGIAS DE TELECOMUNICACION*

Resumen del Trabajo (máximo 250 palabras):

"Hyperledger es un esfuerzo de colaboración de código abierto creado para avanzar en las tecnologías de cadena de bloques de la industria. Es una colaboración global, organizada por La Fundación Linux, que incluye líderes en finanzas, banca, Internet de las cosas, cadenas de suministro, fabricación y tecnología."

Esta es la definición que se da en la web oficial de Hyperledger, es una tecnología cuyo objetivo es crear un entorno donde los de desarrolladores y las empresas acerquen posturas y se puedan coordinar para crear redes de negocio de blockchain.

Hyperledger Fabric es un proyecto de la Fundación Linux que lidera IBM. Es un accionamiento de la tecnología blockchain, diseñado para ser la base de un desarrollo con una gran estabilidad y que cuenta con una gran seguridad.

Hyperledger Composer es un conjunto de herramientas de Hyperledger que utilizan JavaScript para facilitar la creación de aplicaciones de blockchain de Hyperledger Fabric. Pone a disposición del desarrollador una interfaz de usuario llamada Playground que se utiliza en este proyecto para crear el modelo de aplicación y probar la red de negocio o empresarial.

Como resultado de este proyecto se desarrollará y desplegara un ejemplo de aplicación empresarial mediante la utilización y el estudio de todos estos entornos y herramientas.

Abstract (in English, 250 words or less):

“Hyperledger is an open source collaborative effort created to advance cross-industry blockchain technologies. It is a global collaboration including leaders in finance, banking, Internet of Things, supply chains, manufacturing and Technology.”

This is the definition that is given in the official website of Hyperledger, it is a technology whose objective is to create an environment where the developers and companies approach positions and can be coordinated to create blockchain business networks.

Hyperledger Fabric is a project of the Linux Foundation led by IBM. It is a drive of blockchain technology, designed to be the basis of a development with great stability and that is very safe.

Hyperledger Composer is a set of Hyperledger tools that use JavaScript to facilitate the creation of Hyperledger Fabric blockchain applications. It makes available to the developer, a user interface called Playground that is used in this project to create the application model and test the business network.

As a result of this project, an example of business application will be developed and deployed through the use and study of all these environments and tools.

Palabras clave (entre 4 y 8):

Hyperledger, Fabric, Composer, Playground, Blockchain, Aplicación, Negocio, red

Índice

Lista de figuras

1. INTRODUCCIÓN

1.1 CONTEXTO Y JUSTIFICACIÓN DEL TRABAJO

Blockchain es una tecnología que permite la gestión de la información entre distintos dispositivos, compartiendo un registro distribuido, descentralizado y sincronizado entre ellos. Aunque esta tecnología cada vez está más desarrollada aún sigue en fase de experimentación, implantación y pruebas, empezando a mostrar el potencial que puede alcanzar.

Dentro de Blockchain, hay diversas compañías que han desarrollado distintas tecnologías, en este proyecto se ha decidido escoger el Blockchain de IBM, se denomina Hyperledger y la razón de escogerlo es que los usuarios de Hyperledger Fabric sólo necesitan conocimientos de Java Script y la estructura de este código. El proyecto también ofrece SDKs (Software Development Kits), por lo que no hay necesidad de aprender un nuevo idioma. Al utilizar estos lenguajes comunes, es mucho más flexible que los productos de la competencia.

Su versatilidad y funcionalidad fácil de entender atrae a muchas industrias sin importar el tamaño y la experiencia en el Blockchain. Hyperledger en general también está ganando muchos seguidores. Sólo en 2018 se han incorporado 27 nuevas empresas y organizaciones de los más diversos sectores, entre ellas se encuentran IBM, Huawei, Nokia, Samsung, Intel, Consensys, Airbus, Daimler, Deutsche Borse, Cisco, Accenture, ANZ Banco, BNY Mellon, Thomson Reuters, American Express, JP Morgan, BBVA, Blockstream, Netki, Lykke, bloq. También muchos son los bancos que han decido participar, como por ejemplo el banco BBVA y empresas españolas como Tecnalia, 159 desarrolladores en total contribuyeron con casi 3.500 conjuntos de cambios a Hyperledger Fabric.

Hyperledger se ha hecho un hueco dentro de las redes empresariales en diversos espacios, como la sanidad, las finanzas y la cadena de suministro, para desarrollar aplicaciones blockchain que garanticen la privacidad y redes autorizadas descentralizadas.

Se perfila como una tecnología que puede revolucionar la forma en que las empresas controlan el acceso a un blockchain, con una serie de mejoras en las medidas de seguridad. Con lo que se consigue que las transacciones comerciales sean más inteligentes, más rápidas y más seguras.

Blockchain es una solución global que se encuentra con un vacío en términos de legalidad. Por eso nació Alastria, primer consorcio multisectorial promovido por organizaciones e instituciones para el establecimiento de una infraestructura de Blockchain/DLT con Autorización Pública, apoyando servicios con eficacia jurídica en el ámbito nacional y de acuerdo con la normativa europea, es decir, recrear un Blockchain como Etehereum pero a nivel nacional con lo que se podrían tener todas las aplicaciones de Blockchain en un entorno regulado.

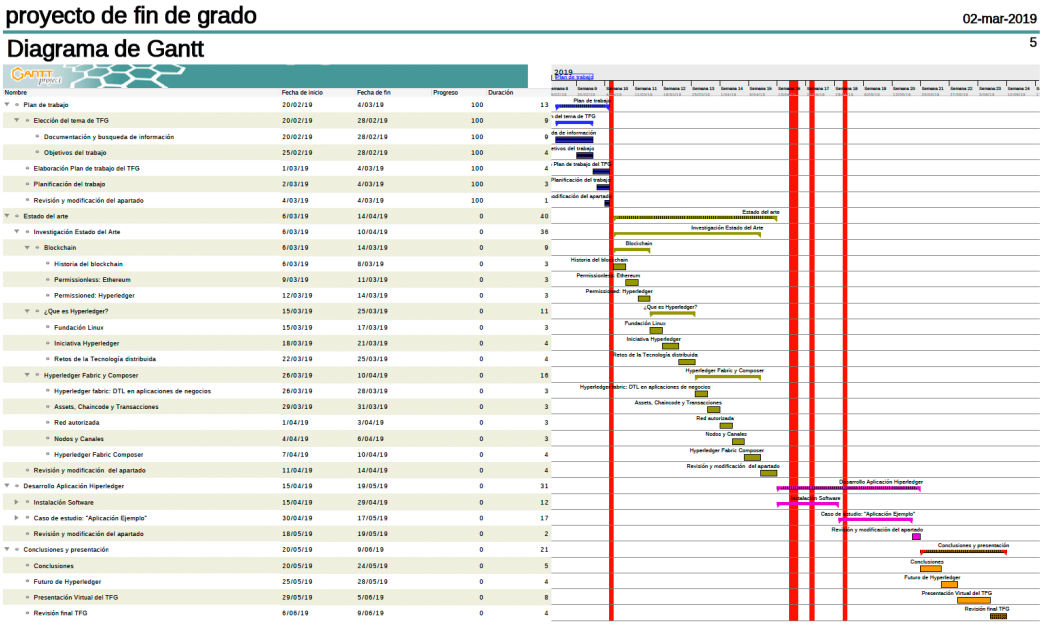
Aunque Alastria es agnóstica en materia tecnológica, se decidió empezar usando Quorum, un derivado de Ethereum que ofrece la posibilidad de hacer una red permisionada y desplegar algoritmos más eficientes. Actualmente está desplegando versiones en otras tecnologías, por ejemplo, están viendo la posibilidad de construir sobre Parity o Hyperledger Fabric.

La razón que impulsa el desarrollo de este TFG es un estudio de la tecnología a fondo tanto de blockchain como de Hyperledger Fabric, que permita obtener un conocimiento técnico y riguroso.

1.2 OBJETIVOS DEL TRABAJO

OBJETIVOS GENERALES

- Análisis y estudio de la tecnología Blockchain. Conocer cuáles son las bases fundamentales de la tecnología y su funcionamiento. Conocer el estado de esta tecnología en la actualidad y qué podríamos hacer con ella.
- Despliegue de una red basada en Hyperledger, mediante Hyperledger Fabric y Composer. Ser capaz de implementar una red Blockchain y sus correspondientes nodos, assets, Chaincode y transacciones. Conocer la infraestructura de la red que lo soporta y sus comunicaciones.
- Instalación de todos los programas y entornos necesarios en los distintos pasos de confección de la red.
- Desarrollo de una demostración real del funcionamiento de la tecnología. Se implementará un caso de uso en el que se aplique todo lo estudiado en



- codificación, de todos los entornos, tanto de front end como de back end necesarios para el desarrollo de toda la operativa diseñada.
- pruebas, de la codificación, para comprobar que todo funciona correctamente.

- documentación, de todo el proceso para que queden reflejados todos los pasos que se van dando, para poder corregir posibles fallos de codificación o de diseño.

1.3 PLANIFICACIÓN DEL TRABAJO

En este TFG hay tres actividades importantes: el estudio de la tecnología, la implantación del caso de uso y la redacción de toda la documentación necesaria.

En la siguiente figura se aprecia un diagrama de Gantt con la planificación de las tareas de este proyecto:

Tarea

Nombre	Fecha de inicio	Fecha de fin	Progreso	Duración
Plan de trabajo	20/02/19	4/03/19	100	13
Elección del tema de TFG	20/02/19	28/02/19	100	9
Documentación y busqueda de información	20/02/19	28/02/19	100	9
Objetivos del trabajo	25/02/19	28/02/19	100	4
Elaboración Plan de trabajo del TFG	1/03/19	4/03/19	100	4
Planificación del trabajo	2/03/19	4/03/19	100	3
Revisión y modificación del apartado	4/03/19	4/03/19	100	1
Estado del arte	6/03/19	14/04/19	0	40
Investigación Estado del Arte	6/03/19	10/04/19	0	36
Blockchain	6/03/19	14/03/19	0	9
Historia del blockchain	6/03/19	8/03/19	0	3
Permissionless: Ethereum	9/03/19	11/03/19	0	3
Permissioned: Hyperledger	12/03/19	14/03/19	0	3
¿Que es Hyperledger?	15/03/19	25/03/19	0	11
Fundación Linux	15/03/19	17/03/19	0	3
Iniciativa Hyperledger	18/03/19	21/03/19	0	4
Retos de la Tecnología distribuida	22/03/19	25/03/19	0	4
Hyperledger Fabric y Composer	26/03/19	10/04/19	0	16
Hyperledger fabric: DTL en aplicaciones de negocios	26/03/19	28/03/19	0	3
Assets, Chaincode y Transacciones	29/03/19	31/03/19	0	3
Red autorizada	1/04/19	3/04/19	0	3
Nodos y Canales	4/04/19	6/04/19	0	3
Hyperledger Fabric Composer	7/04/19	10/04/19	0	4
Revisión y modificación del apartado	11/04/19	14/04/19	0	4
Desarrollo Aplicación Hiperledger	15/04/19	19/05/19	0	31
Instalación Software	15/04/19	29/04/19	0	12
Topología del entorno de desarrollo	15/04/19	15/04/19	0	1
Requisitos para la instalación del entorno de Hyperledger Fabric	15/04/19	17/04/19	0	3
IDE Visual Code	15/04/19	15/04/19	0	1
Instalación de Composer Dev Tools	16/04/19	16/04/19	0	1
Node JS	16/04/19	16/04/19	0	1
Python 2.7	16/04/19	16/04/19	0	1
NVM	16/04/19	16/04/19	0	1
Instalación de Docker o Docker Tool	17/04/19	17/04/19	0	1
Máquina Virtual: Instalación del entorno Hyperledger Fabric Dev	17/04/19	20/04/19	0	2
Instalación de Vargrant	17/04/19	17/04/19	0	1
Estableciendo la máquina virtual mediante VirtualBox y Vargrant	20/04/19	20/04/19	0	1
Native: Instalación del entorno Hyperledger Fabric Dev	22/04/19	22/04/19	0	1
Mac Os: Estableciendo el entorno de desarrollo de Fabric	22/04/19	22/04/19	0	1
Herramientas de Hyperledger Composer para el desarrollo de aplicaciones	24/04/19	29/04/19	0	6
Fabric Composer Playground	24/04/19	24/04/19	0	1
Generador Yeoman Hyperledger	25/04/19	25/04/19	0	1
Roles y tarjetas de la red de negocio	26/04/19	26/04/19	0	1
REST Server	27/04/19	29/04/19	0	3
Descripción general	27/04/19	27/04/19	0	1
Seguridad	28/04/19	28/04/19	0	1
Skeleton Angular usando Yeoman	29/04/19	29/04/19	0	1
Caso de estudio: "Aplicación Ejemplo"	30/04/19	17/05/19	0	17
Modelado de la aplicación	30/04/19	4/05/19	0	4
Implementancón de las APP Clientes	5/05/19	8/05/19	0	4
Implementación de los procesos de transacción	9/05/19	12/05/19	0	4
Desarrollo de las aplicaciones Front End para las aplicaciones de red	13/05/19	15/05/19	0	3
Pruebas del funcionamiento	16/05/19	17/05/19	0	2
Revisión y modificación del apartado	18/05/19	19/05/19	0	2
Conclusiones y presentación	20/05/19	9/06/19	0	21
Conclusiones	20/05/19	24/05/19	0	5
Futuro de Hyperledger	25/05/19	28/05/19	0	4
Presentación Virtual del TFG	29/05/19	5/06/19	0	8
Revisión final TFG	6/06/19	9/06/19	0	4

1.4 BREVE SUMARIO DE PRODUCTOS OBTENIDOS

Como resultado del desarrollo de este proyecto obtendremos una red Blockchain y sus correspondientes nodos, assets, Chaincode y transacciones, mediante la aplicación de la tecnología hyperlededger fabric y mediante la herramienta hyperledger composer.

1.5 BREVE DESCRIPCIÓN DE LOS OTROS CAPÍTULOS DE LA MEMORIA

Capitulo 2.- estado del arte, en este apartado se aborda la situación actual y futura de esta tecnología, así como una explicación de sus aplicaciones y del porque se esta utilizando cada vez mas, en comparación con otras tecnologías anteriores.

Capitulo 3.- Instalación de los entornos y programas necesarios para el desarrollo de la aplicación.

Capitulo 4.- Conclusiones, acerca del desarrollo del proyecto, los logros y fracasos respecto a los objetivos iniciales.

Capitulo 6.- Bibliografía

Capitulo 7.- Webgrafia

2. ESTADO DEL ARTE

2. 1. BLOCKCHAIN

A nivel técnico, blockchain se puede definir cómo un registro contable (ledger) que registra transacciones agrupadas en bloques dentro de una red distribuida de nodos, donde cada nodo mantiene una copia del ledger. Los nodos ejecutan un protocolo de consenso que valida las transacciones, agrupándolas en bloques, y construyendo una cadena de seguridad con hash.

2. 1. 1. HISTORIA DEL BLOCKCHAIN

Blockchain promete resolver fundamentalmente problemas de tiempo y confianza para acometer las ineficiencias y los costos en industrias tales como servicios financieros, cadenas de suministro, logística y sanitarias. Las características clave de blockchain incluyen la inmutabilidad y un libro de registro contable (ledger) distribuido donde las actualizaciones sobre las transacciones se realizan mediante un sistema de confianza basado en consenso, que puede facilitar una auténtica interacción entre múltiples partes, porque solo es posible su actualización mediante la suma de un bloque vinculado al bloque anterior.

Esta interacción digital no solo está limitada por una confianza sistémica, sino que además asegura que la procedencia transaccional del registro mantiene un historial inmutable de interacción entre las partes. Con el diseño del sistema blockchain, se intenta construir un sistema que lleva implícita la confianza, lo que conduce a reducir riesgos, y, con varias construcciones tecnológicas aplicadas, tales como criptografía, encriptación, smarts contracts y consenso, con lo que se consigue infundir una mayor seguridad en el sistema de transacciones.

Inicialmente la tecnología blockchain se gestó e implementó en 2008, por Satoshi Nakamoto, como una infraestructura destinada al bitcoin que mejoraba la confianza digital mediante la descentralización, ya que, a partir de un núcleo se

extraen bloques que, mediante la interconexión forman cadenas mayores cuya finalidad es transportar transacciones.

En la Imagen 1 nos da una idea visual de los hitos en la historia del Blockchain.

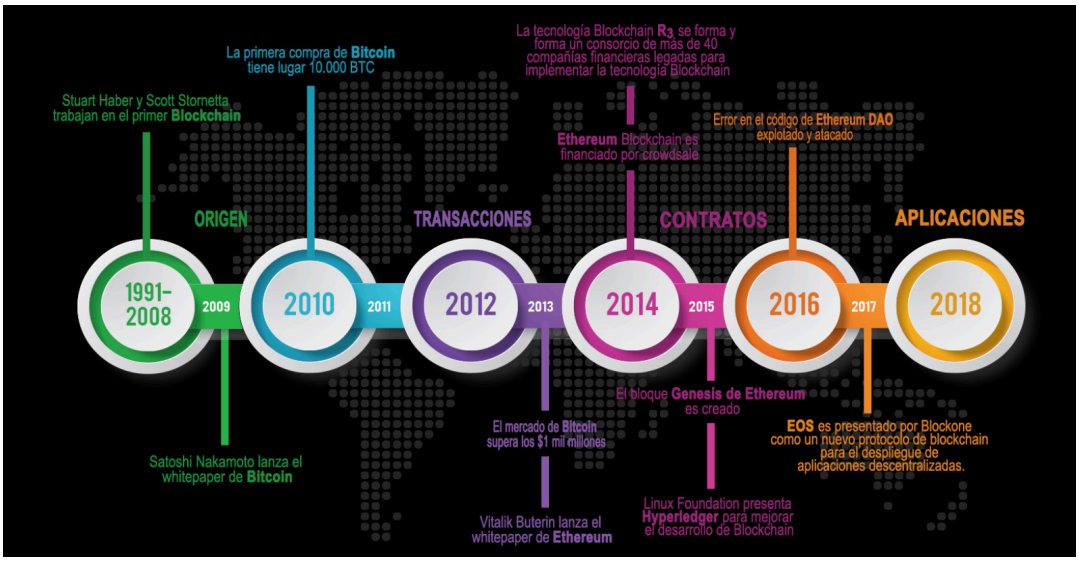


Imagen 1: Historia del Blockchain

Posteriormente Vitalik Buterin, uno de los primeros desarrolladores de bitcoin, comenzó a formar Ethereum, presentado en 2015 como blockchain pública para mejorar las funcionalidades del bitcoin, pasando de ser solamente una criptomoneda a una nueva plataforma para el desarrollo de aplicaciones descentralizadas. Ethereum actualmente es de las tecnologías blockchain más importantes, al incorporar los conceptos de contrato inteligente y aplicaciones descentralizadas.

Existen dos tipos de blockchain, permissionless (públicas) y permissioned (privadas), como vemos reflejado en la Imagen 2.

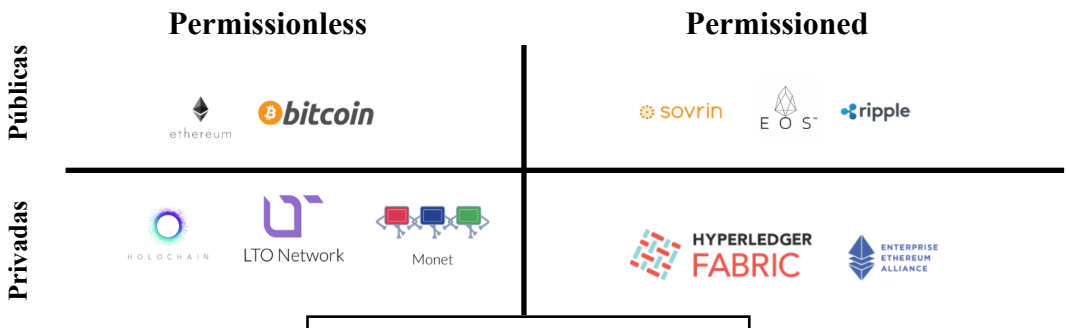


Imagen 2: Blockchains privadas y públicas

2. 1. 1. 1.Permissionless Blockchain: Ethereum y bitcoin

Un blockchain permissionless o público, es un tipo de blockchain completamente descentralizado que no requiere ningún tipo de autorización, solamente es preciso colaborar con la red para mantener el consenso. Todos los miembros tienen el derecho de realizar todas las acciones, incluyendo agregar y validar transacciones, contratos inteligentes, etc…

Las ventajas de este tipo de blockchain son:

- Ofrecen una plataforma descentralizada, lo que significa que la información no se almacena en un repositorio central y cualquier persona puede acceder a la información en cualquier momento y desde cualquier lugar.
- Proporciona firmas inmutables para todos los registros. Todo el intercambio de información y las transacciones se cifran criptográficamente, lo que hace que todo el sistema y los datos permanezcan seguros.
- Anonimato: Los mineros y participantes en la red permanecen en el anonimato.
- Transparencia: Todos los datos e información son visibles para todos.
- Transacciones rápidas y fáciles.

Las desventajas son:

- Anonimato: aunque es una buena señal de que la identidad de las partes comerciales permanezca en el anonimato, esto puede volverse un problema. En caso de que haya un fraude o alguien quiera rastrear a las partes que están negociando, esto se vuelve difícil. Debido a estas características, se puede utilizar blockchain para actividades ilícitas.
- Mecanismo de consenso: este tipo de blockchains, funciona según el principio de Proof-of-Work donde los participantes tienen que resolver el complejo rompecabezas matemático, lo que requiere el consumo de una gran cantidad de recursos, y por lo tanto, es muy costoso.
- Ofrece un tamaño de bloque limitado.

- ✤ No es necesario probar la identidad. Lo único necesario es comprometer la capacidad de procesamiento para convertirse en parte de la red. Cualquier minero que resuelva el rompecabezas puede convertirse en parte del sistema.

Estos inconvenientes del sistema blockchain permissionless hacen que muchas compañías cuestionen el sistema. Por lo que Ethereum, blockchain de este tipo, está cambiando su mecanismo de consenso proof-of-work a proof-of-stake.

Dos de los ejemplos más representativos de este tipo de blockchain son:

Imagen 3 Símbolo de bitcoin

Bitcoin creada el 3 de Enero de 2009, es una tecnología basada en el intercambio punto a punto (P2P) que no necesita un ente gubernamental que lo emita y lo respalde, fue desarrollado por quien o quienes se ocultan tras el seudónimo de Satoshi Nakamoto. Una de las características más importantes de Bitcoin es que, al ser open source y su creador anónimo, tampoco pertenece a ningún individuo o compañía privada. Los usuarios son los que mantienen en funcionamiento su plataforma.

Pero Bitcoin se puede considerar también como un sistema digital, pues fue el primer blockchain con la capacidad de almacenar cualquier tipo de información, desde los bitcoin que se han gastado hasta contratos inteligentes. Esta información esta protegida por un importante sistema criptográfico.

En la figura 3 se muestra el logotipo del Bitcoin.

Imagen 4: Símbolo de la criptomoneda Ether de Ethereum

Ethereum es un blockchain que permite desarrollar contratos inteligentes y aplicaciones descentralizadas, aunque inicialmente fue una mejora de la criptomoneda bitcoin, en la actualidad Ethereum utiliza su propia criptomoneda: el Ether.

Cuyo logotipo se muestra en la imagen 4.

Ethereum permite a los desarrolladores crear y desplegar aplicaciones descentralizadas o Dapp. Bitcoin, por ejemplo, es un Dapp que proporciona a sus usuarios un sistema de efectivo electrónico que permite los pagos en línea.

Debido a que las aplicaciones descentralizadas están compuestas por un código que se ejecuta en una red de blockchain, no están controladas por ninguna entidad individual o central.

