Private blockchain development with

Hyperledger Fabric

by Blockchain team, KBTG

We are

KBank

KBTG
KASIKORN
BUSINESS-TECHNOLOGY GROUP

We use

node.js
docker

HYPERLEDGER
FABRIC

Agenda

9:00 - 12:00

Overall Hyperledger things (1.5 h)

- Hyperledger Overview
- Hyperledger Fabric

Break 15 mins

Environment Setup (1 h)

- Installing the prerequisites

13:00 - 16:00

HLF deployment (1.5 h)

- Demo
- Setup Network & Deploy Chaincode

Break 15 mins

Developer deep diving (1.5 h)

- Chaincode Development
- API Development

Our dream team for DPU X

KBTG KASIKORN BUSINESS-TECHNOLOGY GROUP

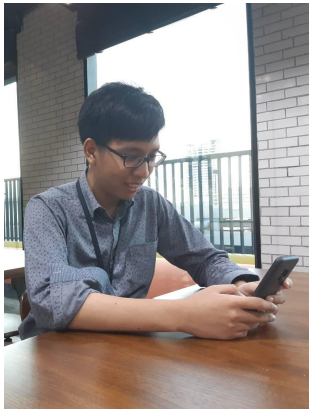
Mentor

Siriwat
Kasamwattanarote, Ph.D.

Tutors

Alongkorn
Chetasumon

Nattawit
Chaiworawitsakul



Kriengsak
Kongkapunya

Pinn
Prugsanapan

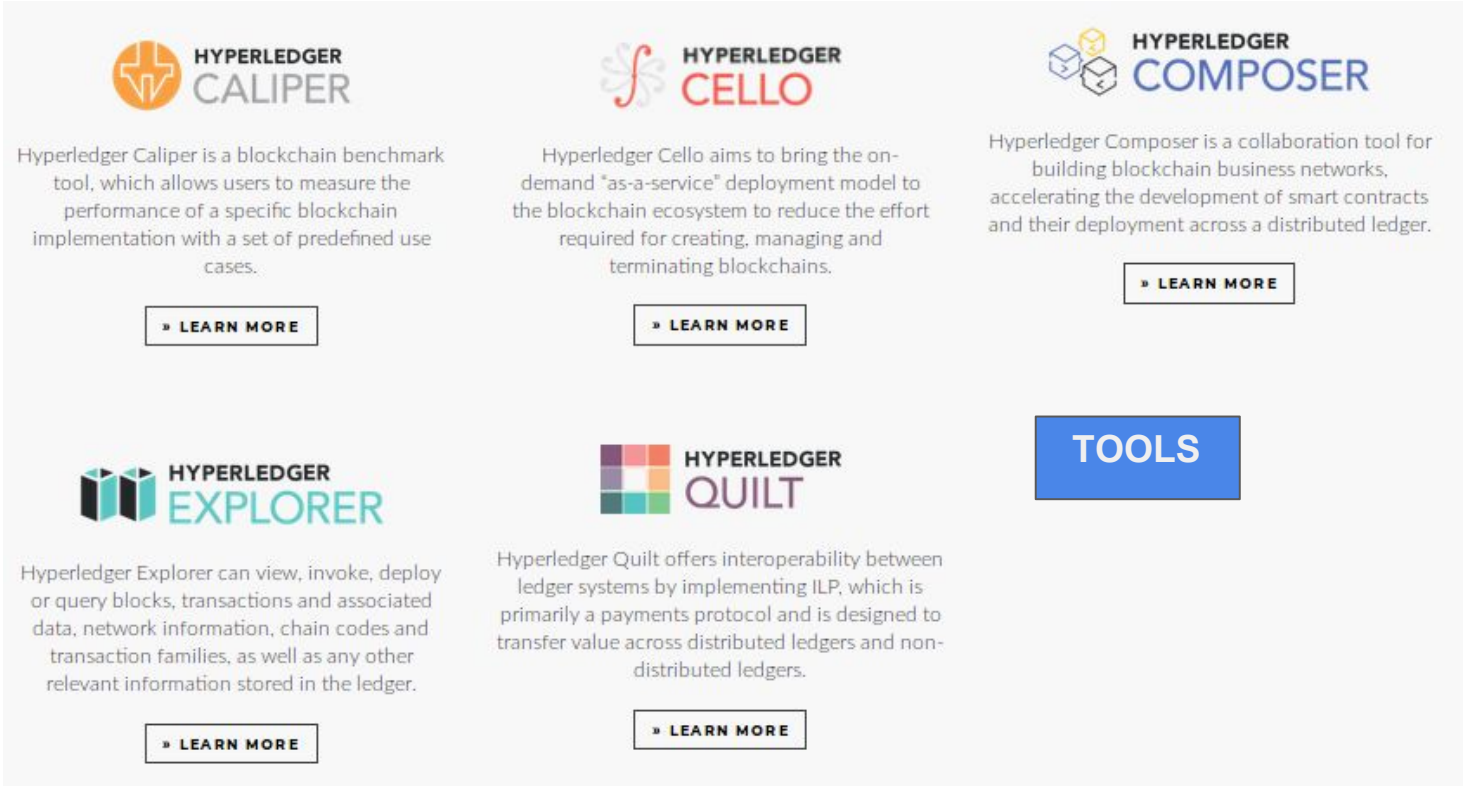
Hyperledger Overview

Hyperledger Overview (1)

Hyperledger

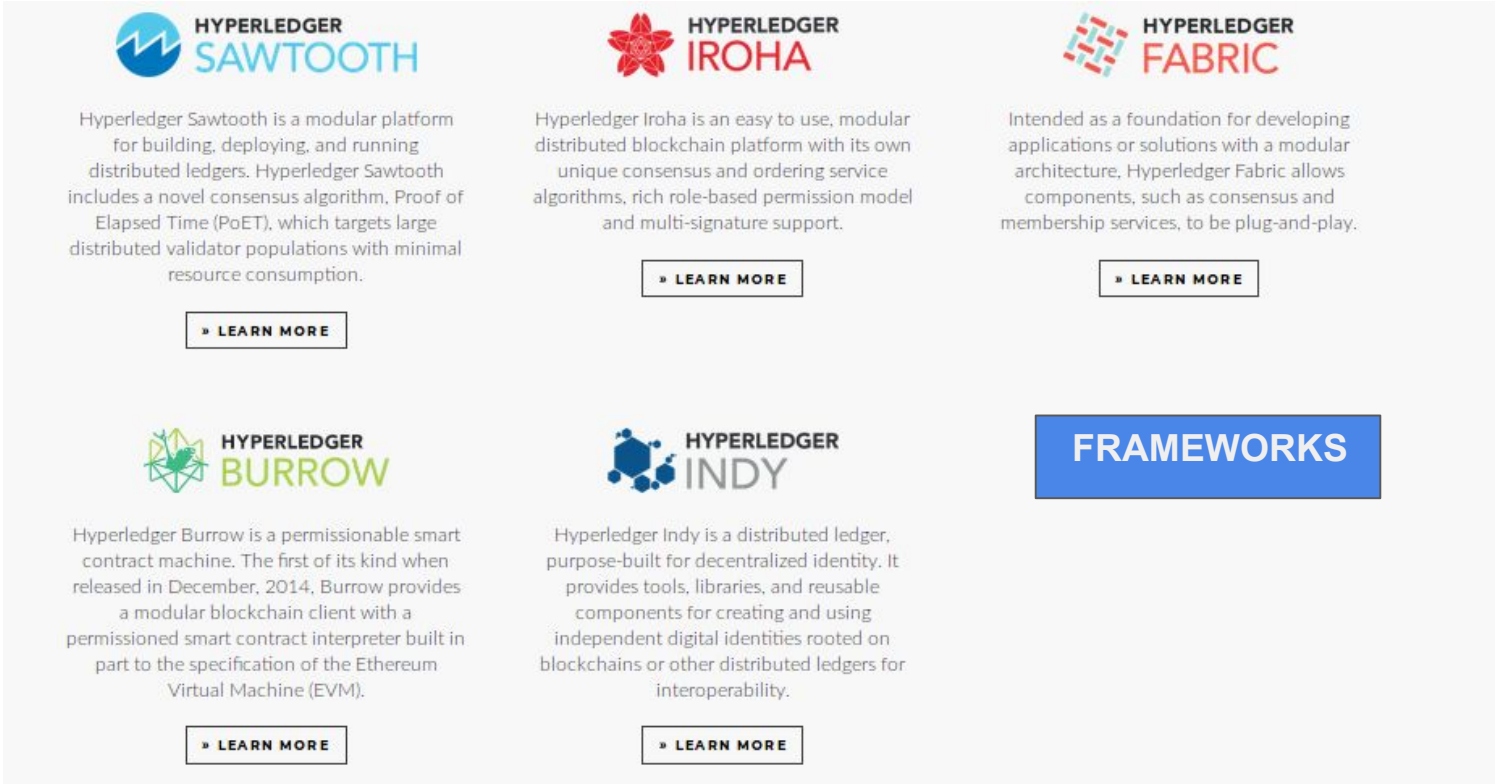
- Open Source Blockchain Project
- Hosted Under Linux Foundation
- Contains 10 projects
- Use Cases
  - Financial, Healthcare, Supply Chain, Retail, Bank Guarantee Etc.
- Communities
  - https://chat.hyperledger.org/home - Rocket Chat
  - https://stackoverflow.com/questions/tagged/hyperledger - Stackoverflow
  - https://github.com/hyperledger - Github

Hyperledger Overview (2)



Reference: https://www.hyperledger.org/projects

Hyperledger Overview (3)



Reference: https://www.hyperledger.org/projects

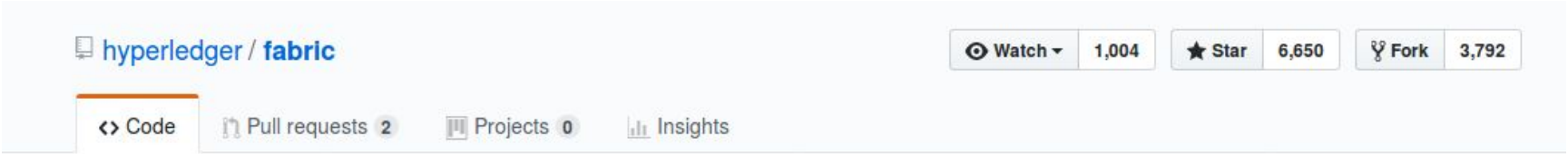
Hyperledger Overview (4)

