文档编号： 第 1版

分册名称：设计报告 第 1册/共1 册

**大宗商品交易市场监管与服务技术研究开发及应用示范**

**课题2——大宗商品电子商务交易平台的主体智能查验、过程监测与可信交易技术**

BaaS区块链即服务设计报告

课题编号：2019YFB1405002

2020-11

目 录

[1.项目简介 4](#_Toc55998573)

[2.研究内容 6](#_Toc55998574)

[3.系统总体设计 8](#_Toc55998575)

[3.1整体功能设计 8](#_Toc55998576)

[3.2区块链节点编排与管理 11](#_Toc55998577)

[3.2.1 私有化部署区块链节点 11](#_Toc55998578)

[3.2.2 云上部署区块链服务 15](#_Toc55998579)

[3.3区块链可信存证服务 16](#_Toc55998580)

[4. 区块链节点编排系统设计 16](#_Toc55998581)

[4.1用户管理 18](#_Toc55998582)

[4.2用户权限管理 18](#_Toc55998583)

[4.3区块链节点编排 18](#_Toc55998584)

[4.3.1 证书 – 配置文件的初始化工作 18](#_Toc55998585)

[4.3.2 Kubernetes通过HTTP请求进行容器编排 18](#_Toc55998586)

[4.3.2.1 kubernetes容器编排组件-Master Node 19](#_Toc55998587)

[4.3.2.2 Kubernetes容器编排组件-Worker Node 20](#_Toc55998588)

[4.3.3区块链节点间的通信 21](#_Toc55998589)

[4.3.4区块链编排组件Master容灾机制 21](#_Toc55998590)

[4.4容器节点的监控 22](#_Toc55998591)

[5.区块链可信存证系统设计 22](#_Toc55998592)

[5.1区块链用户权限管理和交易合法性验证 23](#_Toc55998593)

[5.1.1组织内证书的管理 25](#_Toc55998594)

[5.1.1.1 Fabric CA 25](#_Toc55998595)

[5.1.1.2 用户Fabric CA注册证书 27](#_Toc55998596)

[5.1.1.3 区块链节点通过Fabric CA注册证书 27](#_Toc55998597)

[5.1.2组织内的权限管理 28](#_Toc55998598)

[5.1.2.1 MSP – 证明用户参与区块链中的合法身份 28](#_Toc55998599)

[5.1.2.2 ACLs 权限访问控制 29](#_Toc55998600)

[5.1.2.3 背书策略 30](#_Toc55998601)

[5.2区块链可信存证通道的管理 30](#_Toc55998602)

[5.2.1 配置通道的创世区块 31](#_Toc55998603)

[5.2.2 创建通道 32](#_Toc55998604)

[5.2.3 Peer节点加入通道 32](#_Toc55998605)

[5.2.3 设定锚节点 32](#_Toc55998606)

[5.3区块链服务智能合约的管理 33](#_Toc55998607)

[5.3.1 链码的生命周期 34](#_Toc55998608)

[5.3.1.1 上传链码 35](#_Toc55998609)

[5.3.1.2 打包链码 35](#_Toc55998610)

[5.3.1.3 安装链码 36](#_Toc55998611)

[5.3.1.4 审批链码 36](#_Toc55998612)

[5.3.1.5 提交链码 37](#_Toc55998613)

[5.3.2 外部链码的服务调用 37](#_Toc55998614)

[5.3.2.1 链码的外部构建和启动 38](#_Toc55998615)

[5.3.2.2 外部构建和运行的配置 38](#_Toc55998616)

[5.3.2.3 外部构建运行 39](#_Toc55998617)

[5.4区块链服务调用 40](#_Toc55998618)

[5.4.1区块链可信存证溯源和校验 41](#_Toc55998619)

[5.4.2区块链可信存证入链 41](#_Toc55998620)

[5.4.3 区块链私有数据存证 43](#_Toc55998621)

[5.4.4 区块链请求的负载均衡 43](#_Toc55998622)

[5.5区块链网络的监控 43](#_Toc55998623)

# 1.项目简介

**大宗商品项目遇到的问题？**

1. **最主要的问题是可信存证问题** --- 溯源&&验证。
2. **可信存证区块链服务轻节点**：用户作为区块链的参与者，没有足够的能力部署全节点，但仍需向用户提供区块链可信存证的服务，因此为用户编排和调度轻节点，使用户可以真正的参与到区块链网络中。
3. **可信存证服务节点的部署与调度：**区块链参与者在区块链中的节点部署需要有两种可以选择的方式，即私有化部署和云上部署。大宗商品综合服务平台、清算所、交易平台等需要使用区块链作为可信存证的平台，因此需要为他们提供全节点式的区块链服务节点。
4. **可信存证数据源**：大宗商品综合服务平台、清算所、交易平台等需要使用区块链作为可信存证的平台，需要为他们提供区块链的可信存证和溯源服务。

**可以为大宗商品项目节点部署遇到的问题提供解决方案？**

1. **统筹兼顾**：可以做到管理或编排区块链节点，既可以管理云上orderer、peer等区块链节点，也可以编排用户本地的区块链节点。
2. **全节点部署方案**：全节点即可部署在本地，也可以部署在云平台上。
   * 1. 交易平台、清算所、大宗商品综合服务平台，考虑到安全或者权限方面的需求，可以自己根据自己的需求在本地私有化部署全节点，并参与到区块链网络中。
     2. 交易平台、清算所、大宗商品综合服务平台同时可以选择将自己的区块链节点放到BaaS平台上，直接向他们提供区块链服务。

3. **轻节点部署方案**：轻节点部署在BaaS平台上。

* + 1. 考虑到用户的计算能力和存储能力，用户可以选择直接将区块链节点放到BaaS平台上，透明的、可信的向用户提供区块链服务。

**在大宗商品之外，我们的BaaS管理平台还能做到什么？**

1. **可以提供一套更为通用的区块链服务的解决方案。**为更多的用户提供一站式的服务，实现区块链网络的快速部署和维护，智能合约和业务模式的快速落地。
2. **可以提供场景化的区块链服务解决方案。**区块链服务平台可以在业务领域，如金融领域和食品安全领域的区块链的可信存证与溯源等，为区块链平台的使用者提供可信的，可以溯源的区块链解决方案。

# 2.研究内容

**BaaS部署的技术难点：**

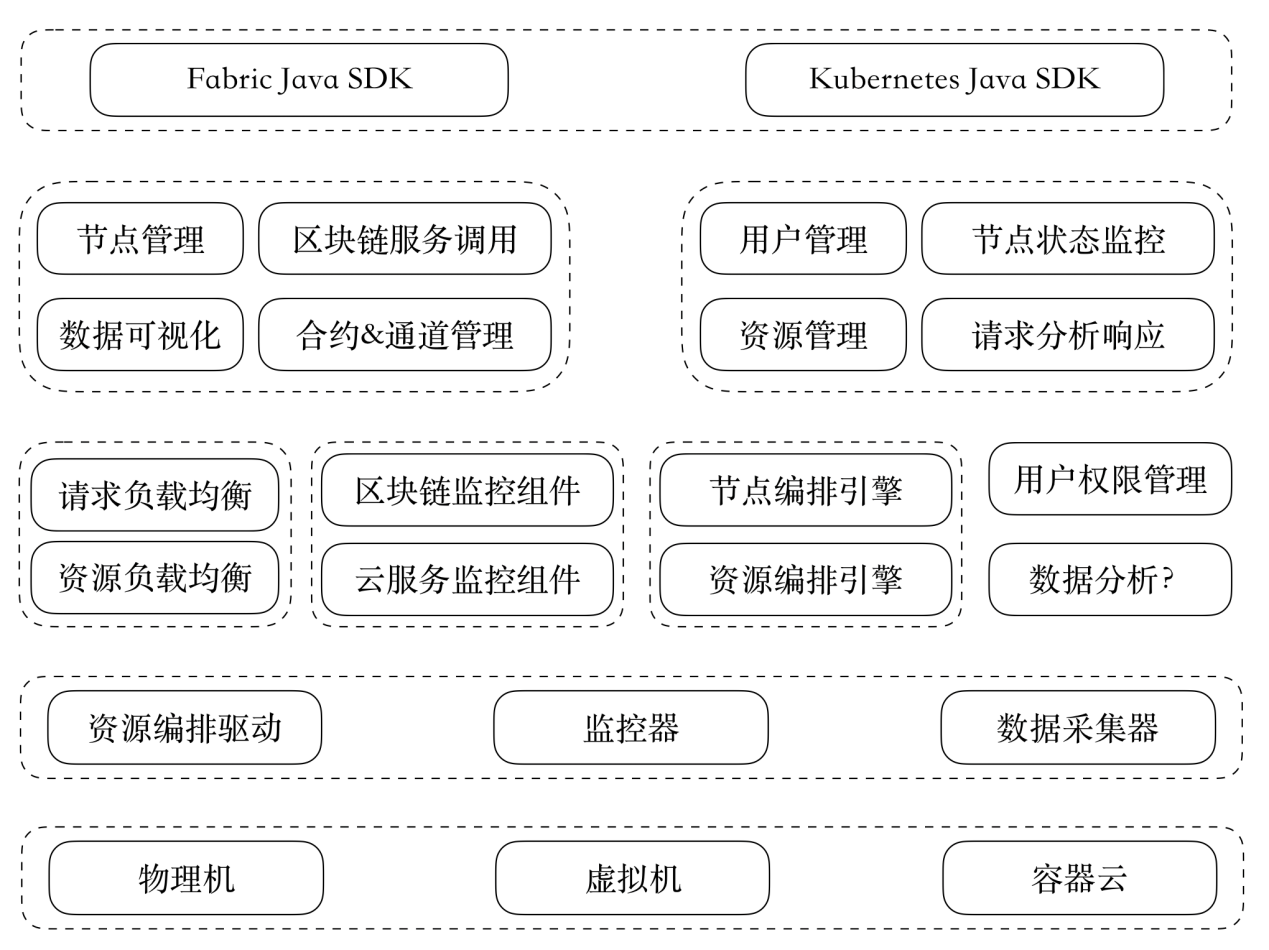
1. 如何和大宗的几个平台结合起来？BaaS如何容纳和管理交易平台和服务平台的节点？
   1. **部署：**我们的BaaS平台的解决方案非常灵活，即支持云上部署，也支持私有化部署，可以根据各个交易平台的具体情况，和使用区块链的场景，进行针对行的提供服务。
   2. **私有化部署：**每个交易平台作为一个Org参与到区块链网络中，会单独为每个公司部署区块链全节点 / 轻节点。
   3. **云上部署：**特别的，每个交易平台或者公司即可将节点部署在本地，也可将节点放到BaaS平台上集中的管理和提供区块链服务。
   4. **扩展：**BaaS平台的扩展是动态的，灵活的根据用户的使用场景，动态的扩展联盟的组织，组织中的节点，以及节点上的各种配置。
   5. **联盟治理：**可以提供一整套的联盟治理的解决方案，包括且不限于联盟创建，联盟邀请，联盟准入，联盟投票等一系列的联盟的生命周期的管理。
   6. **可信服务：**各个交易平台的用户的节点可以放在云上集中的进行区块链轻节点的编排，向用户提供可信的区块链账本溯源的服务。
2. BaaS平台的设计方案如何与区块链的场景结合起来：
   1. 面向各个交易平台 / 清算所 / 综合服务平台：为他们提供部署区块链全节点的服务，可以自由的进行可信存证。
   2. 面向各个交易平台 / 清算所 / 综合服务平台的用户，为他们提供区块链可信存证溯源服务。

**目前的区块链部署方式的缺点是什么？也是我们的优势**

1. 区块链服务过时：大部分的BaaS平台使用的Fabric版本仍停留在1.x版本
   1. 大宗商品BaaS平台设计方案松耦合、模块化高、便于系统升级。
   2. 大宗商品的区块链服务模块提供可插拔的REST API接口式服务，系统耦合度低，可以即使的维护升级。
   3. 大宗商品的区块链节点镜像服务可插拔，只需升级对应的模块组件即可完成系统的升级需求
2. 可扩展性：
   1. 物理资源编排：是否具有大规模场景下的应用部署和管理的能力，是否可以跨区域的对云上硬件资源进行扩展。
3. 资源或服务的调度：
   1. 负载均衡：对于非均匀的资源请求可以实现智能调度，合理的分配系统的资源。
4. 缺乏监控：并没有系统化的区块链节点监控方案
5. 性能弱，对交易的入链/查询的实时性的要求：
   1. 首先没有对fabric源码的改良。
   2. 不可以指定节点进行查询。
   3. 对私有数据的支持性较差，隐私保护不够。
6. 安全性：
   1. 是否可以避免外部攻击和内部攻击？以及是否可以保障系统的容错性？
   2. 是否支持国密算法？
   3. 是否提供隐私数据的加密和授权设施？
7. 中心化身份证书管理机制：统一由一个SDK管理所有的用户的密钥。
   1. 大宗商品平台提供了用户身份与公钥、地址间的映射关系的保存设施，同时可以由相应的证书颁发机构生成证书，满足法律效力的要求。
8. 权限控制、安全管理、隐私保护力度不够：
   1. 我们的BaaS平台针对分布式账本防篡改和安全与隐私进行了加强，可以针对用户的特定场景需求，针对性的增强用户的权限管理、安全控制、隐私保护和监督管理等功能。
   2. 可以通过背书策略、ACLs权限控制、MSP策略等一系列的策略进行权限控制和安全控制。
9. 链码的管理难度高，维护复杂
   1. 停留在1.4的生命周期中，链码更新或升级维护起来非常的负载。
   2. 不可以支持外部链码。

