Blockchain use-cases

Supply Chain

- Food Safety
- Trade Finance
- Asset management and tracking
- Agriculture
- Global Trade
- Anticounterfeiting
- Product Lifecycle management
- Provenance
- Logistics & Shipment
- High Value goods tracking
- Global Trade
- Anticounterfeiting
- Chain of custody tracking

Health Care

- Secured Patient Data Management
- Patient Data exchange & interoperable
- Clinical trials and data security
- Pharmaceutical tracking & purity
- Health research
- Healthcare IoT
- Eliminate fraud, Counterfeit and theft
- Managing Intellectual Property
- Regulatory Compliance
- Medical devices management

Financial

- Escrow
- Equity
- Trading
- Settlements
- Transfer of Value
- Know your customer
- Anti money laundering
- Peer-to-peer Lending
- Exchanges
- International Payments
- Regulatory compliance
- Derivatives Trading

Government

- KYC
- Voting
- Railways
- Vehicle Registration
- Licenses and Registration
- Copyrights
- Data regulation
- Mining Industry
- Taxation
- Legal
- Certificates
- Land Registration
- Transparency and traceability of spending
- Fraud and Corruption management
- Immigration management

Insurance

- Peer-to-peer Insurance
- Automated Insurance claim
- Underwriting
- Client on-boarding
- Car Insurance optimization
- Health Insurance optimization
- Fraud Detection
- Claim prevention
- New Payment model
- Medical record management
- Predictions and Analytics solutions

IoT

- Device to Device payment
- Smart Home
- Autonomous Vehicles
- Improved supply chain
- Security and Auditable
- Improve shipping and logistics
- Food Safety
- Asset Management
- Health & Fitness

Payments

- Micropayments
- Cross-border payments
- Remittance
- Microfinance
- Identity verification
- Audit
- Hedge funds

Media

- Ad Networks
- Media Buying
- Ad Exchanges
- Royalty Payment
- Content Bypassing aggregator
- Pricing for paid content

Agriculture

- Smart farming
- Reduce food waste
- Food Safety
- Food fraud
- Animal Health
- Authenticity
- Subsidies
- Reduce Suicide rate
- Farmers loan process
- Fair pricing

