Chapter 2.1: Architecture for this story

Learning Bluemix & Blockchain

Bob Dill, IBM Distinguished Engineer, CTO Global Technical Sales
David Smits, Senior Certified Architect, IBM Global Business Services

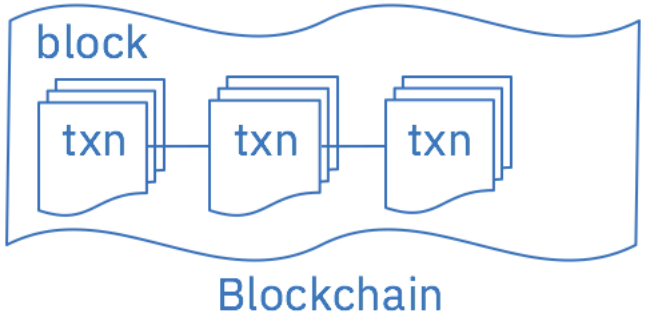
The Plan: 30 minute Chapters with an hour or two of practice

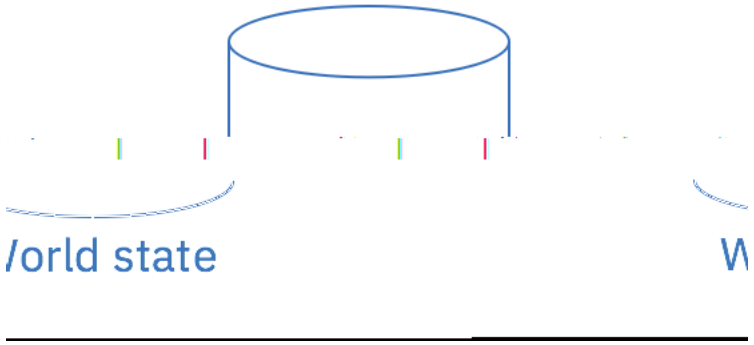
Chapter 1:	What is Blockchain? Concept and Architecture overview
Chapter 2:	What's the story we're going to build
Chapter 2.1:	**A chi ec e f he S**
Chapter 3:	Set up local HyperLedger Fabric V1 development environment
Chapter 4:	Build and test the network
Chapter 5:	Administration User Experience
Chapter 6:	Buyer Support and User Experience
Chapter 7:	Seller Support and User Experience
Chapter 8:	Shipper Support and User Experience
Chapter 9:	Provider Support and User Experience
Chapter 10:	Finance Company Support and User Experience
Chapter 11:	Combining for Demonstration
Chapter 12:	Events and Automating for Demonstration

Components in a blockchain solution

Ledger		A ledger is a channel's chain and current state data which is maintained by each peer on the channel.
Smart Contract	f(abc);	Software running on a ledger, to encode assets and the transaction instructions (business logic) for modifying the assets.
Peer	Network	A broader term overarching the entire transactional flow, which serves to generate an agreement on the order and to confirm the correctness of the set of transactions constituting a block.
permissioned	Membership	Membership Services authenticates, authorizes, and manages identities on a blockchain network.
lock), as well as	Events	Creates notifications of significant operations on the blockchain (e.g. a new b notifications related to smart contracts.
	Systems Management	Provides the ability to create, change and monitor blockchain components
	Wallet	Securely manages a user's security credentials
. Not part of	Systems Integration	Responsible for integrating Blockchain bi-directionally with external systems blockchain, but used with it.