La principal innovación de Ethereum, la Máquina Virtual Ethereum (EVM) es un software completo de Turing que permite a cualquier persona ejecutar cualquier programa independientemente del lenguaje de programación, si se le ha asignado suficiente tiempo y memoria. La máquina virtual Ethereum hace que el proceso de creación de aplicaciones de blockchain sea más fácil y eficiente que nunca, ya que, en lugar de tener que crear una cadena de bloques completamente original para cada nueva aplicación, Ethereum permite el desarrollo de miles de aplicaciones diferentes en una sola plataforma.

Ethereum también se puede utilizar para construir organizaciones autónomas descentralizadas (DAO). Un DAO es una organización completamente autónoma y descentralizada, que se ejecuta mediante código de programación, es decir, una colección de contratos inteligentes escritos en la cadena de bloques Ethereum. El código está diseñado para reemplazar las reglas y la estructura de una organización tradicional, eliminando la necesidad de un control centralizado. Un DAO es propiedad de todos los que compran tokens, pero en lugar de que cada token corresponda a acciones y propiedad, los tokens actúan como contribuciones que otorgan a las personas derechos de voto.

2. 1. 1. 2. PERMISSIONED: HYPERLEDGER

Imagen 5: Simbolo de Hyperledger

En las blockchains permissioned o privadas a diferencia de las públicas vistas anteriormente, la participación en la red no es abierta. Se controla el acceso de los participantes a ciertas funciones de la red mediante roles, que se establecen de forma flexible y que se ajusta a las situaciones particulares.

En la imagen 5 vemos el logotipo de Hyperledger.

El establecimiento de los roles es flexible y puede ajustarse a condiciones particulares, esto quiere decir, que la lectura y la escritura que se efectúa en la base de datos debe ser autorizada total o parcialmente para cada componente.

En este tipo de blockchain no existe la necesidad de utilizar una criptomoneda, ya que no es necesario crear ningún sistema de estimulo como proof-of-work o proof-of-stake.

Las ventajas de este tipo de blockchain, son:

- ✤ Ofrecen un rendimiento más eficiente en comparación con las redes públicas de blockchain. En una red de blockchain, todos los nodos de la red deben calcular y validar las transacciones en la red. Estos nodos a menudo realizan cálculos redundantes para validar transacciones en la red. Debido a que los permissioned blockchains solo requieren que sus nodos miembros validen las transacciones, ofrecen un mejor rendimiento en relación con la cantidad de cómputo utilizada para la validación.
- ✤ Tienen estructuras de gobierno claramente definidas en comparación con las redes públicas de blockchain. Los blockchains públicos funcionan como foros públicos donde cualquier persona que opere un nodo completo tendría algo que decir en el gobierno y las reglas de la red. Como consecuencia, las actualizaciones y los desarrollos en estas redes son increíblemente lentos, ya que varios nodos independientes, actuando en su propio interés, deberían llegar a un consenso para activar una actualización de la red. Sin embargo, los nodos en permissioned blockchains son capaces de avanzar con las actualizaciones mucho más rápido. Si bien a veces puede haber debates sobre las actualizaciones de una red de permissioned blockchains, estos debates son más fáciles de resolver entre un consorcio de empresas en comparación con una arena pública de cientos de miles de personas y entidades.

Como principales desventajas:

- ✤ La seguridad depende completamente de la integridad de sus miembros. Si un grupo interno de miembros en un permissioned blockchain se confabula para cambiar los datos, puede arruinar la integridad de la red en su beneficio.

✤ En comparación con las bases de datos públicas, los permissioned blockchains son más propensas a la censura y la regulación. Debido a que estos blockchains son administrados por un consorcio de empresas, deben cumplir con las mismas regulaciones que sus miembros. De forma que, en teoría, un regulador podría imponer sus leyes censurando las transacciones que considere oportunas. Si bien la amenaza de la regulación puede no afectar las operaciones diarias de las empresas que ya cumplen con las leyes vigentes, evitaría que las empresas que necesitan las características no censurables de los blockchains públicos puedan operar en los permissioned blockchains.

2. 2. ¿QUE ES HYPERLEDGER?

Hyperledger es una permnissioned blockchain o blockchain privada, en el que varios equipos de desarrollo están colaborando para crear código abierto (open source), iniciado en 2015 por la Fundación Linux, que incluye líderes en finanzas, banca, IoT, cadena de suministro, fabricación y tecnología.

2. 2. 1. FUNDACION LINUX

La Fundación Linux desde el año 2000 está dedicada a construir un ecosistema sostenible mediante proyectos de código abierto para acelerar el desarrollo tecnológico y la adopción comercial, no solo para Linux sino también para otros sistemas operativos.

Imagen 6: Proyectos de la Fundación Linux

En la imagen 6 observamos los proyectos destacados de la Fundación Linux.

La Fundación Linux cuenta con personal dedicado y todas las iniciativas se financian a través de las cuotas de más de 800 miembros y más de 50 proyectos.

2. 2. 2. INICIATIVA HYPERLEDGER

Bajo la Fundación Linux, Hyperledger es la incubadora de tecnologías de blockchain para empresas. Todas las iniciativas de Hyperledger se engloban en dos categorías.

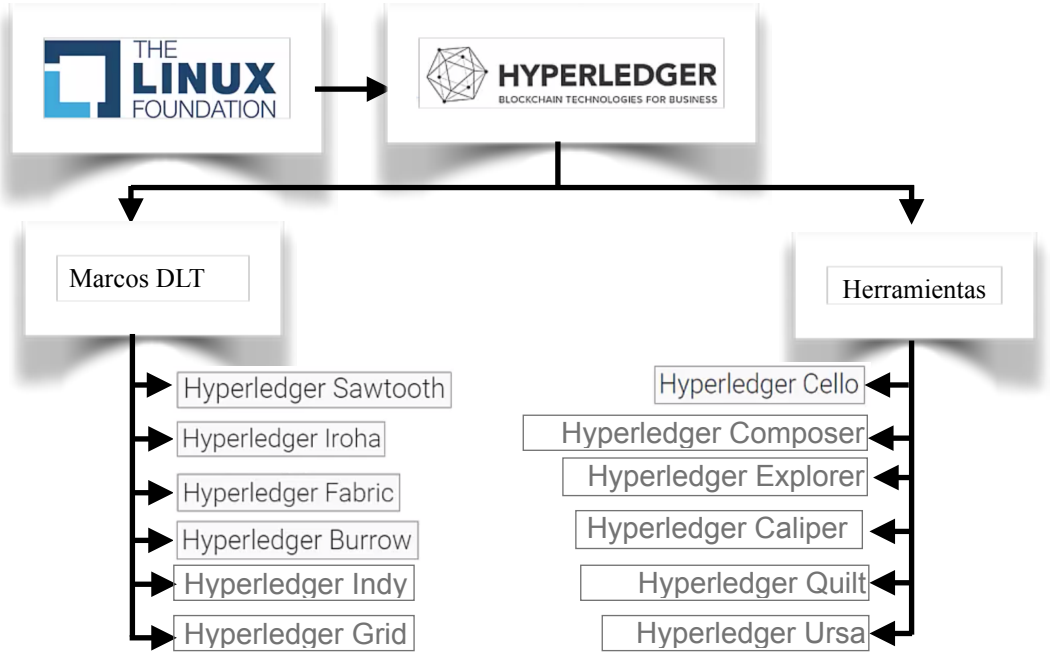


Imagen 7: Proyectos bajo el paraguas de Hyperledger

Todos los proyectos bajo el paraguas de Hyperledger se muestran en la imagen 7.
La primera categoría son los marcos DLT:

- Sawtooth, dirigido por Intel, permite construir, implementar y ejecutar ledgers distribuidos, que proporcionan un registro digital mantenido sin autoridad o implementación central.
- Iroha, es liderado por Suramitsu, está escrito en C ++ e incorpora un algoritmo de consenso llamado Yet Another Consensus y el servicio de pedidos BFT.
- IBM donó el código fuente para su iniciativa blockchain llamada open blockchain y lo combinó con el código fuente de otros socios, como Digital Asset que han incubado el proyecto Hyperledger Fabric.

Hyperledger Fabric permite que el consenso y los servicios, sean plug-and-play, aprovecha la tecnología de contenedores para alojar contratos inteligentes llamados "chaincode" que comprenden la lógica de aplicación del sistema.

- ✳Burrow, fue originalmente contribuido por Monax y copatrocinado por Intel. Hyperledger Burrow proporciona un cliente modular de blockchain con un intérprete de contrato inteligente permissioned, desarrollado según la especificación de la Máquina Virtual Ethereum (EVM).
- ✳Indy, está diseñado específicamente para la identidad descentralizada. Proporciona herramientas, bibliotecas y componentes reutilizables para crear y usar identidades digitales independientes arraigadas en blockchains u otros ledgers distribuidos.
- ✳Grid, es un framework. No es un blockchain ni una aplicación. Grid es un ecosistema de tecnologías, marcos y bibliotecas que trabajan en conjunto, lo que permite a los desarrolladores elegir qué componentes son los más adecuados para su industria o modelo de mercado.

La segunda categoría de iniciativas son las herramientas:

- ✳Caliper, permite a los usuarios medir el rendimiento de una implementación de blockchain específica con un conjunto de casos de uso predefinidos.
- ✳Cello, es una iniciativa que pretende simplificar la creación y gestión de infraestructura blockchain.
- ✳Composer, es para crear aplicaciones de red de negocios utilizando un lenguaje de modelado de alto nivel.
- ✳Explorer, proporciona visibilidad en un sistema de blockchain operacional, para obtener visibilidad de las transacciones, bloques, etc.
- ✳Quilt, tiene como objetivo lograr la interoperabilidad entre diferentes cadenas.
- ✳Ursa, es una biblioteca criptográfica compartida que permitiría evitar la duplicación de otros trabajos criptográficos y, con suerte, aumentar la seguridad en el proceso.

La idea detrás de todas estas herramientas es que serán reutilizables en los distintos ledgers de tecnología distribuida, independientemente de qué tecnología de Hyperledger se utilize.

2. 2. 3. RETOS DE LA TECNOLOGIA

El camino hacia cualquier aplicación de permissioned blockchain comienza con dos decisiones fundamentales: primero, seleccionar qué plataforma de blockchain, y segundo, qué API (application programming interface) utilizar.

Algunos de los retos que hay que enfrentar cuando se selecciona una solución permissioned blockchain, son:

- **Integración**: Al utilizar contratos inteligentes (smart contracts) en la aplicación de permissioned blockchain, probablemente se deba establecer comunicación con sistemas de API externos, lo que se convierte en un ejercicio intensivo de desarrollo.

- **Acceso a los datos**: Los permissioned blockchains son excelentes para escribir información, pero no tanto para leerla. La mayoría de las soluciones de blockchain necesitan interactuar con los datos que se están registrando, pero esos datos son difíciles de acceder y son muy ininteligibles.

- **Privacidad de los datos**: Muchos escenarios de permissioned blockchain operan en industrias reguladas con fuertes restricciones de privacidad de datos. Por eso, proteger y hacer cumplir el control de acceso es un requisito clave.

- **Almacenamiento de datos**: Los blockchains no son la mejor solución para almacenar grandes volúmenes de datos, por lo que normalmente requieren almacenamiento de datos externo.

El problema reside en que las soluciones de almacenamiento descentralizado que funcionan bien para los blockchains públicos no son aplicables en los escenarios de permissioned blockchains.

- **Identidad**: Una de las principales diferencias de las soluciones de permissioned blockchain es que las identidades de los participantes en la red son conocidas. Como resultado, muchos de los complejos protocolos de cómputo de consenso no son necesarios en esos escenarios.

2. 3. HYPERLEDGER FABRIC Y COMPOSER:

Hyperledger Fabric es una tecnología ledger distribuida privada aplicable al negocio, privada es la palabra clave que diferencia a Hyperledger de otras tecnologías Blockchain mas orientadas hacia el dominio público como Ethereum y Bitcoin. La aplicación comercial requeriría que el sistema de ledger distribuida tenga ciertas características que son muy diferentes de las características deseadas en una tecnología de Ledger distribuida pública.

2. 3. 1. HYPERLEDGER FABRIC: DTL EN NEGOCIOS

Hay cuatro características qué hacen que Hyperledger fabric sea adecuado para implementar negocios basados en aplicaciones DTL (distributed ledger technology).

Hyperledger fabric es una red permissioned, por lo que soporta transacciones confidenciales, no es precisa la utilización de criptomonedas y es programable.

Con estas características Hyperledger Fabric consigue establecer confianza, transparencia y responsabilidad entre los participantes en la red.

Hyperledger fabric permite crear una red de permisos, proporciona herramientas para que los propietarios de la red puedan restringir quién puede acceder, y qué pueden hacer en la red. Requiere que los participantes en la red sean conocidos y que, para unirse a la red los participantes deban conseguir permisos de la autoridad reguladora.

Hyperledger es una red privada, esto quiere decir que es necesario asignar identidades al los participantes en el proyecto, las empresas negocian con entidades conocidas, proveedores de materias primas o compradores de bienes.

En algunas industrias, por ley, se supone que las empresas solo interactúan con entidades conocidas, por ejemplo, en la industria bancaria, los bancos deben conocer la identidad de cada cliente.

Luego están las agencias reguladoras que interactúan con las empresas. Una tecnología de contabilidad distribuida se basa en aplicaciones para negocios que requieren soporte para administrar identidades en las redes, la aplicación de

negocio define los roles que se asigna a los participantes y el acceso se concede o no en base a estos roles. Un servicio abstracto denominado proveedor de servicios de miembros (membership service provider) se encarga de generar las credenciales para los distintos participantes.

La identidad de los miembros en la red de Hyperledger se gestiona mediante certificados X509, cuando se crea una identidad se le expide un certificado, cada vez que este participante inicia una transacción se certifican las claves privadas en la firma, de forma que cualquier componente de la red puede validar la autenticidad de la transacción, utilizando la clave pública asociada. Los certificados siguen el proceso típico de emisión y la revocación por parte de las autoridades de certificación en la red.

2. 3. 2. ASSETS, CHAINCODE Y TRANSACCIONES

Los Assets o activos, representan algún tipo de valor que se puede intercambiar en el sistema de blockchain. Cualquier objeto de valor en el mundo real puede representarse como un activo en Hyperledger Fabric siempre que se pueda representar digitalmente. En Hyperledger Fabric la representación de activos puede ser en formato JSON o en formato binario. Los cambios de estado de los assets o activos pueden tener lugar en Hyperledger Fabric solo mediante transacciones bien definidas y codificadas mediante unas construcciones llamadas “chaincode” o código de cadena, conceptualmente es el mismo término que smart contract en otras tecnologías de ledger distribuidas. Las empresas pueden utilizar el chaincode para

automatizar los procesos de negocio, el miembro de la red puede ejecutar el código de cadena en el contexto de una transacción que se registra ledger. La automatización del proceso de negocio mediante código de cadena o chaincode conduce a una mayor eficiencia, transparencia y mayor confianza entre los participantes.

Todas las transacciones se registran en un ledger o registro contable, donde se realiza un seguimiento y también se registran los cambios de estado que tienen lugar en los activos, y, como Hyperledger es una tecnología distribuida, todos los participantes tienen una copia de este ledger o registro contable.

2. 3. 3. NODOS Y CANALES

El concepto de nodo es común en todos las tecnologías blockchain, es un punto final de comunicación en una red. Los nodos son las entidades de comunicación del blockchain o cadena de bloques, que se conectan entre sí para formar la red blockchain, los nodos utilizan algún tipo de protocolo peer to peer para mantener el ledger distribuido sincronizado en toda la red.

Los nodos necesitan certificados válidos para poder comunicarse con la red y los participantes o miembros usan las aplicaciones que se conectan a la red a través de los nodos. La identidad de los participantes no es la misma que la identidad de los nodos.

Cuando el participante ejecuta o invoca una transacción, su certificado se utiliza para firmar esa transacción, sin embargo, el certificado del nodo es utilizado por la red para comprobar si el nodo es seguro o no.

Si el certificado de un nodo ha caducado o ha sido revocado, la transacción firmada por un participante con un certificado válido, se transmite a la red, pero la transacción se rechazará porque el certificado que esta usando el nodo no es válido.

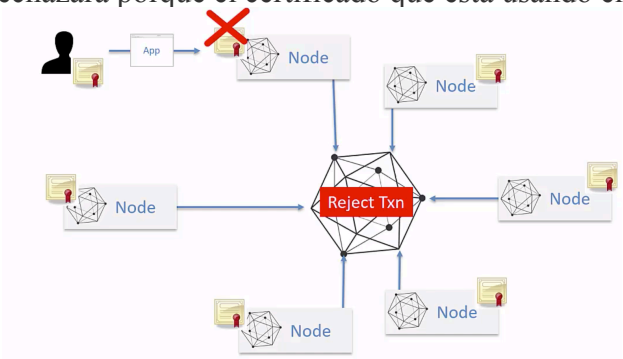


Imagen 8: Ejemplo de transacción rechazada porque un nodo tiene un certificado no válido

En la imagen 8 vemos cómo se rechaza una transacción al contener un nodo inválido. En Hyperledger Fabric, a diferencia de las tecnologías Blockchain de dominio público como Ethereum y Bitcoin, todos los nodos no son iguales.

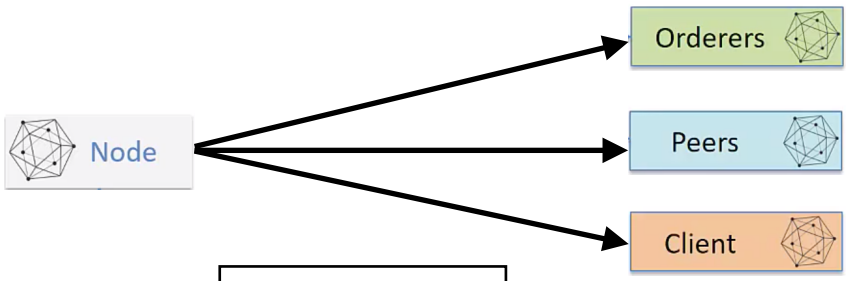


Imagen 9: Tipos de nodos

Hay tres tipos distintos de nodos, como se ve en la imagen 9, el primer tipo es el nodo *cliente*, que usan las aplicaciones para iniciar las transacciones.

El segundo tipo es el nodo *Peer,* que mantienen el ledger sincronizado a través de la red y el tercer tipo, son los nodos *Orderers*, que son los responsables de la distribución de transacciones.

En Hyperledger Fabric el control o visibilidad de las transacciones se restringe mediante la creación de canales privados, de forma que una transacción iniciada en estos canales privados solo serán visibles para los participantes en la misma pero no para todos los demás miembros del blockchain. Los miembros de Hyperledger Fabric pueden crear y participar en multiples canales o redes privadas, pudiendo decidir quién será la autoridad que las valide y qué tipo de políticas se utilizaran para la validación de la transacción.

Los miembros pueden participar en múltiples redes de blockchains de Hyperledger. Los nodos Peers se conectan con los canales y pueden recibir todas las transacciones que se están transmitiendo en ese canal. El canal tiene su propio ledger independiente, de forma que si hay dos canales, se mantienen dos ledgers diferentes en cada uno de ellos, y no hay visibilidad para un peer conectado a un canal en el ledger de otro canal.

2. 3. 4. HYPERLEDGER FABRIC COMPOSER

Hyperledger Composer es una herramienta de desarrollo abierto cuyo objetivo principal es acelerar el desarrollo de las aplicaciones blockchain de Hyperledger. Composer facilita que los equipos creen y administren aplicaciones de red para negocios que se implementan con tecnologías Hyperledger.

La principal ventaja de usar composer es, que además de reducir el tiempo de comercialización, oculta la complejidad de la infraestructura subyacente.

Composer también ofrece capacidad de modelado de negocios mediante un lenguaje que se puede usar fácilmente por miembros no técnicos del equipo como Business Analyst, al permitir que, por ejemplo, los contratos o transacciones inteligentes se pueden codificar en javascript.

El conjunto de herramientas de desarrollo del composer están, por un lado, orientadas a los desarrolladores / arquitectos y a los operadores de la red, personas que necesitan visibilidad en la red, por otro lado, están las herramientas para los administradores, personas que administran las políticas en la red o que crean identidades para los participantes.

Y por ultimo, están las herramientas para el analista de negocios. El desarrollador y el analista de negocios crean las aplicaciones de red utilizando el lenguaje de modelado de composer.

En la creación de una aplicación de negocio mediante composer, existen una serie de fases o runtimes.

En una primera fase, el analista de negocios, utiliza el lenguaje de modelado del composer para crear la red de negocios, el lenguaje utilizado es orientado a objetos para definir el modelo de dominio para la red de negocios. El desarrollador toma el modelo de red empresarial y codifica la especificación de la transacción en el modelo de red empresarial para crear una aplicación final de transacciones basadas en javascript y el modelo de dominio empresarial en el lenguaje de modelado del composer.

En composer existe el concepto de playground, que sería la segunda fase, es un entorno utilizado principalmente por el experto en negocio y por los desarrolladores.

El playground está disponible como una aplicación web, por lo que hay una interfaz de usuario web utilizada para crear el modelo de red empresarial. Este modelo de red empresarial creado por medio del playground se guarda en el almacenamiento local del navegador. El otro propósito para el playground es realizar pruebas simuladas de la aplicación.

Como tercera fase, está el entorno integrado, donde los desarrolladores pueden codificar la aplicación de red empresarial y luego utilizarlo en un simulador integrado basado en nodos para probar la aplicación. Toda la ejecución se lleva a cabo en memoria, el entorno integrado permite la prueba de desarrollo y pruebas unitarias.

3. DESARROLLO APLICACION HYPERLEDGER

3. 1. INSTALACIÓN DEL SOFTWARE

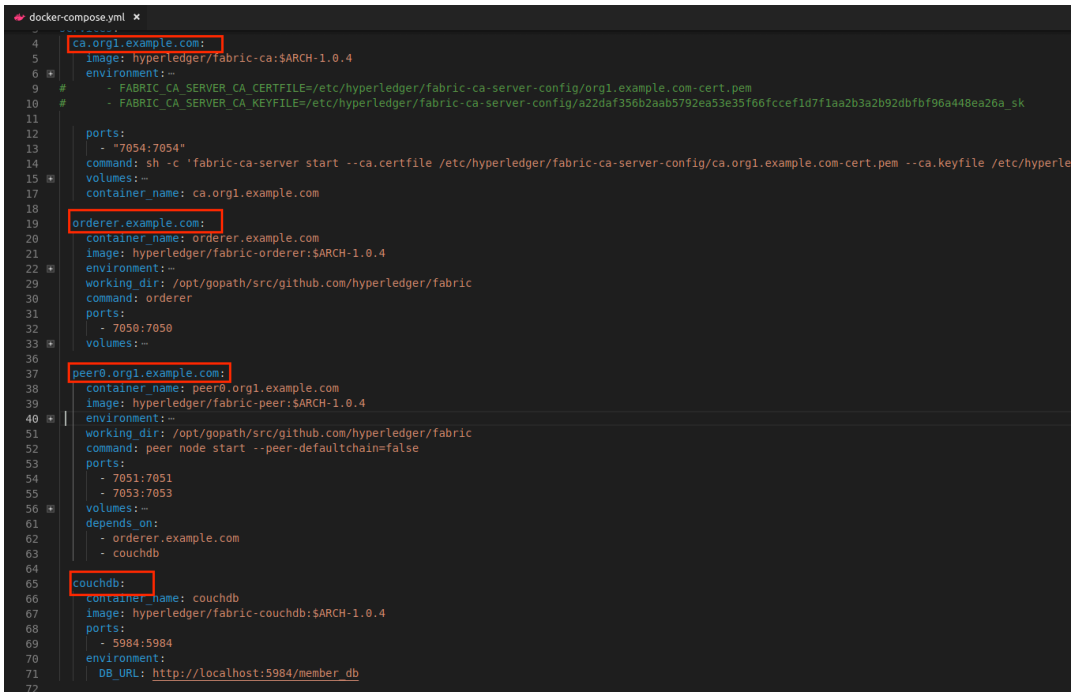
Se deben instalar una serie de programas (sobre los que se ampliará la información en los siguientes apartados), todos ellos open source que hacen posible la posterior instalación tanto de Hyperledger Fabric, como de Hyperledger Composer, y también facilitan el desarrollo de la aplicación final.