Others

- Identity
- Real Estate
- Tourism
- Energy Trading
- Gaming

14 Ways Blockchain Will Transform Banking

Interbank Transactions

Remittance

Smart Contract Enforcement

Crypto Banking

Record Sharing & Storage

Clearing & Settlement

Loan Syndication

Regulatory Technology

KYC/AML

Regulatory Reporting

Trade Finance

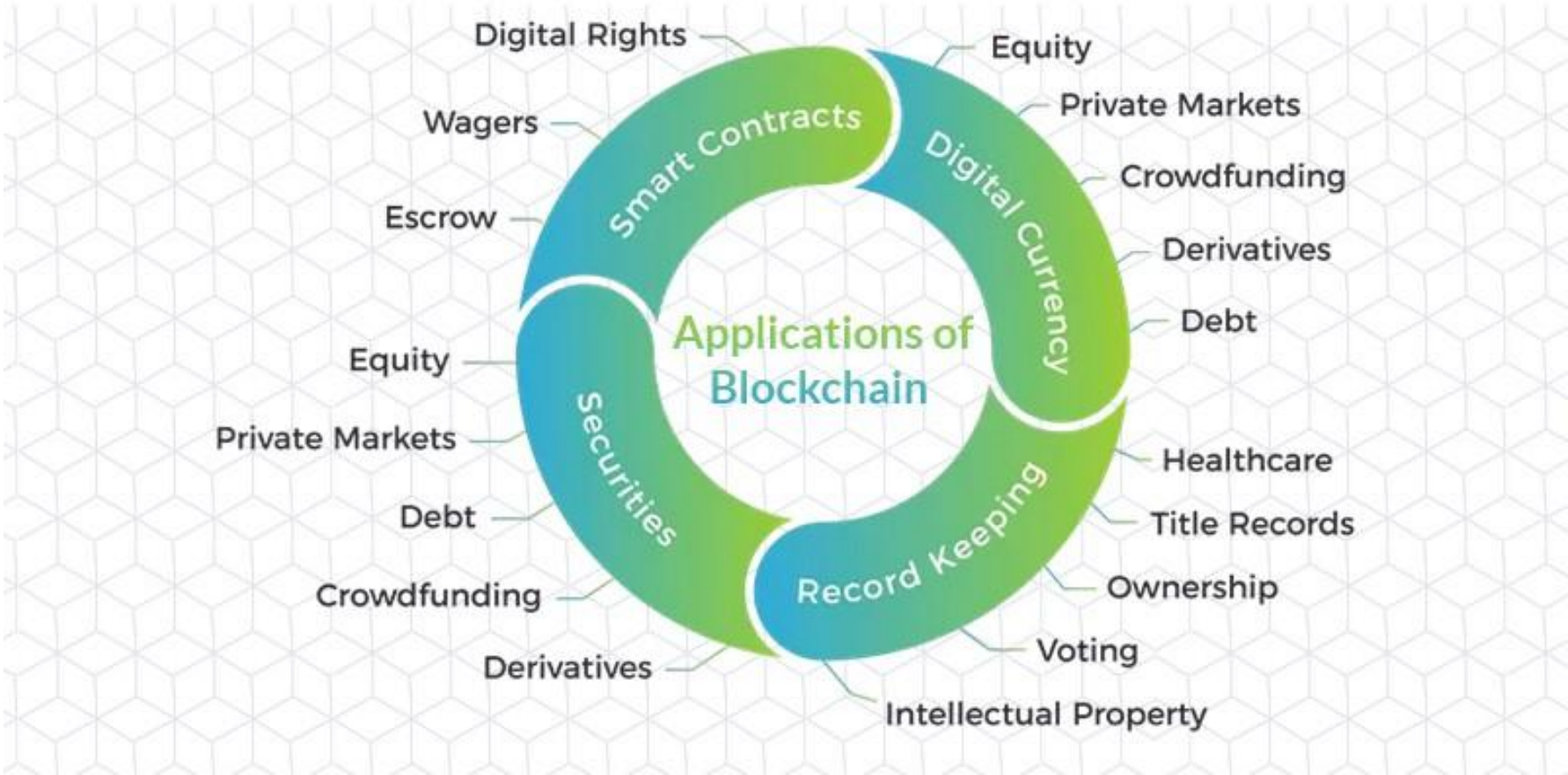
Data Security

Increasing Transparency

Serving The Unbanked

DISRUPTOR DAILY

Blockchain in Banking


Applications of Blockchain
Smart Contracts
Digital Currency
Record Keeping
Securities
Digital Rights
Wagers
Escrow
Equity
Private Markets
Crowdfunding
Derivatives
Debt
Healthcare
Title Records
Ownership
Voting
Intellectual Property
Equity
Private Markets
Debt
Crowdfunding
Derivatives

Steps to build Blockchain solutions

1. Identify the Business problem and Business Value rather than focusing on technology.
2. Choose right Blockchain types Public, Private or Permissioned
    a. Ethereum, EOS
    b. Hyperledger Projects
    c. Stellar etc.
3. Deploy to cloud providers or On-prem
    a. AWS
    b. Azure
    c. IBM

chaincode lifecycle

1. **Package the chaincode:** This step can be completed by one organization or by each organization.
2. **Install the chaincode on your peers:** Every organization that will use the chaincode to endorse a transaction or query the ledger needs to complete this step.
3. **Approve a chaincode definition for your organization:** Every organization that will use the chaincode needs to complete this step. The chaincode definition needs to be approved by a sufficient number of organizations to satisfy the channel's LifecycleEndorsment policy (a majority, by default) before the chaincode can be started on the channel.
4. **Commit the chaincode definition to the channel:** The commit transaction needs to be submitted by one organization once the required number of organizations on the channel have approved. The submitter first collects endorsements from enough peers of the organizations that have approved, and then submits the transaction to commit the chaincode definition.

Policies

ACL

Endorsement policy

Orderer - Raft demo

Private data

Policies

a policy is a set of rules that define the structure for how decisions are made and specific outcomes are reached.

To that end, policies typically describe a **who** and a **what**, such as the access or rights that an individual has over an **asset**.

For example, an insurance policy defines the conditions, terms, limits, and expiration under which an insurance payout will be made. The policy is agreed to by the policy holder and the insurance company, and defines the rights and responsibilities of each party.

in Hyperledger Fabric, policies are the mechanism for infrastructure management. Fabric policies represent how members come to agreement on accepting or rejecting changes to the network, a channel, or a smart contract.