A ledger often consists of two data structures

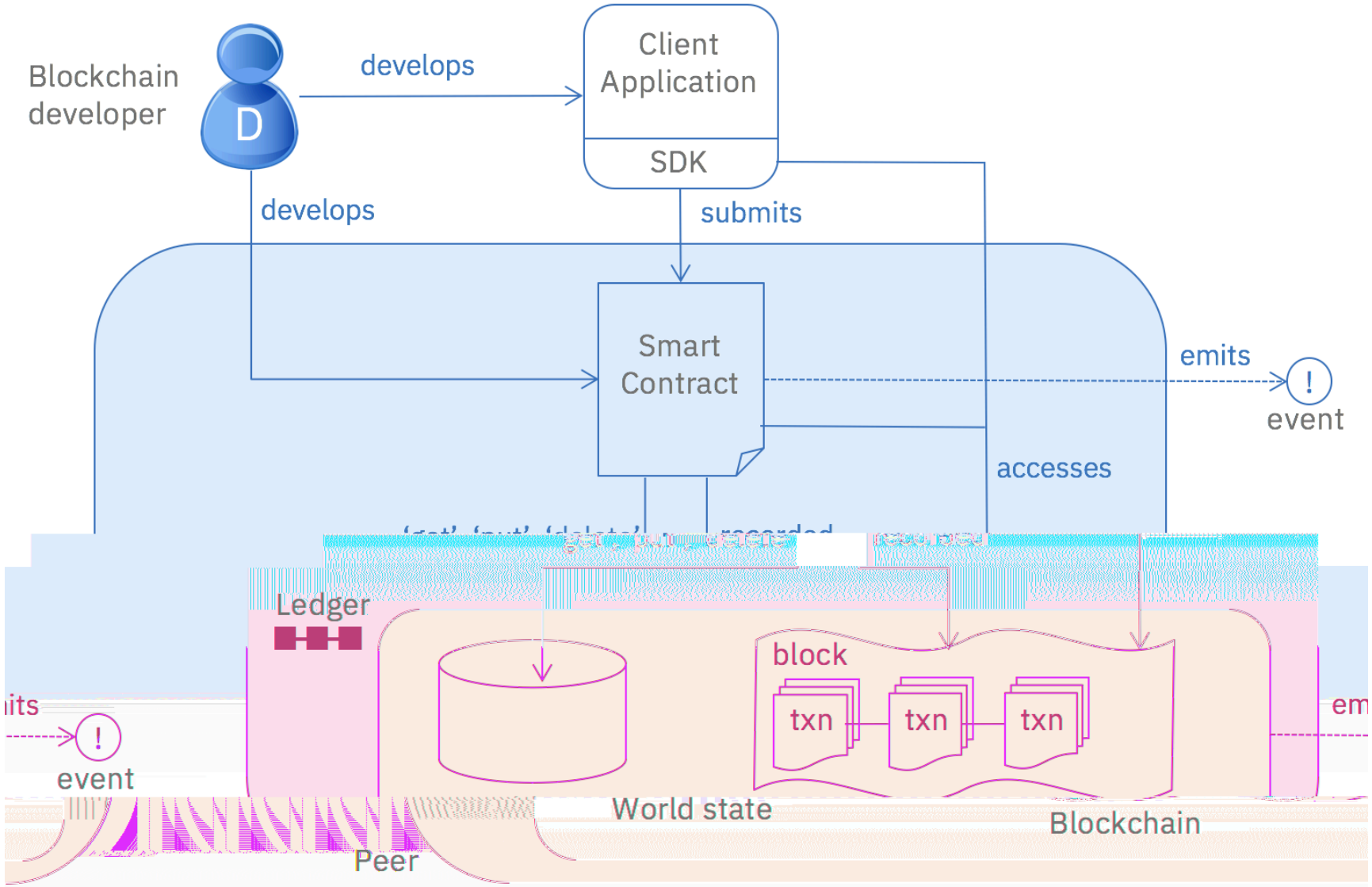




- Blockchain
  - A linked list of blocks
  - Each block describes a set of transactions (e.g. the inputs to a smart contract invocation)
  - Immutable - blocks cannot be tampered

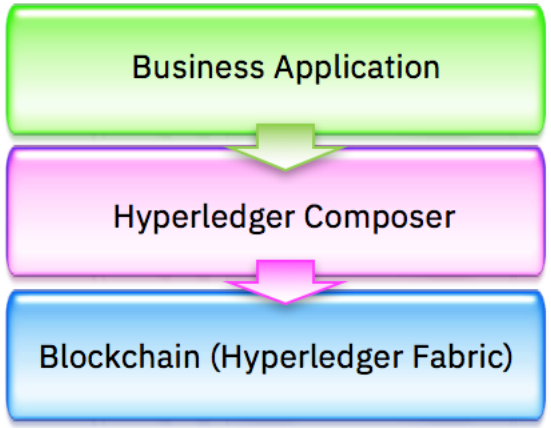
  - World State
    - An ordinary database (e.g. key/value store)
    - Stores the combined outputs of all transactions
    - Not usually immutable

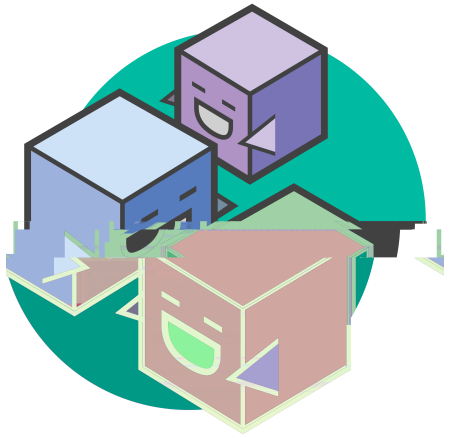
How applications interact (via the Fabric SDK)



How application interact (with Hyperledger Composer)