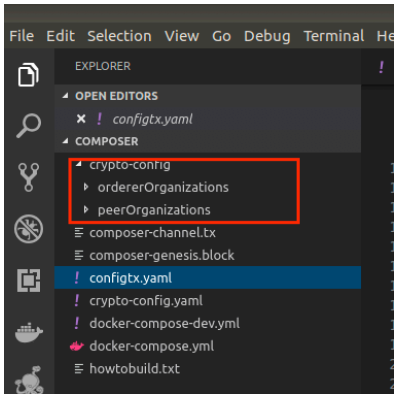
3. 2. TOPOLOGÍA DEL ENTORNO DE DESARROLLO

Especificación de la arquitectura de los componentes que existen en el entorno de desarrollo.

Existen cuatro instancias o partes principales (gestionadas como se verá mas adelante mediante contenedores del programa Docker) que constituyen la infraestructura des entorno de desarrollo. En primer lugar, está la autoridad de certificación, segundo es el orderer, el tercero es el peer , que tiene una dependencia de la cuarta instancia, CouchDB utilizada para los datos de estado.

La configuración de todos estos elementos se gestiona en dos carpetas.

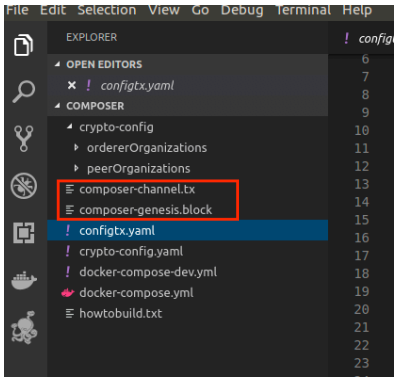




La carpeta crypto tiene todo el material criptológico y la carpeta config con los archivos de configuración.

Las instancias necesitan acceso a canal de información y al bloque genesis que está disponible en la carpeta Composer,

Composer channel.tx, es el archivo de configuración del canal y el Composer genesis.block es el bloque génesis o el bloque cero de la red blockchain.



Todos los componentes de la infraestructura necesitan certificados y estos se controlan con la carpeta crypto-config, donde esta todo el material criptográfico, que se genera con una herramienta denominada *cryptogen*, y para la creación del bloque génesis y el canal de transacción se utiliza *configTx*.

Para explicar esta arquitectura se ha considerado una sola CA (certification authority), pero en un escenario real, donde hay multiples organizaciones que son parte de la red de negocio, puede haber multiples CA, multiples orderers (nunca habrá un único orderer en un entorno de producción), por último, sólo hay un par (peer) en la configuración, y la correspondiente instancia de couchDB que le proporciona la capa de gestión de estado.

Existen también múltiples utilidades de script que están disponibles para la gestión del entorno de desarrollo.

3. 3. REQUISITOS PARA LA INSTALACIÓN DEL ENTORNO

En este apartado se explican los distintos programas y utilidades que se deben instalar previamente a la instalación del entorno de Hyperledger Fabric y Composer.

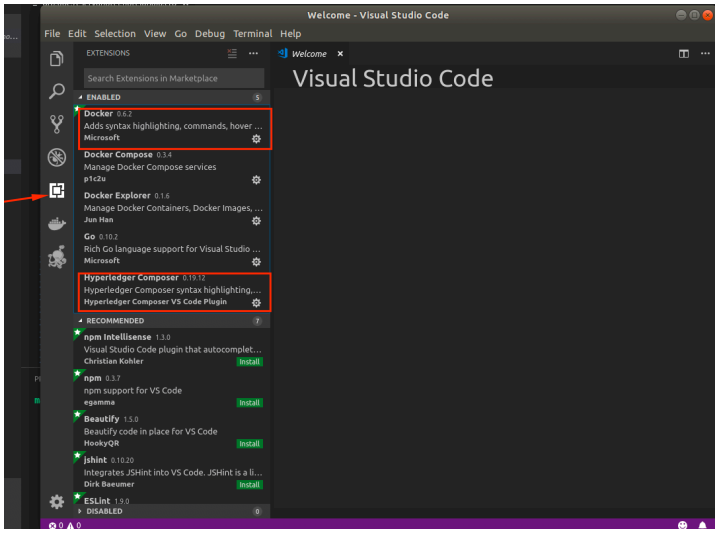
3. 3. 1. IDE VISUAL CODE

Un IDE (Integrated Development Environment ó Entorno de Desarrollo Integrado) es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, puede ser utilizado para varios.Por ejemplo, Dev C++, un IDE para el lenguaje de programación C++.. WebDevStudio, un IDE en línea para el lenguaje de programación C/C++, Eclipse, NetBeans, Visual Studio .Net, RAD Studio XE, ...

Para este proyecto se utilizará Visual Studio Code ya que tiene disponible una extensión de Composer y otra de Docker, lo que facilita el desarrollo para el entorno de Hyperledger.

Se descarga desde https://code.visualstudio.com/download . Después instalamos las extensiones:



3. 3. 2. INSTALACIÓN DE COMPOSER DEV TOOLS

Hyperledger Composer es un conjunto de herramientas que permiten la creación de redes blockchain que simplifican a los desarrolladores la creación de contratos inteligentes o smart contracts, y aplicaciones para resolver los posibles problemas. Se basa en JavaScript, y utiliza herramientas como node.js, npm, CLI y editores. Composer empaqueta todos los archivos en un fichero BNA (Business Network Archive), que permite desplegar la aplicación blockchain fácilmente en varios entornos, ya sean locales o en el cloud, un archivo .bna contiene:

- el modelo que define la aplicación (assets, participantes y transacciones) dentro de ficheros .cto
- la lógica de negocio del modelo (chaincode), es decir la lógica de las transacciones dentro de ficheros .js
- las reglas de acceso a los modelos dentro de ficheros .acl
- las consultas dentro de ficheros .qry
- alguna otra cosa más como un README.md o un package.json

A la hora de elegir un entorno para instalar Hyperledger, es aconsejable utilizar una versión de Linux, en general se recomienda un Ubuntu 16.04 o superior, o un MacOS. Además son necesarios una serie de programas que se enumeran a continuación.

3. 3. 2. 1. NODE JS

Imagen 10: Página de descarga de node

Node JS es un código abierto de JavaScript del lado del servidor, basado en eventos y diseñado para generar aplicaciones web. Node utiliza el motor V8, desarrollado por Google para ejecutar javascript en código de máquina nativo, en lugar de interpretarlo o ejecutarlo como bytecode. Node se ejecuta en Mac OS X, Windows y Linux.

3. 3. 2. 2. PYTHON 2.7

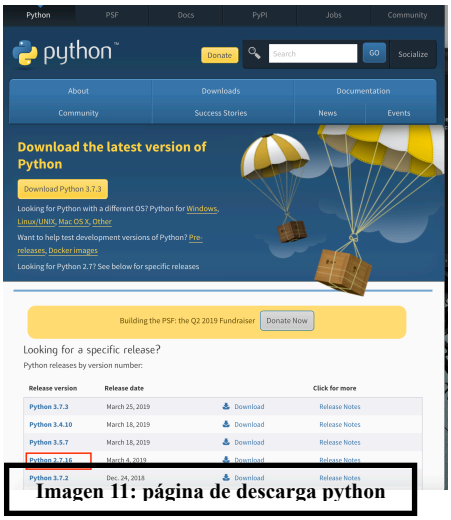


Imagen 11: página de descarga python

Phyton es un lenguaje de programación de alto nivel interactivo e interpretado, de código abierto y de propósito general, multi-plataforma, se adecua a diversos paradigmas de programación y además, es altamente portable.

3. 3. 2. 3. NVM

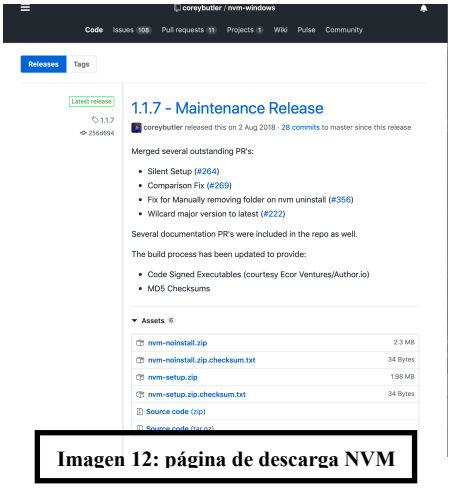


Imagen 12: página de descarga NVM

NVM (Node Version Manager) es una pequeña utilidad que permite mantener múltiples versiones de Node en nuestro sistema las cuales pueden ser cambiadas bajo demanda.Node utiliza versionamiento semántico para cada uno de sus releases. Poseen un proceso muy estructurado para realizar el lanzamiento de una nueva versión.

3. 3. 3. INSTALACIÓN DE DOCKER

Docker es una aplicación que permite crear contenedores ligeros y portables (que contienen los elementos necesarios para que una aplicación se ejecute y la propia aplicación), para que las aplicaciones de software puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando los despliegues.

Imagen 13: logotipo de Docker

En la imagen 13 vemos el logotipo de Docker.

4. MÁQUINA VIRTUAL: INSTALACIÓN DE HYPERLEDGER FABRIC DEV

3. 4. 1. INSTALACIÓN DE VARGRANT

Vagrant es una herramienta open source, disponible para Windows, MacOS X y GNU/Linux, que utiliza *VirtualBox* como motor de máquinas virtuales, y permite generar entornos de desarrollo creando y configurando máquinas virtuales a partir de simples ficheros de configuración. La configuración de la máquina virtual es un fichero de texto plano, por lo que se puede incluir en un repositorio en el control de versiones, junto con el resto del código del proyecto, de forma que clonando este repositorio y ejecutando Vargrant, se tiene disponible el entorno de desarrollo.

3. 4. 2. ESTABLECIENDO LA MÁQUINA VIRTUAL

Virtualbox es un programa de virtualización capaz de instalar en nuestro ordenador cualquier sistema operativo. El programa ha sido creado por la empresa alemana innotek, aunque actualmente se encuentra en manos de Adobe.

Para este proyecto hemos utilizado VirtualBox para la instalación del sistema operativo Ubuntu de Linux en una máquina host basada en MacOS.

3. 5. NATIVE: INSTALACIÓN DEL ENTORNO HYPERLEDGER FABRIC

Para codificar las aplicaciones de blockchain se necesita un entorno de desarrollo, que puede ser creado instalando todos los componentes de Hyperledger Fabric en la máquina, escritorio, portátil, etc… o puede se puede instalar mediante una máquina virtual. Se debe tener en cuenta que Hyperledger Fabric tiene muchas dependencias por lo que antes de poder utilizarlo requiere tener instalado todo el software que hemos mencionado en los apartados anteriores. Los usuarios de Linux pueden configurar muy rápidamente el entorno de desarrollo, ya que la mayoría de los pasos de instalación están automatizados. Los usuarios de Mac están en algún lugar entre los usuarios de Windows y los usuarios de Linux. Con la plataforma Windows se complica, aunque existen muchos problemas que se han informado y resuelto.

Al plantearse la decisión de si optar por una instalación nativa o en máquina virtual, se ha tenido en cuenta el hecho de que la instalación nativa requiere varios softwares acarrea la dedicación de una gran cantidad de tiempo, especialmente si surgen problemas. Por contra, la instalación de la máquina virtual es bastante rápida en comparación con la nativa, además de ser totalmente automatizada. Los choques de versión y otros problemas son comunes en la instalación nativa especialmente en la plataforma Windows, mientras que la instalación de la máquina virtual es autónoma y las versiones se gestionan internamente.

3. 5. 1. MACOS: ESTABLECER EL ENTORNO DE DESARROLLO

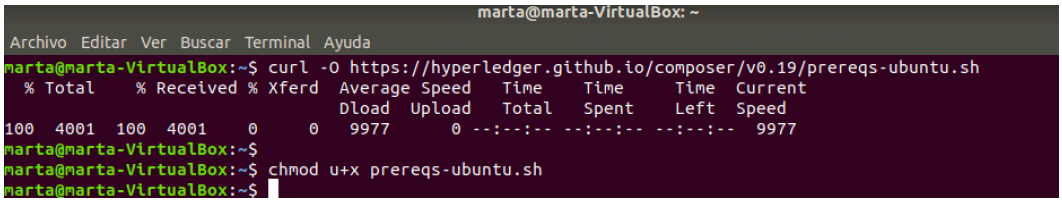
La primera opción para el desarrollo de este proyecto fue realizar la instalación del entorno de Hyperledger de manera nativa en la plataforma con la que trabajo habitualmente que es MacOs, ya que, en principio se puede realizar desde la consola mediante lineas de comando. El método es muy parecido a la instalación en linux aunque cuenta con algunos pasos más que esta última. Debido a que debe haber algún tipo de incompatibilidad entre la versión de Hyperledger y la versión de MacOS Mojave 10.14.5, resultó inviable realizar esta instalación.

3. 6. HERRAMIENTAS DE HYPERLEDGER COMPOSER

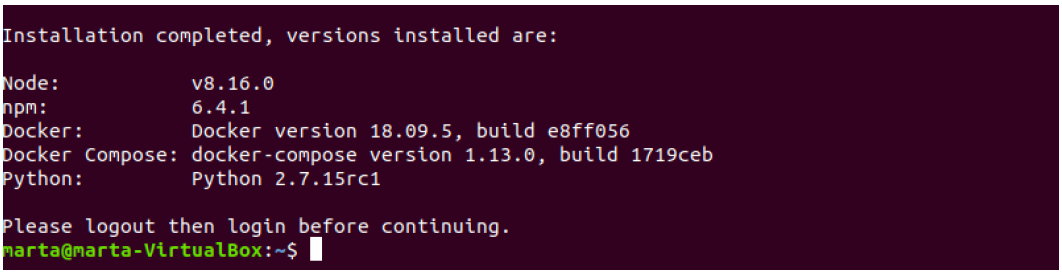
Para la instalación de Hyperledger Fabric, Composer y el resto de herramientas necesarias para la realización de este proyecto, es necesario la instalación de todo el software que se ha mencionado en los apartados anteriores, lo que se denomina "prerequisitos" de la instalación, y que para realizar en la máquina virtual con Linux que voy a utilizar, se realiza mediante los siguientes comandos:

curl -O https://hyperledger.github.io/composer/v0.19/prereqs-ubuntu.sh

chmod u+x prereqs-ubuntu.sh



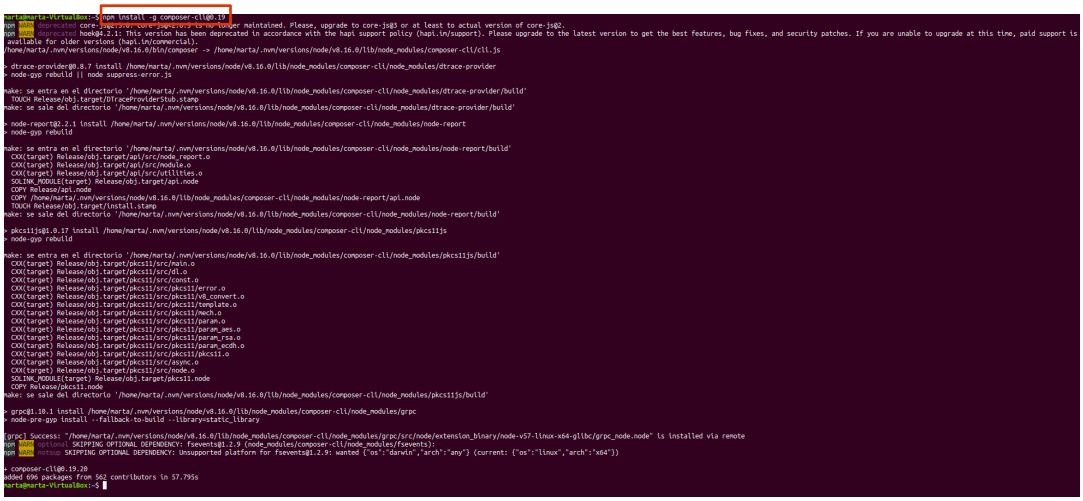
Al ejecutar *./prereqs-ubuntu.sh*, obtenemos:



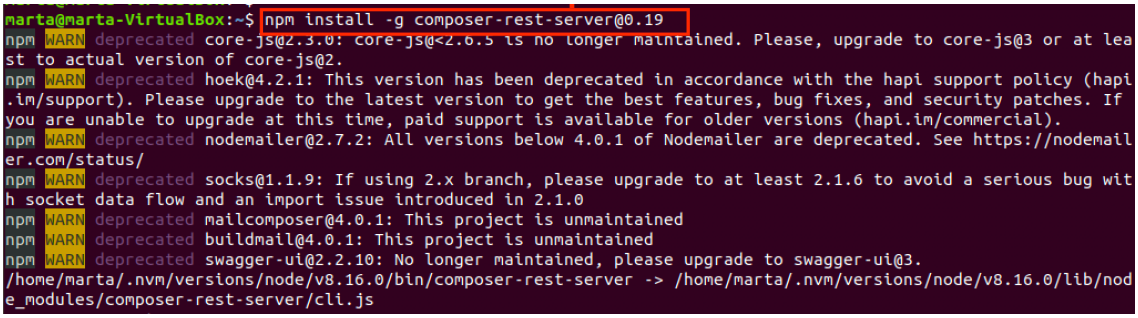
Donde se especifican los paquetes instalados y las correspondientes versiones.

Ahora debemos instalar las herramientas de desarrollo de Hyperledger Composer que utilizaremos para crear la red de negocios (Business Networks), establecer Hyperledger Fabric y desplegar esta red localmente, ya que también se puede desplegar desde Hyperledger Fabric a otros entornos como por ejemplo una plataforma cloud.

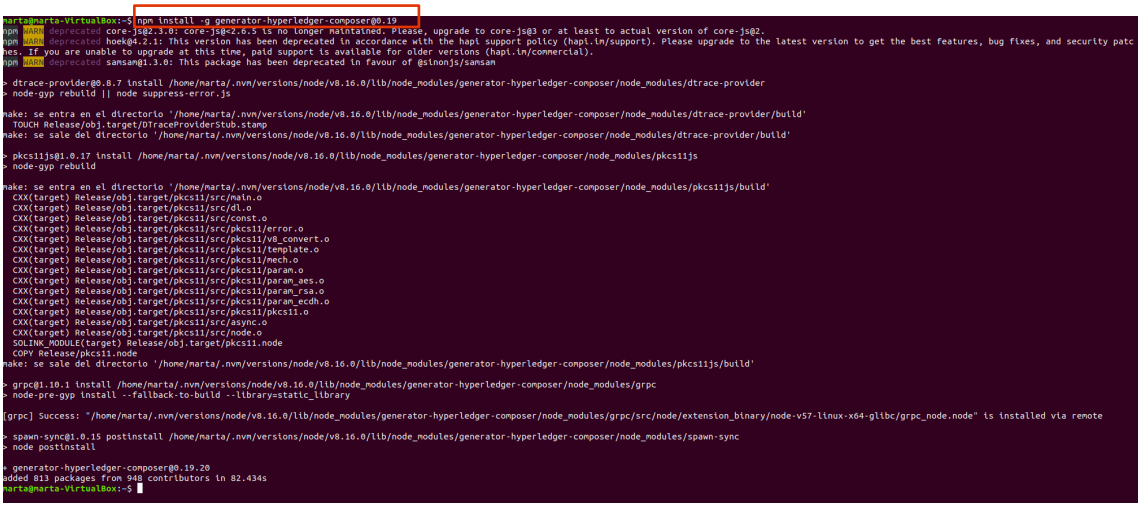
Instalamos los componentes, el más importante el es Composer-CLI, que contiene las operaciones esenciales con el comando *npm install -g composer-cli@0.19*, obteniendo:



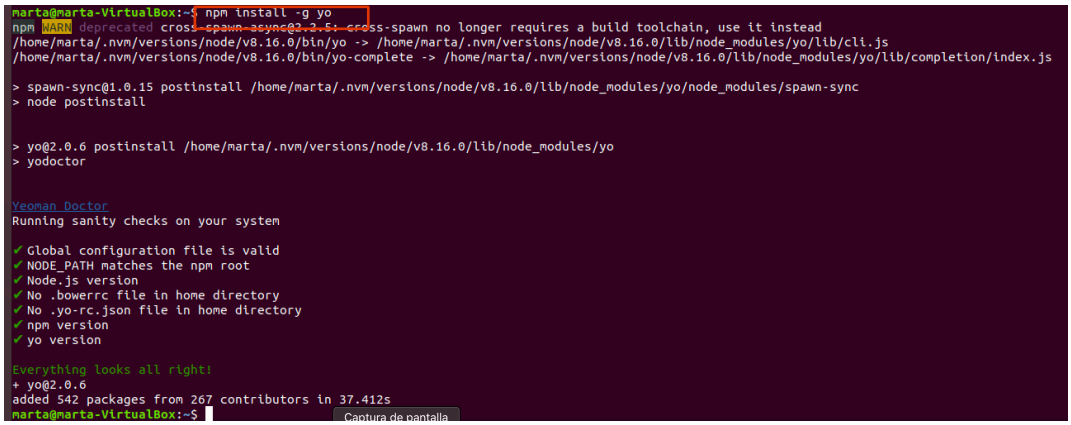
Después instalamos el composer-rest-server para exponer la red de negocio como RestFul API, mediante el comando *npm install -g composer-rest-server@0.19*:



Para generar los assets de la aplicación, instalamos generator-hyperledger-composer, mediante el comando *npm install -g generator-hyperledger-composer@0.19*



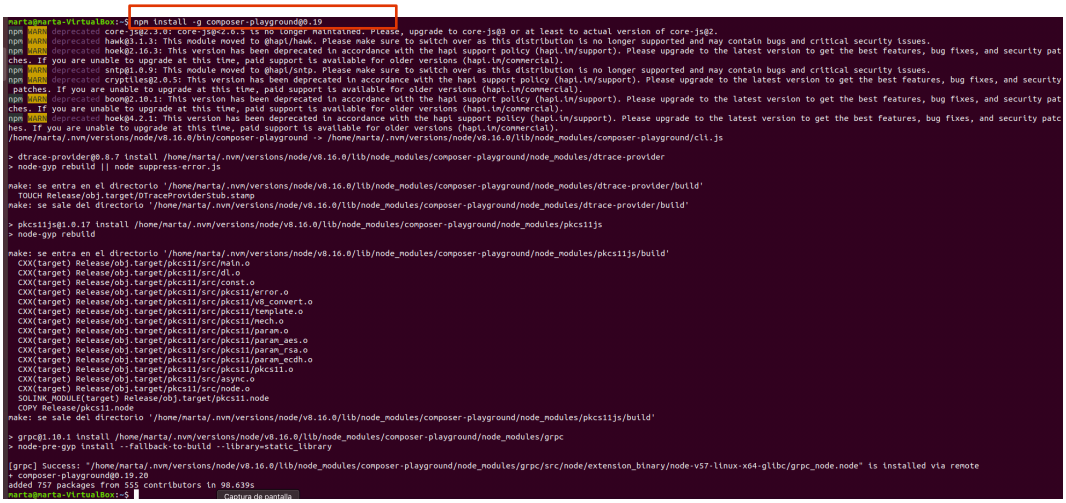
Por último, Yeoman que es una herramienta para generar aplicaciones que utilizará generator-hyperledger-composer, con el comando *npm install -g yo*



3. 6. 1. FABRIC COMPOSER PLAYGROUND

Para ejecutar Hyperledger Composer se utiliza lo que se denomina un "playground", que es un entorno de pruebas donde podemos editar todos los ficheros que se ha desarrollado y desplegarlos en Fabric de manera transparente. Permite modelar y testear una aplicación blockchain fácil y rápidamente.