Hyperledger Composer - https://composer-playground.mybluemix.net/

- Simple, Quick BUT Frequent Update (latest 0.20.x)
- Concept
  - Participant - A user on the blockchain network
  - Asset - An object that will be transfer in the blockchain
  - Transaction - An operation between participants
  - Ledger - A record of asset transfer and change
- Component
  - Model - Define objects in blockchain network
  - Scripts - Business logic to do with asset by participant
  - Access Control - Permission of participant in blockchain network
  - Query - Query Definition

Hyperledger Overview (5)

Hyperledger Fabric



- Enterprise Blockchain Platform
- Permissioned Blockchain
- Flexible Architecture
- Latest Version is release-1.2.0

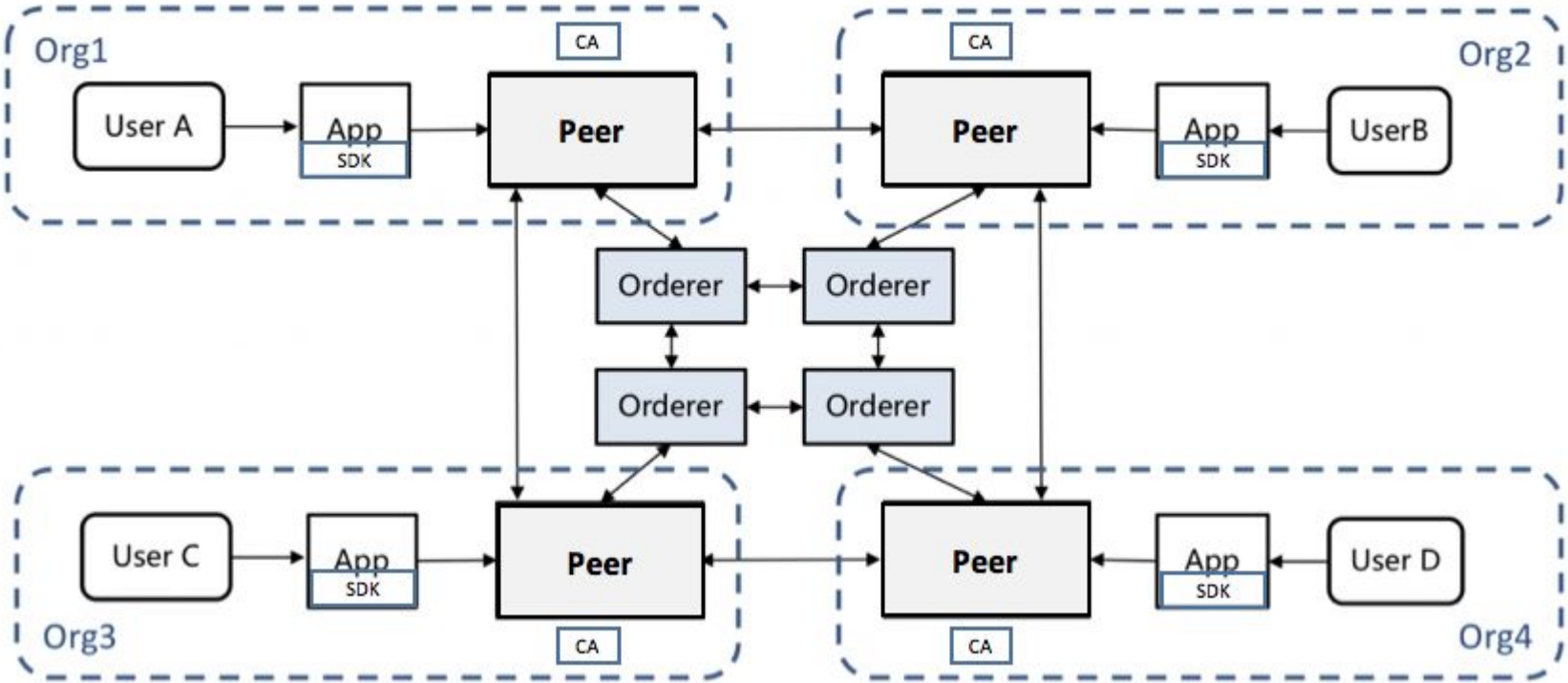
Hyperledger Fabric Overview

Hyperledger Fabric Overview (1)

Hyperledger Fabric Concept

1. Components
2. Transactions Flow
3. Consensus

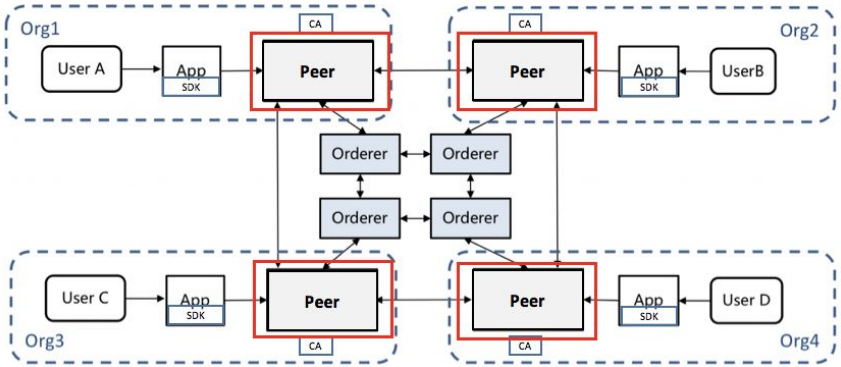
Hyperledger Fabric Overview (1)



http://www.think-foundry.com/deploy-hyperledger-fabric-on-kubernetes-part-1/

Hyperledger Fabric Overview (2)

Component: Peer



Endorsing Peer: Peers that execute the smart contract according to the "Endorsement Policy"

Committing Peer: Peers that write the blocks received from "Orderer"

Anchor Peer: Peers that represent an Organization in a "Channel"

Hyperledger Fabric Overview (3)

Component: Ledgers

Blockchain/Ledger:

- A “Chain” of “Blocks*” that every peer keeps a copy of it
- An “Immutable Blockchain” similar to other blockchain platform that keeps every transactions content.

World State:

- A Key-Value-Store(KVS) database that keeps the latest state.
- Support LevelDB(Default) and CouchDB

*Block is a batch of transactions created by the orderer

Hyperledger Fabric Overview (4)

Component: Smart Contract

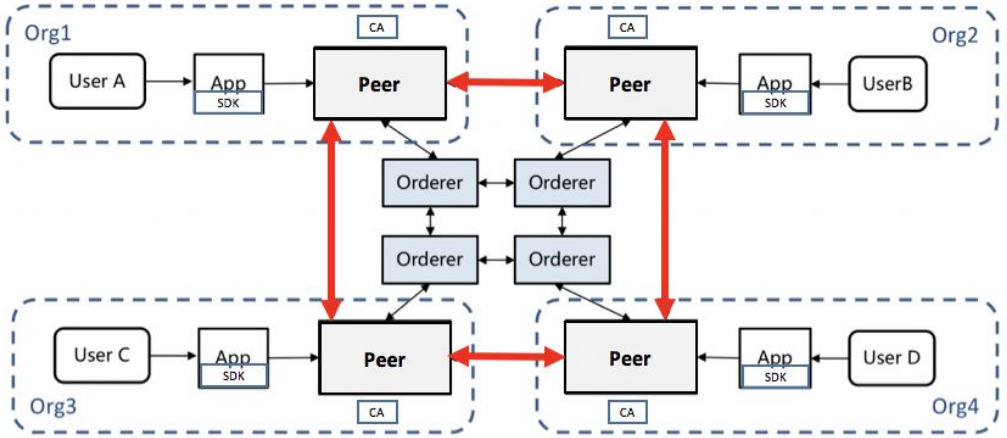
Smart Contract: All instructions/Business logic that can be customized to suites the functional requirements.

“A Smart Contract in Hyperledger Fabric is also known as Chaincode”

Hyperledger Fabric Overview (5)

Component: Channel

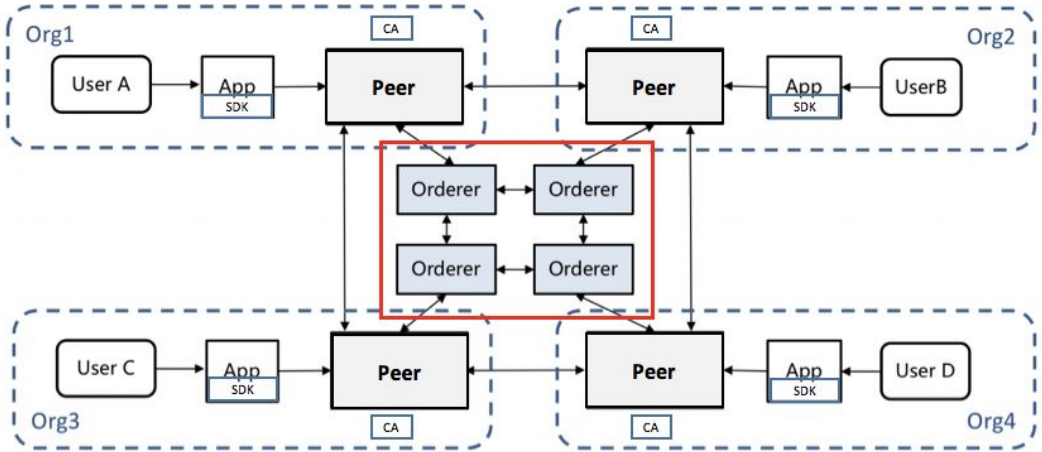
Channel: A subnetwork that connects peers of Organization which provide data privacy between a group of organizations.



Hyperledger Fabric Overview (6)

Component: Ordering Service

Orderer: Node(s) that responsible for making blocks and send it to all the peer in a specific channel.



Hyperledger Fabric Overview (7)

Component: Fabric Certificate Authorities

Hyperledger Fabric has a default CA that issues Public-Key infrastructure(PKI) certificates for each organization and its users.

Note:

1. Can use third-party CA
2. Support HA implementation

Hyperledger Fabric Overview (7)

Component: Fabric Certificate Authorities

Hyperledger Fabric has a default CA that issues Public-Key infrastructure(PKI) certificates for each organization and its users.