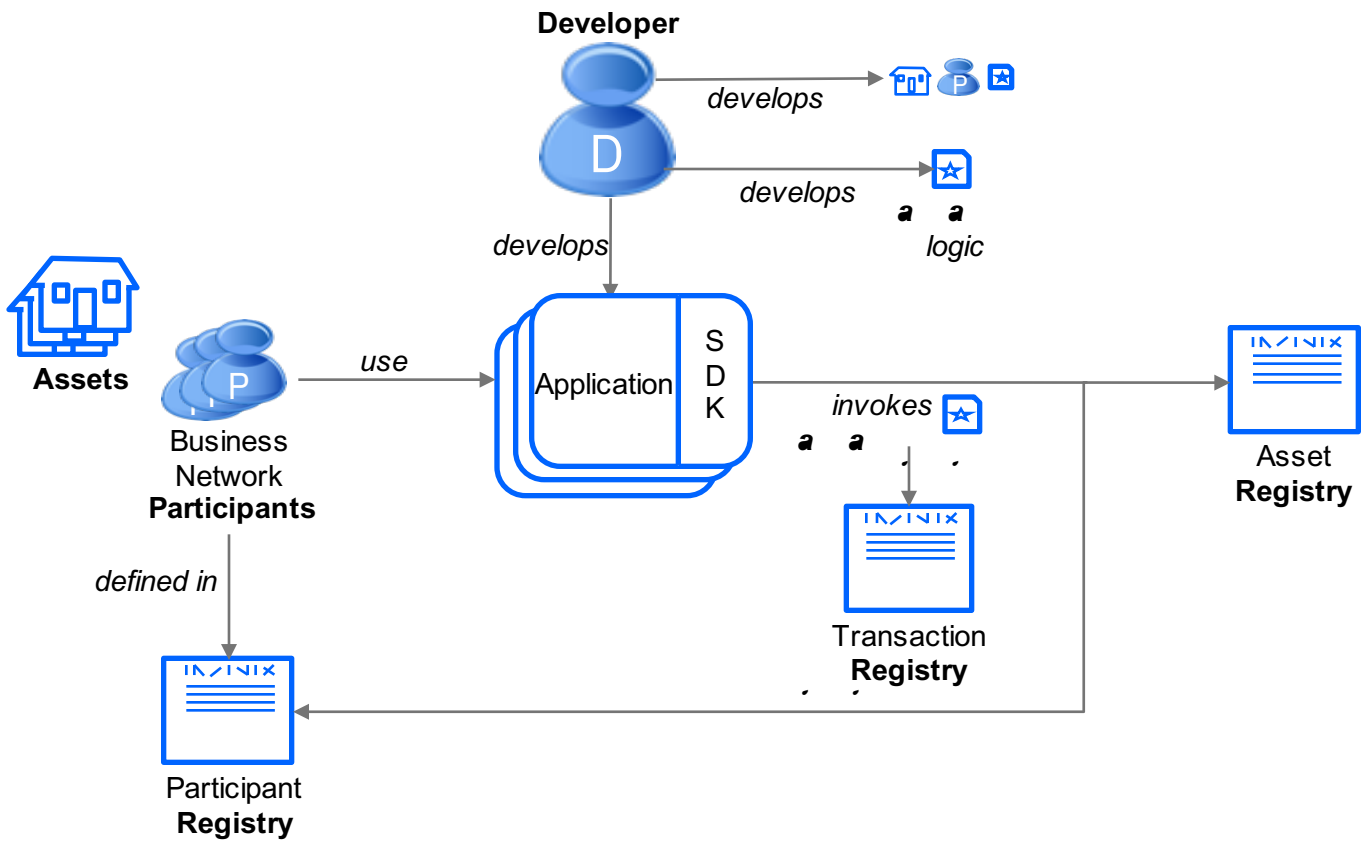
- A suite of bigh level application **ab ac i n** for business networks
- Emphasis on business-centric vocabulary for quick solution creation
- Reduce risk, and increases understanding and flexibility



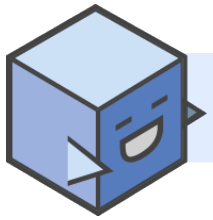


- Features
  - Model your business networks, test and expose via APIs
  - Applications invoke APIs transactions to interact with business network
  - Integrate existing systems of record using loopback/REST
- Fully open and part of Linux Foundation Hyperledger

Key Concepts for the Application Developer

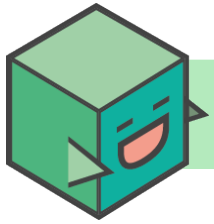


Developer Concepts for Hyperledger Composer



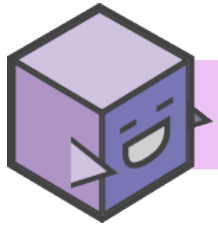
Applications

- Provides front-end for the user
  - May require different applications per participant
- Interacts with the registries
  - Add, delete, update, query
  - Registries persisted on blockchain
  - Connects to blockchain via Javascript client libraries (SDK) or REST



Models

- A domain specific language (.CTO) that defines the type structure of
  - Assets
  - Participants
  - Transactions
- Aims to match how we talk about business networks in the real world