Para instalarlo utilizamos el siguiente comando *npm install -g composer-playground@0.19*



Para instalar Hyperledger Fabric, creamos un directorio y descargamos el archivo .tar.gz o .zip que contiene las herramientas necesarias para la instalación y utilización, primero se crea un directorio llamado fabric-dev-servers con el comando

mkdir ~/fabric-dev-servers && cd ~/fabric-dev-servers, y desde ahí se descarga el repositorio de git con el comando *curl -O https://raw.githubusercontent.com/hyperledger/composer-tools/master/packages/fabric-dev-servers/fabric-dev-servers.tar.gz*,

```
marta@marta-VirtualBox:~/fabric-dev-servers$ curl -O https://raw.githubusercontent.com/hyperledger/composer-tools
/master/packages/fabric-dev-servers/fabric-dev-servers.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 29471  100 29471    0     0  74047      0 --:--:-- --:--:-- --:--:-- 74047
```

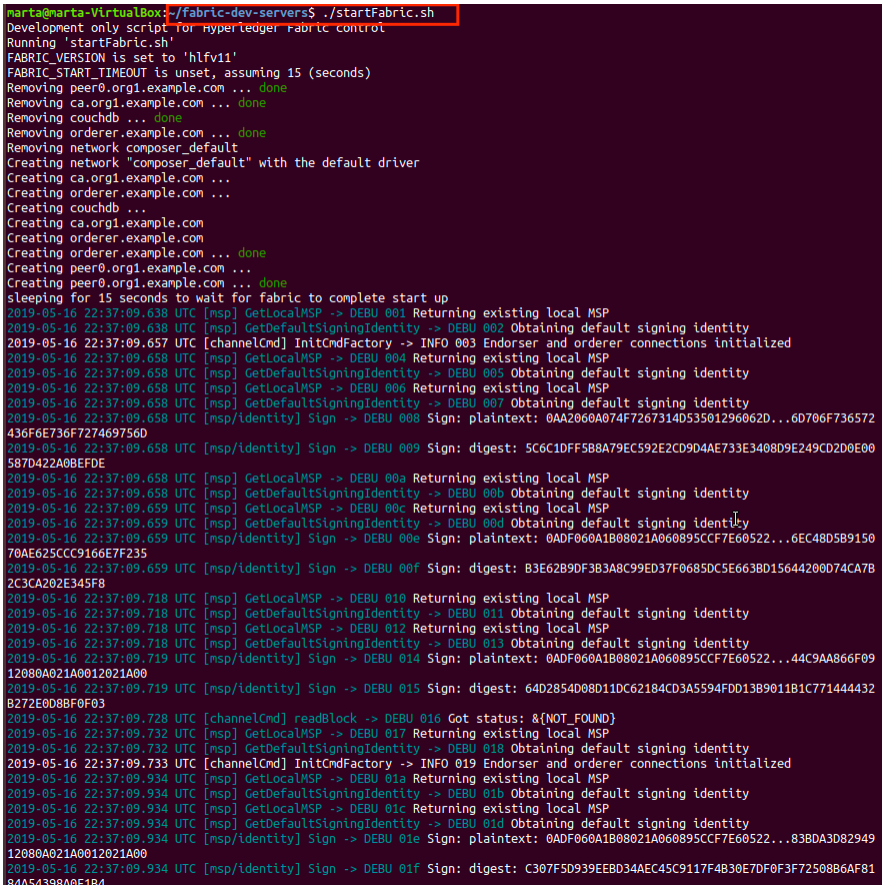
que se tiene que descomprimir mediante el comando *tar -xvf fabric-dev-servers.tar.gz*

```
arta@marta-VirtualBox:~/fabric-dev-servers$ tar -xvf fabric-dev-servers.tar.gz
ackage.json
loader.sh
reateComposerProfile.sh
reatePeerAdminCard.sh
ownloadFabric.sh
tartFabric.sh
topFabric.sh
eardownAllDocker.sh
eardownFabric.sh
abric-scripts/hlfv1/
abric-scripts/hlfv11/
abric-scripts/hlfv12/
abric-scripts/hlfv1/composer/
abric-scripts/hlfv1/createComposerProfile.sh
abric-scripts/hlfv1/createPeerAdminCard.sh
abric-scripts/hlfv1/downloadFabric.sh
abric-scripts/hlfv1/startFabric.sh
abric-scripts/hlfv1/stopFabric.sh
abric-scripts/hlfv1/teardownFabric.sh
abric-scripts/hlfv11/composer/
abric-scripts/hlfv11/createComposerProfile.sh
abric-scripts/hlfv11/createPeerAdminCard.sh
abric-scripts/hlfv11/downloadFabric.sh
abric-scripts/hlfv11/startFabric.sh
abric-scripts/hlfv11/stopFabric.sh
abric-scripts/hlfv11/teardownFabric.sh
abric-scripts/hlfv12/composer/
abric-scripts/hlfv12/createComposerProfile.sh
abric-scripts/hlfv12/createPeerAdminCard.sh
abric-scripts/hlfv12/downloadFabric.sh
abric-scripts/hlfv12/startFabric.sh
abric-scripts/hlfv12/stopFabric.sh
abric-scripts/hlfv12/teardownFabric.sh
abric-scripts/hlfv1/composer/composer-channel.tx
abric-scripts/hlfv1/composer/composer-genesis.block
abric-scripts/hlfv1/composer/configtx.yaml
abric-scripts/hlfv1/composer/creds/
abric-scripts/hlfv1/composer/crypto-config/
abric-scripts/hlfv1/composer/crypto-config.yaml
abric-scripts/hlfv1/composer/docker-compose.yml
abric-scripts/hlfv1/composer/howtobuild.txt
abric-scripts/hlfv11/composer/composer-channel.tx
abric-scripts/hlfv11/composer/composer-genesis.block
abric-scripts/hlfv11/composer/configtx.yaml
abric-scripts/hlfv11/composer/crypto-config/
abric-scripts/hlfv11/composer/crypto-config.yaml
```

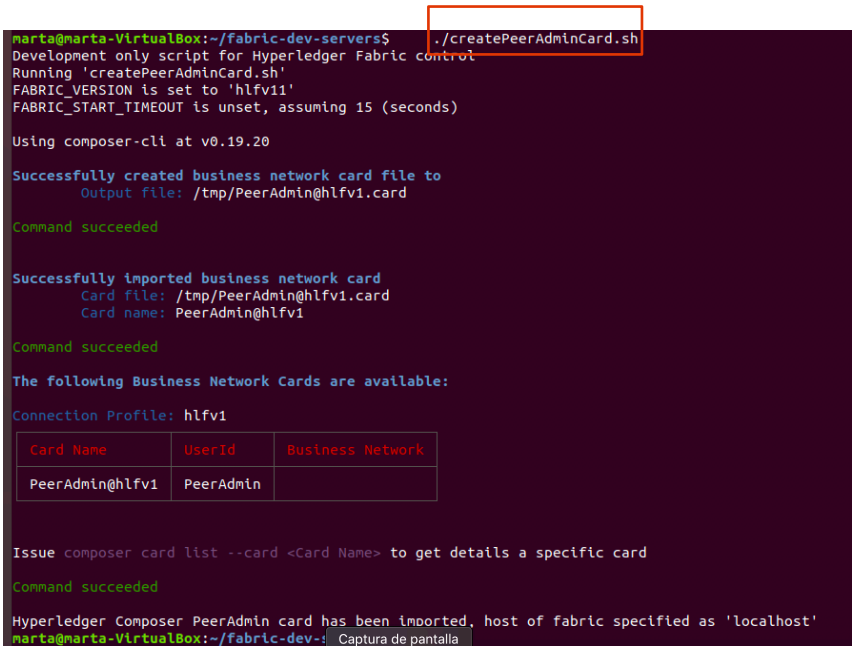
Se utiliza las herramientas que se incluyen en la carpeta que se genera para descargar la rutina de ejecución de Hyperledger Fabric v1.1

```
marta@marta-VirtualBox:~/fabric-dev-servers$ ./downloadFabric.sh
Development only script for Hyperledger Fabric control
Running 'downloadFabric.sh'
FABRIC_VERSION is set to 'hlfv11'
FABRIC_START_TIMEOUT is unset, assuming 15 (seconds)
x86_64-1.1.0: Pulling from hyperledger/fabric-peer
Digest: sha256:57417699ddf50c5ebd47a9a2cc74c0324fbba0281eb1104b9ddd05a67776b01f
Status: Image is up to date for hyperledger/fabric-peer:x86_64-1.1.0
x86_64-1.1.0: Pulling from hyperledger/fabric-ca
Digest: sha256:92f44d0811cddb0d335f7879f7e3b3c4b631f31740c76f3e7b85438c244b03f4
Status: Image is up to date for hyperledger/fabric-ca:x86_64-1.1.0
x86_64-1.1.0: Pulling from hyperledger/fabric-ccenv
Digest: sha256:07818367dc6d4264472d24b21819f9dc4e16e890d81ddfacee0341a22d72050b
Status: Image is up to date for hyperledger/fabric-ccenv:x86_64-1.1.0
x86_64-1.1.0: Pulling from hyperledger/fabric-orderer
Digest: sha256:0c3a3b5ecfd24b513da22bbb77da7b3f5bca9c121cc0ac5c46ba04c97c163654
Status: Image is up to date for hyperledger/fabric-orderer:x86_64-1.1.0
x86_64-0.4.6: Pulling from hyperledger/fabric-couchdb
Digest: sha256:4278269b115cfd0f24251b5381407be9ccdf396c1470c69e1ee2ff16917ac882
Status: Image is up to date for hyperledger/fabric-couchdb:x86_64-0.4.6
```

La primera vez que lo utilicemos, necesitaremos arrancar el script de start, y generar un PeerAdmis card. Para inicializar y finalizar la sesión de desarrollo, utilizamos ./startFabric.sh y ./stopFabric.sh. Cuando finalicemos se debe ejecutar también ./teardownFabric.sh, lo que nos obligara a volver a crear un nuevo PeerAdmin card.

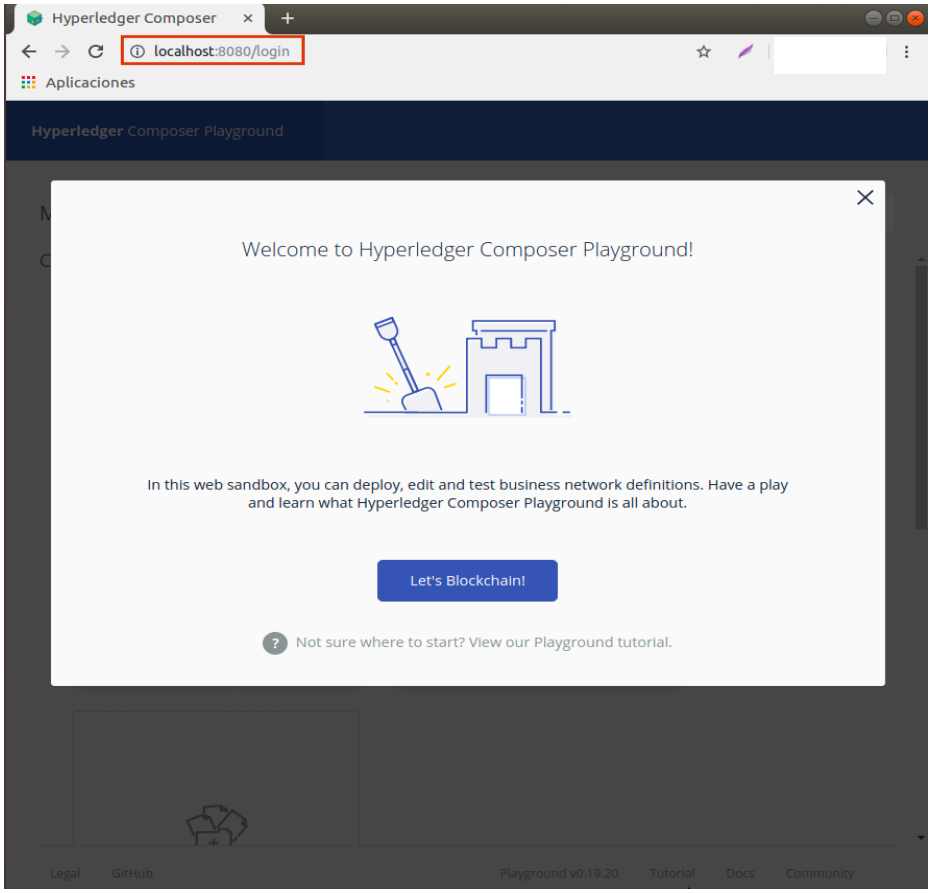


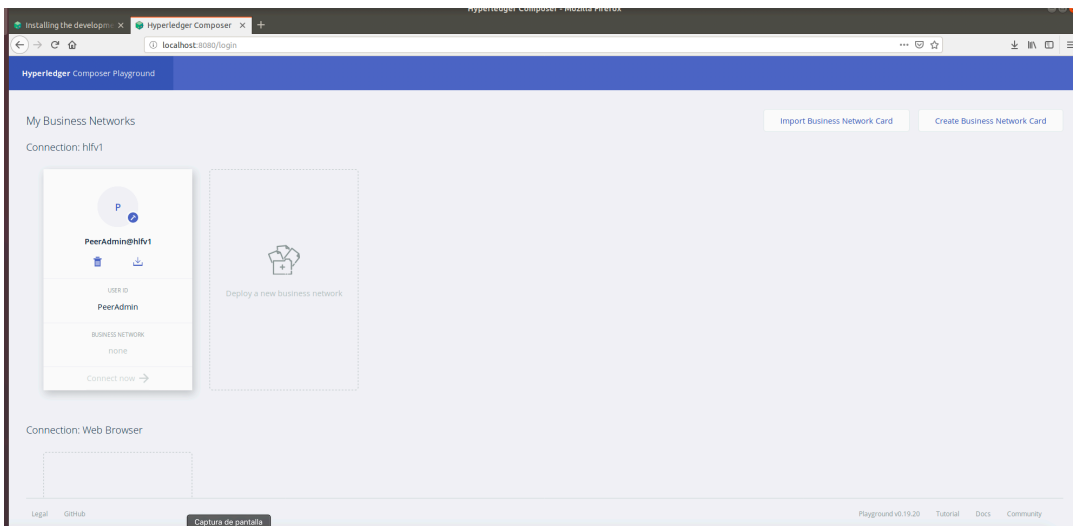
Para crear PeeradminCard, mediante el comando *./createPeerAdminCard.sh*



Para iniciar el playground, ejecutamos *composer-playground* lo que abrirá automáticamente el navegador normalmente en la dirección http://localhost:8080/login

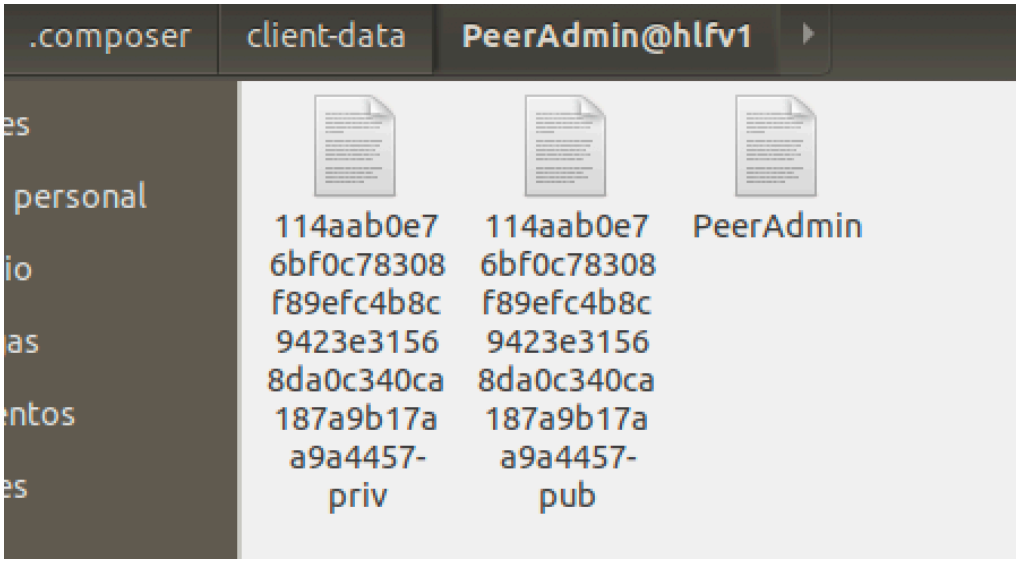
```
marta@marta-VirtualBox:~/fabric-dev-servers$ composer-playground
2019-05-16T22:42:18.990Z INFO    :LoadModule              :loadModule()           Loading composer-wallet-filesystem fro
m /home/marta/.nvm/versions/node/v8.16.0/lib/node_modules/composer-playground/node_modules/composer-wallet-filesystem {}$
2019-05-16T22:42:19.247Z INFO    :PlaygroundAPI           :createServer()         Playground API started on port 8080 {}
$
2019-05-16T22:42:31.305Z INFO    :PlaygroundAPI           :createServer()         Client with ID 'hxBeRu2wBKc4I7g5AAAA'
on host '::1' connected {}$
```





3. 6. 2. ROLES Y TARJETAS DE LA RED DE NEGOCIO

Los usuarios de Hyperledger fabric network pueden realizar acciones en función de sus roles. Hay dos roles de administración para las aplicaciones de la red de negocios.



El primero es Peeradministrator o el administrador de pares (admin peer). Este usuario es responsable de distribuir las actividades a nivel de infraestructura o de nodo y crearlas como parte de la configuración del entorno.

El segundo rol es el administrador de red o el administrador de red. El usuario y este rol es el responsable de llevar a cabo actividades de gestión a nivel de aplicación.
Un usuario en este rol es creado por el administrador de pares.

Los participantes o usuarios de la aplicación de la red de negocio pueden asumir diferentes roles y estos roles se definen como parte del modelo de aplicación. Las acciones que un usuario puede realizar dependen del rol que tenga.

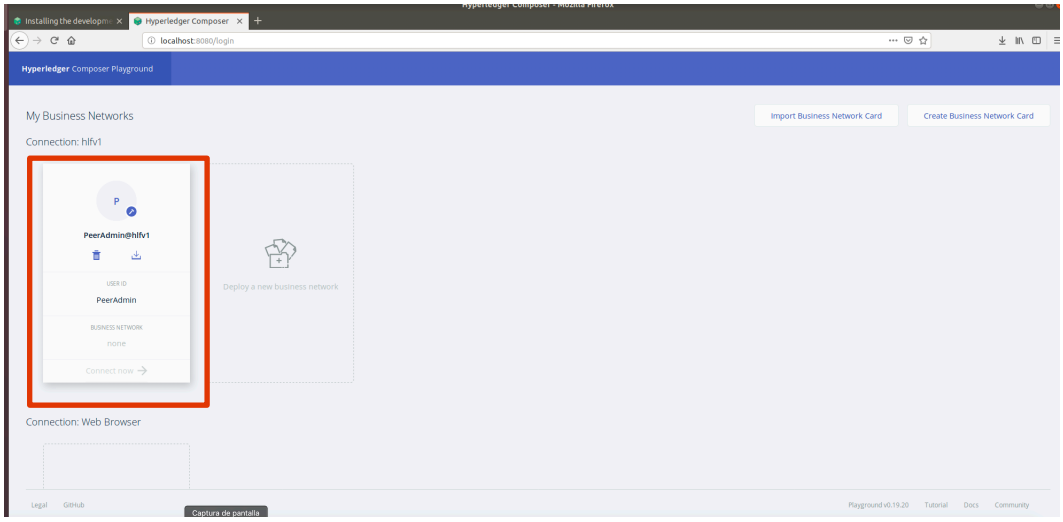
En resumen, el administrador de pares (peers) crea el administrador de red (network administrator) y este, puede crear varios participantes en la aplicación.
El usuario debe tener el rol adecuado, las credenciales y la información sobre cómo llegar a los diversos componentes de la infraestructura y luego también necesitan

algún tipo de aplicación que pueda ser una herramienta de administración, como composer CLI o puede ser una aplicación que utiliza e SDK para la conexión con las aplicaciones de red de negocio y realización de las acciones.

Para que la aplicación pueda funcionar se necesita también lo que se denomina tarjetas o business network cards, que contienen esencialmente las credenciales, las claves, los certificados y un perfil de conexión, necesario para poder conectar a las aplicaciones y componentes de la infraestructura de Hyperledger fabric.

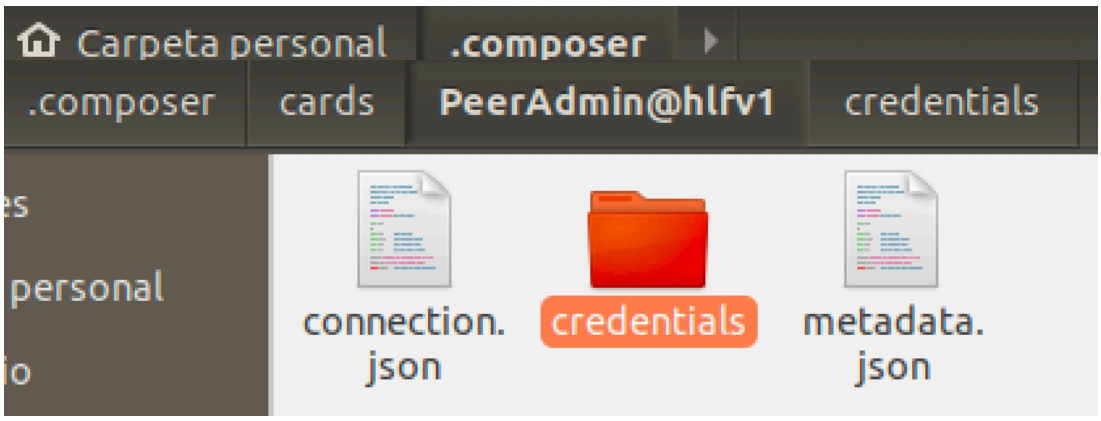
En el perfil de conexión, hay información sobre cómo llegar a la CA y cómo llegar a los peers y los orderers. Un usuario puede tener varias tarjetas o network cards de este tipo configuradas en su máquina para conectarse a diferentes redes empresariales.

En el apartado anterior al instalar el Hyperledger composer playground, este se abría en el navegador mostrando la siguiente pantalla, sonde aparece la card por defecto:



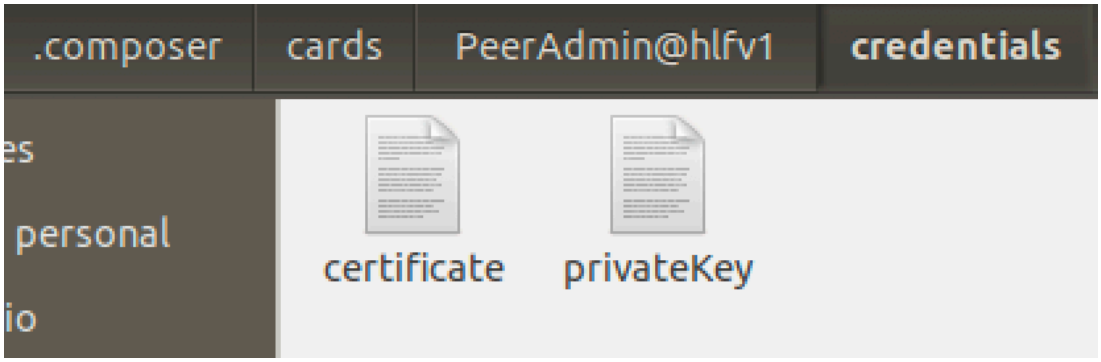
Donde se ve que el nombre de la tarjeta tiene dos partes. Primero hay un nombre que identifica al usuario y luego está el signo @ seguido del nombre de la aplicación o el nombre del nodo.

