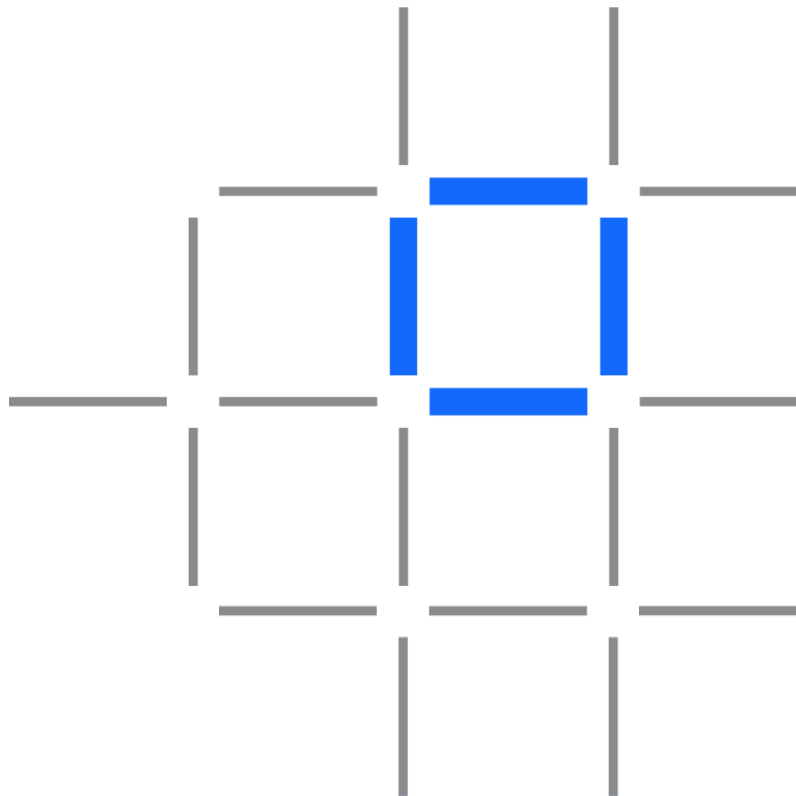
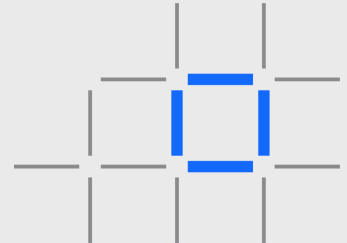
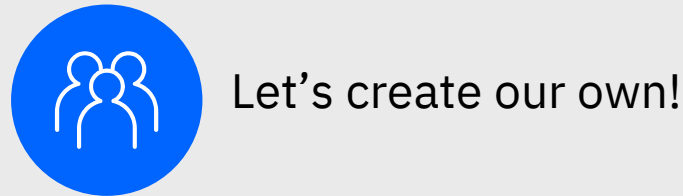
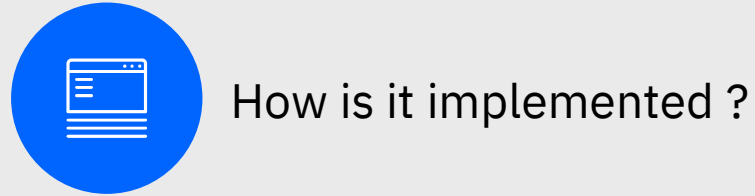
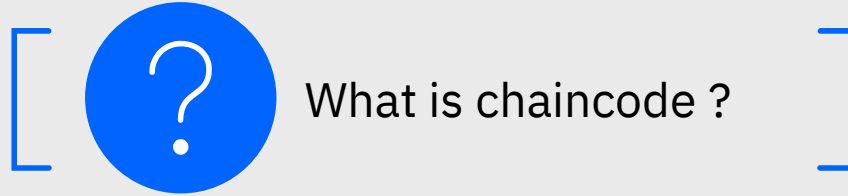


Blockchain Developed

An Introduction to chaincode development

IBM **Blockchain**





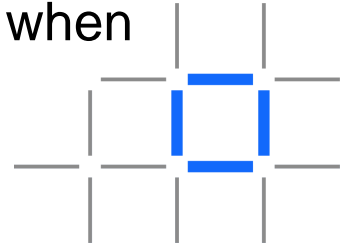
Hyperledger Fabric: Ledger

Ledger:

- A sequence of cryptographically-linked blocks, containing transactions and the current world-state. In addition to data from previous transactions, the ledger also contains the data for currently-running chaincode applications.

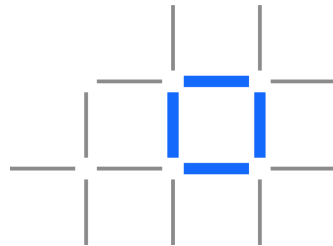
World-state:

- Key-value database used by chaincodes to store their state when executed by a transaction.