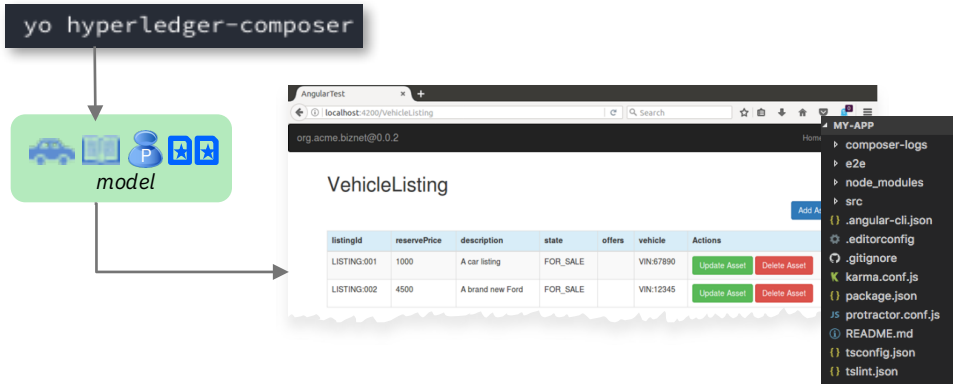
Scripts

- Scripts provide the implementation of transaction processor logic
- Specified in Javascript
- Designed for any reasonable Javascript developer to pick up easily

Creating the Application

- JavaScript applications require() the NPM "composer-client" module
  - This provides the API to access assets, participants and transactions
- RESTful API (via Loopback) also available
- Command-line tool available to generate skeleton command-line or Angular2 application from model
- Also helps with the generation of unit tests to help ensure quality code

```
const BusinessNetworkConnection = require('composer-client')
                                    .BusinessNetworkConnection;
this.bizNetworkConnection = new BusinessNetworkConnection();
this.titlesRegistry = this.bizNetworkConnection.getAssetRegistry
                    ('net.biz.digitalPropertyNetwork.LandTitle')
let landTitle1 = factory.newResource
    ('net.biz.digitalPropertyNetwork', 'LandTitle', 'LID:1148');
landTitle1.owner = ownerRelation;
landTitle1.information = 'A nice house in the country';
this.titlesRegistry.add(landTitle1);
```



Events and Queries

- Events allow applications to take action when a transaction occurs
  - Events are **defined** in models
  - Events are **emitted** by scripts
  - Events are **caught** by applications
- Caught events include transaction ID and other relevant information

```
event SampleEvent {
  --> SampleAsset asset
  o String oldValue
  o String newValue
}
```

```
// Emit an event for the modified asset.
var event = getFactory().newEvent('org.acme.sample', 'SampleEvent');
event.asset = tx.asset;
event.oldValue = oldValue;
event.newValue = tx.newValue;
emit(event);
```

```
businessNetworkConnection.on('SampleEvent', (event) => {
    console.log(event);
}
```

- Queries allow applications to perform complex registry searches
  - They can be statically defined in a separate .qry file or generated dynamically by the application
  - They are invoked in the application using buildQuery() or query()
  - Queries require the blockchain to be backed by CouchDB

```
query selectCommoditiesWithHighQuantity {
  description: "Select commodities
  statement:
    SELECT org.acme.trading.Commodity
      WHERE (quantity > 60)
}
```

```
return query('selectCommoditiesWithHighQuantity', {})
```

Access Control

```
rule EverybodyCanReadEverything {
    description: "Allow all participants read access to all resources"
    participant: "org.acme.sample.SampleParticipant"
    operation: READ
    resource: "org.acme.sample.*"
    action: ALLOW
}
```

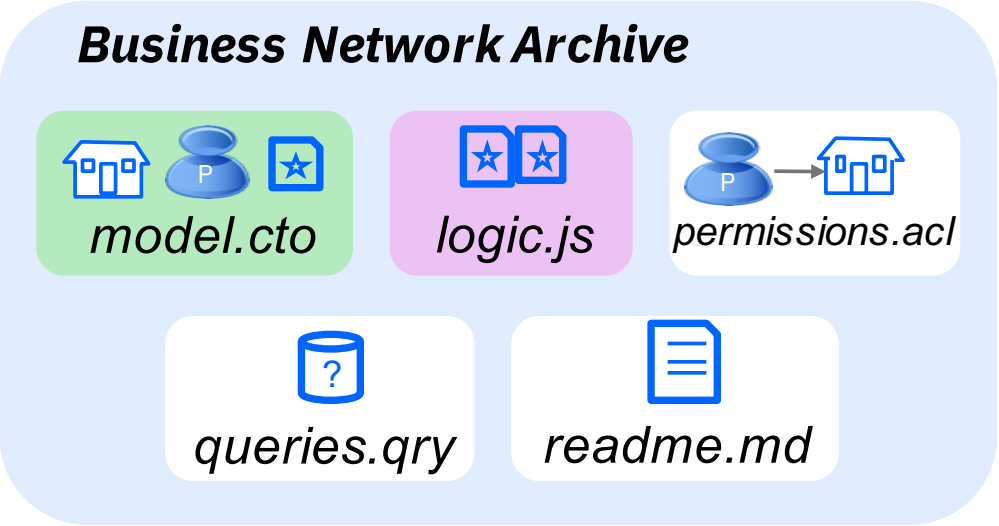
```
rule OwnerHasFullAccessToTheirAssets {
    description: "Allow all participants full access to their assets"
    participant(p): "org.acme.sample.SampleParticipant"
    operation: ALL
    resource(r): "org.acme.sample.SampleAsset"
    condition: (r.owner.getIdentifier() === p.getIdentifier())
    action: ALLOW
}
```

```
rule SystemACL {
  description:  "System ACL to permit all access"
  participant: "org.hyperledger.composer.system.Participant"
  operation: ALL
  resource: "org.hyperledger.composer.system.**"
  action: ALLOW
}
```