La carpeta de Composer tiene dos subcarpetas, la subcarpeta de la card y la subcarpeta de datos del cliente, donde se encuentran algunos archivos clave y certificado.



En la subcarpeta de la card tenemos el archivo connection.json que contiene toda la información relacionada con la conexión en formato JSON y otro archivo llamado metadata.json.

Y luego está la subcarpeta con las credenciales donde se almacena la clave privada y el certificado.



Las cards o tarjetas pueden ser creadas, listadas o eliminadas mediante el comando composer card, y las acciones list, create o delete. También se pueden exportar e importar, entre distintas business networks.

3.6. 3. REST SERVER

Una API RESTful es una interfaz de programa de aplicación (API) que utiliza operaciones HTTP para peticiones de datos GET, PUT, POST y DELETE, es decir, las operaciones CRUD, se basa en la tecnología REpresentational State Transfer (REST), un estilo arquitectónico y un enfoque de las comunicaciones que se utiliza a

menudo en el desarrollo de servicios web ,en resumen es un código que permite que dos programas de software se comuniquen entre sí.

El REST utilizado por los navegadores puede ser considerado como el lenguaje de Internet. Con el aumento del uso de la nube, las APIs están emergiendo para exponer los servicios web. REST es una opción lógica para construir APIs que permiten a los usuarios conectarse e interactuar con los servicios cloud. Las API RESTful son utilizadas por sitios como Amazon, Google, LinkedIn y Twitter.

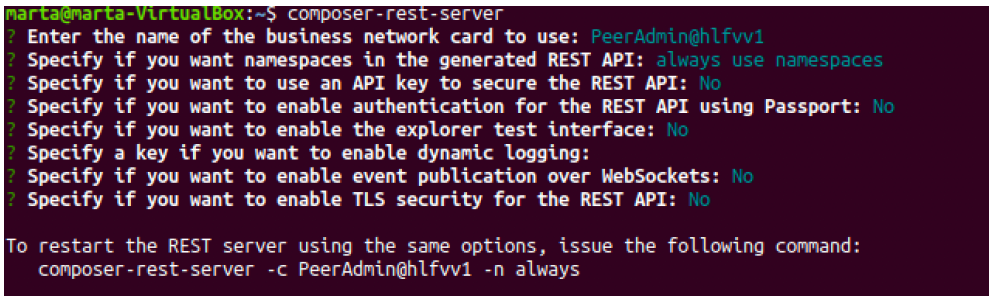
3.6. 3. 1. DESCRIPCIÓN GENERAL

REST Server es un proceso de nodeJS independiente que expone los recursos de red de negocios como API REST.
Esencialmente, es una de las herramientas de Hyperledger Composer que puede utilizar para exponer su modelo de red empresarial en forma de Rest APIs.

El proceso Rest Server se encuentra frente a la aplicación de red de la estructura.

La configuración de la conexión es parte de la configuración del REST Server permitiéndole conectarse con la aplicación de red empresarial.
El desarrollador puede acceder a la url del explorador expuesta por el REST Server para obtener información sobre la API y no solo consultar la información sobre cómo invocar las API REST, también pueden probar las API y realizar cierto nivel de prueba.
Las aplicaciones se pueden conectar directamente al servidor REST en lugar de conectarse con Hyperledger Fabric.
El beneficio de ese enfoque sería que el desarrollador de la aplicación no tendrá que depender de ninguna biblioteca específica de Hyperledger Fabric y el código será mucho más simplificado y fácil de mantener.
Con este tipo de arquitectura, la aplicación puede invocar las transacciones del modelo a través de las API.

El servidor de composer rest puede iniciarse utilizando el comando *composer-rest-server, d*onde se nos va a pedir la introducción de una serie de datos



3. 6. 3. 2. SEGURIDAD DEL REST SERVER

Una vez que se inicia el servidor Rest, cualquier persona que tenga conocimiento de la url podrá acceder a las API y, de hecho, invocar la API, par evitar esto las RESTFul APIs deben configurarse para utilizar TLS o HTTPS y deben requerir autenticación para que el acceso a la API pueda restringirse.

El Rest Server está configurado para la autenticación a través de la configuración. Existen múltiples mecanismos de autenticación para la API REST. El servidor REST no requiere un esquema de autenticación específico debido al hecho de que el servidor REST utiliza el paquete NPM conocido como Passport. Passport permite que las aplicaciones nodeJS implementen sin problemas la autenticación mediante estrategias.

Por lo que, cuando un desarrollador se conecta al servidor REST, debe presentar las credenciales de autenticación según la configuración de la estrategia de autenticación en el servidor REST, que es diferente a la utilizada en el blockchain.

3. 6. 3. 3. SKELETON ANGULAR USANDO YEOMAN

Para utilizar Angular se tener un conocimiento básico y para poder iniciar la aplicación generada, necesitará instalar el CLI Angular.

En esta arquitectura de solución típica, composer se conecta al backend o al tiempo de ejecución compatible a través del SDK de composer y la capa de presentación tiene una aplicación de angular (frontEnd) que se conecta al composer.

En realidad, se puede usar cualquier marco de javascript que no necesariamente tenga que usar angular y no necesariamente tiene que crear una aplicación web que pueda necesitarla donde quiera más seguridad en términos de comportamiento de la aplicación y, en ese caso, puede decida optar por una aplicación de escritorio que utilice el SDK del compositor para conectarse al receptor o directamente a la estructura para que no haya ningún problema, es decir, una aplicación cliente en la que la aplicación de escritorio se conecta directamente al backend de la estructura.
Las ventajas de utilizar Angular con Yeoman, primero, es que puede obtener rápidamente un front y una aplicación, en segundo lugar, la estandarización.El código será consistente, con lo que los desarrolladores podrán comprender el código de los demás rápidamente porque se basa en la misma implementación de línea de base.

Y la tercera es la prueba, por lo que puedes usar la aplicación generada por Yeoman para probar tu propio código.
Hay dos formas en las que puede utilizar el generador de Yeoman con aplicaciones de angular. Las primera, leería el archivo de la red de negocios para obtener la información sobre los diversos activos y recursos y esos activos y recursos se convertirán a JavaScript o representación Typescript que luego utilizará la aplicación angular.
La segunda al iniciar el entorno de Hyperledger Fabric, ejecuta el repositorio del composer y luego ejecuta el generador de aplicaciones angulares de yeoman que se conecta al repositorio para extraer la información sobre las diversas definiciones de activos y así es como crea la representación en typescript.

3. 7. CASO DE ESTUDIO: “APLICACIÓN EJEMPLO”

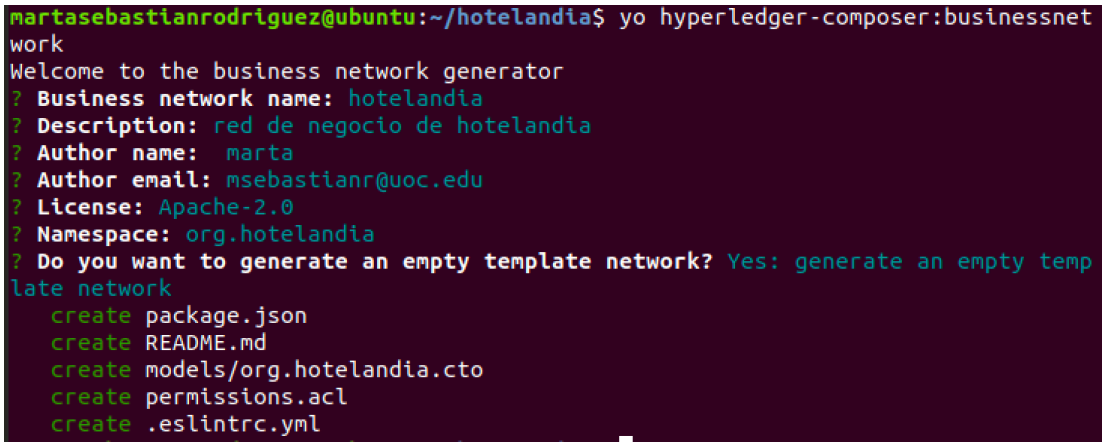
La aplicación ejemplo que voy a desarrollar se basa en un sistema de reserva de habitaciones de hotel.

Partimos de una empresa local que se dedicaba a la reserva de plazas hoteleras a través de internet localmente en Zaragoza. Debido a que están recibiendo muchas peticiones para poder reservar también en otras ciudades españolas incluso a nivel mundial, han decidido unirse a otras empresas similares que operan en otras localidades formando el consorcio que han denominado HOTELANDIA, y buscar una tecnología que les permita crear una plataforma para afrontar de forma segura la transacciones con sus clientes.

El equipo de tecnología ha realizado una intensa investigación y ha decidido que la mejor estrategia es aprovechar Hyperledger Fabric para crear una red de negocios que permita una colaboración B2B eficiente mejorando la experiencia del cliente.

Hyperledger Fabric permite a las empresas socias tener acceso a información actualizada en todo momento, información relativa a las comisiones que generarán a través de contratos inteligentes, e incluso algunos de los flujos de trabajo se construirán en contratos inteligentes que eliminarán la necesidad de usar correos electrónicos.

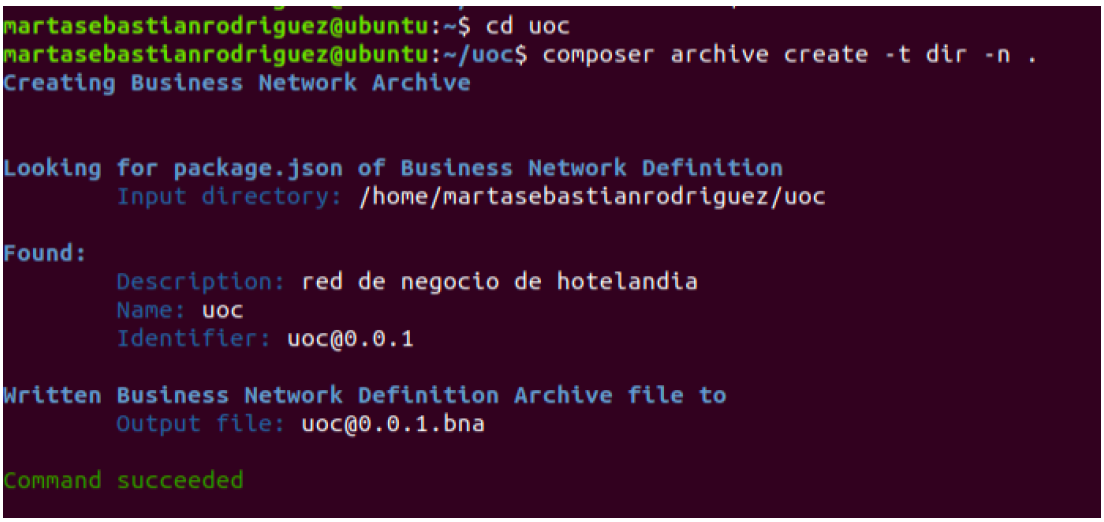
Para crear la estructura de la aplicación utilizamos el comando *$yo hyperledger-composer ,*



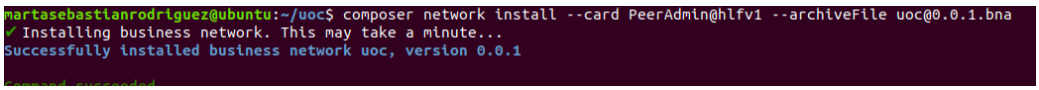
Que muestra una serie de opciones, como son nombre del proyecto y del autor, descripción del proyecto, email del autor, etc.. y después crea los archivos necesarios para su utilización en Fabric

Una vez tenemos la carpeta creada con los archivos de base, vamos a importarlos al Hyperledger playground. Para ello, creamos un archivo .bna (business network archive), Hyperledger lo utiliza para paquetizar los archivos y generar uno nuevo que puede ser desplegado en un red Fabric.Para generarlo, seguimos los siguientes pasos:

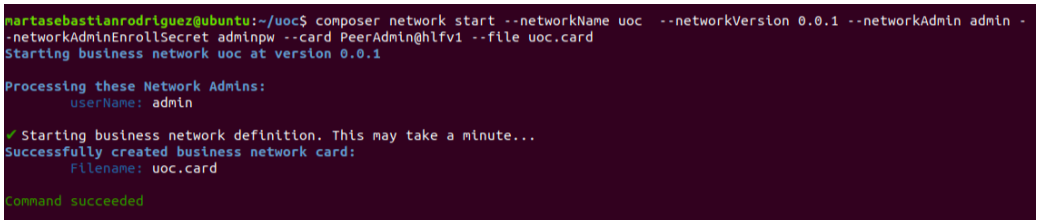
composer archive create --sourceType dir --sourceName ., este comando construye un bna desde el directorio donde se encuentran los archivos de nuestro proyecto.



★ El segundo paso consiste en instalar y desplegar la red en nuestro Fabric local, mediante la conexión PeerAdmin que hemos creado anteriormente al instalar el entorno. Utilizamos el comando: *composer network install --archiveFile uoc@0.0.1.bna --card PeerAdmin@hlfv1*



Para realizar el despliegue utilizamos: *composer network start --networkName uoc --networkVersion 0.0.1 --networkAdmin admin --networkAdminEnrollSecret adminpw --card PeerAdmin@hlfv1 --file uoc.card*



El networkName y networkVersion deben ser las mismas que las especificadas en el package.json o de lo contrario no funciona.

El card generado ha de ser importado a la red de fabric para poder ser utilizado, con el siguiente comando: *composer card import --file uoc.card*

```
martasebastianrodriguez@ubuntu:~/uoc$ composer card import --file uoc.card

Successfully imported business network card
        Card file: uoc.card
        Card name: admin@uoc

Command succeeded
```

Para comprobar que nuestra red está activada y corriendo, utilizamos el siguiente comando: *composer network ping --card admin@uoc —card*

```
martasebastianrodriguez@ubuntu:~/uoc$ composer network ping --card admin@uoc
The connection to the network was successfully tested: uoc
        Business network version: 0.0.1
        Composer runtime version: 0.19.20
        participant: org.hyperledger.composer.system.NetworkAdmin#admin
        identity: org.hyperledger.composer.system.Identity#6a6d2fb86abea3090634aa307585ede08c37c309c50c3220b009415c9a87e76a

Command succeeded
```

3. 7. 1. MODELADO DE LA APLICACIÓN

El lenguaje de modelado de Hyperledger composer es un lenguaje orientado a objetos y se utiliza para definir el modelo de dominio en la red de negocios. Un modelo de dominio empresarial creado con el lenguaje de modelado, define la representación de los participantes, los activos con los que se relacionan estos participantes, las transacciones que realizan los participantes en los activos y los eventos que se emiten como resultado de estas transacciones.

Los modelos se definen en archivos con una extensión .cto. Lo primero que hace en un archivo de modelo es declarar el espacio de nombres, que se aplica a todos los recursos que están definidos en dicho archivo.

Por lo tanto, puede haber varios recursos en el archivo, el espacio de nombres se declara al principio del archivo.

El primer modelo de dominio es Hotelandia, donde hay tres archivos de modelo de dominio, uno contiene el modelo para las empresas socias y el espacio de nombres que utiliza es org.hotelandia.zona, mediante esta clase se definen las localidades partner en el consorcio, con el número de habitaciones disponibles que poseen,

además de otros datos necesarios para el negocio, como el atributo ownershiptype que indica si la empresa local esta en alquiler o comprada por hotelandia.

org.hotelandia.zona.cto

```
namespace org.hotelandia.zona


asset Zona identified by zonaId {
  o String      zonaId

  o Ownership   ownershipType default="LEASED"

  // Number of seats per class
  o Integer     suites      range = [4,]
  o Integer     dobles   range = [6, 20]
  o Integer     individuales    range = [30, ]

  o String      nickName  optional
}

enum Ownership {
  o   LEASED
  o   OWNED
}
```

El segundo tiene el modelo para los hoteles que cada empresa tiene en su zona o localidad, su espacio de nombres sería org.hotelandia.hotel. En el se definen los datos relevantes para la identificación y utilización de los datos de un hotel en concreto.

org.hotelandia.hotel.cto | org.hotelandia.zona.cto

```
namespace org.hotelandia.hotel

import      org.hotelandia.zona.Zona

asset Hotel identified by hotelId {

  o   String           hotelId
  o   String           hotelNumber
  o   Address          address
  -->Zona              zona optional
}

concept Address {
  o   String      code
  o   String      street
  o   DateTime    schedule

}

transaction CreateHotel {
  o   String        hotelNumber
  o   String        code
  o   String      street
  o   DateTime      schedule

}

event HotelCreated {
  o   String      hotelId
}


transaction AssignZona {
  o   String      hotelId
  o   String      zonaId
}


event ZonaAssigned {
  o   String      hotelId
  o   String      zonaId
}
```

El último modelo contiene los participantes, su espacio de nombres será org.hotelandia.participant, lo que recoge los tipos de participantes en la red comercial de la Hotelandia.

org.hotelandia.participant.cto*

```
namespace  org.hotelandia.participant

abstract participant HOTELANDIAParticipant identified by participantKey {
  o String          participantKey
  o Contact         contact
}

concept Contact {
  o String fName
  o String lname
  o String email
}

participant HOTELANDIANetworkAdmin extends HOTELANDIAParticipant {
}

participant HOTELANDIAPersonnel extends HOTELANDIAParticipant {
  o String  department
}

participant B2BPartner extends HOTELANDIAParticipant {

}
```

Este lenguaje de modelado permite especificar ciertas reglas de negocio, el formato string nos permite introducir expresiones regulares para que los datos introducidos se ajusten a un cierto formato.

Se utiliza mediante la expresión regex seguido de la regla de validación, y también mediante los rangos utilizados, por ejemplo, en el número de habitaciones de la clase org.hotelandia.zona.

El modelado también debe incluir la definición de todas las transacciones que pueden ser ejecutadas por los participantes para cambiar el estado de sus activos, cuando se ejecutan las transacciones, los eventos se emiten por lo que también necesitan ser definidos como parte del modelo. Las transacciones y los eventos se desarrollan mediante una clase javascript que se almacena en una carpeta lib en la raíz del proyecto. Las transacciones que he desarrollado para este proyecto, permiten crear hoteles con fecha de ocupación, una función que asigna el HotelId automáticamente desde los datos del hotel, y una función que asigna este hotel a una zona determinada.

```
/**
 * Create Hotel Transaction
 * @param {org.hotelandia.hotel.CreateHotel} hotelData
 * @transaction
 */

function    createHotel(hotelData) {

    var timeNow = new Date().getTime();
    var schedTime = new Date(hotelData.schedule).getTime();
    if(schedTime < timeNow){
        throw new Error("Scheduled time cannot be in the past!!!");
    }



    return getAssetRegistry('org.hotelandia.hotel.Hotel')

        .then(function(hotelRegistry){

            var  factory = getFactory();

            var  NS =  'org.hotelandia.hotel';
            var dt = new Date(hotelData.schedule)

            var month = dt.getMonth()+1;
            if((month+'').length == 1)  month = '0'+month;
            var dayNum = dt.getDate();
            if((dayNum+'').length == 1)  dayNum = '0'+dayNum;



    var  hotelId = hotelData.hotelNumber+'-'+dayNum+'-'+month+'-'+(dt.getFullYear()+'').substring(2,4);



            var  hotel = factory.newResource(NS,'Hotel',hotelId);
            hotel.hotelNumber = hotelData.hotelNumber;
            var address = factory.newConcept(NS,"Address");


            address.code = hotelData.code;
            address.street = hotelData.street;
            address.schedule = hotelData.schedule;



            hotel.address = address;

            var event = factory.newEvent(NS, 'HotelCreated');
            event.hotelId = hotelId;
            emit(event);

            return hotelRegistry.add(hotel);
        });
}


/**
 * Create Hotel Transaction
 * @param {org.hotelandia.hotel.AssignZona} hotelZonaData
 * @transaction
 *
 * **/
function    AssignZona(hotelZonaData){
    var hotelRegistry={}
    return getAssetRegistry('org.hotelandia.hotel.Hotel').then(function(registry){
        hotelRegistry = registry
        return hotelRegistry.get(hotelZonaData.hotelId);
    }).then(function(hotel){
        if(!hotel) throw new Error("hotel : "+hotelZonaData.hotelId," Not Found!!!");
        var   factory = getFactory();
        var   relationship = factory.newRelationship('org.hotelandia.zona','Zona',hotelZonaData.zonaId);
        hotel.zona = relationship;
        return hotelRegistry.update(hotel);
    }).then(function(){
        var event = getFactory().newEvent('org.hotelandia.hotel', 'ZonaAssigned');
        event.hotelId = hotelZonaData.hotelId;
        event.zonaId = hotelZonaData.zonaId;
        emit(event);
    }).catch(function(error){
        throw new Error(error);
    });

}
```

Cuando una transacción es ejecutada , se genera un transactionId y un timestamp (o marca de tiempo), los datos se obtienen del historian y además se genera un evento que se emite asignándole un eventId y un timestamp.

Existe también un archivo donde se almacenan las queries, ya que los datos se utilizan mediante registros que pueden ser consultados mediante estas queries, que utilizan un lenguaje muy parecido al de SQL. Hay dos tipos de queries, las nombradas y las dinámicas. Las nombradas se definen como parte del modelo de la red de negocio y son mostradas en el REST API por el componente REST Server. Las queries dinámicas se crean durante la rutina de ejecución, se necesita el composer API que puede ser usado en la función que procesa la transacción o en el código cliente.

EL archivo queries.qry, contiene las queries que he desarrollado para este proyecto:

queries.qry

```
/**
 * All named queries in this file
 **/

query AllHotels {
  description: "Returns all hotels in the registry"
  statement:
     SELECT  org.hotelandia.hotel.Hotel
}

query HotelByNumber {
  description: "Returns all hotels in the registry"
  statement:
     SELECT  org.hotelandia.hotel.Hotel
     WHERE   (hotelNumber == _$hotel_number)
 }

 query HotelsCode {
  description: "Returns all hotels in the registry"
  statement:
     SELECT  org.hotelandia.hotel.Hotel
     WHERE   (address.code == _$code )
 }

query HotelsCodeOrdered {
  description: "Returns all hotels in the registry"
  statement:
     SELECT  org.hotelandia.hotel.Hotel
     WHERE   (address.code == _$code)
     ORDER BY [hotelNumber ASC]
 }


 query AllZonas {
 description: "Returns all aircrafts in the registry"
 statement:
   SELECT  org.hotelandia.zona.Zona
}
```

Otra parte importante del diseño del proyecto, es la administración de las identidades.

Hyperledger Fabric tiene una implementación de los miembros basada en PKI (public key infraestructure). Cada usuario o participante posee un certificado X509, y existe un proceso para asignar estos certificados, donde el usuario o participante hace una petición para obtener el certificado, la autoridad de registro valida la identidad del participante y manda la petición a la autoridad de certificación o CA, que asigna el certificado que es remitido al participante que lo había solicitado.

Una vez que el participante inicia una transacción un componente de infraestructura de Hyperledger Fabric valida la autenticidad de la transacción comprobando la validez del certificado con la autoridad de validación.

