NOTE: The community has decided to focus on the language SDKs so this design and the related project are no longer actively being worked upon.

Table of Contents

Design of Hyperledger Fabric REST API

Version	Date	State	Description
0.1		Draft	First skeleton design
0.5		Draft	Base REST API design on use cases and not SDK functions
0.6	7 Apr 2017	Draft	Update security, use cases, and HTTP response codes sections
0.7	27 Apr 2017	Draft	Update title, references, REST endpoints, updates from comments
0.8	23 May 2017	Draft	Added use case, responses to comments, and updated endpoints
0.9		In progress	More information on error handling, link to implementation in external references, refine use case D requirements
…			…

Authors

- Martin Cocks, IBM
  Linux Foundation ID: MartinC
  Email: cocksmar@uk.ibm.com
- Chris Poole, IBM
  Linux Foundation ID: chrispoole

Email: chrispoole@uk.ibm.com

Notes for Readers

??? is used where more information or research is needed.
Discussion about the design should be recorded as review comments on this document or within work item FAB-156.

Design to-do

- Validate endpoints with community
- Swagger doc for proposed API
- Further security design

Prerequisites

For the REST API to work it will need the following software to be configured:

- Hyperledger Fabric v1.0
- Loopback v3.0
- fabric-sdk-node
- A Hyperledger Fabric v1.0 network to connect to.

Requirements

- The REST API must be stateless
  - No request may rely on state being created in the REST server from a previous request.
  - The server may contain state from Fabric that is common between all requests and it may store that state during boot or on first reference.
- The REST API must be securable
- The API must be in a RESTful style that acts upon resources.

(Very) High Level Architecture

All connections from API consumers will be over HTTPS, into a Loopback server hosting the Fabric SDK for Node. Here, as usual, GRPC will be employed as the transport to a Hyperledger Fabric peer.

In the context of a particular Fabric Network (consisting of peers and orderers), there may be 1 or more REST endpoints supporting the API described here operating on it. Each REST endpoint will provide a view of the resources within the Fabric Network from its point of view - thus a REST client should consistently use a chosen REST endpoint. Put another way, REST endpoints operating on a given Fabric Network are independent of each other and a client's view of resources is valid only for the REST endpoint that provided them. One example of this is a transaction that has been ~~run~~ committed on one peer but not yet run on

another peer. The REST server associated with the first peer would show a different query result to the REST server associated with the second peer.

An instance of a REST API server will be associated with a specific peer. The assumption being that each company involved in a blockchain network will need to administer which REST API users (which "company IDs") may access the blockchain under which specific blockchain IDs. This configuration should not be visible to other users of the blockchain. Exactly how a "company ID" can be mapped to a blockchain ID will be defined in the security design section. A company may have multiple peers to provide both High Availability (HA) and Continuous Availability (CA).

Design Principles

- The REST API should be defined by a Swagger document to aid consumption
- For REST, PUT should be used for update and POST for creation.
- The API should meet the requirements defined in the use cases.
  - It will not be a 1:1 mapping of gRPC functionality.
  - It will not be a 1:1 mapping of the SDK functionality.
- A REST URI should uniquely identify a resource or set of resources.
  - It should not contain any verbs.
- The API document could be generated by loopback from a loopback model
- The API document could be handcrafted and consumed by loopback to create a model
- Example output from command `lb swagger handcrafted.json`

```
/**
 * Creates a new `Chain` instance with default values.
 * @param {string} chainName Name of the Chain instance to create
 * @callback {Function} callback Callback function
 * @param {Error|string} err Error object
 * @param {chain} result Result object
 */
SwaggerApi.postChainChainName = function(chainName, callback) {
  // Replace the code below with your implementation.
  // Please make sure the callback is invoked.
  process.nextTick(function() {
    var err = new Error('Not implemented');
    callback(err);
  });
}
```

The API doc/loopback model will be created based upon the information in the REST API function summary section below. Validation of parameters will be left to the SDK where possible: duplication of validation code could lead to inconsistencies in what is allowed

in different layers. Validation errors should be passed back to the REST client where possible and be logged in the server.

REST API Versioning

The versioning scheme (within the npm bundle) will use standard `major.minor.micro` syntax. The version of the REST API will be reflected in the base path of the URL, e.g., `/fabric/major.minor/`, but excluding the micro version.

REST API Use Cases

We have identified four end user personas as consumers of this API. Each user may want to "Update Fabric from a …". We have added a fifth case [E] to record the network administration application requirements that have come from the community to hopefully feed into a follow on work item.