- It is possible to restrict which resources can be read and modified by which participants
  - Rules are defined in an .acl file and deployed with the rest of the model
  - Transaction processors can also look up the current user and implement rules programmatically
- ACL rules can be simple (e.g. everybody can read all resources) or more complex (e.g. only the owner of an asset can do everything to it)
- Application supplies credentials (userid/secret) of the participant when connecting to the Fabric network
  - Remember to grant System ACL all access if necessary

Resources are packaged into BNA files

- Business Network Archive (.BNA) is a package of the resources used by Fabric:
  - Model files (.CTO)
  - Transaction processors (.JS)
  - Access Control Lists (.ACL)
  - Static queries (.QRY)
  - Documentation and versioning (.MD)
- It does not contain the client application

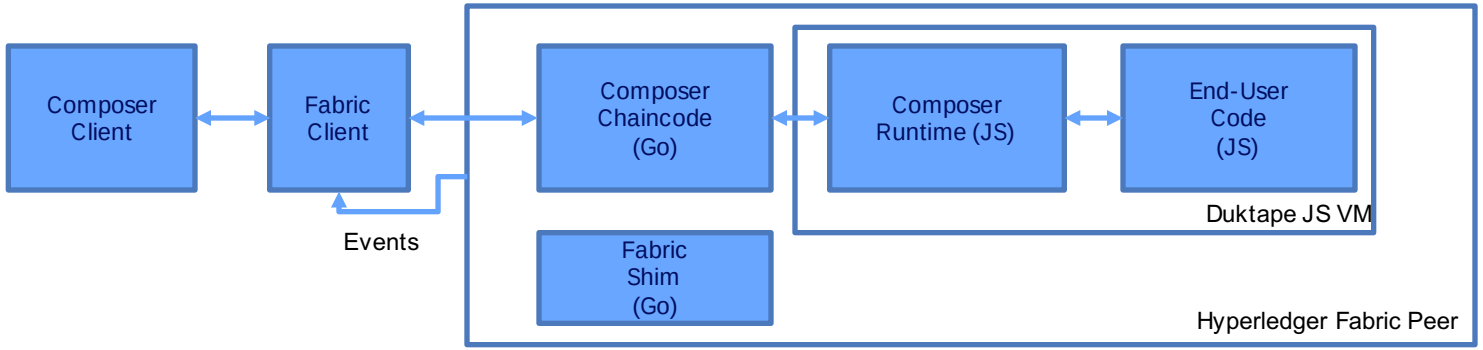


- The BNA simplifies deployment of blockchain and promotion between environments
  - c.f. TAR, WAR, EAR, JAR, BAR…

```
composer archive create –archiveFile my.bna
      --sourceType module --sourceName myNetwork
```