Esto se ha simplificado ya que la card de la red de negocio, contiene las credenciales del criptomaterial que ha sido expedido al usuario.

Para permitir el acceso a los recursos o datos de la aplicación de red de negocio, se utilizan las reglas de control de acceso de composer. Por ejemplo, un participante B2B no debe poder crear ni asignar hoteles a una zona.

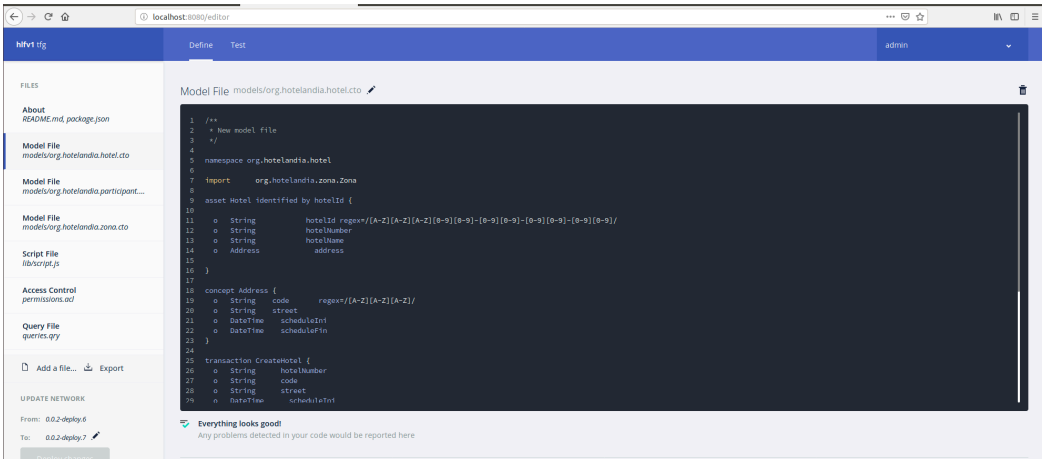
Hay dos formas para implementar el control de acceso, la primera es la via programática donde se implementa como parte de la función que procesa la transacción, el acceso se permite o niega aplicando cierta clase de lógica al contexto del usuario y los datos de la transacción.

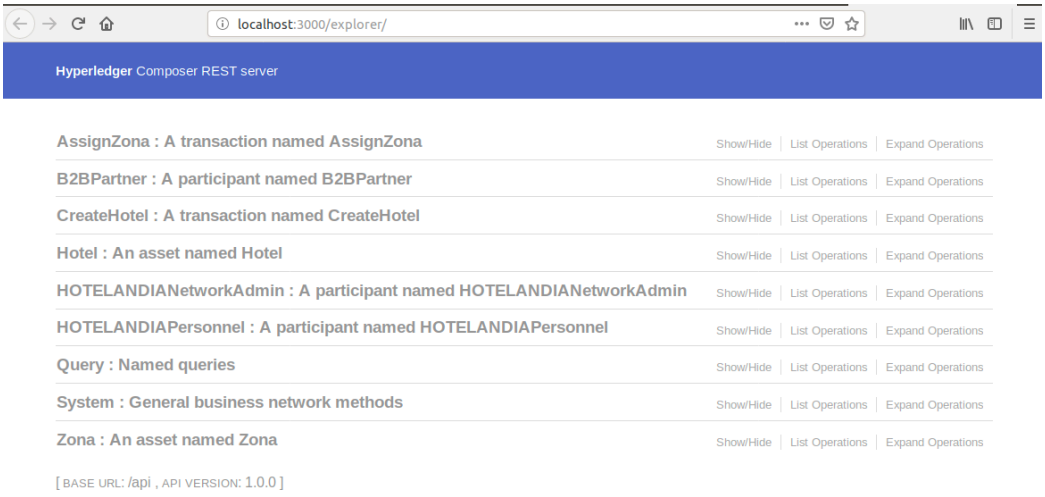
La segunda via es la declarativa en la que las reglas se definen usando el lenguaje de control de acceso de composer, donde se crean las reglas en un fichero que se llama permissions.acl, y se añade a la carpeta raíz del proyecto.

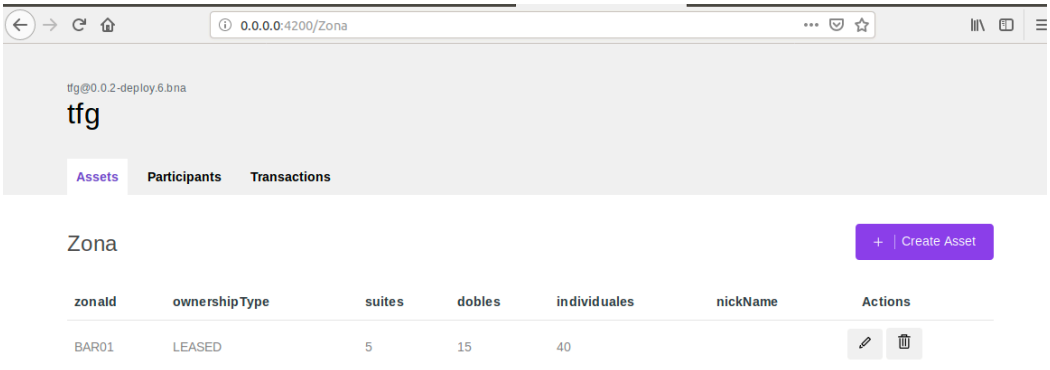
permissions.acl

```
/*
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

rule NetworkAdminUser {
    description: "Grant business network administrators full access to user resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "**"
    action: ALLOW
}

rule NetworkAdminSystem {
    description: "Grant business network administrators full access to system resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}

```

Una vez hemos desplegado todas las clases en nuestro composer local, se obtiene:obtenemos un modelo de aplicación que podemos llamar desde el REST Server, y que a su vez escucha a la app de Angular CLI.







3. 7. 2. IMPLEMENTACIÓN DE LAS APP CLIENTES

En la figura 14, observamos la arquitectura típica de Hyperledger Composer.

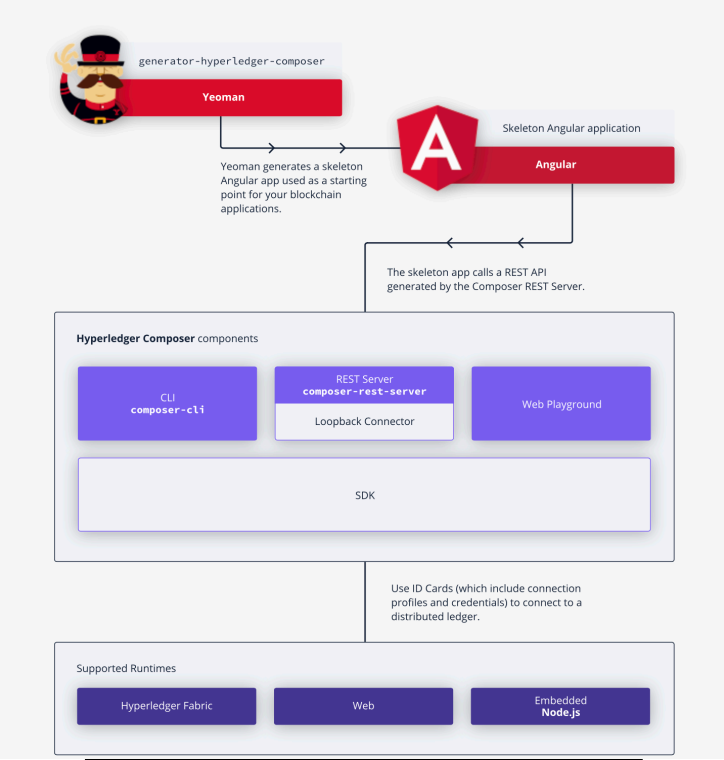


Imagen 14: Arquitectura de Hyperledger Composer

Las APIs se dividen en dos módulos npm:

- composer-client: utilizado para enviar transacciones a una red de negocios o para realizar operaciones de Crear, Leer, Actualizar, Eliminar en activos y participantes. Este módulo se instalaría normalmente como una dependencia local de una aplicación. Proporciona la API que utilizan las aplicaciones empresariales para conectarse a una red empresarial a fin de acceder a los activos, los participantes y las transacciones de presentación. Cuando está en producción, este es el único módulo que necesita ser añadido como una dependencia directa de la aplicación.
- compositor-admin : utilizado para gestionar redes empresariales (instalar, iniciar, actualizar) . Este módulo se instalaría normalmente como una dependencia local de las aplicaciones administrativas. Esta API permite la creación e implementación de definiciones de redes de negocios.

Hyperledger Composer admite la creación de aplicaciones web, Node.js móviles o nativas. Incluye el composer-rest-server (que se basa en la tecnología LoopBack) para generar automáticamente una API REST para una red empresarial, y el complemento de generación de código Hyperledger-Composer para el marco Yeoman que genera una aplicación skeleton angular.

Además, incluye un rico conjunto de API de JavaScript para crear aplicaciones Node.js nativas. El desarrollo de aplicaciones Node.js para trabajar con Hyperledger Composer permite conectarse mediante código a una red empresarial desplegada, crear, leer, actualizar y eliminar activos y participantes, y enviar transacciones.

El SDK JavaScript de Hyperledger Composer es un conjunto de APIs de Node. js que permite a los desarrolladores crear aplicaciones para gestionar e interactuar con las redes empresariales implementadas.

Un usuario interactúa en el tiempo de ejecución de Fabric usando las herramientas de composer que utilizan la card o tarjeta de red de negocio, porque encapsula la información de todas las credenciales, keys, certificados y los perfiles de conexión.

De esta manera el usuario puede tener multiples cards almacenadas en su equipo que las APIs deben poder administrar. Estas cards pueden ser almacenadas en cualquier tipo de almacenamiento, por ejemplo, fileSystem, bases de datos e incluso cloud storage.

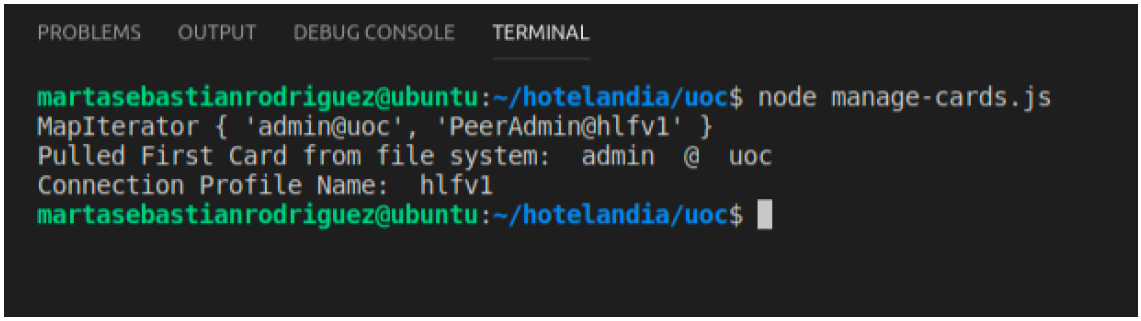
Para utilizar esta información se ha desarrollado la clase manage-cards.js

manage-cards.js

```
'use strict';


// 1. Get the instance of the NetworkCardStoreManager
const NetworkCardStoreManager= require('composer-common').NetworkCardStoreManager;

// 2. Get instance of BusinessNetworkCardStore for filesystem based wallet
var walletType = { type: 'composer-wallet-filesystem' }
const cardStore = NetworkCardStoreManager.getCardStore( walletType );

// 3. Gets all the card
return cardStore.getAll().then(function(cardMap){
    // Print all card names
    console.log(cardMap.keys());

    // 4. Get the name of the first card & then retrieve it
    let firstCard = cardMap.keys().next().value
    // Get the firstCard - returns a promise so check .then()
    return cardStore.get(firstCard);

}).then(function(idCard){

    // get the user and business name
    console.log("Pulled First Card from file system: ", idCard.getUserName(),' @ ', idCard.getBusinessNetworkName())

    // get the connection profile
    console.log("Connection Profile Name: ",idCard.getConnectionProfile().name)

}).catch((error)=>{
    console.log(error)
});

```

Esta clase permite, en primer lugar, obtener la instancia de NetworkCardStoreManager, después crea una instancia de BusinessCardStore para

almacenar el wallet en un archivo, tercero, obtenemos todas las cards instaladas en este archivo y mostramos el nombre por consola, y en cuarto lugar, obtendré la primera card del fileSytem que se mostrará por consola. Para invocar este método utilizo el comando: *node manage.cards.js* desde la consola de Visualcode, obteniendo:



Para que la aplicación pueda actuar en la Fabric Network y las aplicaciones de la red, la aplicación necesita conectar con la ejecución de Fabric. Existen dos clases que hacen esta conexión, Admin Connection y Business Network Connection. Para este proyecto se ha desarrollado la clase admin-connection.js, que será invocada por el resto de clases para ejecutar la conexión con la red de negocio.

Admin Connection está disponible desde el módulo de usuario, una aplicación puede crear una instancia de esta clase para llevar a cabo acciones administrativas en Fabric o realizar acciones administrativas sobre la aplicación de la red de negocio.

Business Network Connection disponible desde el módulo de cliente y es la clase utilizada para ejecutar las transacciones, realizar operaciones CRUD y recibir eventos. Ambas clases requieren llamar a la función de conexión con el nombre de la card.

Para la función de instalación de esta clase, se necesita una instancia de la clase BusinessNetworkDefinition, que es la que llama a la función de instalación para poder invocar la función start.

admin-connection.js

```
'use strict';

// Need the card store instance
const AdminConnection = require('composer-admin').AdminConnection;

// Used as the card for all calls
const   cardNameForPeerAdmin   = "PeerAdmin@hlfv1";
const   cardNameForNetworkAdmin   = "admin@uoc";
const   appToBePinged = "tfg";

// 1. Create Admin Connection object for the fabric
var walletType = { type: 'composer-wallet-filesystem' }
var adminConnection = new AdminConnection(walletType);

// 2. Initiate a connection as PeerAdmin
return adminConnection.connect(cardNameForPeerAdmin).then(function(){

    console.log("Admin Connected Successfully!!!");
    // Display the name and version of the network app
   listBusinessNetwork();
}).catch((error)=>{
    console.log(error);
});


// Extracts information about the network
function listBusinessNetwork(){
    // 3. List the network apps
    adminConnection.list().then((networks)=>{
        console.log("1. Successfully retrieved the deployed Networks: ",networks);

        networks.forEach((businessNetwork) => {
            console.log('Business Network deployed in Runtime', businessNetwork);
         });
        // 4. Disconnect
        return adminConnection.disconnect().then(function(){
            reconnectAsNetworkAdmin();
        });


    }).catch((error)=>{
        console.log("Error=",error);
    });
}

// Ping the network
function reconnectAsNetworkAdmin(){

    // 5. Reconnect with the card for network admin
    return adminConnection.connect(cardNameForNetworkAdmin).then(function(error){

        console.log("2. Network Admin Connected Successfully!!!");
        // 6. Ping the BNA

        adminConnection.ping(appToBePinged).then(function(response){
            console.log("Ping response from "+appToBePinged+" :",response);
            // 7. Disconnect
            adminConnection.disconnect();
        }).catch((error)=>{
            console.log("Error=",error);
        });

    });


```

Para invocar esta clase utilizo el comando: *node admin-connection.js,* obteniendo el siguiente resultado:

```
martasebastianrodriguez@ubuntu:~/hotelandia/uoc$ node admin-connection.js
Admin Connected Successfully!!!
1. Successfully retrieved the deployed Networks:  [ 'uoc' ]
Business Network deployed in Runtime uoc
2. Network Admin Connected Successfully!!!
Ping response from tfg : { version: '0.20.8',
  participant: 'org.hyperledger.composer.system.NetworkAdmin#admin',
  identity: 'org.hyperledger.composer.system.Identity#da96f39b0f684d095eebf6e7eaa8a2a2ee70b6277e2ba4ba074240203029f9e0' }
martasebastianrodriguez@ubuntu:~/hotelandia/uoc$
```

La clase BusinessNetworkDefinition esta disponible en el módulo common, representa a la aplicación de red de negocio, encapsula los metadatos de la aplicación, los archivos de dominio específico (por ejemplo los archivos .cto), y los

códigos que constituyen la lógica de las transacciones. Se ha utilizado para actualizar un despliegue de una versión uoc@0.0.1.bna a uoc@0.0.2.bna.

update-bna.js*

```
'use strict';
/**
 * 1. Create the Admin Connection instance
 * 2. Connect
 * 3. Create the Business Network Definition Object
 * 4. Update the airlinev7 model in runtime
 * 5. Disconnect
 */
const AdminConnection = require('composer-admin').AdminConnection;
const BusinessNetworkDefinition = require('composer-common').BusinessNetworkDefinition;

const cardNameForPeerAdmin = "PeerAdmin@hlfv1";
const appName = "tfg";

const bnaDirectory = "/uoc Project Folder/";

// 1. Create the AdminConnection instance
// Composer 0.19.0 change
var walletType = { type: 'composer-wallet-filesystem' }
const adminConnection = new AdminConnection(walletType);

// 2. Connect using the card for the Network Admin
return adminConnection.connect(cardNameForPeerAdmin).then(function(){
    console.log("Admin Connection Successful!!!");

    // Upgrade the BNA version
    upgradeApp();
}).catch(function(error){
    console.log(error);
});

/**
 * Deploys a network app using the admin connection
 */
function upgradeApp(){
    // 3. Create a Business Network Definition object from directory
    var bnaDef = {}
    BusinessNetworkDefinition.fromDirectory(bnaDirectory).then(function(definition){
        bnaDef = definition;
        console.log("Successfully created the definition!!! ",bnaDef.getName())

        // Install the new version of the BNA
        return adminConnection.install(bnaDef);

    }).then(()=>{

        // 4. Update the application
        // If you do not have the app installed, you will get an error
        console.log("Install successful")
        return adminConnection.upgrade(appName, '0.0.2');

    }).then(()=>{

        console.log('App updated successfully!! ', bnaDef.getName(),'  ',bnaDef.getVersion());

        // 5. Disconnect
        adminConnection.disconnect();

    }).catch(function(error){
        console.log(error);
    });

}
```

La salida cuando ejecutamos esta clase es:

```
martasebastianrodriguez@ubuntu:~/hotelandia/uoc$ node update-bna.js
Admin Connection Successful!!!
Successfully created the definition!!!  uoc
Install successful
```

Se crea una conexión de la aplicación de la red de negocio, usando la instancia de conexión. Se utiliza para la conexión, la desconexión y para comprobar la conectividad mediante un ping a la red. La BussinessNetworkConnection sirve para interactuar con una aplicación de red de negocio desplegada, las funciones utilizadas se usan para recoger información, para comprobar la conectividad mediante la función ping, para acceder a los registros desde runtime, para manejar identidades y para enviar transacciones.

El constructor necesita una instancia de BusinessNetworkCardStore:

new BusinessNetworkConnection(cardStoreObject)

Antes de poder invocar ninguna función, primero se necesita conectar al runtime mediante la función Connect utilizando el nombre de la card como parámetro:

connect(cardName), este card name debe existir en el almacén de cards que se usa para crear la instancia de conexión de red de negocio.

Para desconectar, simplemente se invoca disconnect(), y para comprobar la disponibilidad de la aplicación de red se utiliza la función ping().

businessnetwork-connection-util.js

```
'use strict';

module.exports = {
    // Properties used for creating instance of the BN connection
    cardStore : require('composer-common').FileSystemCardStore,
    BusinessNetworkConnection : require('composer-client').BusinessNetworkConnection,
    // Used for connect()
    cardName : "admin@uoc",

    // Holds the Business Network Connection
    connection: {},

    // 1. This is the function that is called by the app
    connect : function(callback) {

        var cardType = { type: 'composer-wallet-filesystem' }
        this.connection = new this.BusinessNetworkConnection(cardType);

        // Invoke connect
        return this.connection.connect(this.cardName).then(function(){
            callback();
        }).catch((error)=>{
            callback(error);
        });
    },

    // 2. Disconnects the bn connection
    disconnect : function(callback) {
        this.connection.disconnect();
    },

    // 3. Pings the network
    ping : function(callback){
        return this.connection.ping().then((response)=>{
            callback(response);
        }).catch((error)=>{
            callback({}, error);
        });
    }
}
```

Como vemos hay tres funciones en este método, la función connect tiene la función callback cómo parámetro, si se produce la conexión se ejecuta el callback, si la conexión es errónea se ejecuta callback(error). Después están las funciones desconexión y ping, esta última tiene la función callback como argumento con dos

parámetros, response (ping éxito) y error (ping fallido). Para comprobar esta clase, se ha desarrollado la clase de test siguiente:

test-businessnetwork-util.js

```
'use strict';
/**

 * This is for testing the bn-connection-util.js

 */
const bnUtil = require('./businessnetwork-connection-util');

// This creates the business network connection object
// and calls connect() on it. Calls the callback method
// 'main' with error
bnUtil.connect(main);

// Callback function passed to the BN Connection utility
// Error has value if there was an error in connect()
function main(error){
    // 1. Check for the connection error
    if(error){
        console.log(error);
        process.exit(1);
    }

    console.log("1. Successfully Connected !!!");

    // 2. Lets ping
    bnUtil.ping((response, error)=>{
        if(error){
            console.log(error);
        } else {
            console.log("2. Received Ping Response:");
            console.log(response);
        }

        // 3. Disconnect
        bnUtil.disconnect();

        console.log("3. Disconnected");
    });
}

// JavaScript Document
```

Al probarlo en visualCode obtenemos:

```
martasebastianrodriguez@ubuntu:~/hotelandia/uoc$ node test-businessnetwork-util.js
1. Successfully Connected !!!
2. Received Ping Response:
{ version: '0.20.8',
  participant: 'org.hyperledger.composer.system.NetworkAdmin#admin',
  identity: 'org.hyperledger.composer.system.Identity#da96f39b0f684d095eebf6e7eaa8a2a2ee70b6277e2ba4ba0
74240203029f9e0' }
3. Disconnected
```

3. 7. 2. 1. ENVIANDO TRANSACCIONES

Una "*Factory*" se utiliza para crear nuevas instancias de recursos, se accede factory a través de la Business Network Connection y de la Business Network definition. También utilizaremos la clase Resource que representa una instancia de recurso del tipo que se ha definido en el modelo de la aplicación. Para utilizarla se ha desarrollado la clase businessnetwork-factory-submit-transaction.js.

businessnetwork-factory-submit-transaction.js*

```
'use strict';

 // Constant values
 const namespace = "org.hotelandia.hotel";
 const transactionType = "CreateHotel";

// 1. Connect to hotelandia
const bnUtil = require('./businessnetwork-connection-util');
bnUtil.connect(main);

function main(error){

    // Check for error
    if(error){
        console.log(error);
        process.exit(1);
    }

    // 2. Get the Business Network Definition
    let bnDef =  bnUtil.connection.getBusinessNetwork();
    console.log("2. Received Definition from Runtime: ",
                  bnDef.getName()," ",bnDef.getVersion());

    // 3. Get the factory
    let factory = bnDef.getFactory();

    // 4. Create an instance of transaction
    let options = {
        generate: false,
        includeOptionalFields: false
    }
    let HotelId = "ZAR01-01-10-19";
    let transaction = factory.newTransaction(namespace,transactionType,HotelId,options);

    // 5. Set up the properties of the transaction object
    transaction.setPropertyValue('hotelNumber', 'ZAR03');
    transaction.setPropertyValue('code', 'ZAR');
    transaction.setPropertyValue('street' , 'calle tercera');
    transaction.setPropertyValue('schedule' , new Date('2019-10-01T18:18:12.432Z'));

    // 6. Submit the transaction
    return bnUtil.connection.submitTransaction(transaction).then(()=>{
        console.log("6. Transaction Submitted/Processed Successfully!!")

        bnUtil.disconnect();

    }).catch((error)=>{
        console.log(error);

        bnUtil.disconnect();
    });
}
```

La función getBusinessNetwork() se puede usar para extraer información sobre la aplicación desde el tiempo de ejecución. Cuando se invoca esta función, se recibe una instancia de la clase Business Network definition, que se crea desde un archivo o desde el modelo de negocio en un directorio.