The policies that govern the network are fixed at any point in time and can only be changed using the same process that governs the code.

Policies

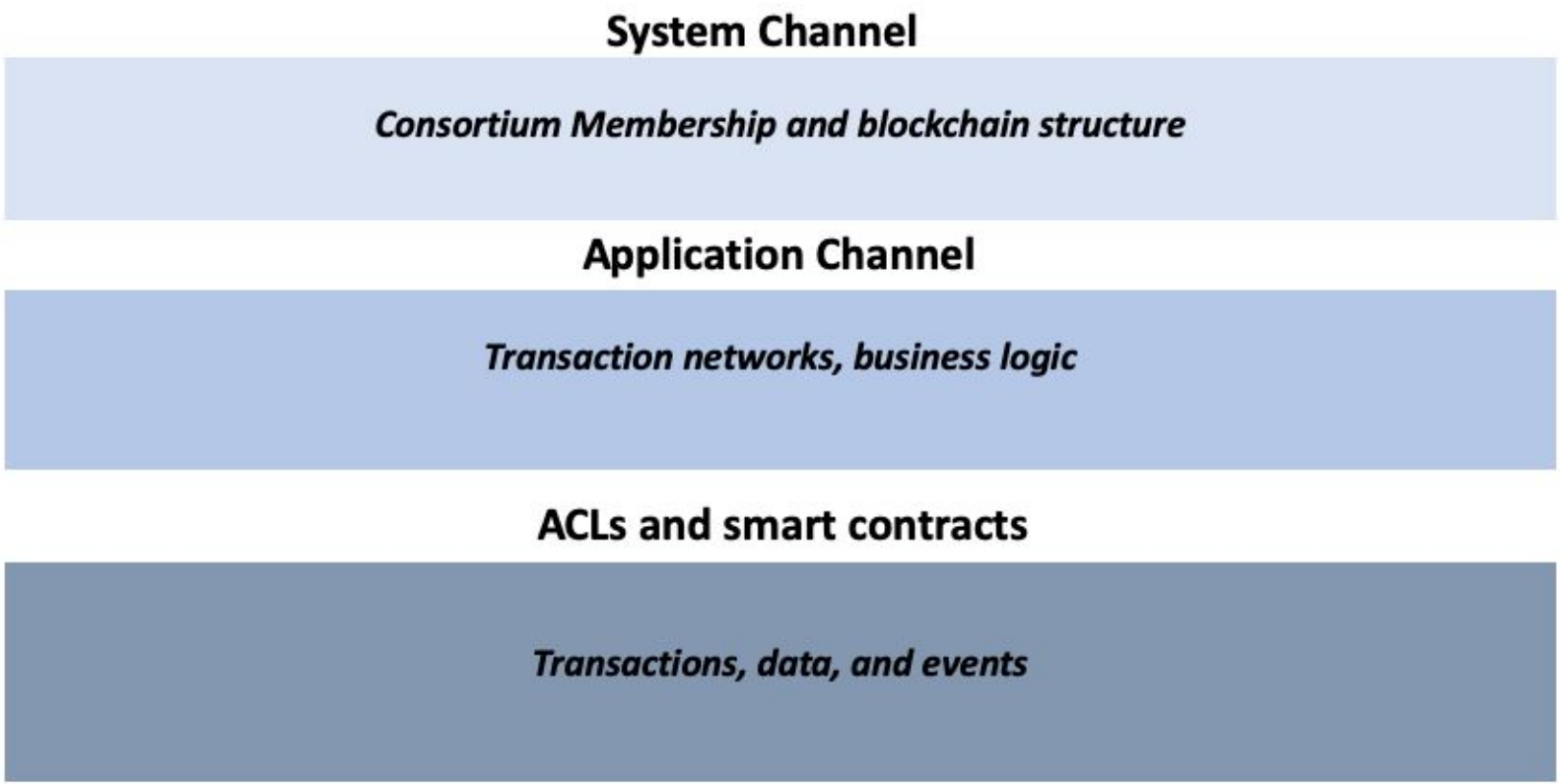
Policies allow members to decide which organizations can access or update a Fabric network, and provide the mechanism to enforce those decisions.

Policies contain the lists of organizations that have access to a given resource, such as a user or system chaincode.