# 3.系统总体设计

## 3.1整体功能设计



BaaS系统功能性模块组成架构

BaaS平台的系统架构主要分为三部分：

1. 应用层 / 业务层：向应用开发者 和 平台管理员提供不同的操作管理能力。
2. 核心层：完成系统的 资源编排、系统监控、数据分析 和 权限管理 区块链网络服务 等重要功能。
3. 底层驱动：访问 和 管理 多种物理资源

a) 对用户提供

1. 权限/证书管理：不同的用户具备不同的 数据/请求 权限（权限管理属于fabric还是kubernetes部分？权限管理管理的是什么权限？管理区块链网络的权限），通过带认证的接口，用户根据自己的权限可以访问区块链网络，安装智能合约并进行调用。

2. 节点管理：管理区块链网络节点，一键式快速创建和部署生产级区块链环境，简化区块链的部署流程和应用配置。

3. 通道管理：可以创建&删除通道，管理通道的生命周期（事实上是区块链的生命周期的管理工作）

4. 智能合约管理：管理智能合约的生命周期，完成一键式的智能合约的部署与调用。

5. 区块链服务：fabric Java sdk 的安装与调用，使用fabric Java sdk提供区块链服务（fabric sdk应该可以调用kubernetes sdk的把？Fabric Java SDK 和 Kubernetes Java SDK 是同级关系嘛？这两个SDK部署在哪里？也是部署在容器中嘛？）

6. 监控服务：监控区块链网络的具体工作情况

1. 区块链监控服务：可以通过 Fabric explorer 等类似的区块链监控工具，也可以通过fabric sdk完成实现具体的功能。

2. 区块链的使用情况包括：区块数目情况，交易数目情况

b) 对管理员提供

1. 用户管理：（这里的用户管理管理的是什么用户？这里的用户实际上指的就是可以创建节点的某一组织内的用户）

2. 资源管理：容器节点的编排功能（计算资源、存储资源、网络资源这些是否支持动态扩容？？），使用kubernetes Java sdk 提供容器编排的服务

3. 节点监控服务：监控区块链网络中各个节点是否在正常的工作。

1. 信息采集组件 - 部署：通过kubernetes部署在sdk业务层、oderer共识节点、peer节点账本存储层。

2. 信息采集组件 - 采集：可以完成采集节点的系统信息（cpu memory network disk）、节点的使用状态（节点访问量、访问耗时 和 节点的健康状态）、业务使用情况（业务访问量、成功率 和 耗时分布）。

3) 平台提供

1. 高性能：区块链节点具备处理高并发的交易的能力（需要对fabric源码进行修改工作，将源码编译成镜像，把镜像交由kubernetes进行管理工作）

2. 可扩展硬件资源：可以跨区域的扩展BaaS平台的硬件资源（Kubernetes可以扩展硬件资源吗？）

3. 资源/请求调度：调度不均匀的用户请求/资源请求（这个资源的请求调度应该通过什么模块完成？）

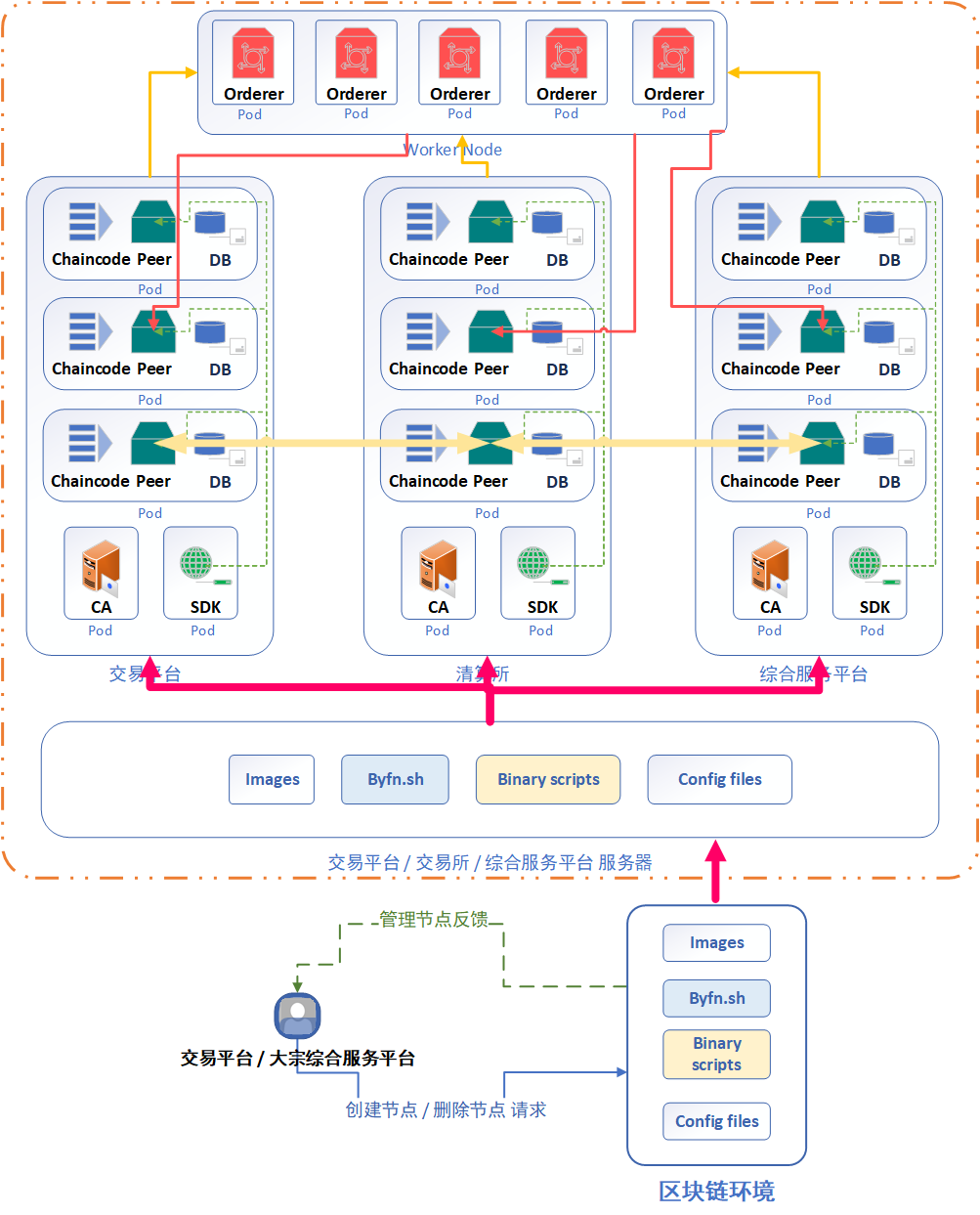
4. 安全保障：

1. 节点工作的安全性

2. 节点中用户数据的安全性

## 3.2区块链节点编排与管理

### 3.2.1 私有化部署区块链节点



Fabric 区块链服务本地部署

将区块链部署在各个交易平台和清算所的服务器本地。将区块链的配置文件、区块链节点镜像、区块链服务启动脚本提前部署在交易平台、清算所、综合服务平台，各个区块链的参与者可以使用区块链服务启动脚本启动区块链服务，并参加到区块链网络中。

区块链服务启动步骤：

1. 生成区块链节点的证书。

|  |
| --- |
| cryptogen generate --config=./crypto-config.yaml |

1. 生成公钥和私钥。

|  |
| --- |
| cp docker-compose-e2e-template.yaml docker-compose-e2e.yaml sed $OPTS "s/CA1\_PRIVATE\_KEY/${PRIV\_KEY}/g" docker-compose-e2e.yaml sed $OPTS "s/CA2\_PRIVATE\_KEY/${PRIV\_KEY}/g" docker-compose-e2e.yaml |

1. 生成通道的创世区块以及区块链网络的配置区块。

|  |
| --- |
| configtxgen -profile TwoOrgsOrdererGenesis -channelID $SYS\_CHANNEL -outputBlock ./channel-artifacts/genesis.block # 生成 channel.tx configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID $CHANNEL\_NAME # 生成 Org1MSPanchors.tx configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID $CHANNEL\_NAME -asOrg Org1MSP # 生成 Org2MSPanchors.tx configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID $CHANNEL\_NAME -asOrg Org2MSP |

1. 初始化容器。

|  |
| --- |
| IMAGE\_TAG=$IMAGETAG docker-compose ${COMPOSE\_FILES} up -d 2>&1 |

1. 初始化区块链网络。
2. 加入区块链网络channel

|  |
| --- |
| docker exec cli peer channel create -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/channel.tx --tls $CORE\_PEER\_TLS\_ENABLED --cafile $ORDERER\_CA |

1. 更新anchor peer

|  |
| --- |
| docker exec cli peer channel update -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/${CORE\_PEER\_LOCALMSPID}anchors.tx --tls $CORE\_PEER\_TLS\_ENABLED --cafile $ORDERER\_CA |

1. 安装链码。

|  |
| --- |
| docker exec cli peer chaincode install -n mycc -v ${VERSION} -l ${LANGUAGE} -p ${CC\_SRC\_PATH} |

1. 实例化链码。

|  |
| --- |
| docker exec cli peer chaincode instantiate -o orderer.example.com:7050 --tls $CORE\_PEER\_TLS\_ENABLED --cafile $ORDERER\_CA -C $CHANNEL\_NAME -n mycc -l ${LANGUAGE} -v 1.0 -c '{"Args":["init","a","100","b","200"]}' -P "AND ('Org1MSP.peer','Org2MSP.peer')" |

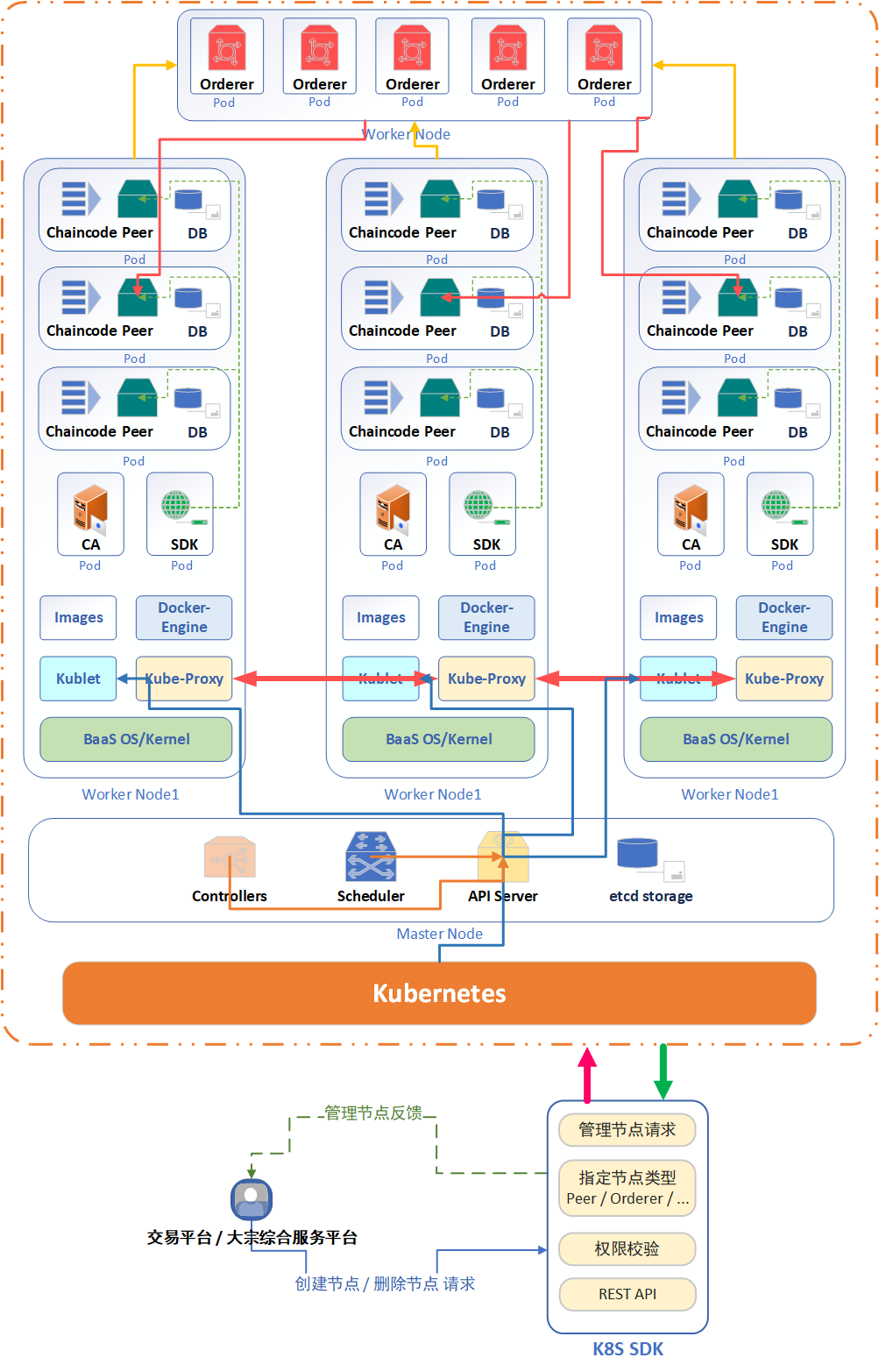
1. 查询链码。

|  |
| --- |
| docker exec cli peer chaincode query -C $CHANNEL\_NAME -n mycc -c '{"Args":["query","a"]}' |