“Hyperledger Fabric CA can also be used to generate the keys and certificates needed to configure an Membership service provider (MSP)”

“MSP will use MSPID that will be unique for each organization to identify which organization the user is in”

Hyperledger Fabric Overview (8)

Component: Hyperledger Fabric SDK

Hyperledger Fabric SDK is used to interact with the network including

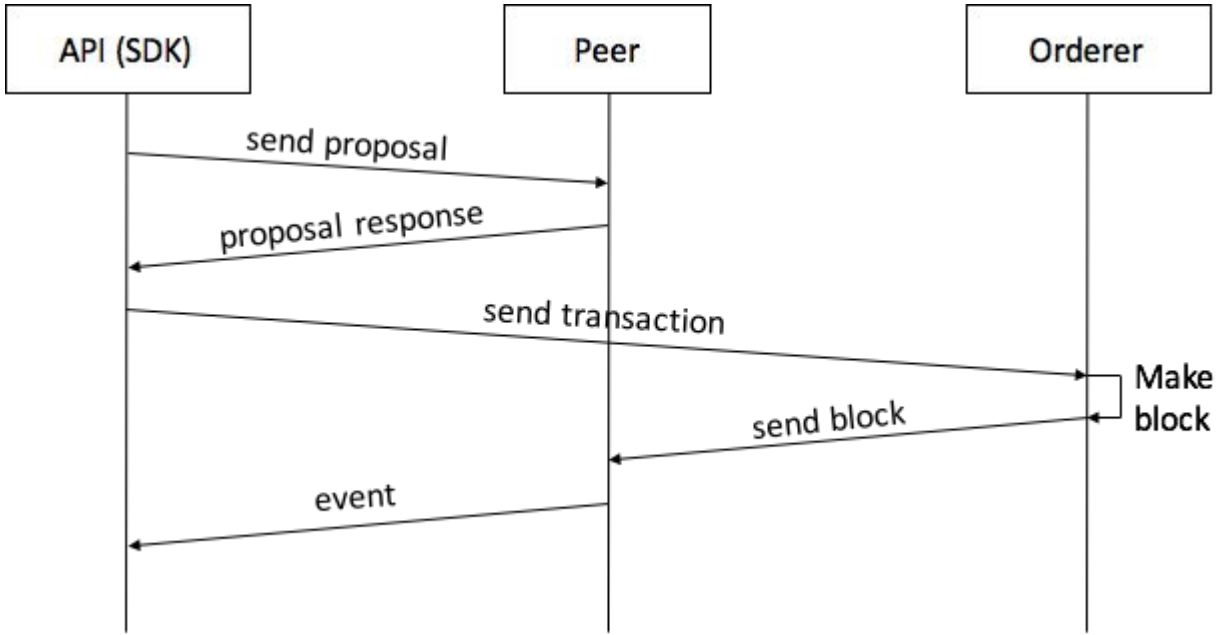
- Transaction Processing
- Membership Service
- Node traversal
- Event Handling

Available in **Node.js**, **JAVA**, Go, Python and REST

https://github.com/hyperledger/fabric-sdk-node, https://github.com/hyperledger/fabric-sdk-java,
https://github.com/hyperledger/fabric-sdk-go, https://github.com/hyperledger/fabric-sdk-py
https://github.com/hyperledger/fabric-sdk-rest

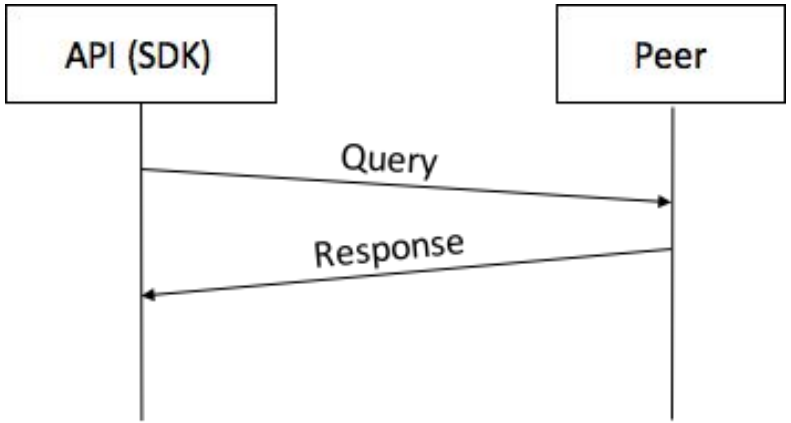
Hyperledger Fabric Overview (8)

Transaction Flow (Invoke)



Hyperledger Fabric Overview (8)

Transaction Flow (Query)

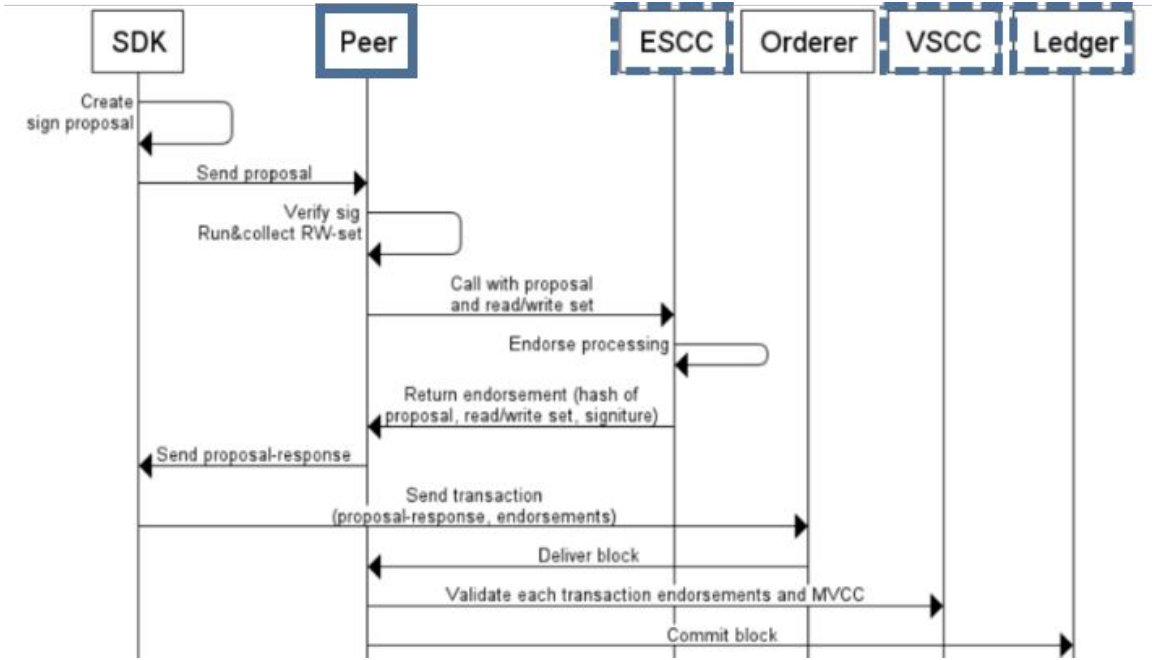


Hyperledger Fabric Overview (8)

Transaction Flow (Invoke)

ESCC =
Endorsement System chaincode

VSCC =
Validation System chaincode

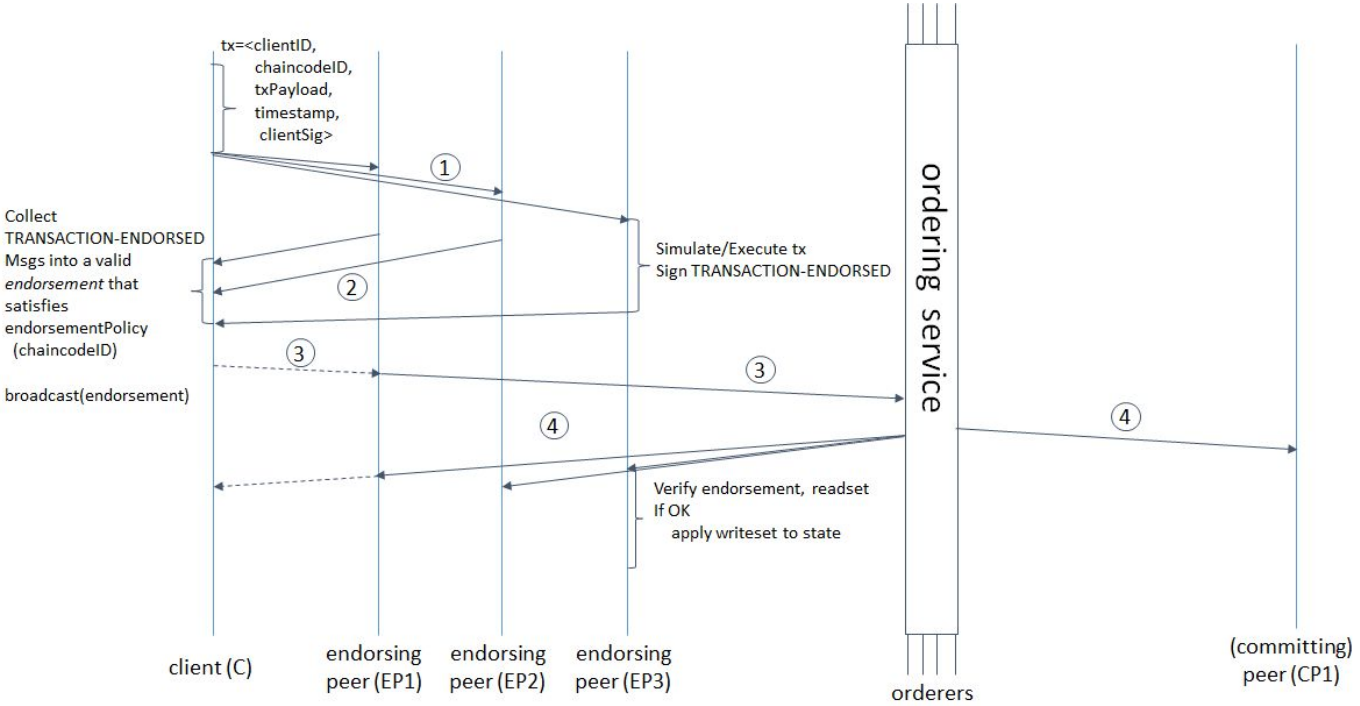


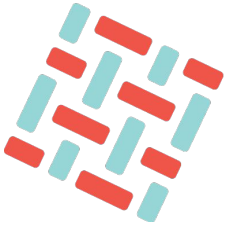
*ESCC, VSCC, Ledger are in Peer

Reference: https://vitalflux.com/blockchain-transaction-hyperledger-blockchain-network/