They also specify how many organizations need to agree on a proposal to update a resource, such as a channel or smart contracts.

Once they are written, policies evaluate the collection of signatures attached to transactions and proposals and validate if the signatures fulfill the governance agreed to by the network.

Hyperledger Fabric Policy Hierarchy



System channel configuration

Every network begins with an ordering **system channel**.

There must be exactly one ordering system channel for an ordering service, and it is the first channel to be created.

The system channel also contains the organizations

The policies in the ordering system channel configuration blocks govern the consensus used by the ordering service and define how new blocks are created.

governs which members of the consortium are allowed to create new channels.

Application channel configuration

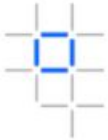
Application *channels* are used to provide a private communication mechanism between organizations in the consortium.

The policies in an application channel govern the ability to add or remove members from the channel.

Application channels also govern which organizations are required to approve a chaincode before the chaincode is defined and committed to a channel using the Fabric chaincode lifecycle.

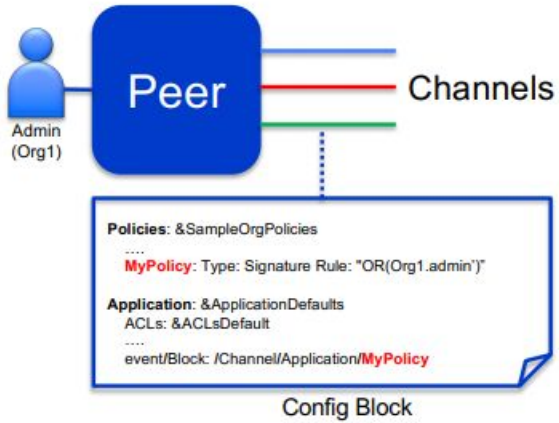
Access control lists (ACLs)

ACL mechanism per channel



Support policy based access control for peer functions per channel

- Access control defined for channel and peer resources:
  - User / System chaincode
  - Events stream
- Policies specify identities and include defaults for:
  - Readers
  - Writers
  - Admins
- Policies can be either:
  - Signature : Specific user type in org
  - ImplicitMeta : “All/Any/Majority” signature types
- Custom policies can be configured for ACLs



https://jira.hyperledger.org/browse/FAB-8727

Access control lists (ACLs)

Network administrators will be especially interested in the Fabric use of ACLs, which provide the ability to configure access to resources by associating those resources with existing policies.

ACLs refer to policies defined in an application channel configuration and extends them to control additional resources.

Fabric uses access control lists (ACLs) to manage access to resources by associating a Policy with a resource. Fabric contains a number of default ACLs

```
SampleSingleMSPChannel:

    Consortium: SampleConsortium

    Application:

        <<: *ApplicationDefaults

        ACLs:

            <<: *ACLsDefault

                event/Block: /Channel/Application/MyPolicy
```

Policies

Policies can be structured in one of two ways: as `Signature` policies or as an `ImplicitMeta` policy.

`Signature` policies

These policies identify specific users who must sign in order for a policy to be satisfied.

```
Policies:
  MyPolicy:
    Type: Signature
      Rule: "OR('Org1.peer', 'Org2.peer')"
```

> This policy construct can be interpreted as: *the policy named `MyPolicy` can only be satisfied by the signature of an identity with role of “a peer from Org1” or “a peer from Org2”*.

`ImplicitMeta` policies

`ImplicitMeta` policies aggregate the result of policies deeper in the configuration hierarchy that are ultimately defined by `Signature` policies.

`Admins` have an operational role. Policies that specify that only Admins — or some subset of Admins — have access to a resource will tend to be for sensitive or operational aspects of the network.

`Writers` will tend to be able to propose ledger updates, such as a transaction, but will not typically have administrative permissions.

`Readers` have a passive role. They can access information but do not have the permission to propose ledger updates nor do can they perform administrative tasks.

`ImplicitMeta` policies

```
Policies:

  AnotherPolicy:

    Type: ImplicitMeta

      Rule: "MAJORITY Admins"
```

Here, the policy `AnotherPolicy` can be satisfied by the `MAJORITY` of `Admins`, where `Admins` is eventually being specified by lower level `Signature` policy.

Endorsement policies

Every chaincode has an endorsement policy which specifies the set of peers on a channel that must execute chaincode and endorse the execution results in order for the transaction to be considered valid.