1. 调用链码。

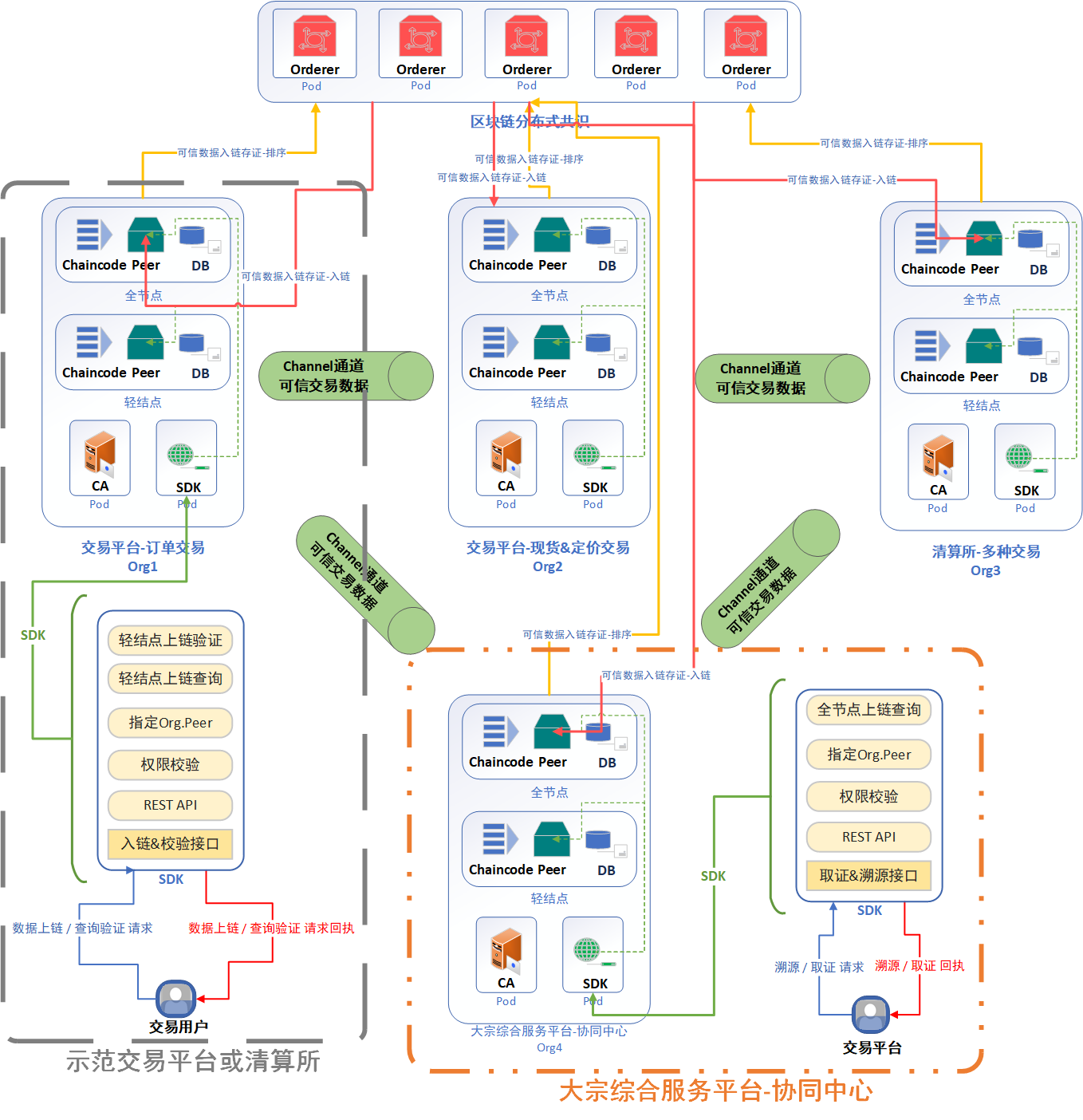
|  |
| --- |
| docker exec cli peer chaincode invoke -o orderer.example.com:7050 --tls $CORE\_PEER\_TLS\_ENABLED --cafile $ORDERER\_CA -C $CHANNEL\_NAME -n mycc $PEER\_CONN\_PARMS -c '{"Args":["invoke","a","b","10"]}' |

### 3.2.2 云上部署区块链服务

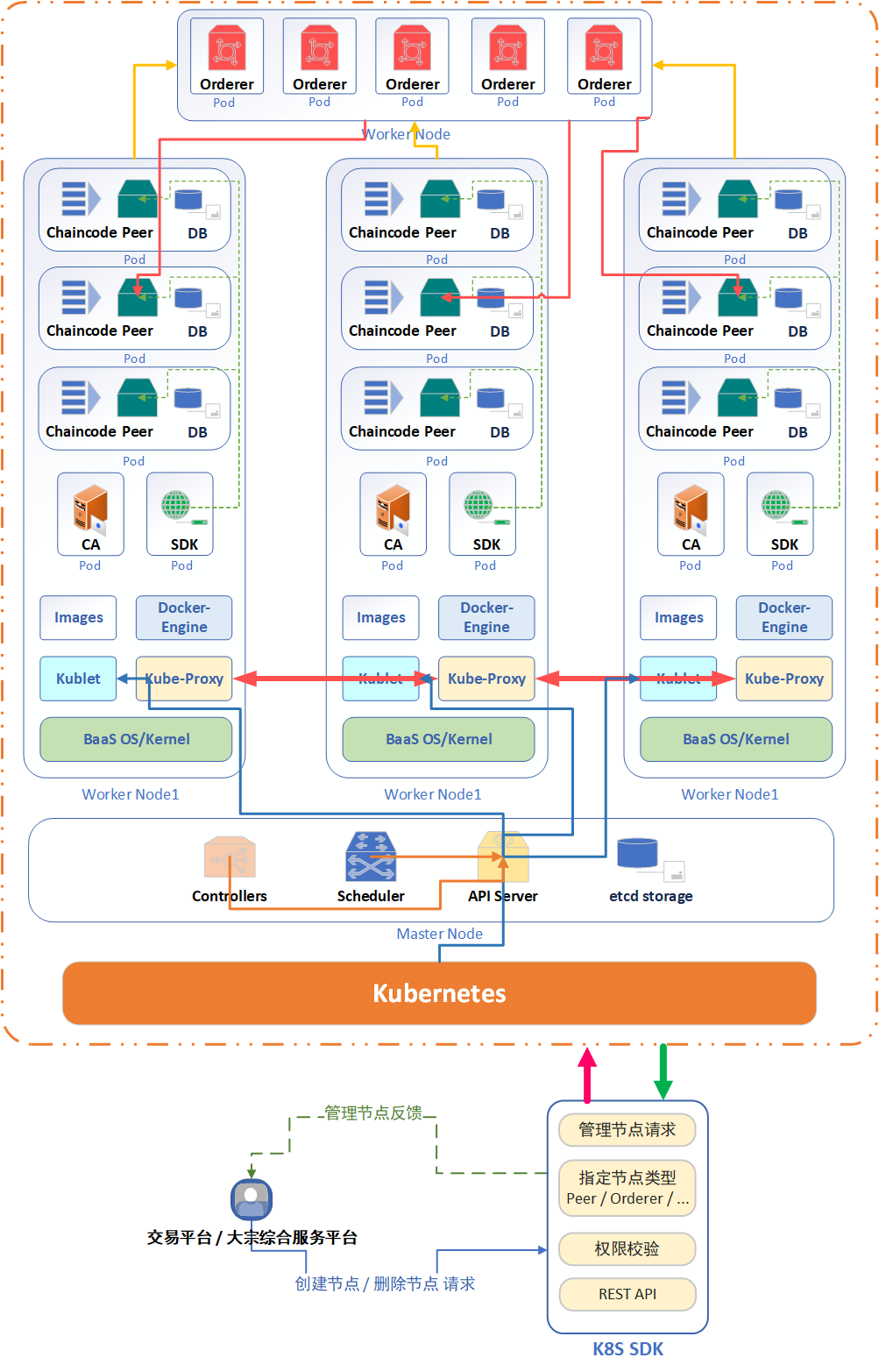


BaaS云平台区块链服务调度

## 3.3区块链可信存证服务



# 4. 区块链节点编排系统设计



区块链云平台系统架构设计

各个小的模块的具体设计方案：

## 4.1用户管理

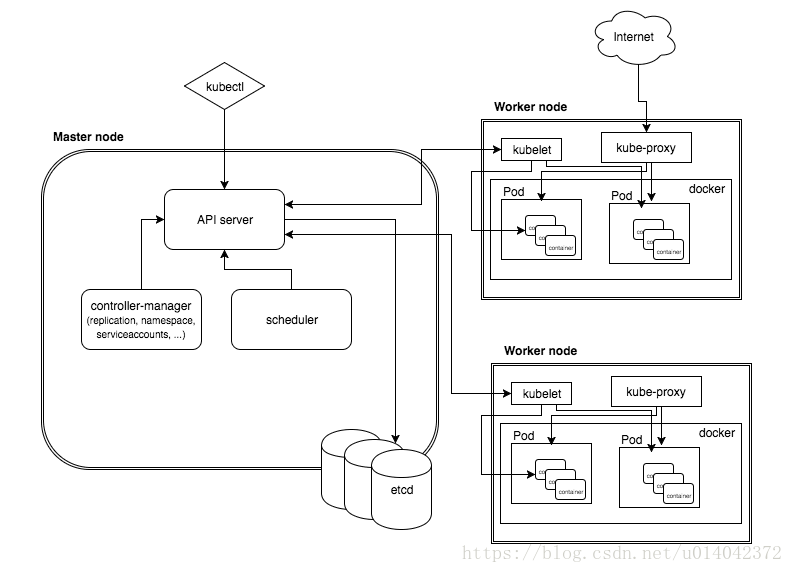
## 4.2用户权限管理

## 4.3区块链节点编排

### 4.3.1 证书 – 配置文件的初始化工作

管理区块链网络节点，一键式快速创建和部署生产级区块链环境，简化区块链的部署流程和应用配置。

### 4.3.2 Kubernetes通过HTTP请求进行容器编排



云上部署区块链节点的技术路线:

1. 准备好一个含有所部署的容器的相关信息的yaml文件，然后通过kubectl客户端工具发送给k8s集群中master结点的API server。
2. API server接收到外部请求并将资源内容存储到ETCD中。
3. Controller组件监控ETCD中的资源变化并作出反应：
   1. replicaSet检查数据库变化，创建yaml文件中所期望数量的pod实例。
   2. Scheduler把新的pods分配到可以运行这些pod的node结点上，形成一个部署方案，并把部署方案传给API server
4. API server根据上述部署方案请求相关node的Kubelet，通过Kubelet把pod运行起来，并把pod的相关信息保存至ETCD。在pod运行过程中，master节点中的Control manager会负责持续管理pod的状态。
5. Kubelet监控数据库变化，管理后续pod的生命周期，继续监视被分配到它所在的节点上运行的那些pod。如果有新的pod分配过来，kubelet则会负责在该节点上运行这个新pod。
6. kuber-proxy运行在集群各个主机上，管理网络通信，如服务发现、负载均衡。例如当有数据或请求发送到主机时，将其路由到正确的pod或容器。

