Burrow是最早Monax开发的项目，后来进入HyplerLedger孵化。Burrow的Incubation proposal中介绍，burrown之前的名字是 eris-db。

从Casey Kuhlman的博文中，可以得知Burrow的定位是一个通用的智能合约执行引擎:

Burrow应当是一个支持授权的合约引擎，它的目标可能是要支持多种类型的合约，或许会成为一个通用的区块链节点系统？

[Caliper](https://www.hyperledger.org/projects/caliper" \o "HyperLedger Caliper)是一个Benchmark工具，华为贡献的，[Measuring Blockchain Performance with Hyperledger Caliper](https://www.hyperledger.org/blog/2018/03/19/measuring-blockchain-performance-with-hyperledger-caliper" \o "Measuring Blockchain Performance with Hyperledger Caliper)。

[Cello](https://www.hyperledger.org/projects/cello" \o "HyperLedger Cello)是一个部署管理平台，也就是现在常说的baas，用来管理组成链的节点，以及部署链、管理链。用户可以直接通过Cello部署合约。

[超级账本HyperLedger的cello项目的部署和使用](http://www.lijiaocn.com/%E6%96%B9%E6%B3%95/2018/04/25/hyperledger-cello.html)

[Composer](https://www.hyperledger.org/projects/composer" \o "HyperLedger Composer)是用于合约、区块链应用开发的工具，用来简化、加快区块链应用的开发过程。

[Explorer](https://www.hyperledger.org/projects/explorer" \o "HyperLedger Explorer)是一个区块链浏览器。

[Indy](https://www.hyperledger.org/projects/hyperledger-indy" \o "HyperLedger Indy)是用于去中心化身份认证(decentralized identity)的分布式账本。[Indy get started](https://github.com/hyperledger/indy-node/blob/stable/getting-started.md)中虚构了一个故事，通过这个故事可以很容易理解 Indy的用途。Indy这个项目挺有意思，是一个挺实在的应用，以后有时间要重点研究下。

[Iroha](https://www.hyperledger.org/projects/iroha" \o "HyperLedger Iroha)也是一个区块链的framework(框架)，是日本的金融科技公司Soramitsu, Hitachi，以及NTT Data、Colu等贡献的

[Sawtooth](https://www.hyperledger.org/projects/sawtooth" \o "HyperLedger Sawtooth)是另一个区块链的framework（框架），采用模块化设计。 framework的竞争很激烈，会上演类似mesos、swarm、kubernetes之间的三国杀嘛？相比Fabric，Sawtooth的一个亮点是能够并行的处理事务：支持事件订阅：

**Fabric目标**

Fabric的主打卖点是：

在permissioned networks（授权网络）上，进行private confidential transactions（私人保密交易）。

Fabric的定位是商业应用、是企业与企业之间的联盟链。联盟链是不能随随便便接入的，必须有准入限制（permissioned）。相比较于比特币的网络是全公开的网络，任何人都可以接入。

**fabric结构**

Fabric组件只有两个：Peer和Orderer，它们就是两个二进制程序。每个参与方可以选择部署一个/多个Peer，或者一个/多个Orderer，或者两个都部署。所有参与方的Peer/Orderer，彼此通信、连接，就组成了Fabric网络。

Peer：每个Peer中都存放全量的数据（账本），也就是完整的链（必须是Peer参与的链，见后面“控制信息的传播范围”）。

Orderer：通过Orderer之间的协商，确保每个Peer都能收到正确的数据。

system chain：Fabric中有一组特殊的链，叫做“system chain”。这组链相当于整个Fabric网络的配置文件，里面记录了所有的Channel信息、参与者的信息。（每个Channel对应一条system chain）

创世块：是system chain的第一个区块。这个区块通常是用configtxgen命令生成的。

Anchor Peer(锚点):每个参与者可以将自己的一个Peer地址写入到system chain中，这样的Peer被称为“锚点”（Anchor Peer）。Orderer们从system chain中获得Anchor Peer的地址，并将形成的共识通知给它们。Anchor Peer通过Gossip协议，将结论八卦给其它的Peer。

Leader Peer：Orderer是将共识发送给每个组织的Leader Peer，Leader Peer是一个组织的多个Peer之间选举出来的，或者组织的主动指定的。Anchor Peer（锚点）主要作用是用于Peer之间的发现。

System Chain与Orderer的存在，使“控制信息的传播范围”成为可能。既然Orderer是输出数据的源头，那么就可以通过Orderer控制信息的传播范围了。