Setting chaincode-level endorsement policies

```
peer lifecycle chaincode approveformyorg --channelID mychannel --signature-policy "AND('Org1.member', 'Org2.member')"
--name mycc --version 1.0 --package-id mycc_1:3a8c52d70c36313cfebbaf09d8616e7a6318ababa01c7cbe40603c373bcfe173
--sequence 1 --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.co
m/msp/tlscacerts/tlsca.example.com-cert.pem --waitForEvent
```

Endorsement policies

For example:

- `AND('Org1.member', 'Org2.member', 'Org3.member')` requests one signature from each of the three principals.
- `OR('Org1.member', 'Org2.member')` requests one signature from either one of the two principals.
- `OR('Org1.member', AND('Org2.member', 'Org3.member'))` requests either one signature from a member of the `Org1` MSP or one signature from a member of the `Org2` MSP and one signature from a member of the `Org3` MSP.
- `OutOf(1, 'Org1.member', 'Org2.member')`, which resolves to the same thing as `OR('Org1.member', 'Org2.member')`.

Private data

Fabric offers the ability to create **private data collections**, which allow a defined subset of organizations on a channel the ability to endorse, commit, or query private data without having to create a separate channel.

private data collection

A collection is the combination of two elements:

The actual private data

A hash of that data

The actual private data

sent peer-to-peer via gossip protocol to only the organization(s) authorized to see it.

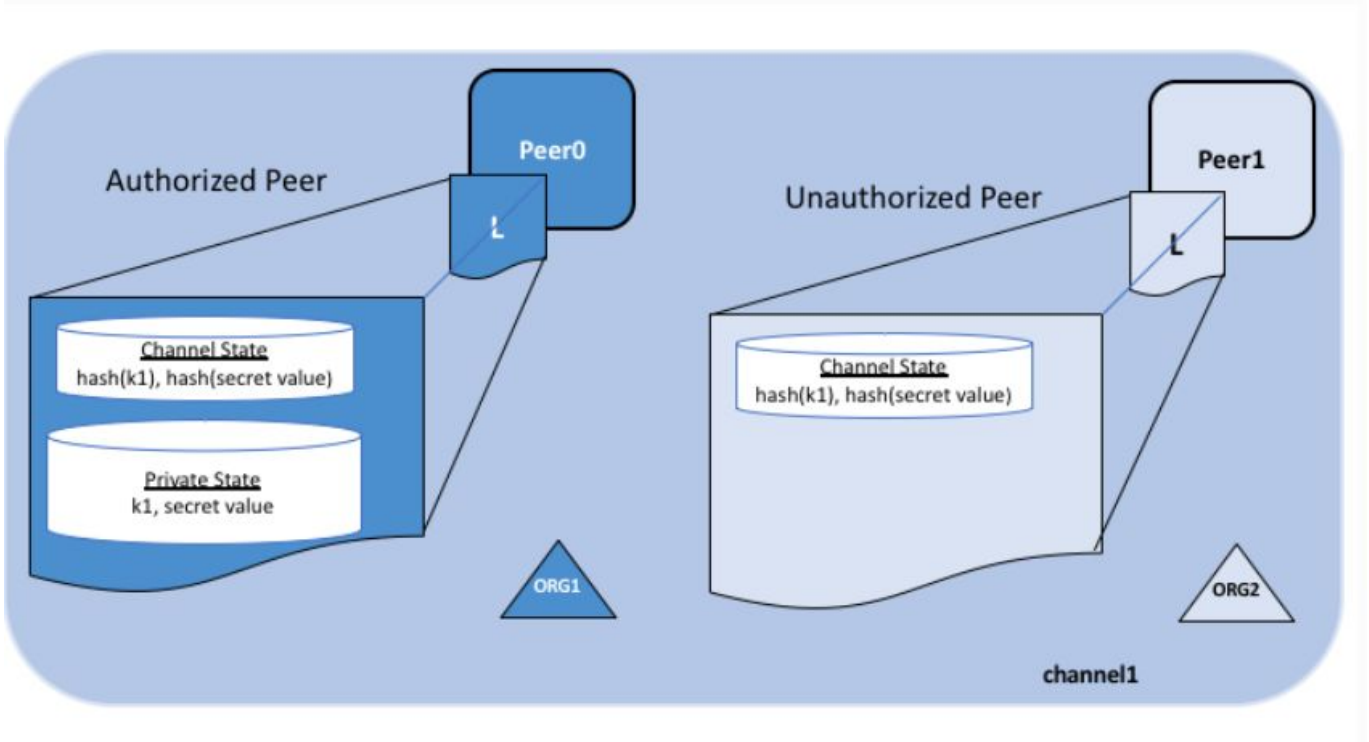
This data is stored in a private state database on the peers of authorized organizations, which can be accessed from chaincode on these authorized peers.