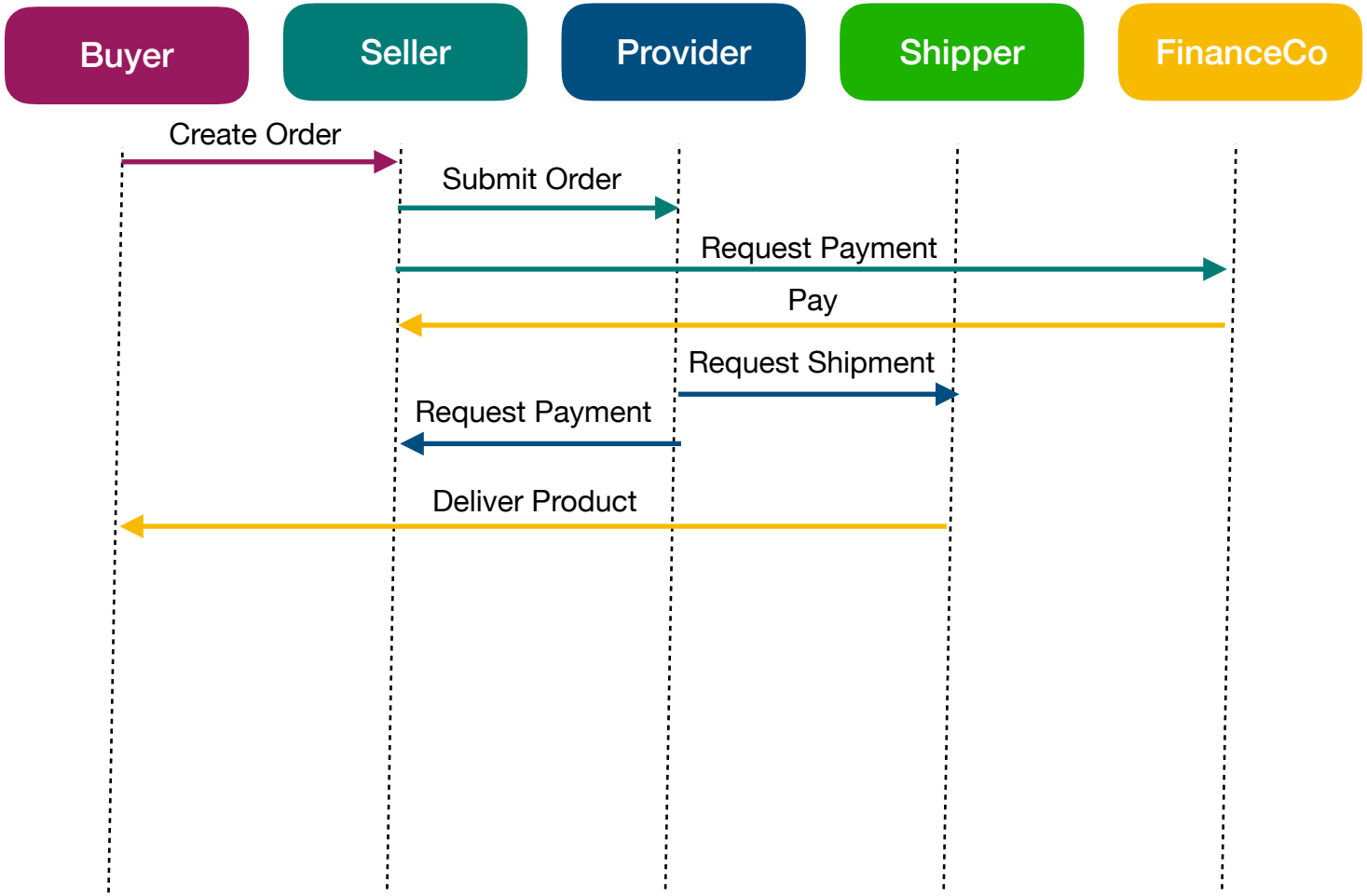
- Create BNA files from Playground or command line
- Build from filesystem or NPM module