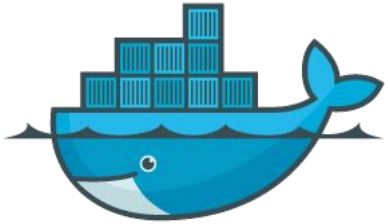
Hyperledger Fabric Overview (9)

Consensus = Permissioned Voting-Based Approach





HYPERLEDGER
FABRIC



docker

Environment Setup

node JS ™

Requirement

Unix based OS

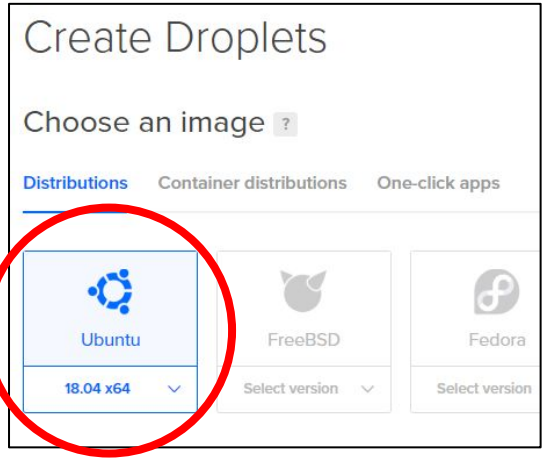
- Linux ✓
- Mac OSX ✓

Windows based OS

- Install VirtualBox
  https://goo.gl/s28DSg
- Install Ubuntu
  https://goo.gl/VoxJzT

Other option

- Free **10$** instance on DigitalOcean
- https://goo.gl/B5Fq7o



Environment Setup (1)

Installing Curl

Download at link.

You can check the version by running the following command:

```
curl --version
```

Link to document: https://goo.gl/Y7eykr

Environment Setup (2)

Docker and Docker Compose

MacOSX, *nix, or Windows 10: Docker Docker version 17.06.2-ce or greater is required.

Download at link.

> Installing Docker for Mac or Windows, or Docker Toolbox will also install Docker Compose. If you already had Docker installed, you should check that you have Docker Compose version 1.14.0 or greater installed. If not, we recommend that you install a more recent version of Docker.

You can check the version by running the following command:

```
docker --version
docker-compose --version
```

> If docker-compose is unavailable on your machine, try installing directly by following link

Environment Setup (3)

Go

Hyperledger Fabric uses the Go Programming Language for many of its components.

Go version 1.10.x is required.

Download at link.

You can check the version by running the following command:

```
go version
```

Set up path:

```
export GOPATH=$HOME/go
export PATH=$PATH:$GOPATH/bin
```

Environment Setup (4)

Node.js Runtime and NPM

Node.js - version 8.9.x or greater

> Node.js version 9.x is not supported at this time.

Download at link.

> Installing Node.js will also install NPM, however it is recommended that you confirm the version of NPM installed. You can upgrade the `npm` tool with the following command:

```
npm install npm@5.6.0 -g
```

Environment Setup (5)

Install Samples, Binaries and Docker Images

The following command will perform the these steps:

1. If needed, clone the hyperledger/fabric-samples repository
2. Checkout the appropriate version tag
3. Install the Hyperledger Fabric platform-specific binaries and config files for the version specified into the root of the fabric-samples repository
4. Download the Hyperledger Fabric docker images for the version specified

```
curl -sSL http://bit.ly/2ysbOFE | bash -s 1.2.0
```

Set up path:

```
export PATH=$(pwd)/bin:$PATH
```

Environment Setup (6)

Clone workshop project

The workshop project is available on github. You can download it by running the following command:

```
git clone https://github.com/linpinn/DPUX-Hyperledger-workshop
```

Demo

Demo

- Hyperledger Network
- Smart Contract (aka. Chaincode)
- API
- Hyperledger-explorer

Setup Network and Chaincode Deployment

Setup Hyperledger Network (1)

Hyperledger Network configuration

- Generate
  - configtx.yaml - Configurations of Peer(s) and Orderer(s)
  - crypto-config.yaml - Certificates format configuration
- Start Network
  - start.sh
  - docker-compose.yaml
- Chaincode Deployment
  - deployNetwork.sh - 3_HLF deployment/network

Setup Hyperledger Network (2)

Hyperledger Network setup steps

1. Install hyperledger fabric dependencies - docker images
2. Start hyperledger fabric components - CA, Peer, Orderer, Cli
3. Install and instantiate chaincode
4. Install application dependencies - node modules
5. Start application

Chaincode Development

Chaincode Development

- Basic go

```
package main

import (
	"fmt"
)

func main() {
	fmt.Println("Hello, 世界")
}
```

Chaincode Development

- Basic go

```
package main

import (
	"fmt"
)

func main() {
	fmt.Println("Hello, 世界")
}
```

Chaincode Development

- Basic go

```
package main

import (
	"fmt"
)

func main() {
	fmt.Println("Hello, 世界")
}
```

Chaincode Development

- Basic go

```
package main

import (
	"fmt"

	"github.com/sirupsen/logrus"
)

func main() {
	logrus.Println("Hello, 世界")
}
```

godoc.org/github.com/sirupsen/logrus

Chaincode Development

- Basic go

```
package main

func main() {
      var a int
      var b int = 10
      var c string = "some string"
      d := 10
      e := "some more string"
}
```

Chaincode Development

- Basic go

```
package main

import (
	"fmt"
)

func main() {
	fmt.Println("1 + 2 =", Add(1, 2))
}

func Add(a int, b int) int {
	return a + b
}
```

Chaincode Development

- Basic go

```
package main

import (
	"fmt"
)

func main() {
	fmt.Println(AddMinus(1, 2))
}

func AddMinus(a int, b int) (int, int) {
	return a + b, a - b
}
```

Chaincode Development

- Basic go

```
package main

import (
	"fmt"
)

func main() {
	c := Circle{5}
	fmt.Println(c.String())
}

type Circle struct {
	Radius int
}

func (c Circle) String() string {
	return fmt.Sprint("I am a circle with radius ", c.Radius)
}
```

Chaincode Development

- Basic go

```
package main

import (
	"fmt"
)

func main() {
	var c Stringer
	c = Circle{5}
	fmt.Println(c.String())
}

type Stringer interface {
	String() string
}
```

Chaincode Development

- Basic go

```
package main

import (
	"fmt"
)

func main() {
	var c Stringer
	c = Circle{5}
	fmt.Println(c.String())
}

type Stringer interface {
	String() string
}
```

Chaincode Development

- Basic go
  - A Tour of Go: https://tour.golang.org
  - Language Specification: https://golang.org/ref/spec
  - Effective Go: https://golang.org/doc/effective_go.html
  - Standard Package: https://golang.org/pkg/

Chaincode Development

- Chaincode API

https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim

Chaincode Development

- Chaincode API

func Start

```
func Start(cc Chaincode) error
```

type Chaincode

```
type Chaincode interface {
    // Init is called during Instantiate transaction after the chaincode container
    // has been established for the first time, allowing the chaincode to
    // initialize its internal data
    Init(stub ChaincodeStubInterface) pb.Response

    // Invoke is called to update or query the ledger in a proposal transaction.
    // Updated state variables are not committed to the ledger until the
    // transaction is committed.
    Invoke(stub ChaincodeStubInterface) pb.Response
}
```

Chaincode interface must be implemented by all chaincodes. The fabric runs the transactions by calling these functions as specified.

Chaincode Development

- Chaincode API

func Start

```
func Start(cc Chaincode) error
```

type Chaincode

```
type Chaincode interface {
    // Init is called during Instantiate transaction after the chaincode container
    // has been established for the first time, allowing the chaincode to
    // initialize its internal data
    Init(stub ChaincodeStubInterface) pb.Response

    // Invoke is called to update or query the ledger in a proposal transaction.
    // Updated state variables are not committed to the ledger until the
    // transaction is committed.
    Invoke(stub ChaincodeStubInterface) pb.Response
}
```

Chaincode interface must be implemented by all chaincodes. The fabric runs the transactions by calling these functions as specified.