#### 4.3.2.1 kubernetes容器编排组件-Master Node

Master节点是区块链节点集群中的网关和中枢枢纽，他的主要主要作用：暴露API接口，跟踪其他服务器的健康状态、以最优方式调度负载，以及编排其他组件之间的通信。

**Master节点如何容灾？**单个的Master节点可以完成所有的功能，但是考虑单点故障的痛点，生产环境中通常要部署多个Master节点，组成Cluster。

**API server**：kubernetes对外的唯一接口，提供HTTP/HTTPS RESTful API，即kubernetes API。所有的请求都需要经过这个接口进行通信。API server主要负责接收、校验并响应所有的REST请求，结果状态被持久存储在etcd当中，是所有资源增删改查的唯一入口，支持的数据类型包括JSON和YAML。

**ETCD**：负责保存k8s集群的配置信息和各种资源的状态信息，当数据发生变化时，ETCD会快速通知k8s相关组件。ETCD是一个独立的服务组件，并不隶属于kubernetes集群。在实际生产环境中，ETCD应该在k8s集群外部以集群方式运行，以确保服务的可用性。ETCD不仅仅用于提供键值数据存储，而且还为其提供了监听（watch）机制，用于监听和推送变更。在kubernetes集群系统中，etcd的键值发生变化会通知倒API Server，并由其通过watch API向客户端输出。

**Contorller manager**：Contorller manager负责管理集群各种资源，保证资源处于预期的状态。Controller manager由多种controller组成，包括replication controller、endpoints controller、namespace controller等 。由Contorller manager完成的主要功能主要包括生命周期功能和API业务逻辑，其中生命周期功能包括Namespace创建和生命周期维护、Event垃圾回收、pod终止相关的垃圾回收、级联垃圾回收及Node垃圾回收等；API业务逻辑包括由ReplicaSet执行的pod扩展等。

**Schduler**：资源调度，负责决定将Pod放到哪个Node上运行。Scheduler在调度时会对集群的结构进行分析，当前各个节点的负载，以及应用对高可用、性能等方面的需求。

#### 4.3.2.2 Kubernetes容器编排组件-Worker Node

Node结点是Kubernetes的工作节点，负责接收来自Master的工作指令，并根据指令相应地创建和销毁Pod对象，以及调整网络规则进行合理路由和流量转发。生产环境中，Node节点可以有很多个。

kubelet是Node的管理者，负责创建和维护pod的生命周期。当Scheduler确定在某个node上运行pod后，会将pod的具体配置信息（image、volume等）发送给该node的kubelet，kubelet会根据这些信息创建和运行容器，并向master报告其运行状态。

kube-proxy是实现kubernetes service的通信与负载均衡机制的重要组件。service接收到请求后，需要kube-proxy转发到pod。每个node都会运行kube-proxy服务，负责将访问的service的TCP/UDP数据流转发到后端的容器。如果有多个副本，kube-proxy会通过Iptables或LVS实现负载均衡。

Container runtime（容器运行环境）：每个node都需要提供一个容器运行环境，它负责下载镜像并运行容器。目前kubernetes支持的容器运行环境包括Docker、RKT、cri-o、Fraki等。

### 4.3.3区块链节点间的通信

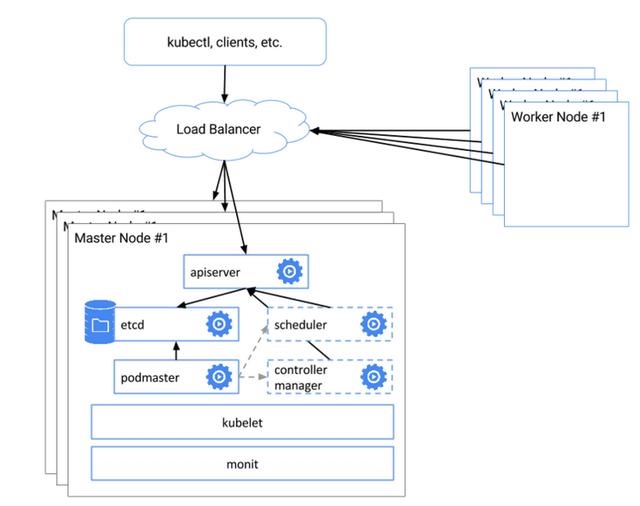
同一pod之内容器间的通信方式：同一pod内的容器共享同一网络空间，可以通过localhost进行相互访问；

不同pod之间的通信方式：通过访问service的虚拟IP进行访问。pod意外故障重启后，IP可能会发生变化，service是建立在一组pod对象之上的资源对象，将多个pod抽象为一个服务，解决了负载均衡和对外统一窗口映射的问题。

客户与pod的通信方式：通过ingress组件，客户可直接通过域名实现对pod的访问。客户的请求首先经过一个可以自动更新配置文件的nginx服务器，由这个服务器将请求负载均衡到对应node的对应端口，从而实现对指定pod的访问。

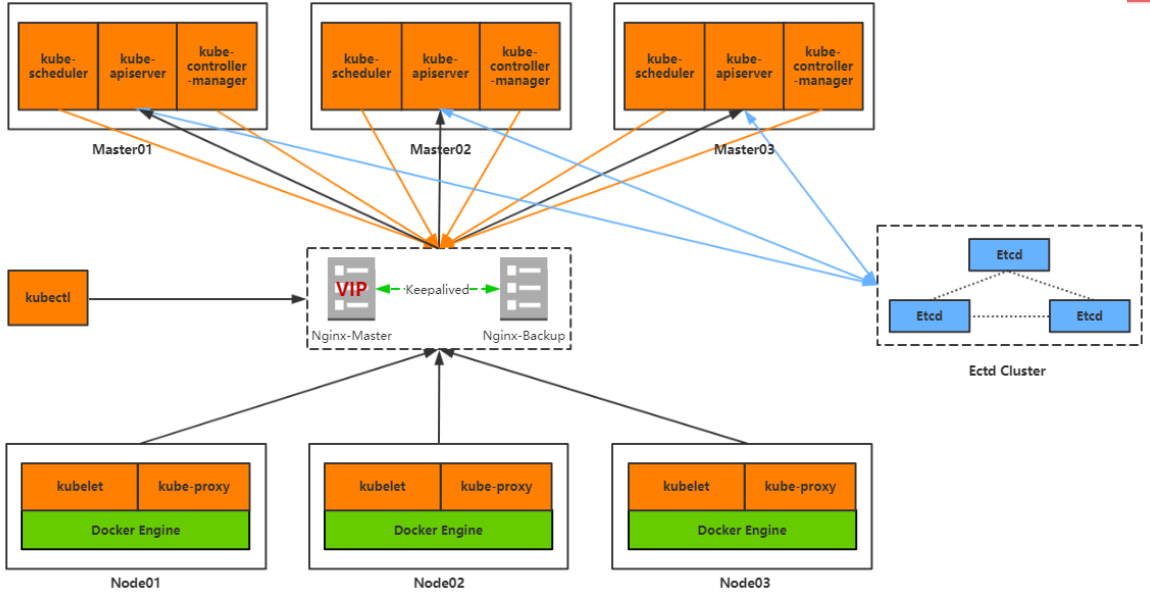
### 4.3.4区块链编排组件Master容灾机制

对于只有一个master节点的k8s集群，如果pod或node出现故障，k8s可以将其恢复；但是如果master节点出现故障，整个集群将陷入瘫痪。为了解决这个问题，实际生产环境中通常采用多主节点的架构搭建k8s集群。



Kubernetes多master节点容灾备份

master结点有4个核心组件，分别为API server、ETCD、Controller manager和scheduler，而高可用的kubernetes集群需要走同时保证这四个组件的高可用性。



Kubernetes多组件节点备份

Controller manager和scheduler：当一个多主节点的kubernetes集群正在运行时，只有一个master节点的Controller manager和scheduler履行其职能，其他master结点的Controller manager和scheduler则处于挂起状态。当正在运行的master结点出现故障时，其他master结点的Controller manager和Scheduler将被唤醒并继续履行其职能。

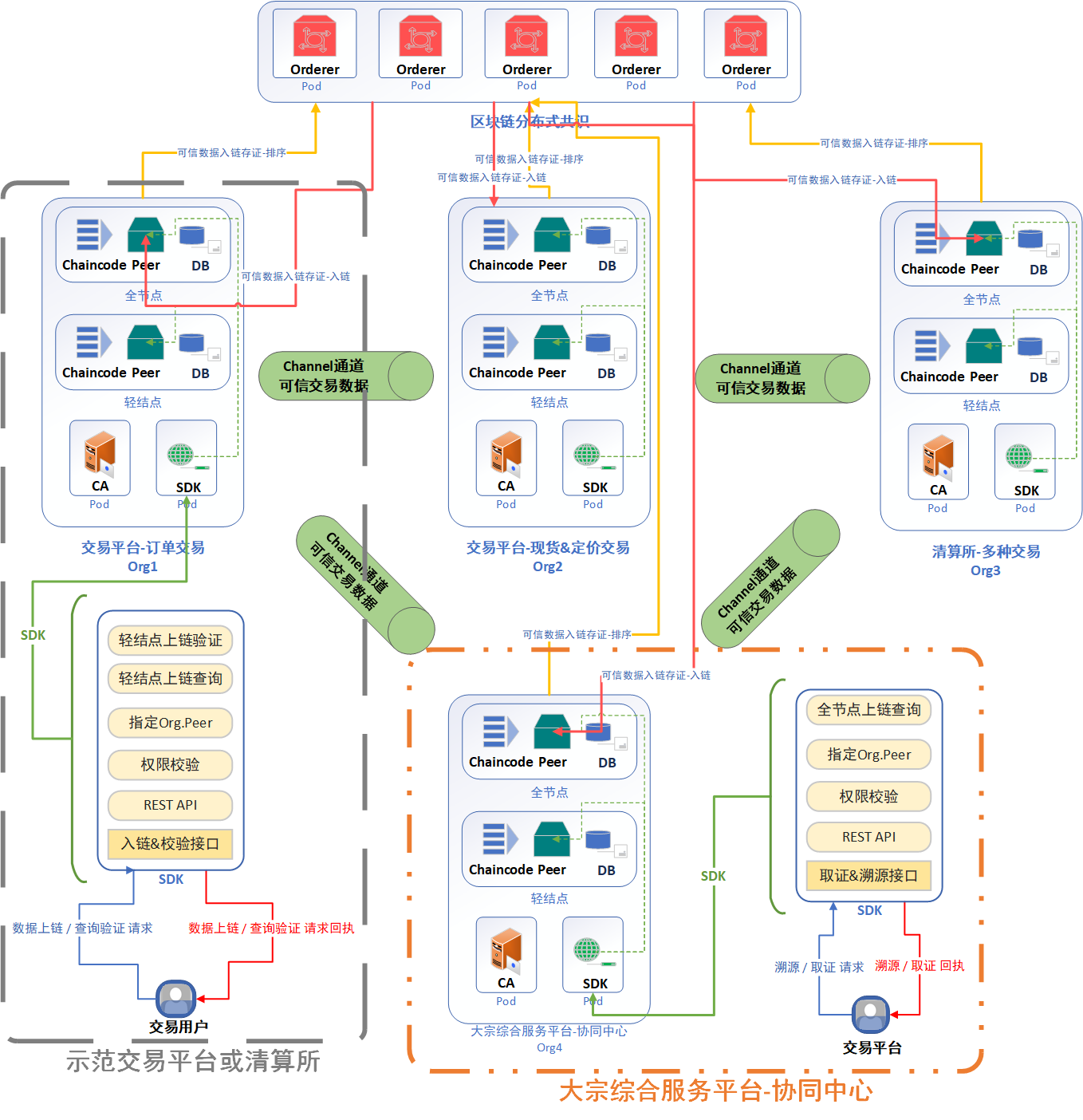
ETCD：当ETCD位于master结点内部时，若发生数据变动则同步更新，若正在使用的ETCD发生故障则改用其他master节点的ETCD。若ETCD在kubernetes外部以集群方式运行，则其自身可以保证自己的高可用性。

API server：当node结点需要访问API server时，将不再像单主结点集群一样直接访问API server，而是连接到负载均衡服务器；kubectl端也通过nginx负载均衡虚拟ip（virtual IP），将指令发送到master端。其中VIP的选择通过keepalive或heartbeat实现。