How Composer Maps to Fabric Chaincode

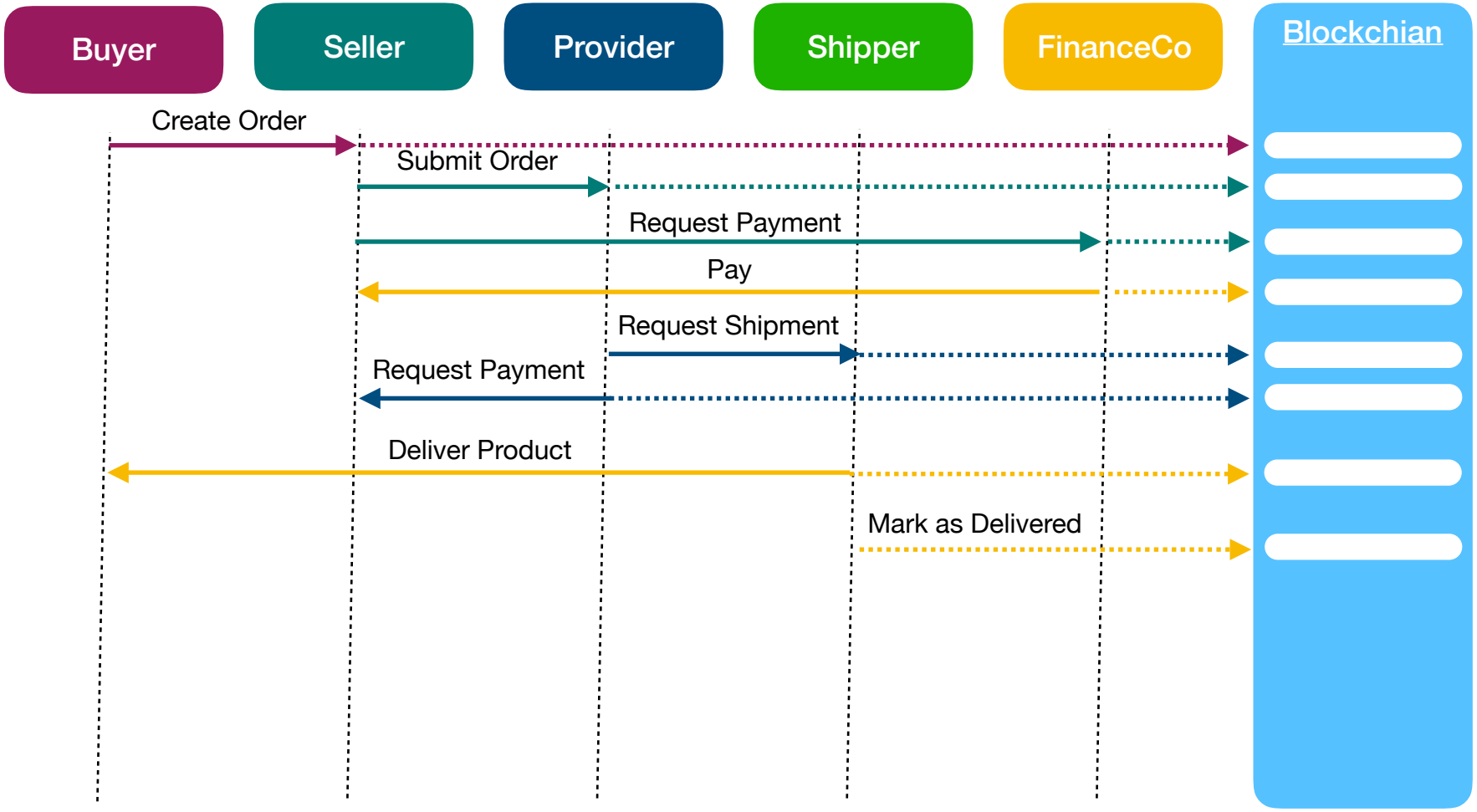


- Each Business Network is deployed to its own chaincode container
  - Container contains a static piece of Go chaincode that starts a Javascript virtual machine running transaction processors
- Browse these containers to view diagnostics information (docker logs)
- Embedded chaincode is not a Composer external interface