[A] Business/SoR Application

This is an application that just wants to store and read updates from an appropriate data store. It does not want to deal with the details of how the data store works.

- Run this update, tell me if it worked.
  - The application will still need to know the name of the "Smart Contract" to use and the name of the channel.
- Query data stored in the blockchain.
- Identify as user A, have the server map it to the appropriate blockchain user.
- Fabric entities the client application needs to know about
  - Channel
  - Chaincode ID (The smart contract to invoke)
  - Data stored in the blockchain (For queries)

[B] "Blockchain aware" Application

This is an application that wants to store and read updates from a blockchain. It wants more granular control over reading and putting data into a blockchain.

- All [A] and the next bullet points.
- Query and invoke on blockchain
- Might target specific peers for endorsement
- Handles each phase of driving a transaction
- Maintains state
- Identify as user A, have the server map it to the appropriate blockchain user.
- Identify as blockchain user B.
- Fabric entities the client application needs to know about
  - Channel

  - Peers (if targeting a specific set of peers for endorsement)
  - Orderers (if targeting a specific set of orderers)
  - Chaincode ID (The smart contract to invoke)
  - Data stored in the blockchain (For queries)

[C] Build System Application

This application wants to update chaincode in a channel. This might be done as part of a devOps pipeline.

- Deploy/Update chaincode
- Query chaincode
- Identify as user A, have the server map it to the appropriate blockchain user.
- Identify as blockchain user X.
- Fabric entities the client application needs to know about
  - Channel
  - Peers (if targeting a specific set of peers for endorsement)
  - Orderers (if targeting a specific set of orderers)
  - Chaincode ID (The smart contract to invoke)
  - Chaincode
  - Data stored in the blockchain (For querying deployed chaincode)

[D] Channel Administration Application

This application manages the peers and orderers that channels use.

- Create and update channels.
  - <added 22 Aug 2017> This requires signatures from each of the orgs involved in the channel. In a real world scenario it is unlikely that a single REST server will be able to sign for more than one org. May need an additional endpoint to return a signature for a config file.
- Identify as user A, have the server map it to the appropriate blockchain user.
- Identify as blockchain user Y.
- Fabric entities the client application needs to know about
  - Channel
  - Peers
  - Orderers

[E]<added in v0.8> Network Administration Application

This is out of scope for the initial implementation but I want to record the requirements for a potential future work item that may want to build upon the capabilities introduced here. The roles involved in the following tasks are user management, peer management, and channel management.

- Create and update users in the Fabric CA authority (authorities??? Can different CAs be used by different Channels?)
- Create/update/delete a Peer in the network
- Create/update/delete a Channel in the network
- Fabric entities the client application needs to know about
  - Channels
  - Peers
  - Users
  - CA
  - ???

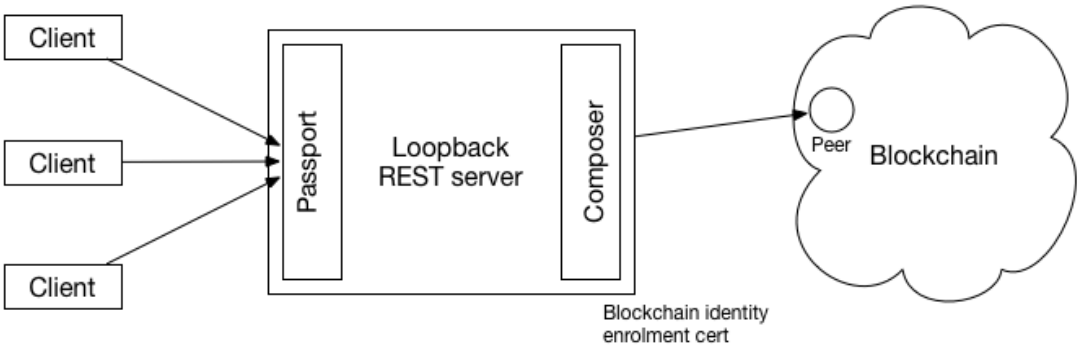
Fabric Documented Use Cases

See use cases.

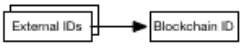
Security Design

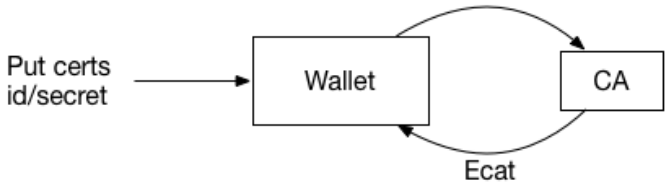
Client and user creation and configuration will not be exposed over REST.