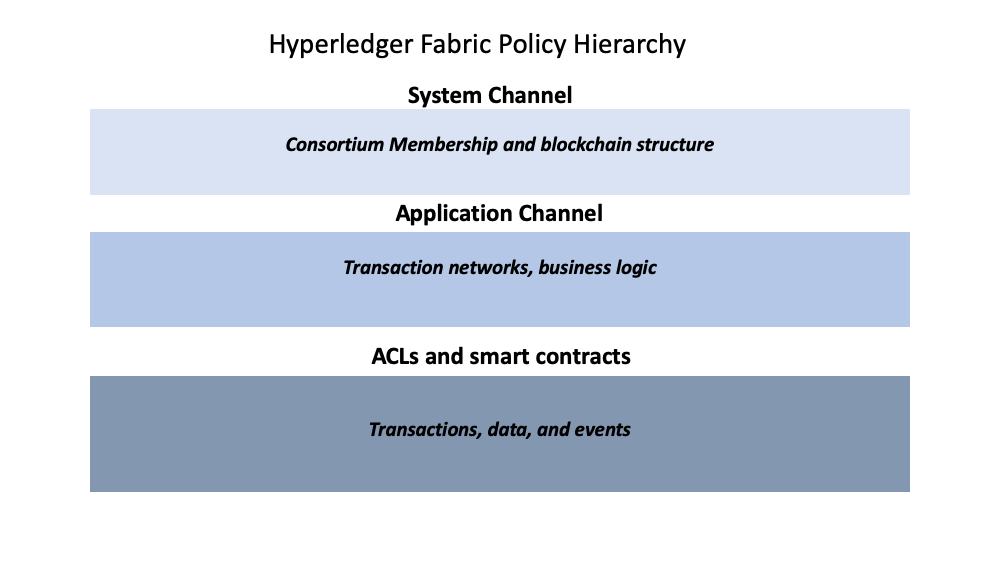
## 4.4容器节点的监控

Kubernetes DashBoard

# 5.区块链可信存证系统设计

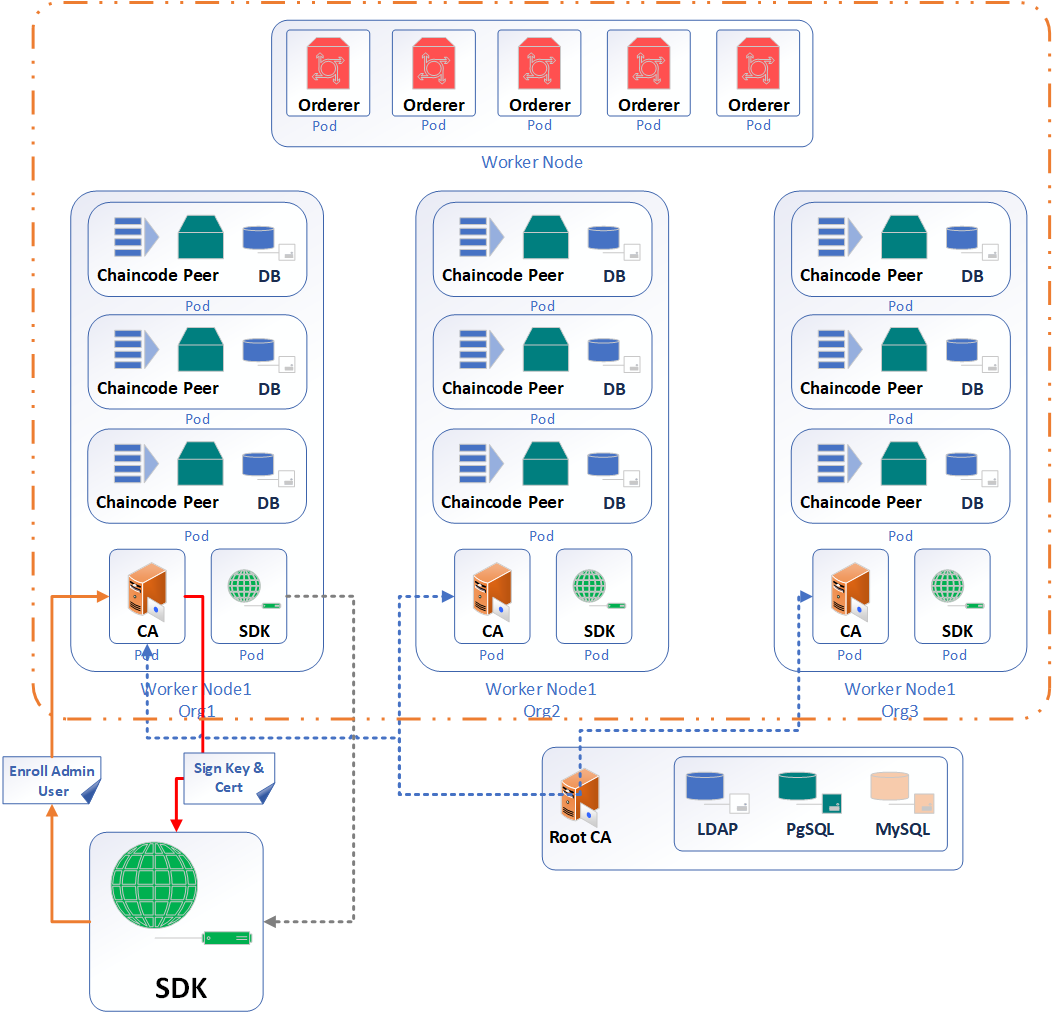


## 5.1区块链用户权限管理和交易合法性验证



Hyperledger Fabric 权限管理体系

### 5.1.1组织内证书的管理



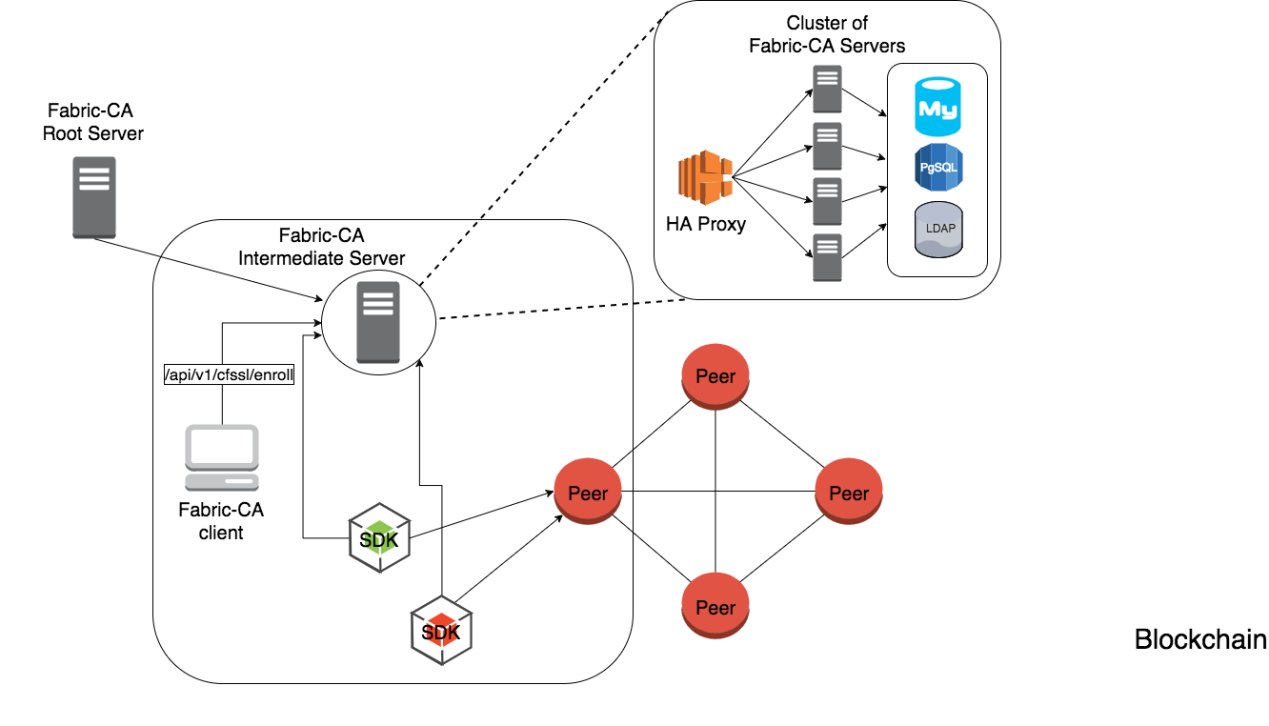
Fabric CA的组成架构

#### 5.1.1.1 Fabric CA

Fabric CA为Hyperledger Fabric行使证书机构的功能。

Fabric CA提供以下功能：

* 身份注册，或者将连接到LDAP作为用户注册；
* 颁发登录证书(ECerts)；
* 颁发交易证书(TCerts)，保证链上交易的匿名性与不可连接性；
* 证书续期与撤销

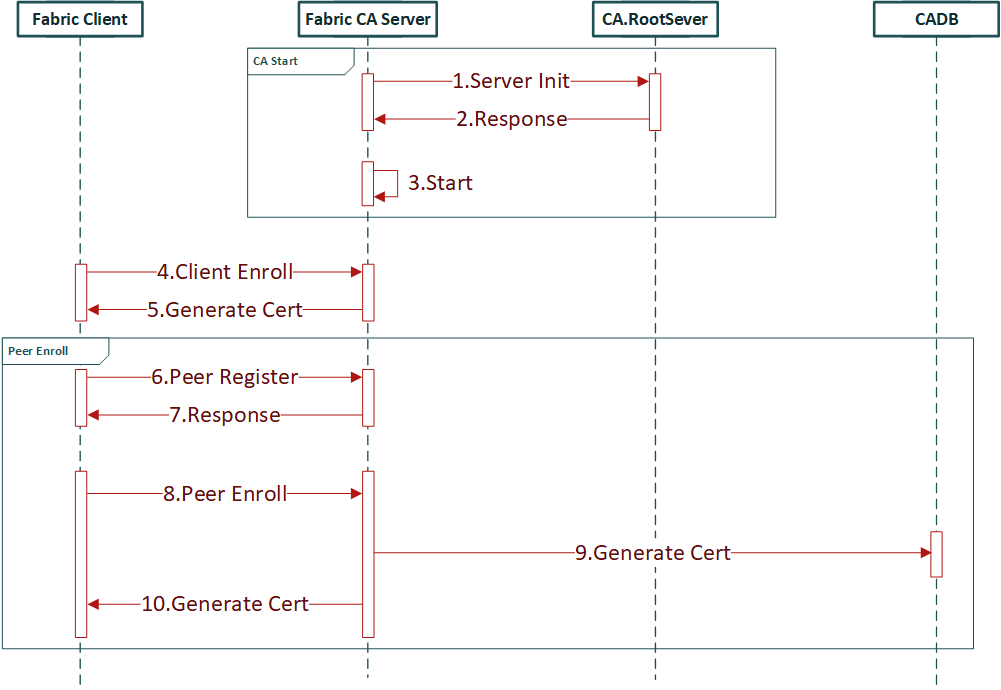


如何与Fabric SDK进行交互？

有两种方式与 Fabric CA 服务端交互：通过 Fabric CA 客户端，或者 Fabric SDK，所有与 Fabric CA 的交互都是通过 REST APIs 来实现的。REST APIs 的swagger说明文档见 fabric-ca/swagger/swagger-fabric-ca.json。

Fabric CA 客户端或者 SDK 可能会连接到 Fabric CA 集群中某一个 Fabric CA 服务端，这一部分可以通过上图右上部分获得更好的理解。客户端连接的是一个 HA 代理节点，这个 HA 代理节点为 Fabric CA 集群作负载均衡。所有的 Fabric CA 服务端共享同一个数据库。数据库用来保存用户和证书信息。如果配置了 LDAP，那么用户信息将会保存在 LDAP 中，而不是数据库中。