联盟链：企业与企业之间的链，不全公开，接入需要验证接入。

私链：也就是Channel，Orderer的data目录，就会发现每个Channel都会有一个独立的链。

Peer的data目录中，只会有Peer所加入的Channel的链：

Orderer中的链与Peer中的链是不同的。Oderer的链中存放的是Channel的配置，是system chain

Peer的链中存放的是Channel的数据。比对一下它们的内容就可以知道。

Peer与Orderer交织，数据更新请求在Orderer之间来回穿梭，最终被送往了Anchor Peer，大嘴巴的Anchor Peer转身通知了身边的所有Peer。

Fabric网络：组成网络的每一个Peer和Orderer，都是得到了批准的、实名的，想要访问这个网络的用户也需要得到批准，并实名签署自己的操作。这就是Fabric网络。

**合约**

参与者们在现实世界里达成共识。这个共识，更像谈判后签署的合同，在区块链中，被称为合约。

落实到Fabric，合约就是分布在每个Peer上的容器。

首先使用Go或者js编写处理逻辑，代码中约定了数据的修改方式。然后这些代码会被打包、签署，并提交到Fabric中，最终成为Peer上的一个Docker容器。

合约是可以升级、改动的,只有在当初签署了合约的人员都同意的情况下，才能更改合约。

背书策略的设计，通过背书的策略，可以约定只有当多个Peer上的合约都得出同样的结果时，对合约的调用才能被接受，只对自己的Peer动手脚是不行的。

合约需要在每个Peer上进行安装的，但激活只需要一次。安装合约其实就是将打包的合约上传到了Peer上（当然这中间还有一些其它检查）：

**MSP**

它是一个包含了根证书、证书（经过CA签署的公钥）和私钥的目录,在组织、组件和用户，三个地方会用到。

组织的msp目录中包含的全是可以公开的证书，没有私钥。因组织的msp目录,是要被写入到system chain中的,被包含在组织msp目录中的证书，主要被用来验证用户证书的有效性。

用户的证书的有效性：必须是用根证书对应的私钥签署的，否则就会被认定为无效，并拒绝服务。

组件msp目录的使用者是Fabric网络中的peer和orderer程序

用户msp目录的使用者是Fabric网络之外的个人或者客户端。

**Fabric实现网络的准入方式：**

必须向拥有根证书私钥的机构，申请一个证书。通过这个限制，保证Fabric网络中的参与者的身份都是真实的（参与者的证书被盗用除外）。

访问Fabric网络的个人或者客户端，要有一个账号；

组成Fabric的网络的每个Peer和Orderer也要有一个账号。

MSP中存放的就是每个账号的私钥和证书。

cryptogen命令 和 FabricCA组件：Msp目录可以用Fabric提供的cryptogen命令生成，但是用cryptogen命令生成是非常不灵活的，生产环境中，应当使用Fabric提供的另一个组件FabricCA。

注意FabricCA组件：不属于Fabric网络，它是网络之外的一个用来签署证书的服务。借助FabricCA，还可以实现账号的分级，简单说就是一个组织的管理员，可以自由地创建属于该组织的子账号。（这是刚需）

**配置：**

配置分两种：组件的配置文件、system chain中的配置区块

1组件的配置文件：就是orderer的配置文件orderer.yaml 以及peer的配置文件core.yaml

2 system chain中的配置区块

Configtxgen命令：能将二进制的创世块转换成json格式，生成创世块（system chain的第一个区块）。

configtxgen -inspectBlock ./genesisblock

从Fabric的网络中也可以读取到指定channel的当前配置：

./peer.sh channel fetch config config\_block.pb -c mychannel -o orderer.example.com:7050

只不过读出来的文件都是protobuf格式的，需要在另一个工具configtxlator的帮助下，将其转换为json格式：

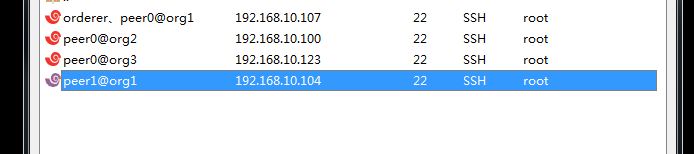
configtxlator proto\_decode --input config\_block.pb --type common.Block

更新system chain的过程更繁琐，以在Channel中添加一个新的成员为例。

在Channel中添加一个新的成员，实质就是修改Fabric网络中记录的Channel配置。这个过程需要好几步：

生成新成员的json格式的配置文件

* 从Fabric中读取Channel最新的配置
* 将读取的Channel配置转成json格式后，将新成员的json配置加入其中
* 将修改后的Channel配置和修改前的Channel配置都转成protobuf格式，然后用configtxlator生成更新文件
* 将更新文件由protobuf格式转换成json格式，加上包含channel信息的信封后，再转换成protobuf格式
* 将加信封的protobuf格式的更新文件发送给Channel中其它组织的管理员，让他们用自己的私钥进行签署
* 最后由其中一个管理员将得到足够签署的更新文件提交到网络



Fabric浏览器