La clase getFactory() se invoca desde Business Network definition recuperado del tiempo de ejecución y devuelve una instancia de factory que se puede usar para crear instancias de recursos que se han definido en la business network application.

Hay cinco funciones en la clase de factory, la primera es newResource(namespace, type, identifier) que toma el espacio de nombres, el tipo y el identificador como parámetros que se asignarán a la nueva instancia, recurso disponible en el módulo común. La siguiente función es el newConcept(namespace, type, options) que tienen como parámetros el espacio de nombres y el tipo de concepto definido en la aplicación de red empresarial, esta clase Concept también pertenece al módulo común. La siguiente función es newRelationship(namespace, type, identifier) que tiene el espacio de nombres, el tipo y el identificador, como atributos, y también pertenece al módulo común. La clase Relationship() representa un puntero al recurso identificado por el identificador y esta instancia de relación puede asignarse a un campo que es un campo de relación.

La función newEvent(namespace, type, id, options) tiene el espacio de nombres, el tipo, la ID del evento que es opcional y algunas opciones como parámetros y crea una instancia Resource del tipo de evento.

La quinta función es el newTransaction(namespace, type, identifier, options) que tiene el espacio de nombres, el tipo y el identificador y las opciones como parámetros. La llamada a esta función crea una instancia de la clase Resource con el tipo igual a la transacción, que puede enviarse para su procesamiento utilizando la instancia de business network connection conectada al tiempo de ejecución y para lo que se utiliza la función submitTransaction(transaction) que utiliza el objeto transacción creado con la función newTransaction().

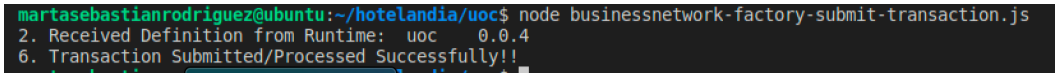
La clase Resource está disponible en el módulo común, representa una instancia de recurso que tiene un tipo *type, que* aquí se refiere a las definiciones de recursos creadas como parte del modelo.

Por ejemplo, en el caso del modelo de Hotelandia, hemos definido un activo de tipo Zona y el activo de tipo hotel y también hemos definido una transacción createHotel, de modo que createHotel es el tipo de recurso, luego la instancia de createHotel se puede crear utilizando el método factory.newTransaction.

En este código que se ha desarrollado se envía una transacción de createHotel. Se realiza en seis pasos:

Primero se usa businessnetwork-connection-util para crear la conexión a Hotelandia. Luego se obtiene la business network definition del tiempo de ejecución utilizando la definición Get Business() y entonces se obtiene la factory de la Business Network definition, se crear una instancia con la función newTransaction(), y se establecen los valores en el objeto transaction y luego se llama a la función submitTransaction() de la business network connection.

La salida que obtengo al ejecutar el código es:



3. 7. 2. 2. REGISTROS

La clase Registry es una clase abstracta que representa un registro en el tiempo de ejecución de Composer, esta disponible en el módulo cliente o Client.

Se crean múltiples registros en la aplicación de red de negocios del tipo de Resources o recursos definidos en el modelo de aplicación, por ejemplo, registro de participantes o registros de activos

Cada vez que se crea una instancia del participante, se agrega una instancia al registro de participantes.

De manera similar, los activos se agregan al registro de Activos. Registro es una clase abstracta que representa un registro en el tiempo de ejecución del compositor.

Hay múltiples clases concretas de implementación de registro.

El primero es el registro de activos que representa el registro para los activos definidos en el modelo, en el caso de las Hotelandia, tenemos los registros de las zonas y los hoteles.

Luego tenemos el registro de participantes, hay un personal de Hotelandia y el registro B2BPartner.

El Registro de transacciones como su nombre indica es para las transacciones, en el proyecto, el registro createHotel de Hotelandia.

El registro de historia es un registro de solo lectura que almacena los registros de historia de las transacciones.

Y luego está el registro de identidades, que de nuevo es de solo lectura y administra un conjunto de identidades en la cadena.

Para obtener acceso a la instancia de registro, la aplicación necesita conectarse al tiempo de ejecución de fabric utilizando la the business network connection y luego invocar getRegistry(FQ Registry Name) con el nombre de registro completo en la business network connection. La respuesta a esta llamada es una instancia de la clase de registro.

Existe otro conjunto de funciones que son específicas del tipo de registro para obtener la instancia de registro. Estas funciones son el getAssetRegistry(Registry Name) donde debe especificar solo el nombre del registro, getParticipantRegistry(Registry Name) y el getTransactionRegistry(Registry Name).

Hay dos funciones get adicionales: getHistorian() y getIdentityRegistry(). Aquí no tiene que especificar los nombres del registro ya que solo hay una instancia del historiador y una instancia del registro de identidad.

El siguiente conjunto de funciones permite que las aplicaciones reciban una lista de registros del tipo específico de recursos. getAllAssettRegistries() devolverá todos los registros de activos en la aplicación, getAllParticipantRegistries() y getAllTransactionRegistries() que devuelven una lista de instancias de registro.

En el código desarrollado hay seis partes:

El primero es conectarse al tiempo de ejecución de Fabric utilizando businessnetwork-connection-util.js.

Luego se obtienen los registros de activos e se imprime el nombre de esos registros de activos, lo mismo sucede con el participante y los registros de transacciones.

Se obtiene el historial y el registro de identidad y se imprimen los nombres completos de los mismos.

get-registries.js

```
'use strict';

const bnUtil = require('./businessnetwork-connection-util');

// This creates the business network connection object
// and calls connect() on it. Calls the callback method
// 'main' with error
bnUtil.connect(main);

// This gets invoked after the promise for connect is
// resolved. Error has value if there was an error in connect()
function main(error){
    // Check for the connection error
    if(error){
        console.log(error);
        process.exit(1);
    }

    // This is where you would code the app specific stuff
    // Connection is available bnUtil.connection
    // to disconnect use bnUtil.disconnect()

    // 2. Get All asset registries...arg true = include system registry
    bnUtil.connection.getAllAssetRegistries(false).then((registries)=>{
        console.log("Registries");
        console.log("==========");
        printRegistry(registries);
        // 3. Get all the participant registries
        return bnUtil.connection.getAllParticipantRegistries(false);
    }).then((registries)=>{
        printRegistry(registries);

        // 4. Get all the transaction Registries
        return bnUtil.connection.getAllTransactionRegistries();
    }).then((registries)=>{
        printRegistry(registries);

        // 5. Get the Historian Registry
        return bnUtil.connection.getHistorian();

    }).then((registry)=>{
        console.log("Historian Registry: ", registry.registryType, "   ", registry.id);

        // 6. Get the Identity Registry
        return bnUtil.connection.getIdentityRegistry();

    }).then((registry)=>{
        console.log("Identity Registry: ", registry.registryType, "   ", registry.id);

        bnUtil.connection.disconnect();
    }).catch((error)=>{
        console.log(error);
        bnUtil.connection.disconnect();
    });
}

// Utility function to print information about the registry
function printRegistry(registryArray){
    registryArray.forEach((registry)=>{
        console.log(registry.registryType, "   ", registry.id);
    });
}
```

La salida que obtengo al ejecutar el código es:

```
martasebastianrodriguez@ubuntu:~/hotelandia/uoc$ node get-registries.js
Registries
==========
Asset     org.hotelandia.hotel.Hotel
Asset     org.hotelandia.zona.Zona
Participant     org.hotelandia.participant.B2BPartner
Participant     org.hotelandia.participant.HOTELANDIANetworkAdmin
Participant     org.hotelandia.participant.HOTELANDIAPersonnel
Transaction     org.hotelandia.hotel.AssignZona
Transaction     org.hotelandia.hotel.CreateHotel
Historian Registry:  Asset     org.hyperledger.composer.system.HistorianRecord
Identity Registry:  Asset     org.hyperledger.composer.system.Identity
```

3. 7. 2. 3. RESOURCES O RECURSOS

Las nuevas instancias de la clase Resource se crean usando la clase factory y las que ya existen se reciben desde el registro usando la clase Registry.

add-resources.js

```
'use strict';
const zonaNamespace = 'org.hotelandia.zona';
const zonaType = 'Zona';

// 1. Connect
const bnUtil = require('./businessnetwork-connection-util');
bnUtil.connect(main);

function main(error){
    // Check for the connection error
    if(error){
        console.log(error);
        process.exit(1);
    }
    // 2. Get the zona AssetRegistry
    return bnUtil.connection.getAssetRegistry(zonaNamespace+'.'+zonaType).then((registry)=>{
        console.log('1. Received Registry: ', registry.id);

        // Utility method for adding the zonas
        addZonas(registry);

    }).catch((error)=>{
        console.log(error);
        // bnUtil.disconnect();
    });
}
/**
 * Creates two resources instances
 * @param {*} registry This is of type AssetRegistry
 */
function    addZonas(registry){
    // 3. This Array will hold the instances of zonas resource
    let    zonas = [];
    const  bnDef = bnUtil.connection.getBusinessNetwork();
    const  factory = bnDef.getFactory();
    // Instance#1
    let    zonaResource = factory.newResource(zonaNamespace,zonaType,'BAR');
    zonaResource.setPropertyValue('ownershipType','LEASED');
    zonaResource.setPropertyValue('suites',10);
    zonaResource.setPropertyValue('dobles',10);
    zonaResource.setPropertyValue('individuales',50);
    // Push instance to  the zonas array
    zonas.push(zonaResource);

    // Instance#2
    zonaResource = factory.newResource(zonaNamespace,zonaType,'BAR');
    // You may use direct assignment instead of using the setPropertyValue()
    zonaResource.ownershipType='OWNED';
    zonaResource.suites=15
    zonaResource.dobles=20;
    zonaResource.individuales=100;
    // Push instance to  the zonas array
    zonas.push(zonaResource);

    // 4. Add the zona resource to the registry
    return registry.addAll(zonas).then(()=>{
        console.log('Added the Resources successfully!!!');
        bnUtil.disconnect();
    }).catch((error)=>{
        console.log(error);
        bnUtil.disconnect();
    });
}
```

Hay varias funciones getter en la clase add-resource. getType() devuelve el tipo de instancia, getNamespace() devuelve el namespace del tipo de resource, getFullyQualifiedType() devuelve el namespace.name del recurso, getIdentifier() devuelve el identificador de la instancia, getFullyQualifiedIdentifier() devuelve el namespace# seguido del Id de la instancia, instanceOf(Fully Qualified Type) devuelve true o false dependiendo de si el argumento encaja con el tipo cualificado del recurso, isRelationship() chequea que la instancia represente una relationship, también devuelve true o false. setIdentitier(new-identifier) se usa para cambiar la identidad de la instancia, setPropertyValue(prop-name, value) se

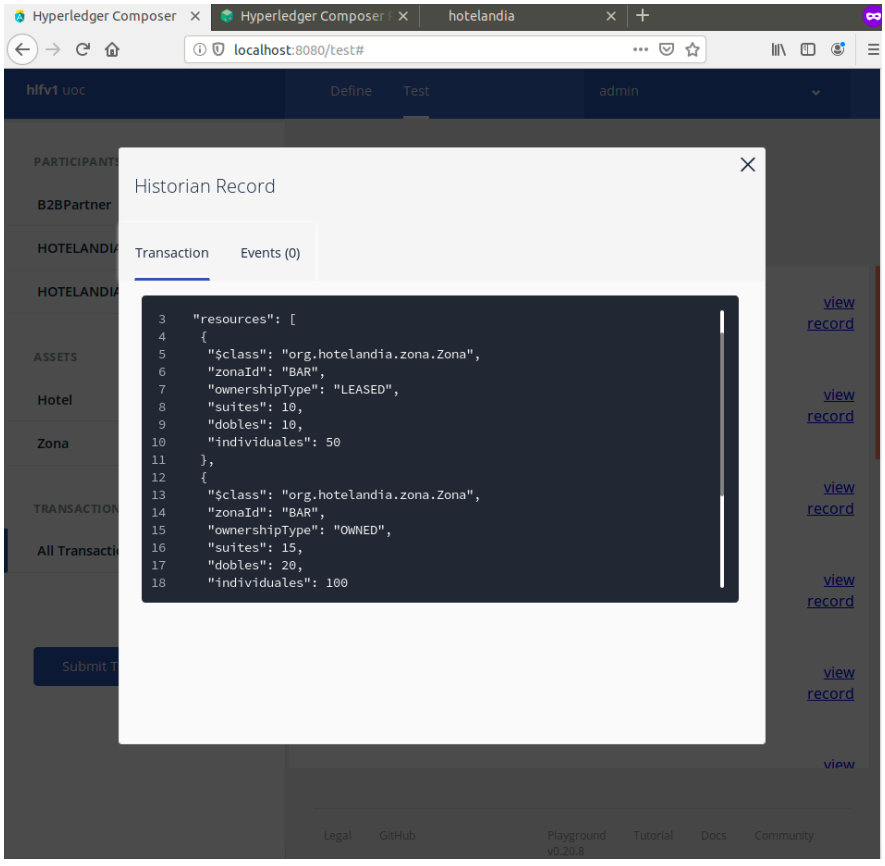
usa para establecer los valores de propiedad en la instancia, addArrayValue(prop-name, value) permite añadir un nuevo elemento al array de propiedades.

En el código desarrollado se crean dos instancias del recurso zona usando la factory y la función de inicialización setProperty y después se invoca la función addAll para añadir las instancias creadas al tiempo de ejecución .

Al ejecutar el código la respuesta que obtengo es:

```
martasebastianrodriguez@ubuntu:~/hotelandia/uoc$ node add-resources.js
1. Received Registry:  org.hotelandia.zona.Zona
Added the Resources successfully!!!
martasebastianrodriguez@ubuntu:~/hotelandia/uoc$
```

Y se puede comprobar su inserción en el histórico de transacciones:



3. 7. 2. 4. CONSULTANDO LOS REGISTROS

Composer soporta dos tipos de consultas, el primer tipo son las named queries, consultas expuestas por Composer Rest Server como REST Api y el segundo tipo son las queries dinámicas, creadas en tiempo de ejecución.

La primera parte del código desarrollado invoca a una named query “allHotels” qué es accesible en el modelo de negocio de Hotelandia, primero se crea la conexión con nuestra business network para llamar a la función query AllHotels.

La segunda parte del código, ejecuta una consulta dinámica, que devuelve una instancia de zona con id específico

client-query.js

```
'use strict';


const bnUtil = require('./businessnetwork-connection-util');

// #1 Connect to the airlinev8
bnUtil.cardName='admin@uoc';
bnUtil.connect(main);

function main(error){
    // for clarity sake - ignored the error

    // #2 Execute the named query : AllHotels

    return bnUtil.connection.query('AllHotels').then((results)=>{

        console.log('Received hotel count:', results.length)

        var   statement = 'SELECT  org.hotelandia.zona.Zona WHERE (zonatId == _$id)';

        // #3 Build the query object
        return bnUtil.connection.buildQuery(statement);

    }).then((qry)=>{

        // #4 Execute the query
        return bnUtil.connection.query(qry,{id:'ZAR01'});
    }).then((result)=>{
        console.log('Received zona count:', result.length);
        if(result.length > 0) console.log(result[0].zonaId);
        bnUtil.connection.disconnect();
    }).catch((error)=>{
        console.log(error);
        bnUtil.connection.disconnect();
    });
}
```

Como resultado se obtiene:

```
martasebastianrodriguez@ubuntu:~/hotelandia/uoc$ node client-query.js
Received hotel count: 3
Received zona count: 1
ZAR
```

3. 7. 2. 5. TEST UNITARIOS

El marco de trabajo de Composer soporta tres ejecuciones de entornos en tiempo de ejecución. La primera es Fabric, el segundo es el playground y el tercero es el entorno de tiempo de ejecución integrado que es adecuado para los test unitarios.

El entorno de tiempo de ejecución integrado simula el tiempo de ejecución de la estructura. Para utilizar el tiempo de ejecución incorporado, se debe configurar un código junto con el código de test unitario.

unittest-harness.js

```
module.exports = {
    // Removed MemoryCardStore & Added NetworkCardStore 0.19.0
    // MemoryCardStore: require('composer-common').MemoryCardStore,
    NetworkCardStore: require('composer-common').NetworkCardStoreManager,
    IdCard: require('composer-common').IdCard,
    AdminConnection: require('composer-admin').AdminConnection,
    BusinessNetworkDefinition: require('composer-common').BusinessNetworkDefinition,
    BusinessNetworkConnection: require('composer-client').BusinessNetworkConnection,
    CertificateUtil: require('composer-common').CertificateUtil,

    info: 'UT-Harness-NOT-Initialized!!!',
    debug: false,

    // Objects created for testing
    adminConnection: {},
    businessNetworkDefinition: {},
    businessNetworkConnection: {},

    initialize: function (folder, callback) {
        // 1 Setup the PeerAdmin Card to be used by the admin connection
        idCard = this.setupPeerAdminCard();

        const cardStore = require('composer-common').NetworkCardStoreManager.getCardStore( { type: 'composer-wallet-inmemory' } );
        this.adminConnection = new this.AdminConnection({ cardStore: cardStore });
        this.businessNetworkConnection = new this.BusinessNetworkConnection({ cardStore: cardStore });//cardType);

        // 4. Import the PeerAdmin Card to the Memory card store
        const peerAdminCardName = "PeerAdmin"
        return this.adminConnection.importCard(peerAdminCardName, idCard).then(() => {

            this.log("PeerAdmin Card imported Successfully!!");

            // 5. Connect to the embedded runtime with PeerAdmin Card
            return this.adminConnection.connect(peerAdminCardName);

        }).then(() => {

            // Admin connection successfull
            this.log("Admin Connection Successful!!")

            // 6. Create the Business Network Def from directory
            return this.BusinessNetworkDefinition.fromDirectory(folder)
        }).then(definition => {
            this.businessNetworkDefinition = definition;
            this.info = definition.metadata.packageJson.name + '@' + definition.metadata.packageJson.version;


            // 7. Install the Composer runtime for the new business network
            return this.adminConnection.install(this.businessNetworkDefinition);//.getName());
        }).then(() => {
            this.log("Runtime Install Successful!!");

            const startOptions = {
                networkAdmins: [
                    {
                        userName: 'admin',
                        enrollmentSecret: 'adminpw'
                    }
                ]
            };

            // 8. Start runtime - will receive admin card on resolution
            return this.adminConnection.start(this.businessNetworkDefinition.getName(),this.businessNetworkDefinition.getVersion
            startOptions);
        }).then((networkAdminCardMap) => {
            this.log("Start Successful - network admin card received!!");
            this.networkAdminCardName = `admin@${this.businessNetworkDefinition.getName()}`;

            // 9. Import the network admin card
            return this.adminConnection.importCard(this.networkAdminCardName, networkAdminCardMap.get('admin'));
        }).then(() => {

            this.log("Imported the Network Admin Card!!! =" + this.networkAdminCardName);

            // 10. Connect the business network
            return this.businessNetworkConnection.connect(this.networkAdminCardName);

        }).then(() => {

            this.log("Business connection successful!!!");

            // 13. Invoke the callback function
            callback(this.adminConnection, this.businessNetworkConnection, this.businessNetworkDefinition);

        }).catch((error) => {
            this.log("Errored!!", error)
        });
    },


    // This prints message to console if the debug=true
    log: function (msg, error) {
        if (this.debug) {
            console.log('UTH: ', msg);
        }
        if (error) console.log('UT Harness Error: ', error);
    },

    disconnect: function(){
        this.businessNetworkConnection.disconnect();
        this.adminConnection.disconnect();
    },

    /**
     * Sets up the Card store and the Network idCard
     */
    setupPeerAdminCard: function () {

        const connectionProfile = {
            name: 'embedded',
            'x-type': 'embedded'
        };

        // 1.2 Metadata
        const metaData = {
            version: 1,
            userName: 'PeerAdmin',
            roles: ['PeerAdmin', 'ChannelAdmin']
        }


        // Generate certificates for use with the embedded connection
        const credentials = this.CertificateUtil.generate({ commonName: 'admin' });

        // 2. Create the IDCard
        idCard = new this.IdCard(metaData, connectionProfile);

        // 3. Set the credentials
        idCard.setCredentials(credentials)

        // 4. Create the admin object
        return idCard;
    }


}
```

Antes de que se ejecute el código de prueba, la aplicación de red se implementa en el tiempo de ejecución integrado y luego el código de test unitario se ejecuta contra la aplicación de red de negocios desplegada.

Primero se debe crear una tarjeta de administración de peers y esta tarjeta de administrador de peers tiene un perfil de conexión en el que el tipo está configurado como incrustado. El siguiente paso es importar esta tarjeta PeerAdmin en el almacén de tarjetas y usarlo para crear un objeto AdminConnection., que será utilizado para implementar los archivos de modelo para el proyecto en el tiempo de ejecución integrado. Una vez que se completa la implementación, se crea la tarjeta de administración de red y esa tarjeta de administración de red se importa al almacén de tarjetas. Después de eso, se crea el objeto business network connection y, con la tarjeta de administración de red, se crea la conexión entre el objeto de red de negocios y el tiempo de ejecución integrado.

Para simplificar, se he creado una plantilla de test unitario unittest-harness.

Primero se crea una función de inicialización se invoca en la plantilla de unit test con la carpeta modelo (que apunta a la ruta de la red de negocio), y una función de devolución de llamada que toma como argumentos la admin connection, la business network connection, y la versión. Cuando se invoca la función de devolución de llamada, simplemente imprimimos el nombre de la aplicación y la versión de la aplicación en la consola.

Luego se invoca la función getAllAssetRegistrier en la business network connection, se trata de una prueba de demostración, donde simplemente se imprimen el nombre de todos los registros.

La clase de test desarrollada es:

```
test-unittest-harness.js
1
2   const utHarness = require('./unittest-harness');
3
4   var modelFolder = '/home/martasebastianrodriguez/hotelandia/uoc';
5   //var modelFolder = '';
6
7
8   // Set this to false for suppressing the UTH messages
9   utHarness.debug=true;
10  utHarness.initialize(modelFolder, (adminCon, bnCon, definition)=>{
11
12      // Get the registry names
13          console.log("BNA =",definition.getName(),'@',definition.getVersion());
14
15          // Lets geth the registries
16          return bnCon.getAllAssetRegistries(false).then((registries)=>{
17              registries.forEach((registry)=>{
18                  console.log(registry.id)
19              });
20
21          });
22  });
```

Y la salida que se obtiene al ejecutar la clase en VisualCode es:

```
martasebastianrodriguez@ubuntu:~/hotelandia/uoc$ node test-unittest-harness.js
UTH:  PeerAdmin Card imported Successfully!!
UTH:  Admin Connection Successful!!
UTH:  Runtime Install Successful!!
UTH:  Start Successful - network admin card received!!
UTH:  Imported the Network Admin Card!!! =admin@uoc
UTH:  Business connection successful!!!
BNA = uoc @ 0.0.4
org.hotelandia.hotel.Hotel
org.hotelandia.zona.Zona
```

3. 7. 3. ESTRATEGIA DE AUTENTICACIÓN EN REST SERVER

El REST server puede ser asegurado mediante dos maneras recomendadas como buenas prácticas, una consiste en utilizar HTTPS en lugar de HTTP. La segunda consiste en habilitar un sistema de autenticación sobre el REST server, a través de Passport o pasaporte.