#### 5.1.1.2 用户Fabric CA注册证书



用户通过Fabrci CA注册证书

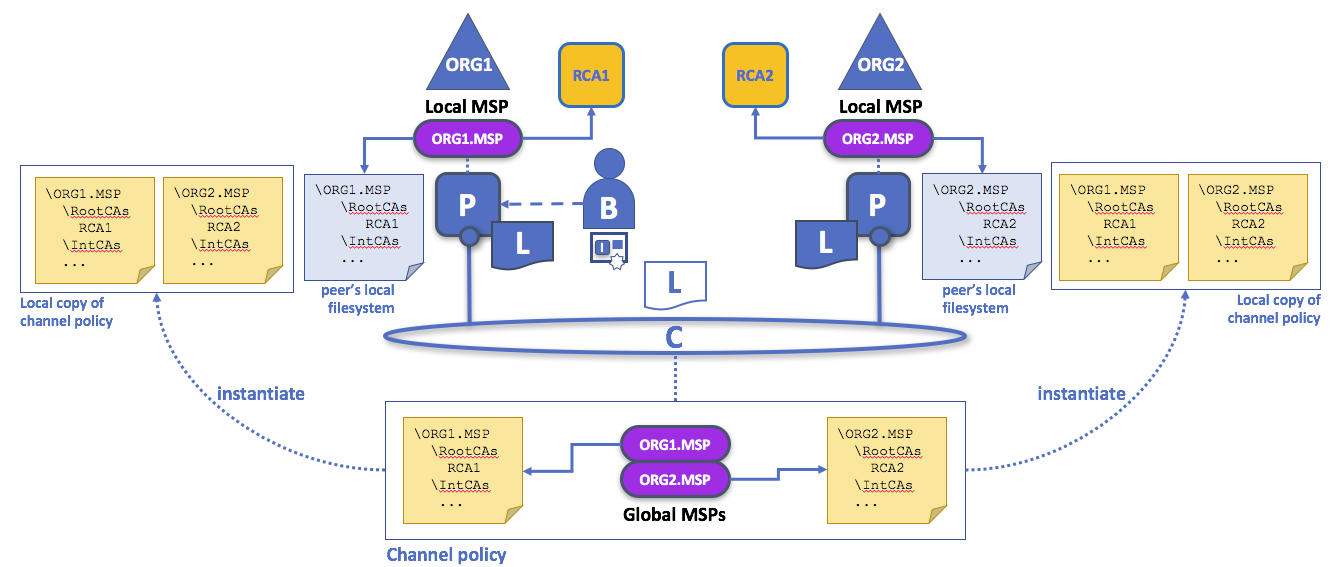
#### 5.1.1.3 区块链节点通过Fabric CA注册证书

这个图在书上P155

具体的实施措施在fabric-samples中的test-network下的使用ca替代crypto-generate 部分。

### 5.1.2组织内的权限管理

#### 5.1.2.1 MSP – 证明用户参与区块链中的合法身份



Fabric是一种只有被信任的节点才能参与到其中的区块链平台，因此参与者必须证明自己的身份。通过生成公钥和私钥的方法解决了怎样证明自己身份的问题。MSP用来解决：在不公开私钥的前提下，使网络上的其他节点信任该节点的问题。MSP定义了网络的参与者应该信任的一些组织。另外MSP给一些member提供了一些权限。

MSP完成的是用户的权限管理的功能。MSP定义的是一组规则，它规定了一些身份是可以被区块链网络的其他参与者信任的。并不是你有身份就可以参与到区块链网络中，MSP列举了哪些member可以参与到区块链网络（Channel, org, etc.）中。

MSP和CA之间的关系：CA产生代表用身份的identity，MSP则定义了一组permissioned identity是可以合法参与到区块链网络中的。除此之外，MSP可以给一个identity赋予一些权限，说白了MSP就是管理了一些具有某些权限的permissioned identity列表。比如每个人都有一个身份证，但是这并不代表你有参与人民代表大会的权利，而MSP维护了一个可参与人民代表大会的permissioned identity list。在用户向CA请求一个证书的时候，必须制定好该用户的权限/角色（client//Admin/ Peer/Orderer ）， MSP由CA创建，CA专门为组织创建证书和MSP。MSP 能够通过列举member的identity的方式来识别：哪些root CA 和intermediate CA可以定义一个域内的可信成员member。MSP也可以识别哪些CA有权利给member发放有效力的证书。

#### 5.1.2.2 ACLs 权限访问控制

Fabric 使用权限控制列表（ACLs）来管理资源的访问权限，在ACLs中通过给资源关联的策略来实现资源访问控制。默认的访问控制在 configtx.yaml 中，这个文件由 configtxgen 用来编译通道配置。访问控制可以通过两种方式中的一种来更新：编辑 configtx.yaml 自身，这会把 ACL 的改变传递到所有新通道；或者通过特定通道的通道配置来更新访问控制。

什么是资源？在Fabric中用户访问到的用户链码，系统链码，或者事件流源被视为系统的资源，因此应该被执行访问控制。

什么是策略？策略是 Fabric 运行的基础，因为它们根据 与完成请求所需资源相关联的策略 来检查 与请求关联的身份（或身份集）。背书策略用来决定一个交易是否被合适地背书。通道配置中定义的策略被引用为修改策略以及访问控制，并且在通道配置本身中定义。策略可以采用以下两种方式之一进行构造：作为 Signature 策略或者 ImplicitMeta 策略。

Signature 策略标示了要满足策略而必须签名的用户。例如：一个名为 MyPolicy 的策略只有被 “ Org1 的节点” 或着 “ Org2 的节点”签名才可以通过。

|  |
| --- |
| Policies:  MyPolicy:  Type: Signature  Rule: “Org1.Peer OR Org2.Peer” |

Signature 策略

ImplicitMeta 策略聚合配置层次结构中更深层次的策略结果，这些策略最终由签名策略定义。他们支持默认规则，比如“组织中大多数管理员”。这些策略使用的语法和 Signature 策略不同但是依旧很简单： <ALL|ANY|MAJORITY> <sub\_policy> 。比如，AnotherPolicy 策略可以通过 MAJORITY Admins （大多数管理员同意）的方式来满足。这里 Admins 是在通过更低级的 Signature 策略来满足的。

|  |
| --- |
| Policies:  AnotherPolicy:  Type: ImplicitMeta  Rule: "MAJORITY Admins" |

ImplicitMeta策略

#### 5.1.2.3 背书策略

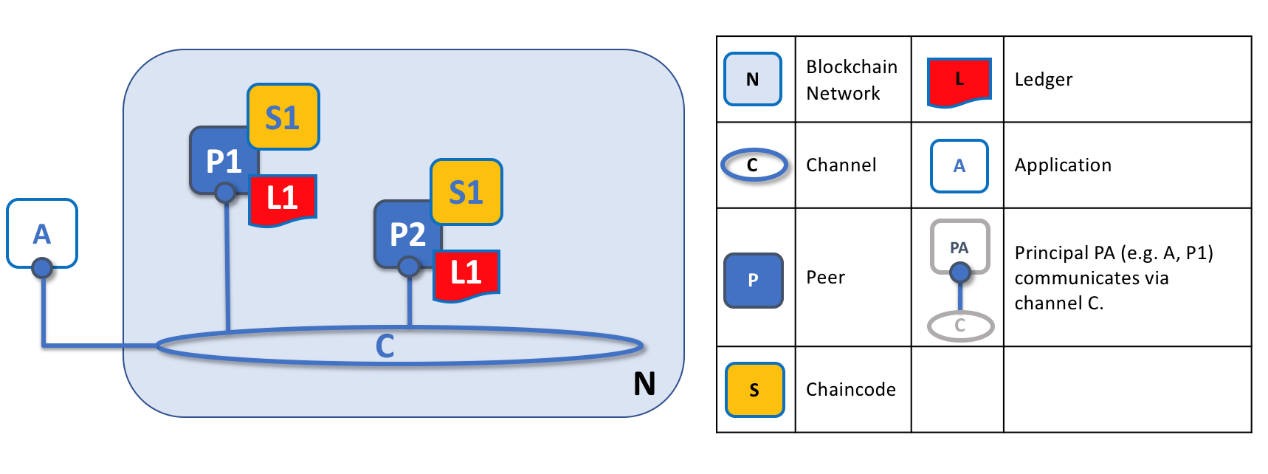
每个链码都有背书策略，指定通道上的一组 Peer 必须执行链码，并且为执行结果进行背书，证明交易是有效的。这些背书策略指定了必须为 Proposal 进行背书的组织。

作为 Peer 进行交易验证的一部分，每个 Peer 的检查确保了交易保存了合适数量的背书，并且是指定背书节点的背书。这些背书结果的检查，同样确保了它们是有效的（比如，从有效的证书得到的有效签名）。

链码级背书策略：

键级背书策略：

## 5.2区块链可信存证通道的管理



Fabric 通道

Hyperledger Fabric 的通道是两个或多个特定网络成员之间通信的专用“子网”，用于进行私有和机密的交易。通道由成员(组织)、每个成员的锚点节点、共享账本、链码应用程序和排序服务节点定义。网络上的每个交易都在一个通道上执行，在这个通道上，每一方都必须经过身份认证和授权才能在该通道上进行交易。加入通道的每个 Peer 节点都有 MSP 提供的身份，MSP 为每个节点授权访问通道中的其他节点和服务。

### 5.2.1 配置通道的创世区块

Fabric SDK通过构建通道创建交易并将交易提交给排序服务来创建通道。通道创建交易指定通道的初始配置，并由排序服务用于写入通道创世块。尽管可以手动构建通道创建交易文件。configtxgen工具通过读取定义通道配置的configtx.yaml文件，然后将相关信息写入通道创建交易中来工作。

在configtx.yaml文件中指定新通道的通道配置。建立通道配置所需的信息在configtx.yaml文件中以Key-value读写友好的形式指定。configtxgen工具使用configtx.yaml文件中定义的通道配置文件来创建通道配置，并将其写入protobuf格式，然后由Fabric读取。

configtx.yaml文件中指定新通道的通道配置应该包含以下信息：

* **Organizations**: 可以加入通道的组织。每个组织都有对用于建立通道MSP的密钥信息的引用。
* **Ordering service:** 哪些排序节点将构成网络的排序服务，以及它们将用于同意一致交易顺序的共识方法。该文件还包含将成为排序服务管理员的组织。
* **Channel policies**: 文件的不同部分共同定义策略，这些策略将控制组织与通道的交互方式以及哪些组织需要批准通道更新。
* **Channel profiles**: 每个通道配置文件都引用configtx.yaml文件其他部分的信息来构建通道配置。使用预设文件来创建Orderer系统通道的创世块以及将被Peer组织使用的通道。为了将它们与系统通道区分开来，Peer组织使用的通道通常称为应用通道。

configtxgen工具使用configtx.yaml文件为系统通道创建完整的创世块。因此，系统通道配置文件需要指定完整的系统通道配置。用于创建通道创建交易的通道配置文件仅需要包含创建应用通道所需的其他配置信息。

在Fabric网络中创建的第一个通道是系统通道。系统通道定义了形成排序服务的Orderer节点集合和充当排序服务管理员的组织集合。系统通道还包括属于区块链联盟的组织。联盟是一组Peer组织，它们属于系统通道，但不是排序服务的管理员。联盟成员可以创建新通道，并包括其他联盟组织作为通道成员。

要部署新的排序服务，需要系统通道的创世块。创世块用于部署单个Orderer节点，该Orderer节点使用该块创建系统通道并形成网络的排序服务。

### 5.2.2 创建通道

在部署了网络的节点并创建了Orderer系统通道后，即为Peer组织创建新通道。因为系统通道用作创建应用通道的模板，所以系统通道中定义的排序节点成为新通道的默认共识者集合。排序服务的管理员成为该通道的Orderer管理员。可以使用通道更新在共识者者集合中添加或删除Orderer节点和Orderer组织。

使用Fabric SDK将通道创建交易提交给排序服务。在排序服务创建通道之前，排序服务将检查提交请求的身份的许可。默认情况下，只有属于系统通道的联盟组织的管理员身份才能创建新通道。

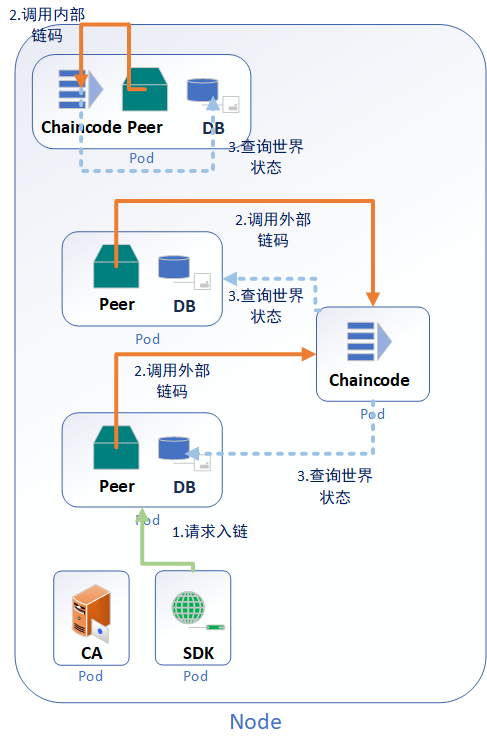
### 5.2.3 Peer节点加入通道

创建通道后，我们可以让Peer加入通道。属于该通道成员的组织可以使用peer channel fetch命令从排序服务中获取通道创世块。然后，组织可以使用创世块，通过peer channel join命令将Peer加入到该通道。一旦Peer加入通道，Peer将通过从排序服务中获取通道上的其他区块来构建区块链账本。

### 5.2.3 设定锚节点

组织的Peer加入通道后，他们应至少选择一个Peer成为锚定节点。为了利用诸如私有数据和服务发现之类的功能，需要Peer锚节点。每个组织都应在一个通道上设置多个锚节点以实现冗余。