The ordering service is not involved here and does not see the private data. Note that because gossip distributes the private data peer-to-peer across authorized organizations, it is required to set up anchor peers on the channel, and configure CORE_PEER_GOSSIP_EXTERNALENDPOINT on each peer, in order to bootstrap cross-organization communication.

A hash of that data

which is endorsed, ordered, and written to the ledgers of every peer on the channel. The hash serves as evidence of the transaction and is used for state validation and can be used for audit purposes.

Private data



Collection members may decide to share the private data with other parties if they get into a dispute or if they want to transfer the asset to a third party.

The third party can then compute the hash of the private data and see if it matches the state on the channel ledger

In some cases, you may decide to have a set of collections each comprised of a single organization.

For example an organization may record private data in their own collection, which could later be shared with other channel members and referenced in chaincode transactions.

When to use a collection within a channel vs. a separate channel

Use **channels** when entire transactions (and ledgers) must be kept confidential within a set of organizations that are members of the channel.

Use **collections** when transactions (and ledgers) must be shared among a set of organizations, but when only a subset of those organizations should have access to some (or all) of the data within a transaction.

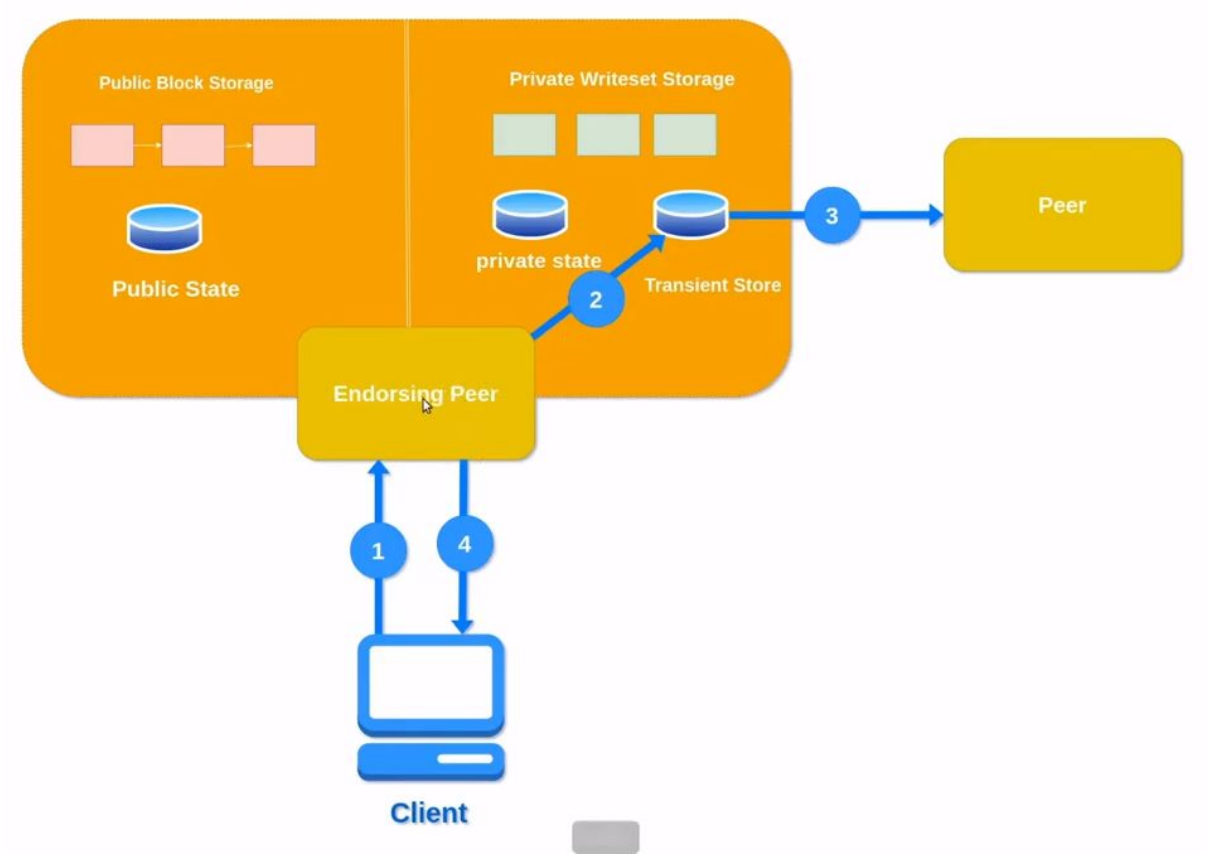
Additionally, since private data is disseminated peer-to-peer rather than via blocks, use private data collections when transaction data must be kept confidential from ordering service nodes.