Passport es un framework que soporta múltiples estrategias de autenticación. Una vez que el servidor REST está habilitado para la autenticación cada vez que un usuario se conecta a una aplicación para el servidor REST, debe proporcionar sus credenciales.

Hay más de 300 módulos de autenticación disponibles para el marco del pasaporte. En el contexto empresarial, los esquemas de autenticación mas comúnmente utilizados son OAUTH2.0, SAML y la autenticación basada en LDAP.

Actualmente OAUTH2.0 es la estrategia de autenticación más utilizada para la seguridad de la API REST. OAUTH2.0 se puede utilizar con un proveedor de identidad empresarial para autenticar a sus usuarios.

El flujo OAUTH2.0. funciona de la siguiente manera, el usuario se conectará a la aplicación que, a su vez, invocará una API en el servidor REST y la primera llamada que se realize sin autenticación llevará a un "401:Error no autorizado". En ese momento, la aplicación redirigirá al usuario a una página de inicio de sesión dónde proporcionará sus credenciales.

La página de inicio de sesión realizará una llamada al proveedor de identidad OAUTH2.0 para validar las credenciales. Después, el proveedor de identidad emite un token de acceso y, a través de una url de devolución de llamada, enviará el token de acceso a la aplicación. Toda la invocación de la API desde la aplicación incluirá el token de acceso que recibió del proveedor de identidad. El servidor REST al recibir una llamada validará el token de acceso con el tiempo de ejecución del pasaporte y, si se determina que el token de acceso es válido, se otorgará el acceso al tiempo de ejecución de la estructura.

3. 7. 3. 1. OAUTH MEDIANTE GITHUB

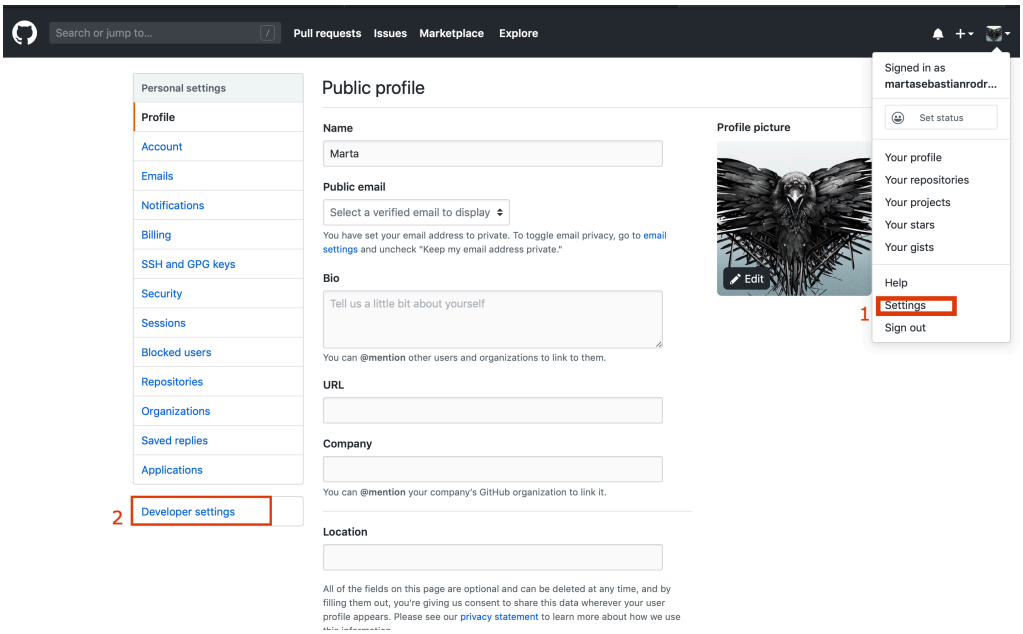
Añadiremos seguridad al REST Server tal y como hemos visto en el apartado anterior mediante el proveedor de identidad GITHUB.

En el contexto de GitHub, la pantalla de inicio de sesión proviene del sitio web de GitHub y el proveedor de identidad es GitHub. Se realiza en cuatro pasos:

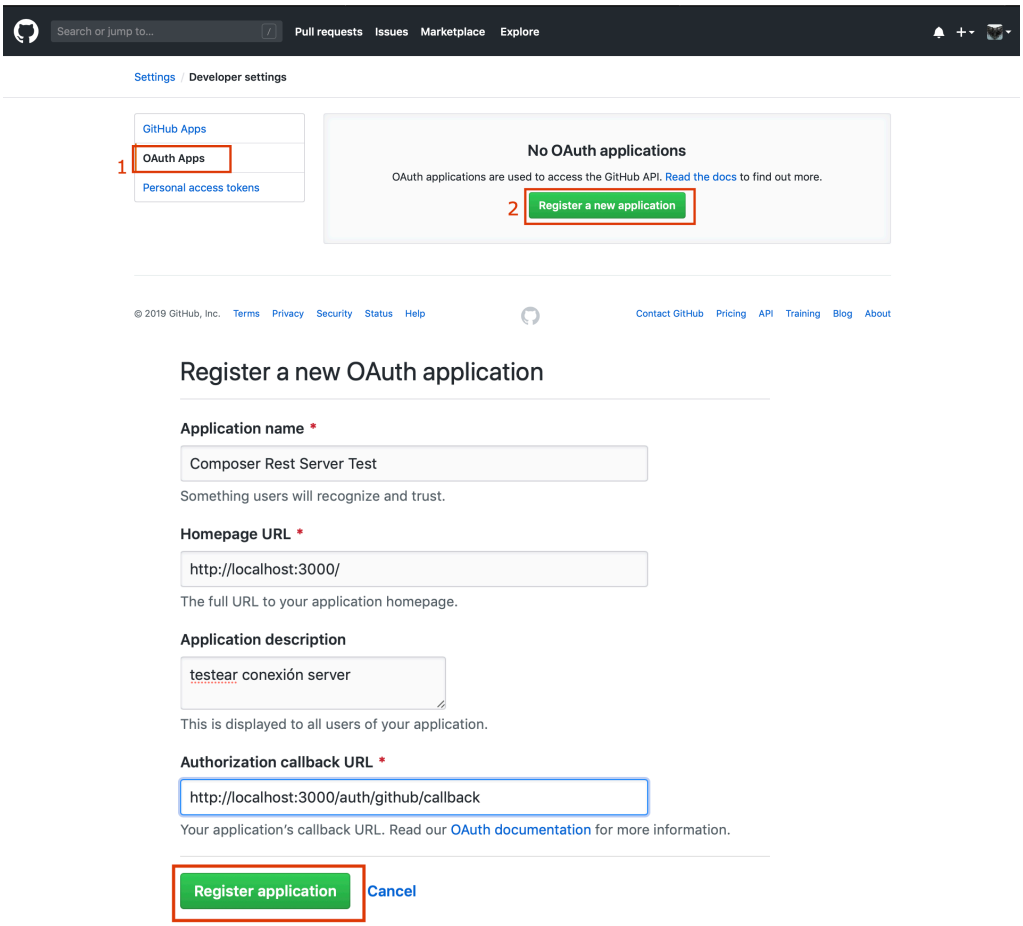
Primer paso.- se instala el paquete de autenticación para Github, mediante el comando: npm install -g passport-github.

```
martasebastianrodriguez@ubuntu:~/hotelandia/uoc$ npm install -g passport-g
ithub
+ passport-github@1.1.0
added 7 packages from 3 contributors in 9.702s
```

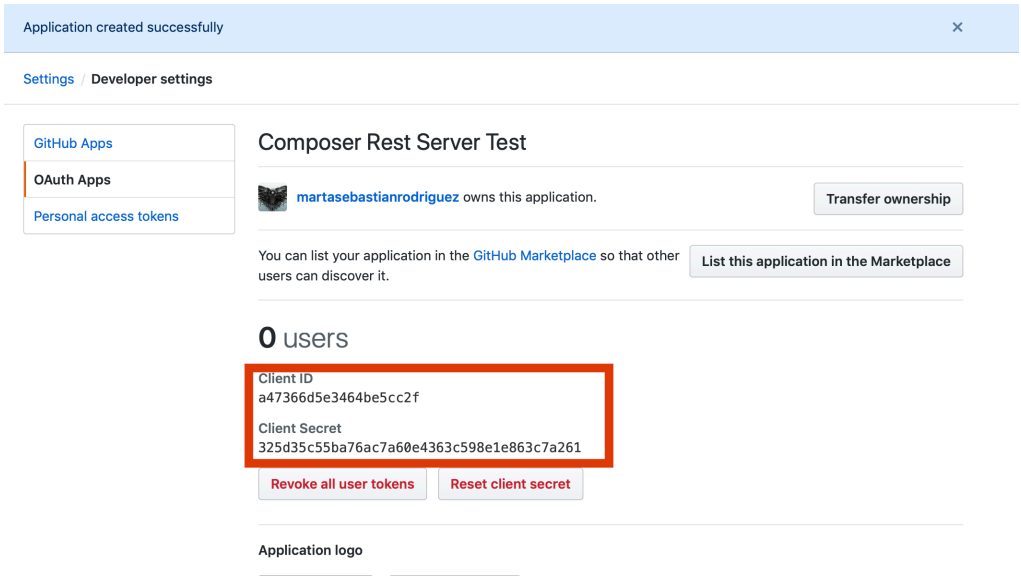
Segundo paso.- hay que registrarse para OAUTH2.0 en GitHub.com. Para esto debemos crear una cuenta en Github, clickamos en nuestra foto de perfil, en settings y seleccionamos developer settings.



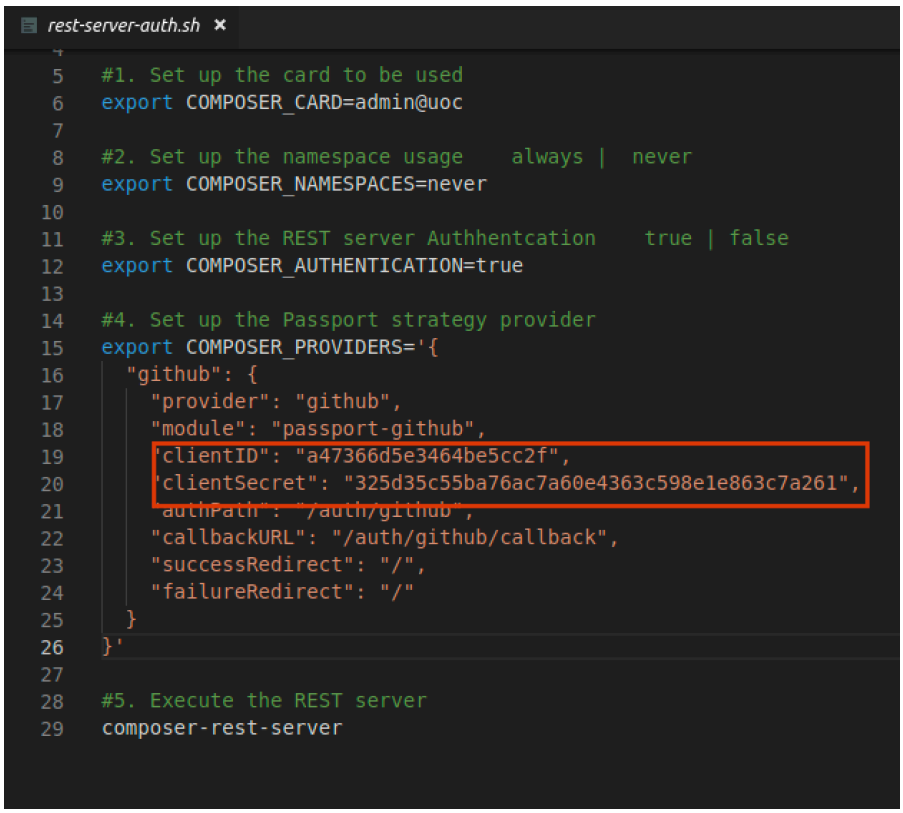
Después, registramos una nueva aplicación OAUTH, con los siguientes datos:



Obtenemos así el client Id y el Client Secret, que se utilizará para obtener la conexión:



Si se recuerda cómo se generaba el composer rest server, debíamos introducir una serie de variables de entorno. Se ha desarrollado un shell-script que introducirá y lanzará el composer rest server con las variables de entorno que necesitamos para la autenticación.



Se han introducido los datos de client ID y client Secret obtenidos anteriormente en Github. Cuando lo ejecutamos, nos abre la dirección de rest server, obtenemos el error 401, por no tener el token de acceso para la conexión.

```
martasebastianrodriguez@ubuntu:~$ ./rest-server-auth.sh
Discovering types from business network definition ...
Discovering the Returning Transactions..
Discovered types from business network definition
Generating schemas for all types in business network definition ...
Registering named query: AllHotels
Registering named query: HotelByNumber
Registering named query: HotelsCode
Registering named query: HotelsCodeOrdered
Registering named query: AllZonas
Generated schemas for all types in business network definition
Adding schemas for all types to Loopback ...
Swagger: se pasa por alto el tipo desconocido "Zona".
Swagger: se pasa por alto el tipo desconocido "Address".
Swagger: se pasa por alto el tipo desconocido "Zona".
Swagger: se pasa por alto el tipo desconocido "Zona".
Swagger: se pasa por alto el tipo desconocido "Zona".
Swagger: se pasa por alto el tipo desconocido "Zona".
Swagger: se pasa por alto el tipo desconocido "Zona".
Swagger: se pasa por alto el tipo desconocido "Zona".
Swagger: se pasa por alto el tipo desconocido "Zona".
Swagger: se pasa por alto el tipo desconocido "Zona".
Added schemas for all types to Loopback
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

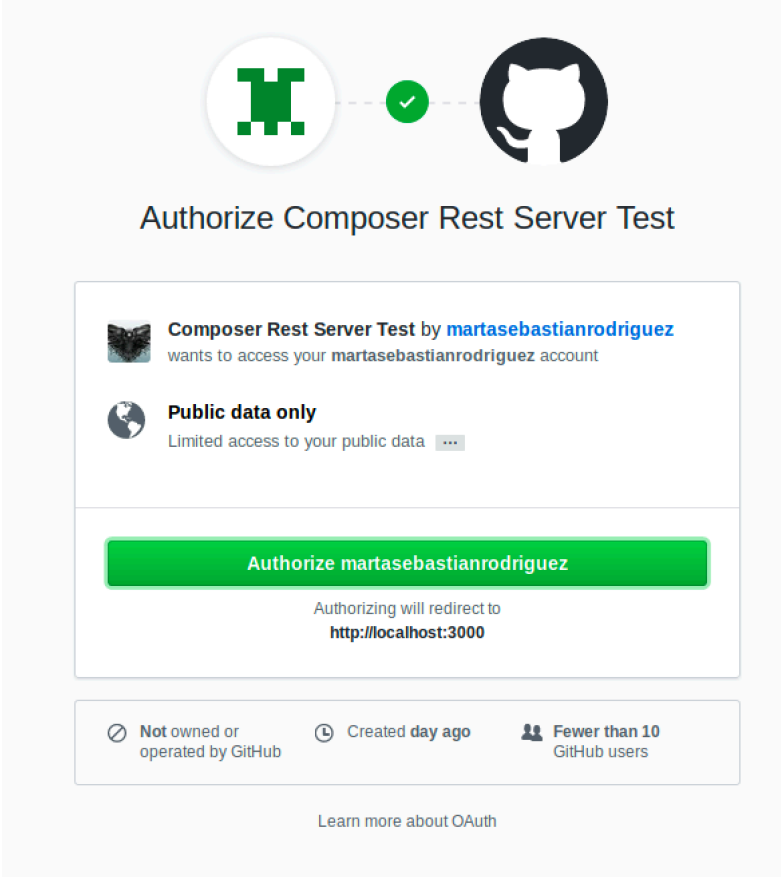
Response Body


```
{
  "error": {
    "statusCode": 401,
    "name": "Error",
    "message": "Authorization Required",
    "code": "AUTHORIZATION_REQUIRED",
    "stack": "Error: Authorization Required\n
  }
}
```

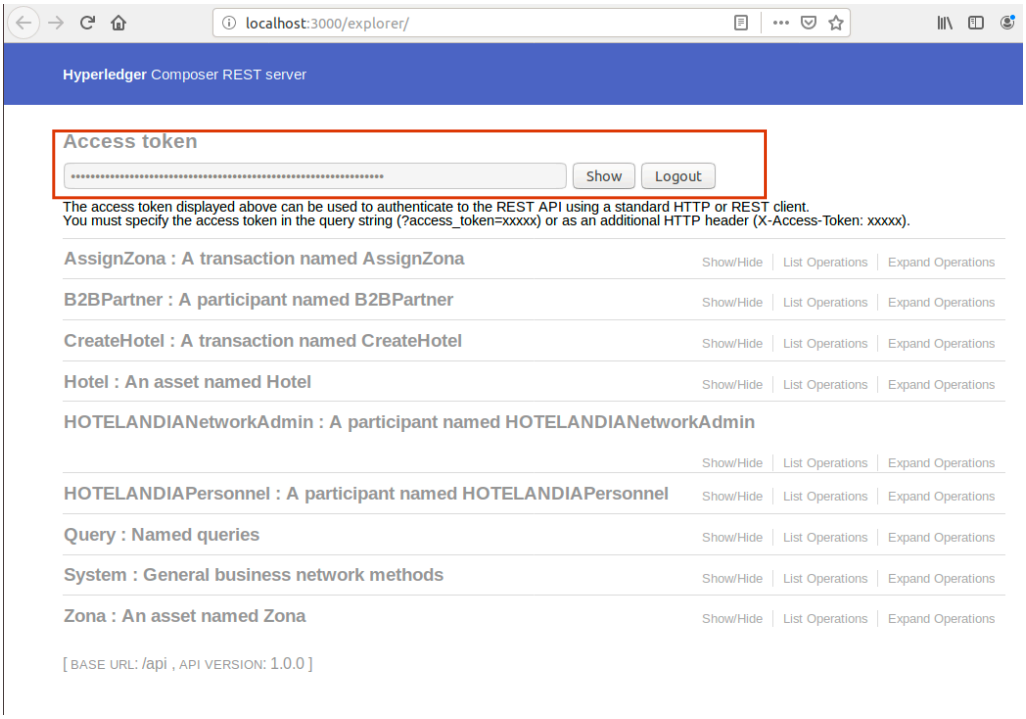

Response Code

401

Para obtener este token, debemos cambiar la url de composer de http://localhost:3000/Explorer a http://localhost:3000/auth/github, entonces se redirigirá a una página donde se piden las credenciales de GitHub, y si se autoriza al usuario:



Y al redirigirnos de nuevo, obtenemos el token de acceso.



Desarrollo ahora un shell script que nos pedirá el token de acceso cuando se ejecute el script anterior, y obtenga las named queries:

test-rest-server-auth.sh

```
#!/bin/bash

if [ -z "$1" ]; then
    echo 'Usage:  rest-server-auth.sh   <Access Key>'
    exit 1
fi

echo   -e '\n\n'
echo   'Executing using Token in Query:'
echo   ''
curl -X GET --header 'Accept: application/json' 'http://localhost:3000/api/Hotel'?access_token=$1
echo   -e '\n\n'
echo   'Executing using Token in Header:'
echo   -e '\n\n'
curl -X GET --header 'Accept: application/json' --header "x-access-token: $1" 'http://localhost:3000/api/Hotel'
echo   -e '\n\n'
```

Vemos el resultado:

```
martasebastianrodriguez@ubuntu:~$ ./test-rest-server-auth.sh
Usage:  rest-server-auth.sh   <Access Key>
martasebastianrodriguez@ubuntu:~$ ./test-rest-server-auth.sh NsDCDluriF6HRT1HH84CF1NokyApIOfeAVRLpARlvDWQeYarbMrirQWhEkiPvx9W



Executing using Token in Query:

[{"$class":"org.hotelandia.hotel.Hotel","hotelId":"ZAR03-01-10-19","hotelNumber":"ZAR03","address":{"$class":"org.hotelandia.hotel.Address","code":"ZAR","street":"calle tercera","schedule":"2019-10-01T18:18:12.432Z"}}]


Executing using Token in Header:



[{"$class":"org.hotelandia.hotel.Hotel","hotelId":"ZAR03-01-10-19","hotelNumber":"ZAR03","address":{"$class":"org.hotelandia.hotel.Address","code":"ZAR","street":"calle tercera","schedule":"2019-10-01T18:18:12.432Z"}}]


martasebastianrodriguez@ubuntu:~$
```

4. Conclusiones

En este proyecto ha habido un intenso trabajo de investigación y de desarrollo, lo cual ha sido un gran reto personal. Se ha conseguido desarrollar y poner en funcionamiento un permissioned blockchain con Hyperledger Fabric e Hyperledger Composer, desplegar una red de negocio y de todo lo que conlleva este sistema.

Han existido problemas relacionados con la tecnología, es difícil la instalación del entorno de Hyperledger fabric y composer en sistemas operativos diferentes a Linux, tales como windows y MacOS. Se ha tenido que realizar utilizando una máquina virtual con varios desktops de linux para simular servidores y clientes. Este problema conllevó una gran perdida de tiempo que luego ha sido difícil recuperar, pero el resultado final es el esperado y planificado.

La planificación inicial se desarrolló sin problema hasta llegar a la parte de instalación de requisitos, y software, donde como se ha comentado anteriormente surgieron graves problemas, pero en la última fase del proyecto se recuperó el tiempo perdido, ya que el desarrollo no ha sufrido más retrasos, y realmente el resultado final se acerca bastante a lo que tenía planificado en un principio.

Quizás en un desarrollo futuro se podría añadir el estudio y despliegue de transacciones multiusuario y de multiples identidades que en este proyecto se han pasado muy por encima.

5. Bibliografía

"Hands - On Blockchain with Hyperledger".

Autores: Mitin Gaur, Luc Desrosiers, Petr Novotny, Venkatraman Ramakrishna, Anthocy O'Dowd and Salman A. Based.

ED.: Packt

Ciudad: Birmingham-Mumbai.

Junio 2018.

6. Webgrafía

The LINUX Foundation:

https://www.linuxfoundation.org/

visitada desde el 15/03/2019

Hyperledger:

https://www.hyperledger.org/

visitada desde el 15/03/2019

The rise of private permissionless blockchains:

https://medium.com/ltonetwork/the-rise-of-private-permissionless-blockchains-part-1-4c39bea2e2be

Instalación de pre-requisitos y entorno de desarrollo Hyperledger:

https://hyperledger.github.io/composer/v0.19/installing/installing-prereqs.html

visitada desde el 15/04/2019 hasta el 2/05/2019

Descarga NodeJS

https://nodejs.org/en/

visitada desde el 16/04/2019

Descarga Python 2.7

https://www.python.org/download/releases/2.7/

visitada desde el 16/04/2019

Descarga NVM

https://github.com/nvm-sh/nvm/blob/master/README.md

visitada desde el 16/04/2019

Descarga Docker

https://www.docker.com/

Visitada el 16/04/2019

How to build a blockchain network using Hyperledger Fabric and Composer:

https://www.freecodecamp.org/news/how-to-build-a-blockchain-network-using-hyperledger-fabric-and-composer-e06644ff801d/

visitada desde el 2/05/2019 hasta el 19/05/2019

Hyperledger Composer wiki

https://github.com/hyperledger/composer/wiki

visitada desde el 2/05/2019 hasta el 19/05/2019

Wikipedia:

- https://es.wikipedia.org/wiki/Hyperledger
- https://es.wikipedia.org/wiki/Cadena_de_bloques
- https://es.wikipedia.org/wiki/Bitcoin
- https://es.wikipedia.org/wiki/Ethereum

Hyperledger Composer playground on-line:

https://composer-playground.mybluemix.net/login

visitada desde el 20/05/2019 hasta el 29/05/2019