## 5.3区块链服务智能合约的管理



Fabric的链码结构

从应用程序开发人员的角度来看，智能合约与账本一起构成了 Hyperledger Fabric 区块链系统的核心。账本包含了与一组业务对象的当前和历史状态有关的事实，而智能合约定义了生成这些被添加到账本中的新事实的可执行逻辑，主要用于操作账本上的数据，用户的应用程序通过链码与 Fabric 账本数据进行交互。

BaaS区块链服务平台的几大优势：

* **链码的相关参数需要多个组织达成一致**： 与其他使用1.x 版本的BaaS平台相比，如Hyperledger cello一个组织就能决定链码的参数（比如背书策略），通道内其他组织成员只能拒绝安装链码，因此不能参与和该链码相关的交易。新的链码生命周期会更加灵活，它既支持原有的这种中心化的信任模型，同时还支持去中心化的方式：在链码生效前需要通道内一定数量的组织对链码参数达成共识。
* **更安全的链码升级过程**： 在Hyperledger cello为主的区块链服务平台中，他们的链码生命周期中一个组织就能完成链码的升级交易，不管其他组织是否有安装升级的链码是否想升级，即升级交易可能由单个组织发起这会给尚未安装新链码的通道成员带来风险。在新的模型下，链码升级需要得到足够数量的组织同意之后才能升级。
* **更合理的背书策略升级**： 在Hyperledger cello为主的区块链服务平台中，要想升级背书策略，只能进行一次完整的链码升级过程，这意味着要重新打包链码、安装链码、升级链码。我们的BaaS平台的链码生命周期中可以避免这些冗余的步骤，支持在不重新打包或重新安装链码的情况下，更改背书策略或私有数据集合配置。用户还可以利用新的默认背书策略，该策略要求获得通道上大多数组织的背书。在通道中添加或删除组织时，会自动更新此策略。
* **可查验的链码包**： 在我们的BaaS平台中，新的链码生命周期可以将链码打包成更易阅读的 tar 包。这样查看链码信息或者组织间协调安装事宜会更方便。
* **使用一个链码包在通道内启动多个链码实例**： 在Hyperledger cello为主的区块链服务平台中，标识一个通道内的链码实例所用的名称和版本号两个要素是在安装链码时指定的。而我们的BaaS平台中，你可以利用一个安装好的包使用不同的名字在一个通道或多个通道内多次部署。
* **通道成员之间的链码包不需要为同一个**：在链码所模拟出的读写集相同的情况下，在BaaS平台中不同的组织可以自己扩展链码，例如为了他们组织的利益执行不同的验证。只要有所需数量的组织的链码执行结果匹配并为交易背书，该交易就会被验证并提交到帐本中。这还允许组织按照自己的时间单独推出小的修补程序，而不需要整个网络同步进行。

### 5.3.1 链码的生命周期

BaaS平台的链码生命周期的管理：

1. 上传链码：BaaS平台的管理员用户上传自己的链码，经过平台中的各个组织审批同意后，参与各方均遵守相同的智能合约的业务规范。
2. 打包链码：通过 peer 可执行文件或者 sdk 或者其他工具将链码源码打包成一个 .tar.gz包。tar包中包含两部分：一个 “Chaincode-Package-Metadata.json” 元数据文件以及另一个包含链码源码的 tar 包。“Chaincode-Package-Metadata.json” 文件中包含链码语言类型（Type）、链码路径（Path）以及标识链码的标签（Label）三个要素，比如："Path":"github.com/chaincode/fabcar/go","Type":"golang","Label":"fabcarv1"}。
3. 安装链码：在需要 参与背书的链码 或 查询账本 的组织的peer上 安装链码。安装成功后会返回一个链码识别号，标识号由链码标签和 hash 值组成。这个识别号后续也可以通过查询已安装链码的方式得到。
4. 审批链码定义：链码受链码定义约束，审批链码定义就是对这些定义的要素进行确认。需要使用这个链码的组织 需要对这个链码进行审批，只有链码的定义得到足够量的组织同意后，才能被启用，满足策略LifecycleEndorsement定义，默认为”大多数“。
5. 提交链码定义：链码定义被足够数量的组织审批同意后，由一个组织将这个链码定义提交到通道上：先从同意的组织哪里收集满足策略的背书，然后提交交易以确认这个链码定义。

#### 上传链码

前端：用户上传链码时需要指定关于链码的配置信息：

|  |  |
| --- | --- |
| 名称 | Mycc |
| 描述 | Abstore |
| 版本 | V1 |
| 链码运行时语言 | Golang |
| 链码源码压缩包.zip | Chaincode - mycc.zip |

#### 打包链码

SDK后端：接受到用户上传的链码的具体信息，将链码源码打包成一个tar包，需要满足一下要求：

1. 文件扩展名为 .tar.gz
2. tar 包中包含两部分：一个名为 “Chaincode-Package-Metadata.json” 元数据文件以及另一个包含链码源码的 tar 包。
3. “Chaincode-Package-Metadata.json” 文件中包含链码语言类型（Type）、链码路径（Path）以及标识链码的标签（Label）三个要素，示例：

"Path":"github.com/chaincode/fabcar/go","Type":"golang","Label":"fabcarv1"}

#### 安装链码

SDK 后端：使用 Admin 用户为组织内需要这个链码背书或账本查询的每一个 peer 节点安装打包好的链码。安装成功后Peer会返回一个链码识别号，标识号由链码标签和 hash 值组成。这个识别号后续也可以通过查询已安装链码的方式得到。

#### 审批链码

Peer审批链码定义：链码受链码定义约束，审批链码定义就是对这些定义的要素进行确认。需要使用这个链码的组织 需要对这个链码进行审批，只有链码的定义得到足够量的组织同意后，才能被启用，满足策略LifecycleEndorsement定义，默认为”大多数“。

链码定义的要素：

* Name: 调用链码时使用的名称；
* Version: 链码版本号，链码有变化则更新版本号；
* Sequence: 序号，一个链码每被定义一次序号增加一；
* Endorsement Policy: 背书策略；
* Collection Configuration: 私有数据集的配置文件路径；
* Initialization: 是否启动执行链码初始化函数；
* ESCC/VSCC Plugins: 背书及验证系统链码插件；

除此之外，链码定义还包括一个链码识别号（既上一步安装链码得到的），但是这个不是必须的。如果一个组织还没有安装链码，它可以不带链码识别号对这个链码定义进行审批，另外不同许这个链码识别号不一样，这也就是说，不同组织可以安装不一样的链码，但是可以作为对等的链码来使用，只要保证他们的读写集一致。

每一个需要用到这个链码的组织都需要进行链码定义的审批，审批动作由 Admin 身份执行，同组织内只需要执行一次，其他所有 peer 节点都可以使用。如果两个组织发送的链码定义不一致，则这个链码定义不会生效。

#### 提交链码

后端SDK：当一个链码定义收集到足够数量的组织的审批同意，则可以由其中任一组织发起提交交易，使得链码定义生效。提交过程由两步构成，第一步到相关组织的背书节点上执行背书收集背书签名（这个过程应该是通过系统链码帮助完成的），第二步将链码定义和背书发到排序服务，提交到通道上，使其生效。

后端SDK：一个链码定义是否已有足够数量的同意，可以通过checkcommitreadiness 命令查询。

### 外部链码的服务调用

我们的BaaS平台支持新的外部链码调用。外部链码启动器功能使运营者能够使用他们选择的技术构建和启动链码。使用外部构建器和启动器可以带来如下几种优势：

* **消除 Docker 守护进程依赖：**Fabric 以前的版本要求 Peer 节点能够访问 Docker 守护进程，以便构建和启动链码，但是 Peer 节点进程所需的特权在生产环境中可能是不合适的。
* **容器的替代品**：不再要求链码在 Docker 容器中运行，可以在运营者选择的环境（包括容器）中执行。
* **可执行的外部构建器**：操作员可以提供一组可执行的外部构建器，以覆盖 Peer 节点构建和启动链码方式。
* **作为外部服务的链码**：传统上，链码由 Peer 节点启动，然后连接回 Peer 节点。现在可以将链码作为外部服务运行，例如在 Kubernetes pod 中，Peer 节点可以连接到该 pod，并利用该 pod 执行链码。

#### 链码的外部构建和启动

在 2.0 中外部构建和启动功能是基于 Heroku 的 Buildpacks 实现的。buildpack 可以将源码等资料转化为一个可运行的实体，它包含一系列的脚本或者小程序。通常它包含如下几个部分：

1. bin/detect：用于前置检测和判断，最终判定当前链码是否用这个构建器来构建并启动。比如检测当前链码的语言类型，是否适合这个构建器来构建；
2. bin/build：用来执行构建动作，将链码相关的原始资料转化为一个可运行的实体，比如用 go build 将一个 go编写的链码编译为一个可执行程序；
3. bin/release（可选）：将链码相关的元数据提供给 peer，目前有两种类型的数据是 peer 可以从 这个步骤的产出中消费的：1. couchDB 的索引数据，一般放在和源码一起打包的 META-INF 文件夹里；2. 连接到外部链码服务相关的连接信息 connection.json；
4. bin/run（可选）：通过修改 core.yaml 中的相关配置，让 peer 使用外部构建器和启动器将链码启动起来。

#### 外部构建和运行的配置

利用Fabric对外部链码构建和启动机制的支持，现在 fabric 可以支持直接在外部运行链码服务，完全独立于 peer 服务，这样链码的运行形式完全“自主可控”。

如果将链码在外部运行，理论上不需要把链码安装到 peer 节点了，但是需要 peer 能够知道这个外部链码服务的存在并能够与之建立链接，所以仍要准备包含了这个外部链码相关信息的链码包并利用外部构建器和运行器在安装的过程中提供给 peer，只是不再需要链码的源码了。2.0 中链码包都是标准的 .tar.gz 格式，链码包mycc.tar.gz需要包含以下文件：

|  |
| --- |
| mycc.tar.gz  |\\_metadata.json --- 1  \\_code.tar.gz  |\\_ metadata/ --- 2  \\_connection.json --- 3 |

metadata.json：定义链码基本信息，它与正常用 peer lifecycle package 生成的链码包中的 metadada.json 的区别是不用配置 path 项，另外 type 项需要定义一个特殊的用于 bin/detect 识别的类型。比如：{"path":"","type":"external","label":"mycc"}。

metadata：couchDB 索引，如果没有可以不包含这个目录（不要用空目录）。

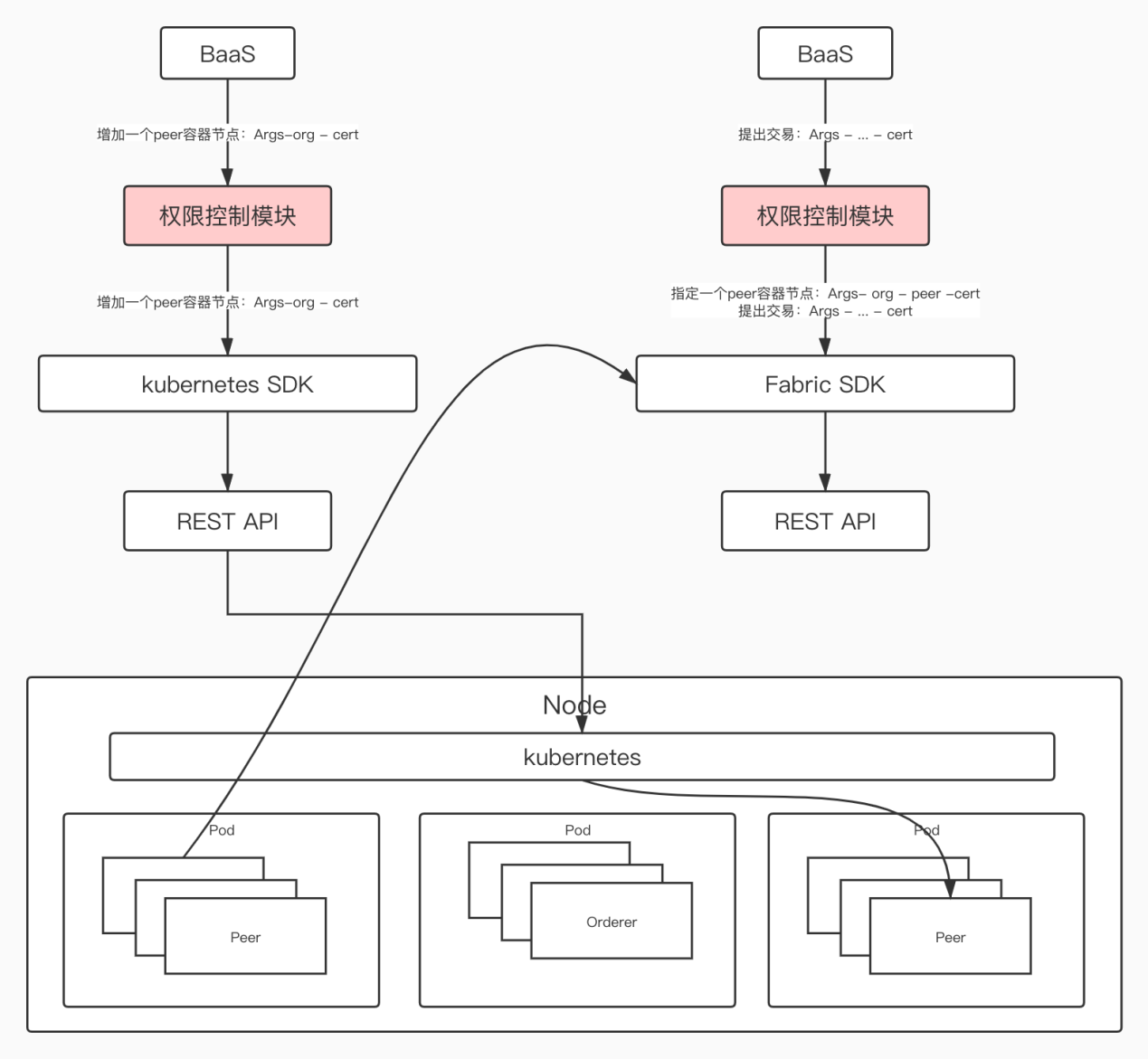
connection.json：定义链接外部链码服务的信息，bin/build和bin/release 将这个文件逐步提供给 peer，它包含以下要素：