Chaincode Development

- Chaincode API

type ChaincodeStubInterface

```
type ChaincodeStubInterface interface {
    // GetArgs returns the arguments intended for the chaincode Init and Invoke
    // as an array of byte arrays.
    GetArgs() [][]byte

    // GetStringArgs returns the arguments intended for the chaincode Init and
    // Invoke as a string array. Only use GetStringArgs if the client passes
    // arguments intended to be used as strings.
    GetStringArgs() []string

    // GetFunctionAndParameters returns the first argument as the function
    // name and the rest of the arguments as parameters in a string array.
    // Only use GetFunctionAndParameters if the client passes arguments intended
    // to be used as strings.
    GetFunctionAndParameters() (string, []string)

    // GetArgsSlice returns the arguments intended for the chaincode Init and
    // Invoke as a byte array
    GetArgsSlice() ([]byte, error)

    // GetTxID returns the tx_id of the transaction proposal, which is unique per
    // transaction and per client. See ChannelHeader in protos/common/common.proto
    // for further details.
    GetTxID() string

    // GetChannelID returns the channel the proposal is sent to for chaincode to process
    // This would be the channel_id of the transaction proposal (see ChannelHeader
    // in protos/common/common.proto) except where the chaincode is calling another on
    // a different channel
    GetChannelID() string

    // InvokeChaincode locally calls the specified chaincode `Invoke` using the
    // same transaction context; that is, chaincode calling chaincode doesn't
    // create a new transaction message.
    // If the called chaincode is on the same channel, it simply adds the called
    // chaincode read set and write set to the calling transaction.
    // If the called chaincode is on a different channel,
    // only the Response is returned to the calling chaincode; any PutState calls
    // from the called chaincode will not have any effect on the ledger; that is,
    // the called chaincode on a different channel will not have its read set
    // and write set applied to the transaction. Only the calling chaincode's
    // read set and write set will be applied to the transaction. Effectively
    // the called chaincode on a different channel is a `Query`, which does not
    // participate in state validation checks in subsequent commit phase.
    // If `channel` is empty, the caller's channel is assumed.
    InvokeChaincode(chaincodeName string, args [][]byte, channel string) pb.Response

    // GetState returns the value of the specified `key` from the
    // ledger. Note that GetState doesn't read data from the writeset, which
    // has not been committed to the ledger. In other words, GetState doesn't
    // consider data modified by PutState that has not been committed.
    // If the key does not exist in the state database, (nil, nil) is returned.
    GetState(key string) ([]byte, error)

    // PutState puts the specified `key` and `value` into the transaction's
    // writeset as a data-write proposal. PutState doesn't effect the ledger
    // until the transaction is validated and successfully committed.
    // Simple keys must not be an empty string and must not start with null
    // character (0x00), in order to avoid range query collisions with
    // composite keys, which internally get prefixed with 0x00 as composite
    // key namespace.
    PutState(key string, value []byte) error

    // DelState records the specified `key` to be deleted in the writeset of
    // the transaction proposal. The `key` and its value will be deleted from
    // the ledger when the transaction is validated and successfully committed.
    DelState(key string) error

    // GetStateByRange returns a range iterator over a set of keys in the
    // ledger. The iterator can be used to iterate over all keys
    // between the startKey (inclusive) and endKey (exclusive).
    // The keys are returned by the iterator in lexical order. Note
    // that startKey and endKey can be empty string, which implies unbounded range
    // query on start or end.
    // Call Close() on the returned StateQueryIteratorInterface object when done.
    // The query is re-executed during validation phase to ensure result set
    // has not changed since transaction endorsement (phantom reads detected).
    GetStateByRange(startKey, endKey string) (StateQueryIteratorInterface, error)

    // GetStateByPartialCompositeKey queries the state in the ledger based on
    // a given partial composite key. This function returns an iterator
    // which can be used to iterate over all composite keys whose prefix matches
    // the given partial composite key. The `objectType` and attributes are
    // expected to have only valid utf8 strings and should not contain
    // U+0000 (nil byte) and U+10FFFF (biggest and unallocated code point).
    // See related functions SplitCompositeKey and CreateCompositeKey.
    // Call Close() on the returned StateQueryIteratorInterface object when done.
    // The query is re-executed during validation phase to ensure result set
    // has not changed since transaction endorsement (phantom reads detected).
    GetStateByPartialCompositeKey(objectType string, keys []string) (StateQueryIterator

    // CreateCompositeKey combines the given `attributes` to form a composite
    // key. The objectType and attributes are expected to have only valid utf8
    // strings and should not contain U+0000 (nil byte) and U+10FFFF
    // (biggest and unallocated code point).
    // The resulting composite key can be used as the key in PutState().
    CreateCompositeKey(objectType string, attributes []string) (string, error)

    // SplitCompositeKey splits the specified key into attributes on which the
    // composite key was formed. Composite keys found during range queries
    // or partial composite key queries can therefore be split into their
    // composite parts.
    SplitCompositeKey(compositeKey string) (string, []string, error)

    // GetQueryResult performs a "rich" query against a state database. It is
    // only supported for state databases that support rich query,
    // e.g.CouchDB. The query string is in the native syntax
    // of the underlying state database. An iterator is returned
    // which can be used to iterate (next) over the query result set.
    // The query is NOT re-executed during validation phase, phantom reads are
    // not detected. That is, other committed transactions may have added,
    // updated, or removed keys that impact the result set, and this would not
    // be detected at validation/commit time.  Applications susceptible to this
    // should therefore not use GetQueryResult as part of transactions that update
    // ledger, and should limit use to read-only chaincode operations.
    GetQueryResult(query string) (StateQueryIteratorInterface, error)

    // GetHistoryForKey returns a history of key values across time.
    // For each historic key update, the historic value and associated
    // transaction id and timestamp are returned. The timestamp is the
    // timestamp provided by the client in the proposal header.
    // GetHistoryForKey requires peer configuration
    // core.ledger.history.enableHistoryDatabase to be true.
    // The query is NOT re-executed during validation phase, phantom reads are
    // not detected. That is, other committed transactions may have updated
    // the key concurrently, impacting the result set, and this would not be
    // detected at validation/commit time. Applications susceptible to this
    // should therefore not use GetHistoryForKey as part of transactions that
    // update ledger, and should limit use to read-only chaincode operations.
    GetHistoryForKey(key string) (HistoryQueryIteratorInterface, error)

    // GetPrivateData returns the value of the specified `key` from the specified
    // `collection`. Note that GetPrivateData doesn't read data from the
    // private writeset, which has not been committed to the `collection`. In
    // other words, GetPrivateData doesn't consider data modified by PutPrivateData
    // that has not been committed.
    GetPrivateData(collection, key string) ([]byte, error)

    // PutPrivateData puts the specified `key` and `value` into the transaction's
    // private writeset. Note that only hash of the private writeset goes into the
    // transaction proposal response (which is sent to the client who issued the
    // transaction) and the actual private writeset gets temporarily stored in a
    // transient store. PutPrivateData doesn't effect the `collection` until the
    // transaction is validated and successfully committed. Simple keys must not be
    // an empty string and must not start with null character (0x00), in order to
    // avoid range query collisions with composite keys, which internally get
    // prefixed with 0x00 as composite key namespace.
    PutPrivateData(collection string, key string, value []byte) error
```

Chaincode Development

- Input

```
// GetArgs returns the arguments intended for the chaincode Init and Invoke
// as an array of byte arrays.
GetArgs() [][]byte

// GetStringArgs returns the arguments intended for the chaincode Init and
// Invoke as a string array. Only use GetStringArgs if the client passes
// arguments intended to be used as strings.
GetStringArgs() []string

// GetFunctionAndParameters returns the first argument as the function
// name and the rest of the arguments as parameters in a string array.
// Only use GetFunctionAndParameters if the client passes arguments intended
// to be used as strings.
GetFunctionAndParameters() (string, []string)

// GetArgsSlice returns the arguments intended for the chaincode Init and
// Invoke as a byte array
GetArgsSlice() ([]byte, error)
```

Chaincode Development

- Processing

```
// GetState returns the value of the specified `key` from the
// ledger. Note that GetState doesn't read data from the writeset, which
// has not been committed to the ledger. In other words, GetState doesn't
// consider data modified by PutState that has not been committed.
// If the key does not exist in the state database, (nil, nil) is returned.
GetState(key string) ([]byte, error)

// PutState puts the specified `key` and `value` into the transaction's
// writeset as a data-write proposal. PutState doesn't effect the ledger
// until the transaction is validated and successfully committed.
// Simple keys must not be an empty string and must not start with null
// character (0x00), in order to avoid range query collisions with
// composite keys, which internally get prefixed with 0x00 as composite
// key namespace.
PutState(key string, value []byte) error

// DelState records the specified `key` to be deleted in the writeset of
// the transaction proposal. The `key` and its value will be deleted from
// the ledger when the transaction is validated and successfully committed.
DelState(key string) error
```

```
// GetHistoryForKey returns a history of key values across time.
// For each historic key update, the historic value and associated
// transaction id and timestamp are returned. The timestamp is the
// timestamp provided by the client in the proposal header.
// GetHistoryForKey requires peer configuration
// core.ledger.history.enableHistoryDatabase to be true.
// The query is NOT re-executed during validation phase, phantom reads are
// not detected. That is, other committed transactions may have updated
// the key concurrently, impacting the result set, and this would not be
// detected at validation/commit time. Applications susceptible to this
// should therefore not use GetHistoryForKey as part of transactions that
// update ledger, and should limit use to read-only chaincode operations.
GetHistoryForKey(key string) (HistoryQueryIteratorInterface, error)
```

Chaincode Development

- Result

type Chaincode

```
type Chaincode interface {
    // Init is called during Instantiate transaction after the chaincode container
    // has been established for the first time, allowing the chaincode to
    // initialize its internal data
    Init(stub ChaincodeStubInterface) pb.Response

    // Invoke is called to update or query the ledger in a proposal transaction.
    // Updated state variables are not committed to the ledger until the
    // transaction is committed.
    Invoke(stub ChaincodeStubInterface) pb.Response
}
```

Chaincode interface must be implemented b
functions as specified.

func Success

```
func Success(payload []byte) pb.Response
```

func Error

```
func Error(msg string) pb.Response
```

Chaincode Development

- Composite key

```
// GetStateByRange returns a range iterator over a set of keys in the
// ledger. The iterator can be used to iterate over all keys
// between the startKey (inclusive) and endKey (exclusive).
// The keys are returned by the iterator in lexical order. Note
// that startKey and endKey can be empty string, which implies unbounded range
// query on start or end.
// Call Close() on the returned StateQueryIteratorInterface object when done.
// The query is re-executed during validation phase to ensure result set
// has not changed since transaction endorsement (phantom reads detected).
GetStateByRange(startKey, endKey string) (StateQueryIteratorInterface, error)

// GetStateByPartialCompositeKey queries the state in the ledger based on
// a given partial composite key. This function returns an iterator
// which can be used to iterate over all composite keys whose prefix matches
// the given partial composite key. The `objectType` and attributes are
// expected to have only valid utf8 strings and should not contain
// U+0000 (nil byte) and U+10FFFF (biggest and unallocated code point).
// See related functions SplitCompositeKey and CreateCompositeKey.
// Call Close() on the returned StateQueryIteratorInterface object when done.
// The query is re-executed during validation phase to ensure result set
// has not changed since transaction endorsement (phantom reads detected).
GetStateByPartialCompositeKey(objectType string, keys []string) (StateQueryIteratorI

// CreateCompositeKey combines the given `attributes` to form a composite
// key. The objectType and attributes are expected to have only valid utf8
// strings and should not contain U+0000 (nil byte) and U+10FFFF
// (biggest and unallocated code point).
// The resulting composite key can be used as the key in PutState().
CreateCompositeKey(objectType string, attributes []string) (string, error)

// SplitCompositeKey splits the specified key into attributes on which the
// composite key was formed. Composite keys found during range queries
// or partial composite key queries can therefore be split into their
// composite parts.
SplitCompositeKey(compositeKey string) (string, []string, error)
```

