Blockchain Unchained

A guide to developing smart contracts

IBM **Blockchain**

Blockchain education series

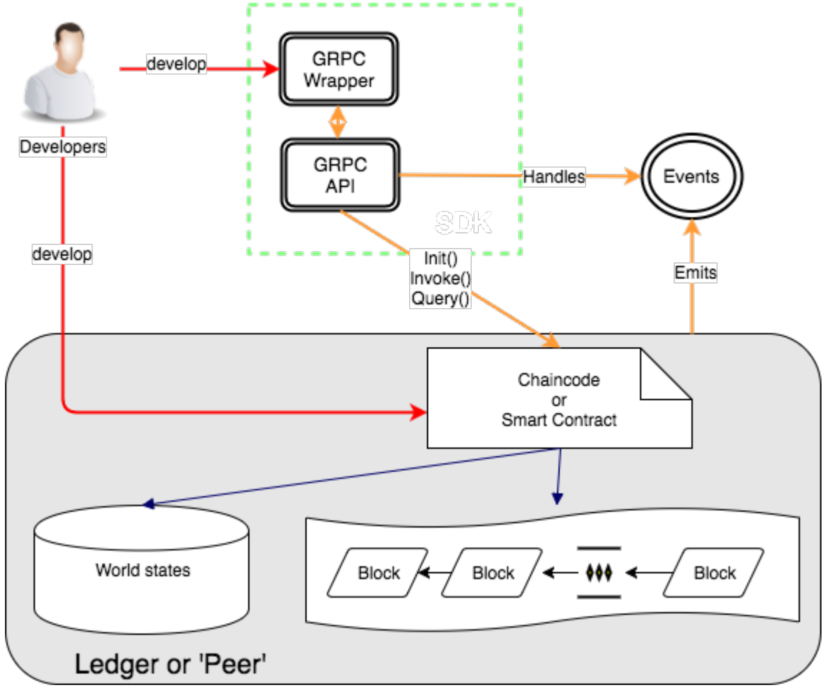
V5.0, 4 October 2017

Chaincode development approaches 1/2

Developer writes low level codes using the GRPC API calls to interact with the ledger

Developer would develop or use example chaincode to test SDK

Developer can use docker for testing



Chaincode development approaches 2/2

Developer builds UI and Business Logic

Integrate with the SDK

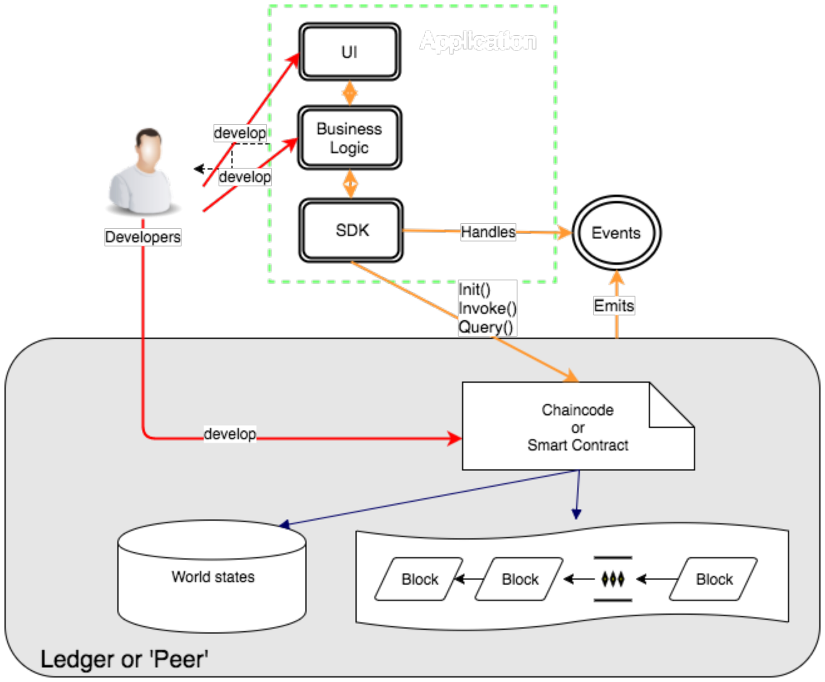
Use Cases for this scenario:

– Support specific language bindings (such as Swift, C++, Fortan, etc)

– Integration with legacy system

– Plugins for toolchains

– Limited platform



Write a first chaincode

- Asset Management
  - ➢ Create Asset (invoke, createCar)
  - ➢ Change Asset Owner (invoke, initLedger, changeCarOwner)
  - ➢ Query for Asset (query, queryAllCars/queryCar)
- Example based on fabcar.go

Write a first chaincode

Basic chaincode structure

– *Import statements*

– *Functions (Main, Init, Invoke, Query)*

– *Data modelling*

Write a first chaincode

Basic chaincode structure

- ***Import statements***
- *Functions (Main, Init, Invoke, Query)*
- *Data modelling*

Write a first chaincode

Import Statement

The import statement lists a few dependencies that you will need for your chaincode to build successfully.

- **fmt -** contains Println for debugging/logging.
- **errors** - standard go error format.
- **encoding/json -** Package strings implements simple functions to manipulate UTF-8 encoded strings and import JSON packages
- **strconv -** Package strconv implements conversions to and from string representations of basic data types.
- **strings -** Package strings implements simple functions to manipulate UTF-8 encoded strings.

The **shim package** provides APIs that let your chaincode interact with the underlying blockchain network to access state variables, transaction context, caller certificates and attributes, and to invoke other chaincodes, among other operations.

Structs are typed collections of fields. They're useful for grouping data together to form records.

Write a first chaincode

Basic chaincode structure

– *Import statements*

– ***Functions (Main, Init, Invoke, Query)***

– *Data modelling*

Write a first chaincode

Main() Function

– The starting point for any Go program

– used for bootstrapping/starting the chaincode

– sets up the communication between this chaincode and the peer that deployed it.

– The SmartContract is the struct that is required to implement the shim.Chaincode interface, which has three methods — Init, Query, and Invoke — for it to be considered a valid Chaincode type by the shim package.

Write a first chaincode

Basic chaincode structure

– *Import statements*

– ***Functions (Main, Init, Invoke, Query)***

– *Data modelling*

Write a first chaincode

Init() Function

– The Init method is called when the chaincode is first deployed onto the blockchain network

– will be executed by each peer that deploys its own instance of the chaincode.

Write a first chaincode

Basic chaincode structure

– *Import statements*

– ***Functions (Main, Init, Invoke, Query)***

– *Data modelling*

Write a first chaincode

Invoke() Function

– The Invoke method is invoked whenever the state of the blockchain is to be modified.

– All invocations of this method are recorded on the blockchain as transactions, which ultimately get written into blocks.

– The structure of Invoke - a function and an array of arguments.

Write a first chaincode

Basic chaincode structure

– *Import statements*

– ***Functions (Main, Init, Invoke, Query)***

– *Data modelling*

Write a first chaincode

Query() Function

- The Query method is invoked whenever any read/get/query operation needs to be performed on the blockchain state.
- The Query method is not intended to change the state of the underlying blockchain, and hence does not run within a transactional context.
- its invocations are not recorded on the blockchain.

Write a first chaincode

Basic chaincode structure

- *Import statements*
- *Functions (Main, Init, Invoke, Query)*
- ***Data modelling***

Write a first chaincode

Data Modelling

– World state is stored in a key value store and takes in a byte array as the value, which can be used to store a serialized JSON structure.

– data models would eventually need to be converted into a JSON string.

– The annotation for each field, for example, json:"assetname" acts like metadata for the marshal/unmarshal API, which will use these annotations to map each field with its corresponding json string equivalent representation.

Write a first chaincode

Functions explained

- *Query-like : function queryCar()*
  - *Defining args*
  - *Using stub.getstate*

- *Invoke-like : function initLedger(), createCar(), changeCarOwner*
  - *Defining args*
  - *Using stub.putstate*
  - *Marshalling/Unmarshalling JSON data*