Embedded logic that encodes the rules for specific types of network transactions. Developers write chaincode applications and deploy them to the network. End users then invoke chaincode through a client-side application that interfaces with a network peer, or node. Chaincode runs network transactions, which if validated, are appended to the shared ledger and modify world state.

- Consist of **code** and **data**.
- Chaincode is the term used to describe a smart contract in Hyperledger
- Chaincode can currently be written in Go.
- Chaincode **must** be deterministic!



Chaincode: deterministic

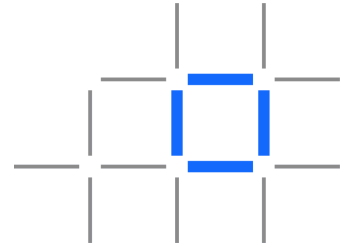
Chaincode **must** be deterministic.

Chaincode is run on each peer within the blockchain network, and when invoked on each peer for the same transaction, they must write the same values to the world-state.

Things to watch for are:

- Date/Time stamps
- Random values

Chaincode must not call external systems.



Chaincode : programming language

As the time of writing, Golang is the only option for chaincode development.

Hyperledger Fabric 1.1 (to be released) is expected to include Javascript chaincode support.

Java chaincode support was removed after 1.0.0-alpha2 for lack of testing. It will eventually come back.

[FAB-3850] disable java chaincode as its WIP [Browse files](#)

Java CC has made good progress with <https://jira.hyperledger.org/browse/FAB-3218>. However its not quite ready yet. Its better to disable it for alpha2 and enable it when its more usable.

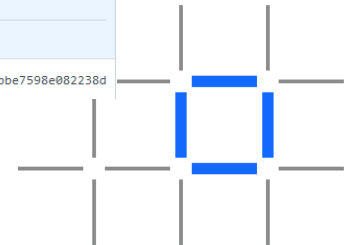
The disable will add endorser and CLI checks to error out if accessing java chaincode. We may just prevent install/instantiate/upgrade of Java CC to make it less invasive (ie, not checking invoke as we cannot invoke what's not instantiated).

Change-Id: I6109833cc9276c5d7f0679b9987e43677125778a
Signed-off-by: Srinivasan Muralidharan <muralisr@us.ibm.com>

master (#15) v1.0.0-rc1 v1.0.0-alpha2

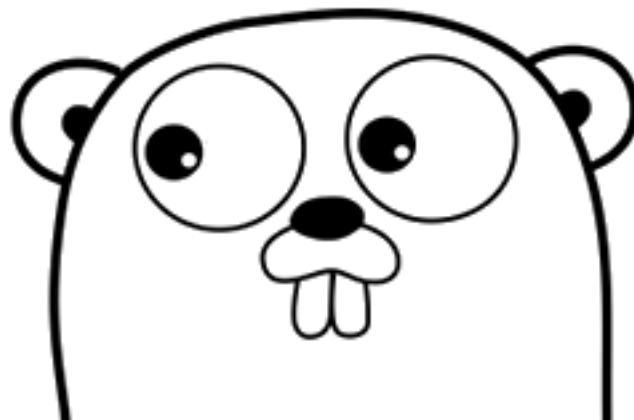
muralisrini committed on May 11

1 parent f3bb8b7 commit 29e0c4083f1f95d4f3513ecd4b7598e082238d

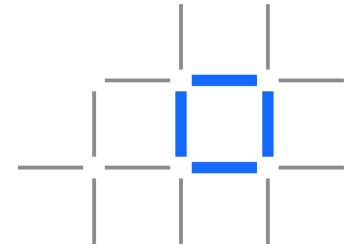


Go Overview

- Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.
- Hyperledger Fabric is built using Go.
- Available for Windows, Linux and Mac
- Download from <https://golang.org/dl/>
- Some quotes about Go:
 - “Simple, minimal syntax”
 - “Garbage collection built-in”
 - “A flexible interface system”
 - “Easy concurrency support via goroutines”
 - “Fast compilation times”
 - “Simple compile build/run procedures”
 - “Statically linked binaries that are simple to deploy”
 - “Fun to write, fast to run”
 - “Go is familiar to most developers making on-boarding easier.”
 - “Go is, in essence, a perfected version of C”



IBM Blockchain

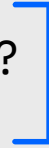




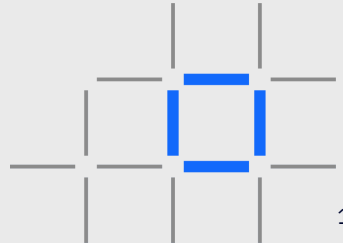
What is chaincode ?