Chaincode Development

- Composite key

```
// GetStateByRange returns a range iterator over a set of keys in the
// ledger. The iterator can be used to iterate over all keys
// between the startKey (inclusive) and endKey (exclusive).
// The keys are returned by the iterator in lexical order. Note
// that startKey and endKey can be empty string, which implies unbounded range
// query on start or end.
// Call Close() on the returned StateQueryIteratorInterface object when done.
// The query is re-executed during validation phase to ensure result set
// has not changed since transaction endorsement (phantom reads detected).
GetStateByRange(startKey, endKey string) (StateQueryIteratorInterface, error)

// GetStateByPartialCompositeKey queries the state in the ledger based on
// a given partial composite key. This function returns an iterator
// which can be used to iterate over all composite keys whose prefix matches
// the given partial composite key. The `objectType` and attributes are
// expected to have only valid utf8 strings and should not contain
// U+0000 (nil byte) and U+10FFFF (biggest and unallocated code point).
// See related functions SplitCompositeKey and CreateCompositeKey.
// Call Close() on the returned StateQueryIteratorInterface object when done.
// The query is re-executed during validation phase to ensure result set
// has not changed since transaction endorsement (phantom reads detected).
GetStateByPartialCompositeKey(objectType string, keys []string) (StateQueryIteratorI

// CreateCompositeKey combines the given `attributes` to form a composite
// key. The objectType and attributes are expected to have only valid utf8
// strings and should not contain U+0000 (nil byte) and U+10FFFF
// (biggest and unallocated code point).
// The resulting composite key can be used as the key in PutState().
CreateCompositeKey(objectType string, attributes []string) (string, error)

// SplitCompositeKey splits the specified key into attributes on which the
// composite key was formed. Composite keys found during range queries
// or partial composite key queries can therefore be split into their
// composite parts.
SplitCompositeKey(compositeKey string) (string, []string, error)
```

Chaincode Development

- Composite key

```
// GetStateByRange returns a range iterator over a set of keys in the
// ledger. The iterator can be used to iterate over all keys
// between the startKey (inclusive) and endKey (exclusive).
// The keys are returned by the iterator in lexical order. Note
// that startKey and endKey can be empty string, which implies unbounded range
// query on start or end.
// Call Close() on the returned StateQueryIteratorInterface object when done.
// The query is re-executed during validation phase to ensure result set
// has not changed since transaction endorsement (phantom reads detected).
GetStateByRange(startKey, endKey string) (StateQueryIteratorInterface, error)

// GetStateByPartialCompositeKey queries the state in the ledger based on
// a given partial composite key. This function returns an iterator
// which can be used to iterate over all composite keys whose prefix matches
// the given partial composite key. The `objectType` and attributes are
// expected to have only valid utf8 strings and should not contain
// U+0000 (nil byte) and U+10FFFF (biggest and unallocated code point).
// See related functions SplitCompositeKey and CreateCompositeKey.
// Call Close() on the returned StateQueryIteratorInterface object when done.
// The query is re-executed during validation phase to ensure result set
// has not changed since transaction endorsement (phantom reads detected).
GetStateByPartialCompositeKey(objectType string, keys []string) (StateQueryIteratorI

// CreateCompositeKey combines the given `attributes` to form a composite
// key. The objectType and attributes are expected to have only valid utf8
// strings and should not contain U+0000 (nil byte) and U+10FFFF
// (biggest and unallocated code point).
// The resulting composite key can be used as the key in PutState().
CreateCompositeKey(objectType string, attributes []string) (string, error)

// SplitCompositeKey splits the specified key into attributes on which the
// composite key was formed. Composite keys found during range queries
// or partial composite key queries can therefore be split into their
// composite parts.
SplitCompositeKey(compositeKey string) (string, []string, error)
```

type StateQueryIteratorInterface

```
type StateQueryIteratorInterface interface {
    // Inherit HasNext() and Close()
    CommonIteratorInterface

    // Next returns the next key and value in the range and execute query iterator.
    Next() (*queryresult.KV, error)
}
```

StateQueryIteratorInterface allows a chaincode to iterate over a set of key/value pairs returned by range and execute query

type CommonIteratorInterface

```
type CommonIteratorInterface interface {
    // HasNext returns true if the range query iterator contains additional keys
    // and values.
    HasNext() bool

    // Close closes the iterator. This should be called when done
    // reading from the iterator to free up resources.
    Close() error
}
```

CommonIteratorInterface allows a chaincode to check whether any more result to be fetched from an iterator and close it when done.

Chaincode Development

- Composite key

```
// GetStateByRange returns a range iterator over a set of keys in the
// ledger. The iterator can be used to iterate over all keys
// between the startKey (inclusive) and endKey (exclusive).
// The keys are returned by the iterator in lexical order. Note
// that startKey and endKey can be empty string, which implies unbounded range
// query on start or end.
// Call Close() on the returned StateQueryIteratorInterface object when done.
// The query is re-executed during validation phase to ensure result set
// has not changed since transaction endorsement (phantom reads detected).
GetStateByRange(startKey, endKey string) (StateQueryIteratorInterface, error)

// GetStateByPartialCompositeKey queries the state in the ledger based on
// a given partial composite key. This function returns an iterator
// which can be used to iterate over all composite keys whose prefix matches
// the given partial composite key. The `objectType` and attributes are
// expected to have only valid utf8 strings and should not contain
// U+0000 (nil byte) and U+10FFFF (biggest and unallocated code point).
// See related functions SplitCompositeKey and CreateCompositeKey.
// Call Close() on the returned StateQueryIteratorInterface object when done.
// The query is re-executed during validation phase to ensure result set
// has not changed since transaction endorsement (phantom reads detected).
GetStateByPartialCompositeKey(objectType string, keys []string) (StateQueryIteratorI

// CreateCompositeKey combines the given `attributes` to form a composite
// key. The objectType and attributes are expected to have only valid utf8
// strings and should not contain U+0000 (nil byte) and U+10FFFF
// (biggest and unallocated code point).
// The resulting composite key can be used as the key in PutState().
CreateCompositeKey(objectType string, attributes []string) (string, error)

// SplitCompositeKey splits the specified key into attributes on which the
// composite key was formed. Composite keys found during range queries
// or partial composite key queries can therefore be split into their
// composite parts.
SplitCompositeKey(compositeKey string) (string, []string, error)
```

type StateQueryIteratorInterface

```
type StateQueryIteratorInterface interface {
    // Inherit HasNext() and Close()
    CommonIteratorInterface

    // Next returns the next key and value in the range and execute query iterator.
    Next() (*queryresult.KV, error)
}
```

StateQueryIteratorInterface allows a chaincode to iterate over a set of key/value pairs returned by range and execute query.

type KV

```
type KV struct {
    Namespace string `protobuf:"bytes,1,opt,name=namespace" json:"namespace,omitempty"`
    Key       string `protobuf:"bytes,2,opt,name=key" json:"key,omitempty"`
    Value     []byte `protobuf:"bytes,3,opt,name=value,proto3" json:"value,omitempty"`
}
```

KV -- QueryResult for range/execute query. Holds a key and corresponding value.

Chaincode Development

- Caller identity

```
// GetCreator returns `SignatureHeader.Creator` (e.g. an identity)
// of the `SignedProposal`. This is the identity of the agent (or user)
// submitting the transaction.
GetCreator() ([]byte, error)
```

Chaincode Development

- Caller identity

```
// GetCreator returns `SignatureHeader.Creator` (e.g. an identity)
// of the `SignedProposal`. This is the identity of the agent (or user)
// submitting the transaction.
GetCreator() ([]byte, error)
```

```
import (
        "bytes"
        "crypto/x509"
        "encoding/pem"
        "fmt"
)

func GetX509Cert(stub shim.ChaincodeStubInterface) (*Certificate, error) {
        creator, err := stub.GetCreator()
        if err != nil {
                return nil, err
        }
        start := bytes.Index(creator, []byte("-----BEGIN CERTIFICATE-----"))
        if start == -1 {
                return nil, fmt.Errorf("No Certificate found")
        }
        block, _ := pem.Decode(creator[start:])
        if block == nil {
                return nil, fmt.Errorf("cannot decode pem block from creator
cert")
        }

        return x509.ParseCertificate(block.Bytes)
}
```

Chaincode Development

- Timestamp

```
// GetTxTimestamp returns the timestamp when the transaction was created. This
// is taken from the transaction ChannelHeader, therefore it will indicate the
// client's timestamp and will have the same value across all endorsers.
GetTxTimestamp() (*timestamp.Timestamp, error)
```

type Timestamp

```
type Timestamp struct {
    // Represents seconds of UTC time since Unix epoch
    // 1970-01-01T00:00:00Z. Must be from 0001-01-01T00:00:00Z to
    // 9999-12-31T23:59:59Z inclusive.
    Seconds int64 `protobuf:"varint,1,opt,name=seconds,proto3" json:"seconds,omitempty"`
    // Non-negative fractions of a second at nanosecond resolution. Negative
    // second values with fractions must still have non-negative nanos values
    // that count forward in time. Must be from 0 to 999,999,999
    // inclusive.
    Nanos                int32    `protobuf:"varint,2,opt,name=nanos,proto3" json:"nanos
    XXX_NoUnkeyedLiteral struct{} `json:"-"`
    XXX_unrecognized     []byte   `json:"-"`
    XXX_sizecache        int32    `json:"-"`
}
```

Chaincode Development

- Timestamp

```
// GetTxTimestamp returns the timestamp when the transaction was created. This
// is taken from the transaction ChannelHeader, therefore it will indicate the
// client's timestamp and will have the same value across all endorsers.
GetTxTimestamp() (*timestamp.Timestamp, error)
```

```
import “time”

func GetTimestamp(stub shim.ChaincodeStubInterface) (time.Time, error) {
      bkkLoc, err = time.LoadLocation("Asia/Bangkok")
      if err != nil {
            return time.Time{}, error
      }

      t, err := stub.GetTxTimestamp()
      if err != nil {
            return time.Time{}, errors.Wrap(err)
      }
      return time.Unix(t.Seconds, int64(t.Nanos)).In(bkkLoc), nil
}
```

Chaincode Development

- Transaction ID and event

```
// GetTxID returns the tx_id of the transaction proposal, which is unique per
// transaction and per client. See ChannelHeader in protos/common/common.proto
// for further details.
GetTxID() string
```

```
// SetEvent allows the chaincode to set an event on the response to the
// proposal to be included as part of a transaction. The event will be
// available within the transaction in the committed block regardless of the
// validity of the transaction.
SetEvent(name string, payload []byte) error
```