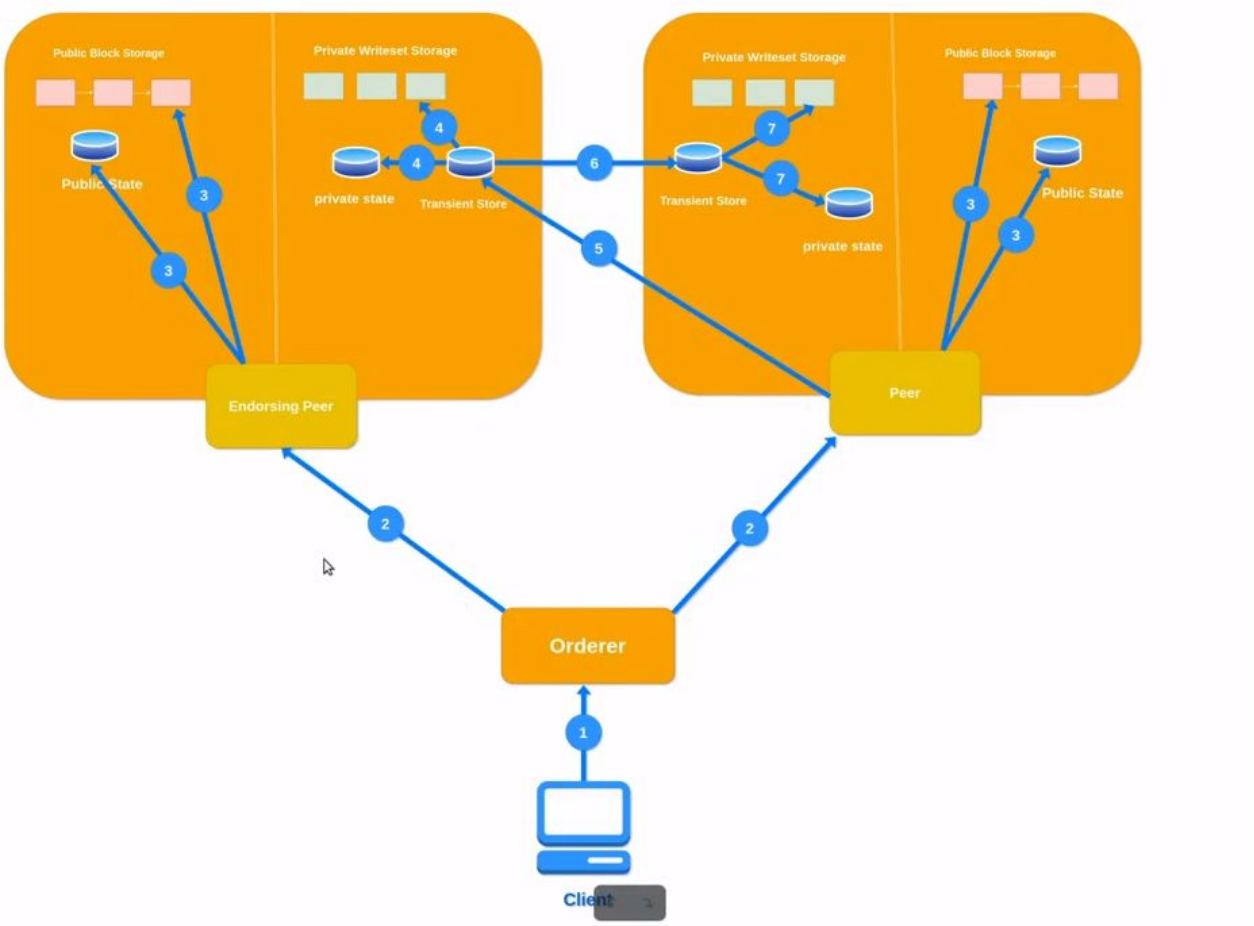
A use case to explain collections

Consider a group of five organizations on a channel who trade produce:

- **A Farmer** selling his goods abroad
- **A Distributor** moving goods abroad
- **A Shipper** moving goods between parties
- **A Wholesaler** purchasing goods from distributors
- **A Retailer** purchasing goods from shippers and wholesalers

https://hyperledger-fabric.readthedocs.io/en/release-2.2/private-data/private-data.html


Public Block Storage
Public State
Private Writeset Storage
private state
Transient Store
Peer
3
2
Endorsing Peer
1
4
Client


Public Block Storage
Private Writeset Storage
Private Writeset Storage
Public Block Storage
Public State
private state
Transient Store
Transient Store
private state
Public State
Endorsing Peer
Peer
Orderer
Client
1
2
2
3
3
3
3
4
4
5
6
7
7

- Exercise: Advanced fabcar
  - Add state change functionality - New -> Inuse-> Recycled
  - Add price details
  - Delete car
  - Audit the car lifecycle - pull history of transactions for a given car - transaction log
- Exercise: Document management system
  - Capture details in Fabric - Create Chaincode called docmanagement
  - Allow pdf document to be upload
  - Capture price details in Private data
  - Integration with Unbound key Management

DevOps

Kubernetes

Helm

ArgoCD

Install Kubectl and Minikube

minikube start

minikube status

kubectl get nodes

cd kubernetes-course/first-app/

kubectl create -f helloworld.yml

kubectl describe pod nodehelloworld.example.com

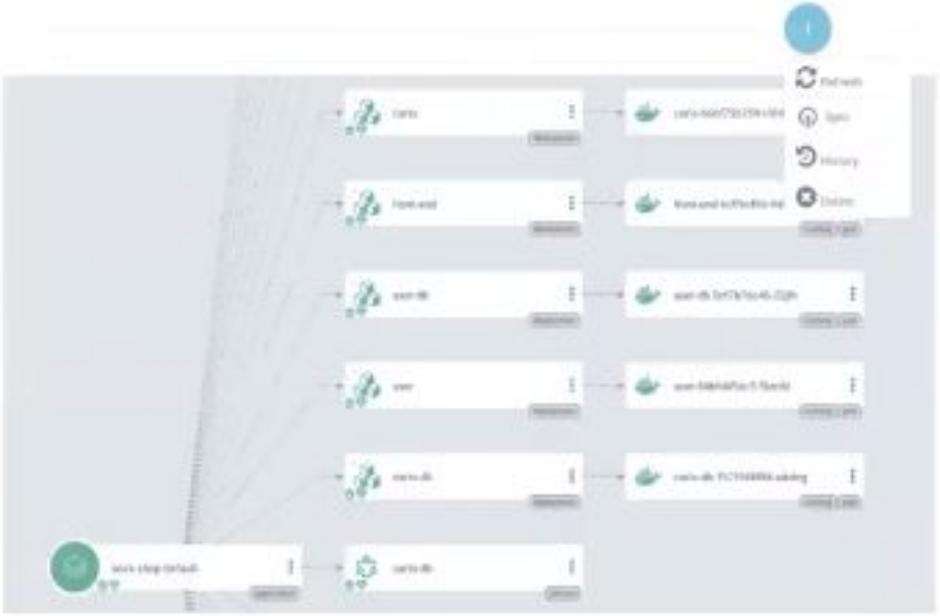
kubectl port-forward nehelloworld.example.com 8081:3000

curl localhost:8081

Argo CD - GitOps Continuous Delivery for Kubernetes

What is Argo CD?

- GitOps Kubernetes Deployment tool
- Uses git repos as the source of truth
- Declarative (**K8s YAML, ksonnet, Helm**)
- Implemented as a K8s controller and CRD
- Enterprise grade (SSO, RBAC, auditability, compliance, security)



How does Argo CD work?

2 Easy Steps

- Connect a git repo containing your application manifests
- Sync your app