Credit, Transaction and Invoice Flows

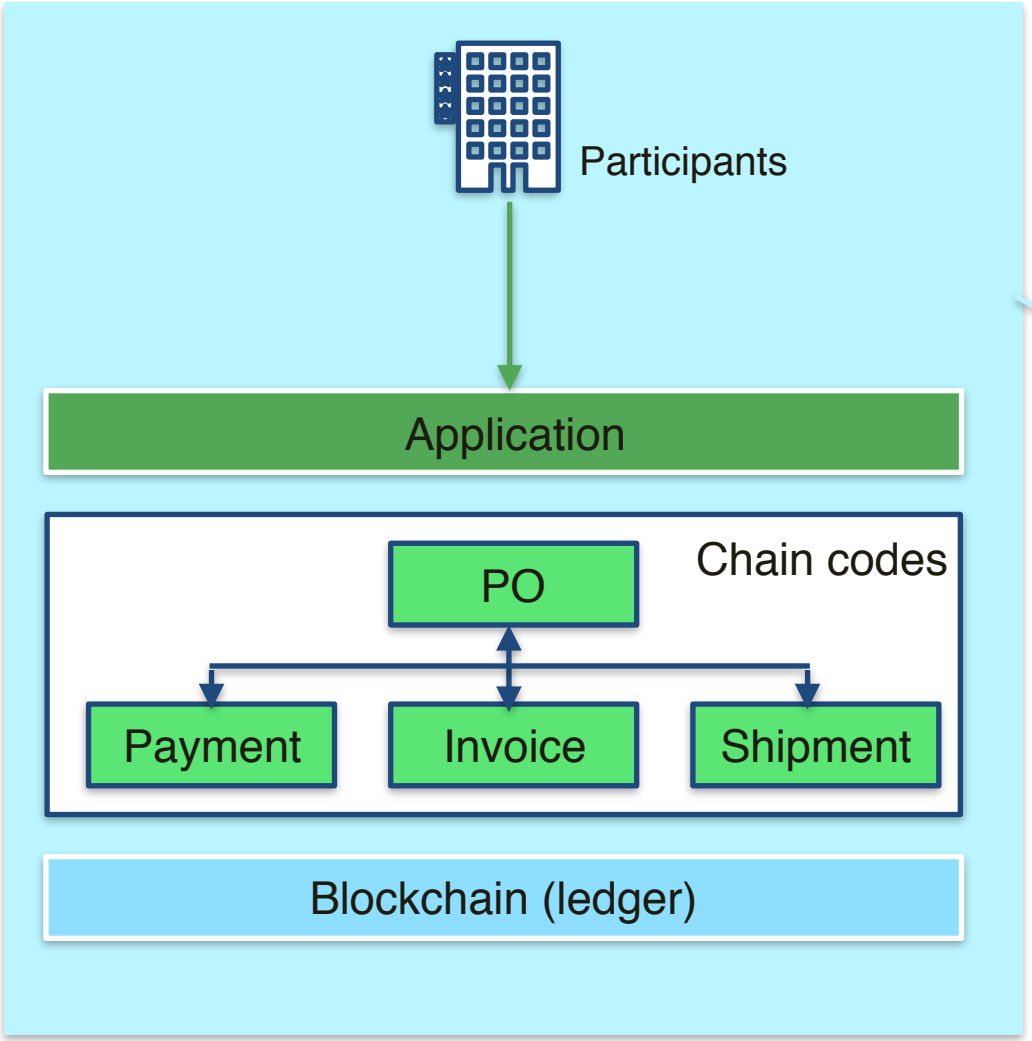


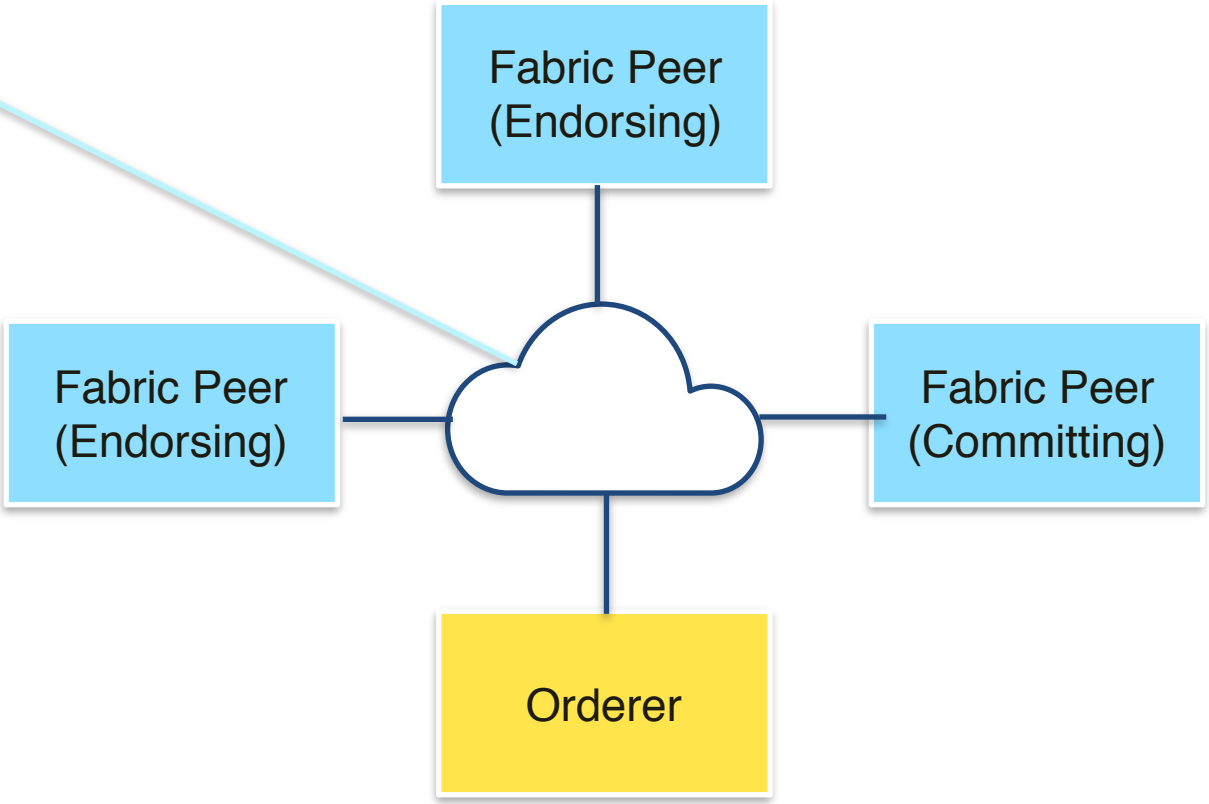
Credit, Transaction and Invoice Flows



Data Model Entity Access

Architecture Overview





The Plan: 30 minute Chapters with an hour or two of practice

Chapter 1:	What is Blockchain? Concept and Architecture overview
Chapter 2:	What's the story we're going to build
Chapter 2.1:	Architecture for the Story
Chapter 3:	Set up local HyperLedger Fabric V1 development environment
Chapter 4:	Build and test the network
Chapter 5:	Administration User Experience
Chapter 6:	Buyer Support and User Experience
Chapter 7:	Seller Support and User Experience
Chapter 8:	Shipper Support and User Experience
Chapter 9:	Provider Support and User Experience
Chapter 10:	Finance Company Support and User Experience
Chapter 11:	Combining for Demonstration
Chapter 12:	Events and Automating for Demonstration

Chapter 3: Set up local HyperLedger V1 development environment

Learning Bluemix & Blockchain

Bob Dill, IBM Distinguished Engineer, CTO Global Technical Sales
David Smits, Senior Certified Architect, IBM Global Business Services