Chaincode Development

- Cross channel processing

```
// InvokeChaincode locally calls the specified chaincode `Invoke` using the
// same transaction context; that is, chaincode calling chaincode doesn't
// create a new transaction message.
// If the called chaincode is on the same channel, it simply adds the called
// chaincode read set and write set to the calling transaction.
// If the called chaincode is on a different channel,
// only the Response is returned to the calling chaincode; any PutState calls
// from the called chaincode will not have any effect on the ledger; that is,
// the called chaincode on a different channel will not have its read set
// and write set applied to the transaction. Only the calling chaincode's
// read set and write set will be applied to the transaction. Effectively
// the called chaincode on a different channel is a `Query`, which does not
// participate in state validation checks in subsequent commit phase.
// If `channel` is empty, the caller's channel is assumed.
InvokeChaincode(chaincodeName string, args [][]byte, channel string) pb.Response
```

Chaincode Development

- CouchDB query

```
// GetQueryResult performs a "rich" query against a state database. It is
// only supported for state databases that support rich query,
// e.g.CouchDB. The query string is in the native syntax
// of the underlying state database. An iterator is returned
// which can be used to iterate (next) over the query result set.
// The query is NOT re-executed during validation phase, phantom reads are
// not detected. That is, other committed transactions may have added,
// updated, or removed keys that impact the result set, and this would not
// be detected at validation/commit time.  Applications susceptible to this
// should therefore not use GetQueryResult as part of transactions that update
// ledger, and should limit use to read-only chaincode operations.
GetQueryResult(query string) (StateQueryIteratorInterface, error)
```

Chaincode Development

- CouchDB query

```
type marbleStore struct {
  ObjectType string    `json:"docType"`
  Storename  string    `json:"storename"`
  Owner      owner     `json:"owner"`
  Employees  int       `json:"employees"`
  Marbles    []marble  `json:"marbles"`
}

type owner struct {
  Name       string    `json:"name"`
  LastName   string    `json:”lastname”`
}

type marble struct {
  ...
}
```

Search by storename

```
{
    "selector": {
        "storename": "tom’s marble"
    }
}
```

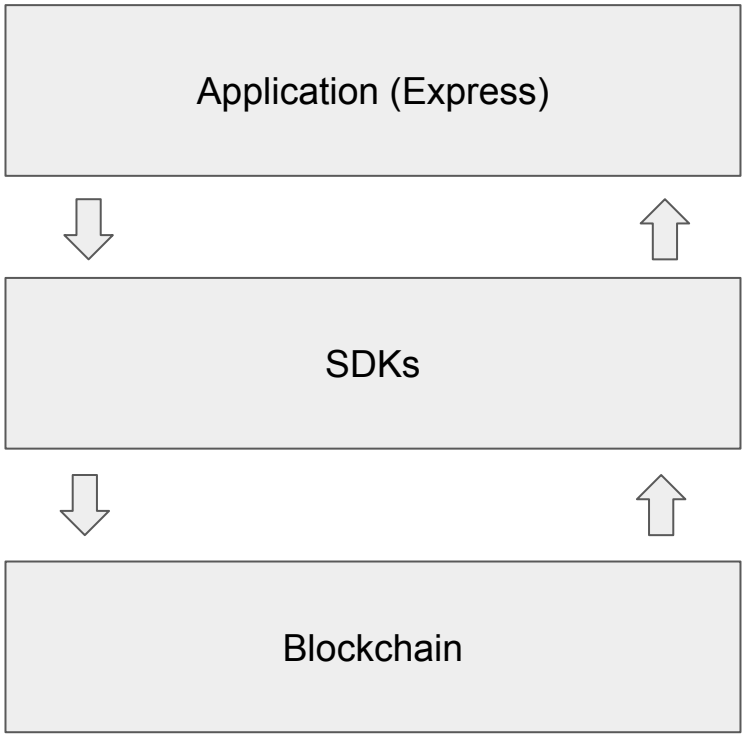
Search by owner’s name

```
{
    "selector": {
        "owner.name": "tom"
    }
}
```

https://medium.com/wearetheledger/hyperledger-fabric-couchdb-fantastic-queries-and-where-to-find-them-f8a3aecef767

API Development

API Development (1)



API Development (2)

fabric-ca-client

- allow our app to communicate with the ca server and retrieve identity material

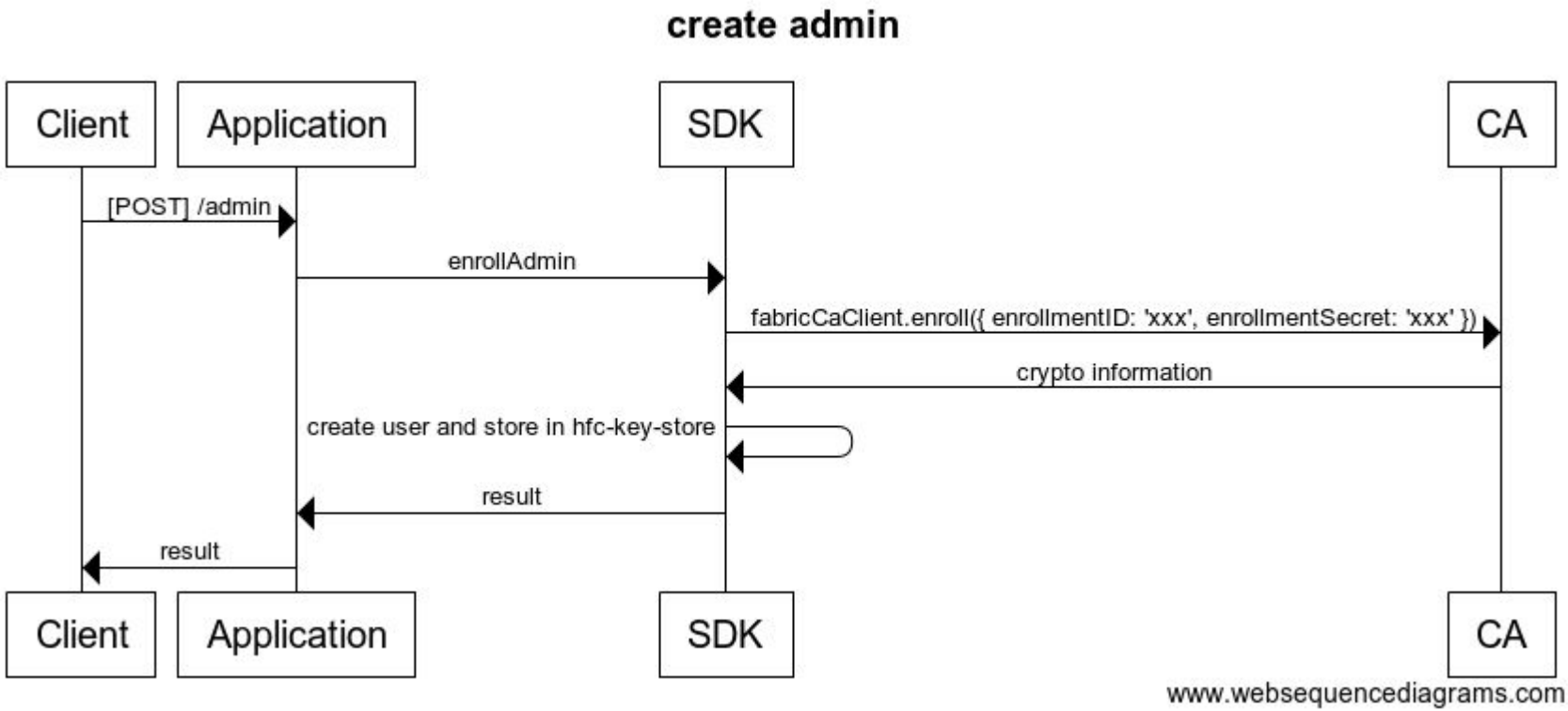
fabric-client

- allow us to load identity material and talk to peer and ordering service

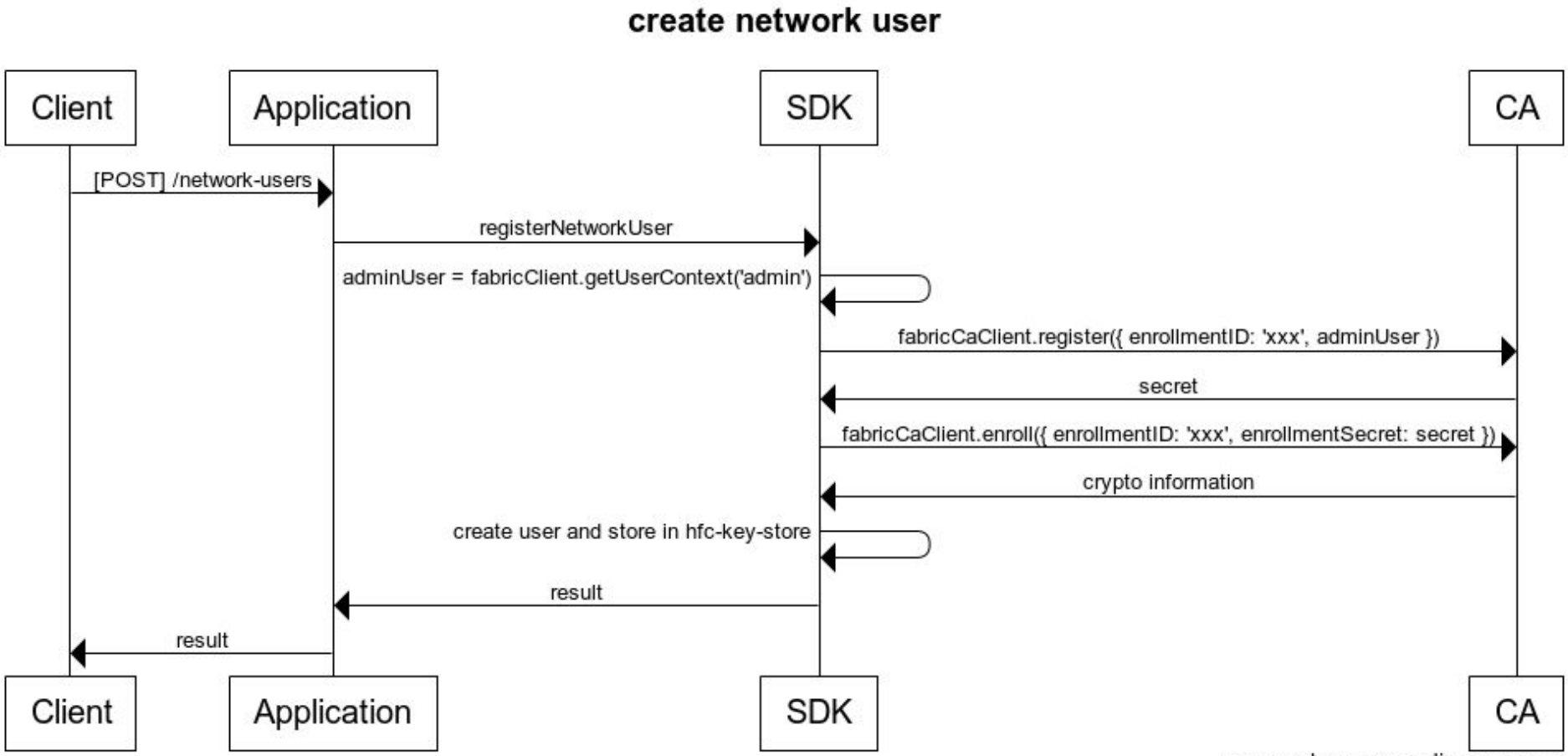
API Development (3)