|  |
| --- |
| {  "address": "host.docker.internal:9999", // 外部链码服务地址  "dial\_timeout": "10s",  "tls\_required": false, // 这里必须是 js 的 bool 类型，官网示例中用了双引号，变成字符串，会导致出错，下面同理  "client\_auth\_required": false,  "client\_key": "-----BEGIN EC PRIVATE KEY----- ... -----END EC PRIVATE KEY-----",  "client\_cert": "-----BEGIN CERTIFICATE----- ... -----END CERTIFICATE-----",  "root\_cert": "-----BEGIN CERTIFICATE---- ... -----END CERTIFICATE-----"  } |

#### 外部构建运行

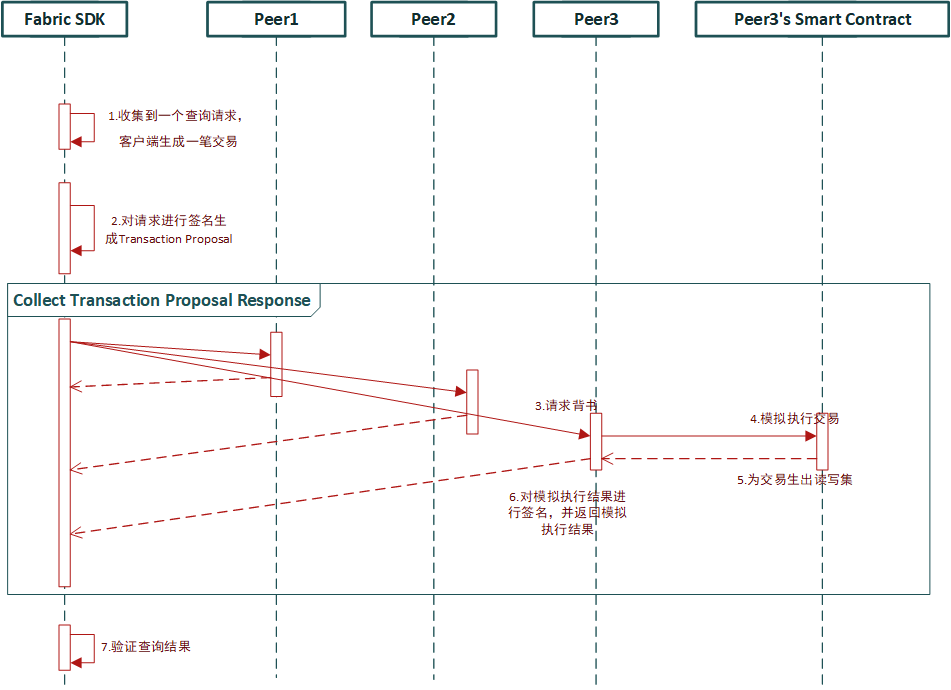
构建外部链码并启动的流程图

## 5.4区块链服务调用



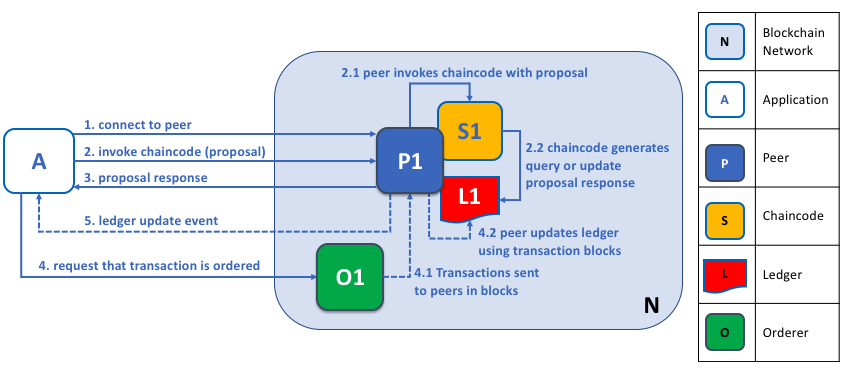
BaaS区块链服务流程

### ·5.4.1区块链可信存证溯源和校验

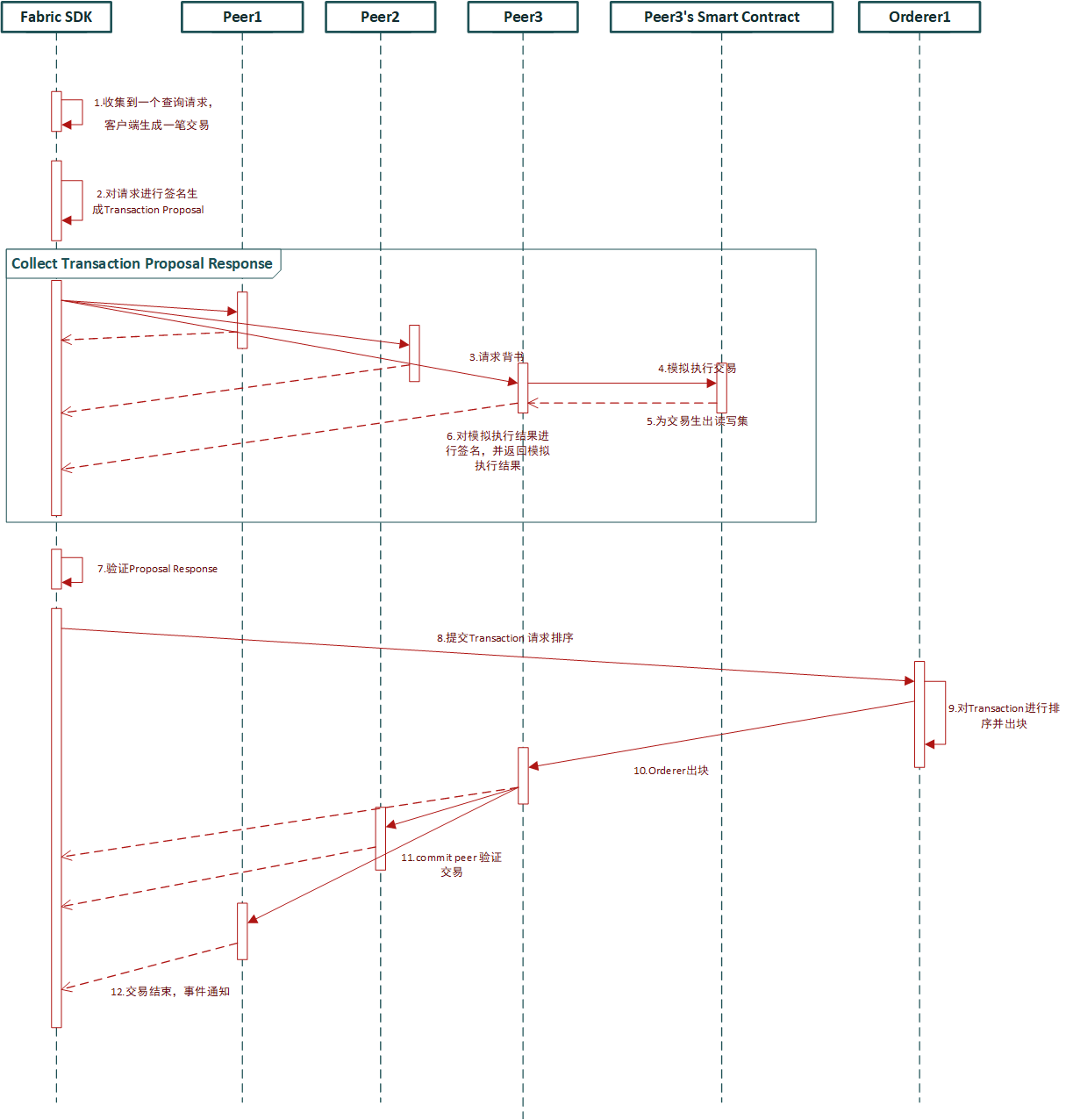


Fabric SDK的查询过程

### 5.4.2区块链可信存证入链



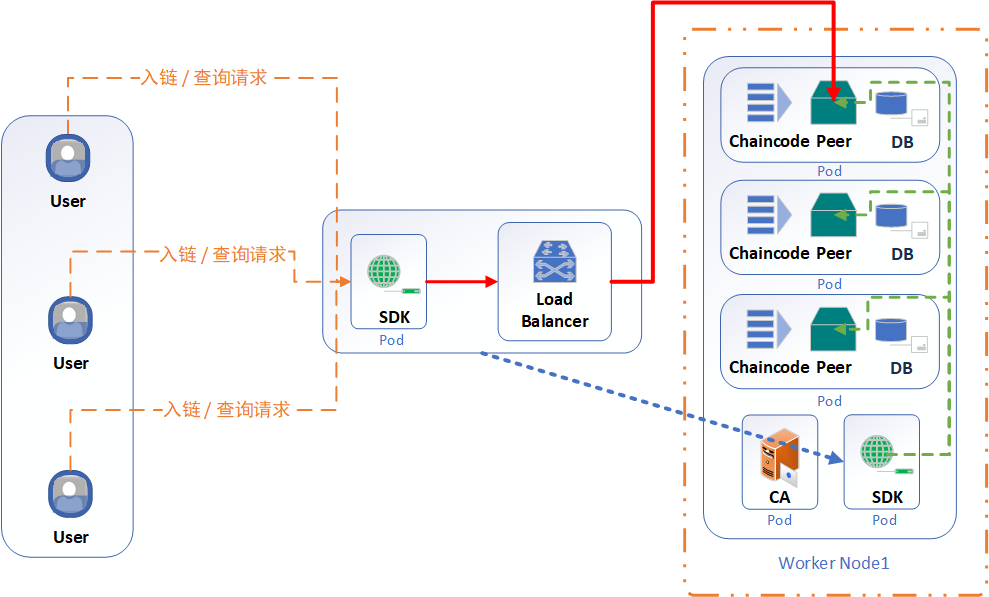
Fabric SDK 请求入链的过程



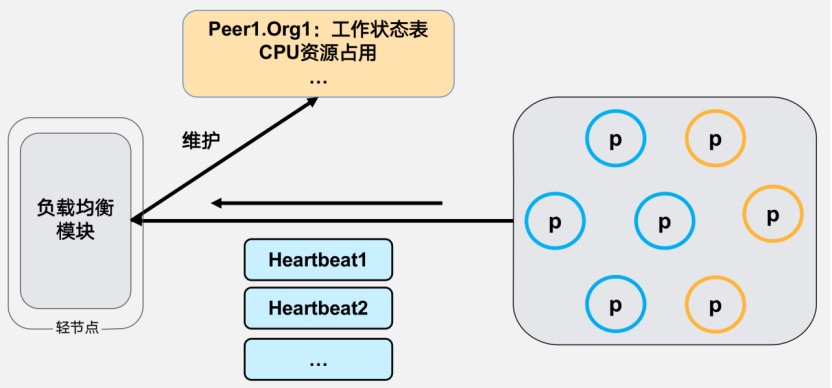
Fabric SDK 请求入链序列图

### 5.4.3 区块链私有数据存证

### 5.4.4 区块链请求的负载均衡



全节点查询和入链的负载均衡



负载均衡的具体机制

## 5.5区块链网络的监控

参考Fabric Explorer