How is it implemented ?



Let's create our own!



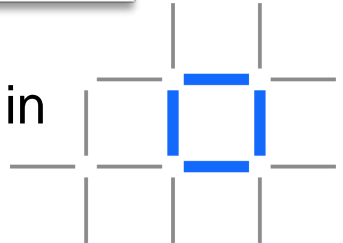
Chaincode interface

- Each chaincode must implement the chaincode interface
 - Interface is defined in [core/haincode/shim/interfaces.go](#) (from line 28 onwards)
 - Chaincode methods are called in response to received transactions.

- For example:

```
import (  
    "errors"  
    "fmt"  
    "strconv"  
  
    "github.com/hyperledger/fabric/core/chaincode/shim"  
)
```

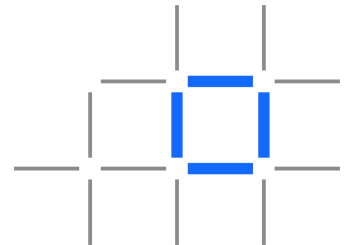
- Other methods to access and modify the ledger are defined in [ChaincodeStubInterface](#)



Chaincode functions

Fabric chaincode consists of the 2 main functions:

- **Init()** – Called when the chaincode is deployed. Can be used to initialize the ledger.
- **Invoke()** – Transaction added to the blockchain and creates/updates/deletes information in the worldstate



The simplest possible chaincode would look like this:

```
package main

import (
    "fmt"
    "github.com/hyperledger/fabric/core/chaincode/shim"
    pb "github.com/hyperledger/fabric/protos/peer"
)

type HelloWorld struct {
}

func (t *HelloWorld) Init(stub shim.ChaincodeStubInterface) pb.Response {
    return shim.Success(nil)
}

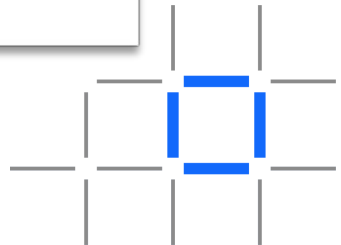
func (t *HelloWorld) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
    return shim.Success(nil)
}

func main() {
    err:= shim.Start(new(HelloWorld))
    if err!= nil{
        fmt.Printf("Error starting HelloWorld chaincode: %s", err)
    }
}
```

Chaincode : Error handling

Many functions return an error as the 2nd parameter.

```
Bvalbytes, err := stub.GetState(B)
if err != nil {
    return nil, errors.New("Failed to get state")
}
if Bvalbytes == nil {
    return nil, errors.New("Entity not found")
}
Bval, _ = strconv.Atoi(string(Bvalbytes))
```



World state : Update

Updating and querying the world-state is done simply within the chaincode.

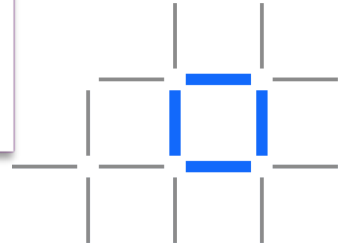
To update the world-state:

```
err := stub.PutState(key, value)
```

where key is a string, and value is an array of bytes

Example:

```
// Write the state back to the ledger
err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
if err != nil {
    return nil, err
}
```



World state: Query

Updating and querying the world-state is done simply within the chaincode.

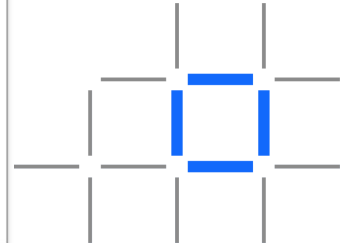
To query the world-state:

```
result, error := stub.GetState(key)
```

where key is a string, and value is an array of bytes

Example:

```
Bvalbytes, err := stub.GetState(B)
if err != nil {
    return nil, errors.New("Failed to get state")
}
if Bvalbytes == nil {
    return nil, errors.New("Entity not found")
}
Bval, _ = strconv.Atoi(string(Bvalbytes))
```

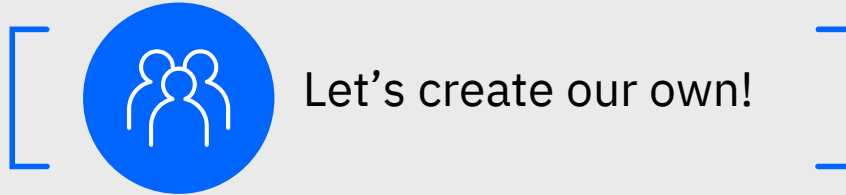




What is chaincode ?



How is it implemented ?



Let's create our own!