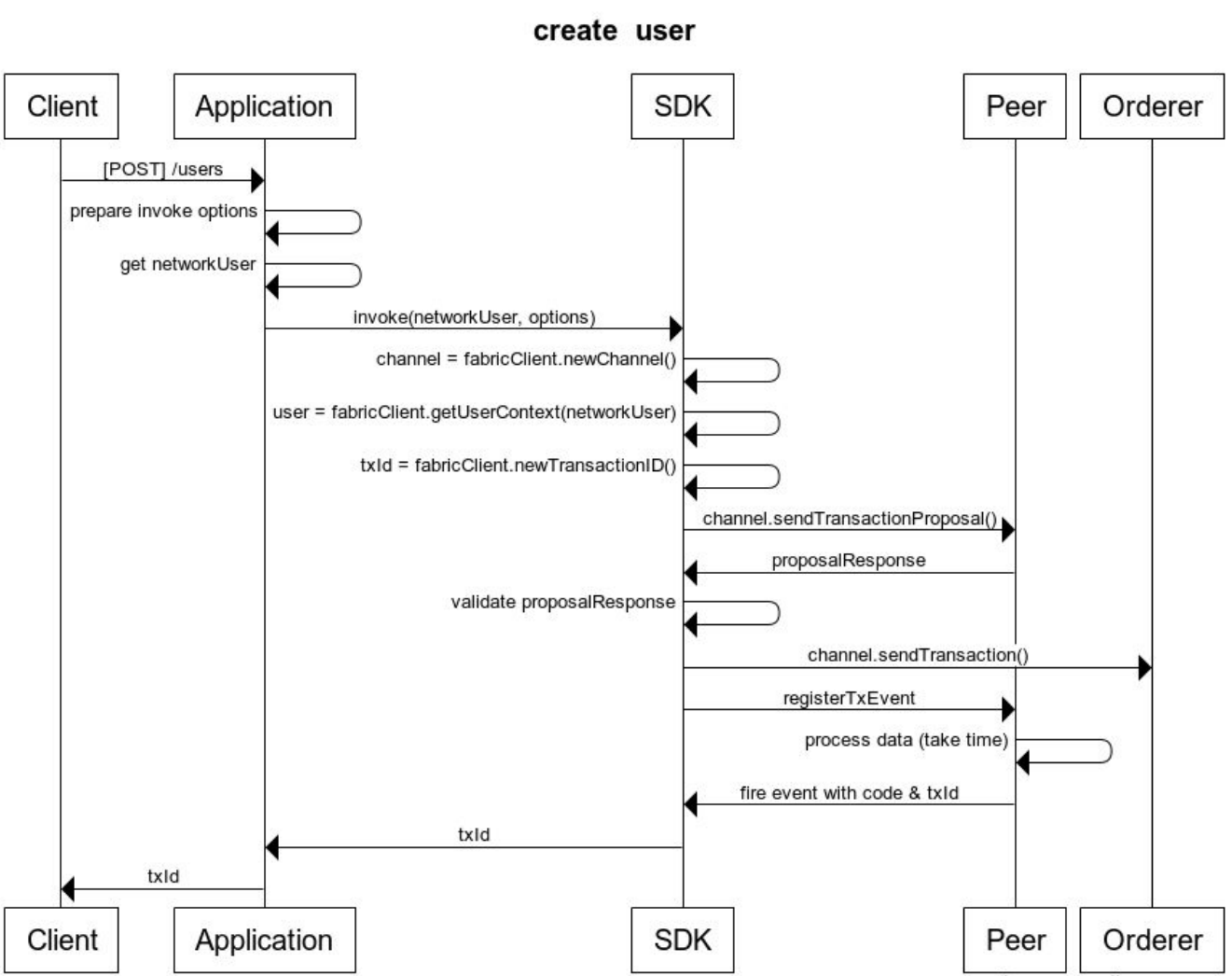
- Enroll Admin
- Register Network User
- Invoke
- Query

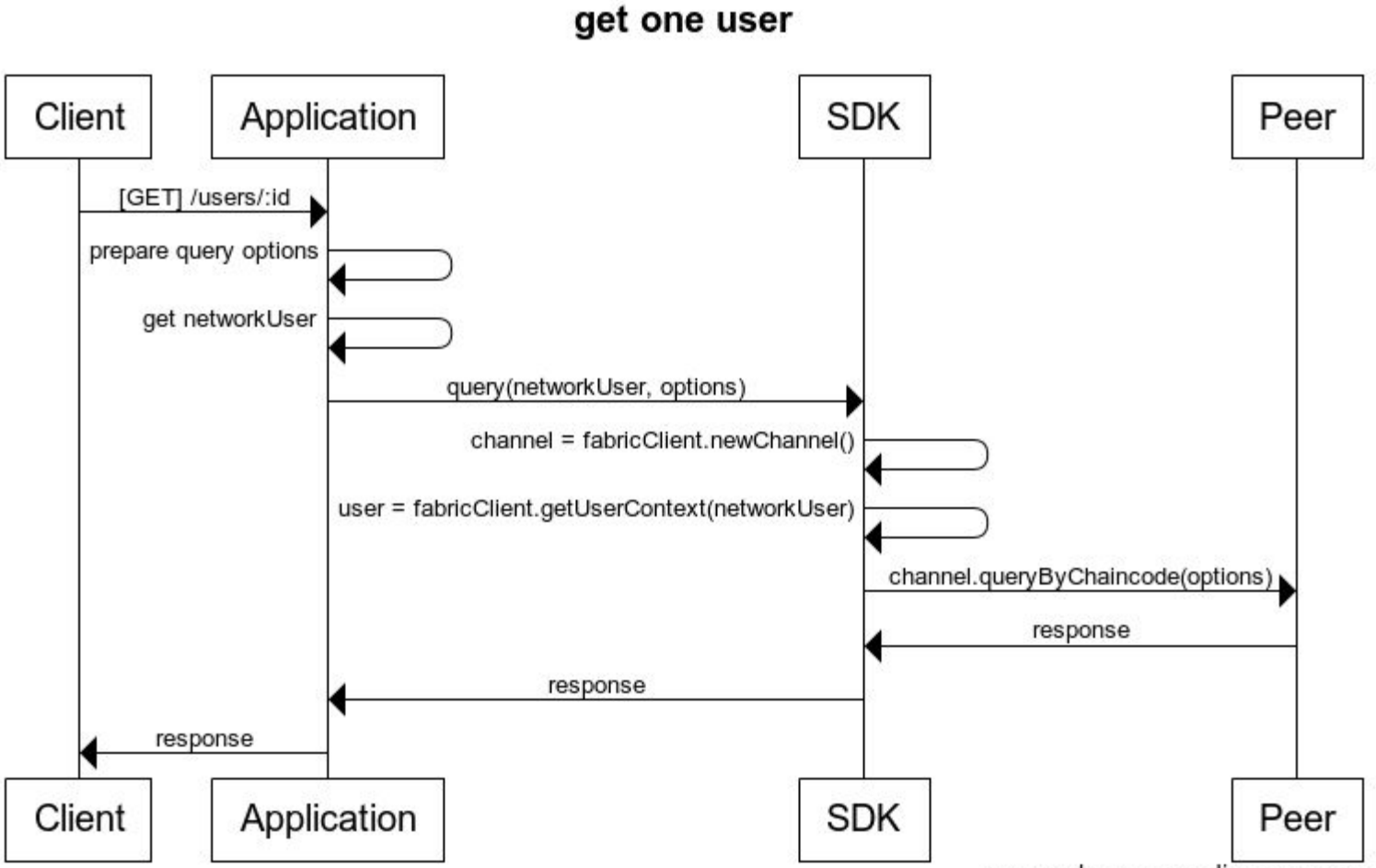
API Development (4)



API Development (5)




create user
Client
Application
SDK
Peer
Orderer
[POST] /users
prepare invoke options
get networkUser
invoke(networkUser, options)
channel = fabricClient.newChannel()
user = fabricClient.getUserContext(networkUser)
txId = fabricClient.newTransactionID()
channel.sendTransactionProposal()
proposalResponse
validate proposalResponse
channel.sendTransaction()
registerTxEvent
process data (take time)
fire event with code & txId
txId
txId
Client
Application
SDK
Peer
Orderer
www.websequencediagrams.com


get one user
Client
Application
SDK
Peer
[GET] /users/:id
prepare query options
get networkUser
query(networkUser, options)
channel = fabricClient.newChannel()
user = fabricClient.getUserContext(networkUser)
channel.queryByChaincode(options)
response
response
response
Client
Application
SDK
Peer
www.websequencediagrams.com

Enjoy your first !!

Hyperledger Fabric

Q&A

We are

KBank

KBTG
KASIKORN
BUSINESS-TECHNOLOGY GROUP

We use

node
docker

HYPERLEDGER
FABRIC