Hyperledger Composer intends to use *Passport* to do the authentication of REST clients. For compatibility purposes this REST API exposed through loopback should do the same. Basic auth and client certs should be supported for compatibility with users of the API: this should be transparent and handled by Passport. High level architecture is described below:



External ID	Blockchain ID
johnsmith@github	johnsmith@biz.net





Key points: The REST API provider will be the client from the SDK's perspective in the initial (prototyping) phase. One fabric user enrollment certificate will be used to drive all requests into the fabric-sdk—at least initially. The client and associated credentials to use will be configured in the REST API provider set up, a keyValueStore will probably also be part of the configuration. (i.e., the wallet. This component is likely to be required by Hyperledger Composer too, so a shared service employing its own APIs can be used here.) The REST API provider **MUST** require clients to authenticate their identity so that only authorized users are able to use the REST API and drive SDK functions as the configured client. When a REST request is translated into an SDK function call the configured client will be used when required.

Hyperledger Composer have similar challenges with securing REST APIs for their business networks. We intend to work with them and arrive at an API that looks and feels similar, as much as is appropriate.

Handling Asynchronous Interaction with Fabric

For the first phase the REST calls will be synchronous and will block until the Promise is resolved and the result will be returned on the call. If a client request times out

then the information returned in the promise will be lost to them.

To resolve the issue of a timeout causing context to be lost a future phase will look at the options for allowing the REST calls to be asynchronous, either through returning a token for a promise or providing a register callback mechanism. To allow the caller to determine whether or not to run a request synchronously an optional query parameter will be added to the command(s) that can be run asynchronously.

REST API Design Considerations

A stateless REST API can't use an instance reference (in the url or message) for an SDK object it acts upon as the object won't exist in the REST API provider's memory. The REST API may reference objects that exist within Fabric that the SDK may query.

To keep the REST API stateless requires passing serialized objects in some requests and responses. An example would be a proposal response that would need to be passed on a "commit" request to the orderers.

To keep the interface of the REST API cleaner this API will query data from Hyperledger Fabric when possible rather than require it to be passed on a request/response. The performance of querying information for each REST request can be considered in a follow on phase. Techniques such as caching can be considered at that point.

This design does not consider server specific configuration to be state as it can be used across unrelated requests and is not updated by the REST API functions.

PUT is used for updates and POST is used for create.

REST API Function Summary

Related Information

Fabric Client information, and mapping from a REST client will be configured in the loopback REST API provider application.

Lifecycle API

- Run a transaction end to end [A,B]
  - `POST /channels/{channelName}/transactions`
  - Related SDK function: *Chain.createTransaction() does not appear to exist in the SDK*
  - Related SDK function: *Chain.sendTransactionProposal(request), Chain.sendTransaction(request)*
  - input: transaction data

  - output: transaction result
- Send a transaction to the channel's orderer(s) [B]
  - `POST /channels/{channelName}/transactions`
  - NOTE: This is the same URL as end to end, action will be determined by payload content.
  - Related SDK function: *Chain.sendTransaction(request)*
  - input: transaction data and, endorsement results
  - output: transaction result
  - **This requires an improvement to the SDK or to store sdk class instances between requests which requires session management in the REST server**
    - https://jira.hyperledger.org/browse/FAB-5156 raised to request sdk improvement.
    - Session management should not be required when the input data should be sufficient.
- ~~Endorse a transaction on the channel's peers [B]~~
  - ~~`PUT /channels/{channelName}/proposal`~~
  - ~~Related SDK function: *Chain.sendTransactionProposal(request)*~~
  - ~~optional query string: peers=[i,j,k]~~
  - ~~input: transaction result~~
  - ~~output: array of endorsement results~~
- Endorse a transaction on the channel's peers [B]
  - NOTE: This is an RPC style call as it does not act on a resource
    - *The alternative is to do REST style calls to an ephemeral proposal. See previous main bullet*
    - *This call can have no asynchronous option as there is nothing created that can be queried to determine if the request completes, fails or is still in-progress.*
    - ***??? How long can this call take? Is sync only okay?***
  - POST /channels/{channelName}/endorse
  - optional query string: peers=[i,j,k]
    - Use to target specific peers.
  - input: Transaction data
  - output: Array of endorsements
- Install new chaincode onto a peer [C,D]
  - `POST /chaincodes?peers=[i,j,k]`
  - input: chaincode id
  - output: result
- Install an update to chain code on a peer [C,D]
  - `PUT /chaincodes?peers=[i,j,k]`
  - input: chaincode id

  - output: result
- Endorse new chaincode with the channel's peers [C,D]
  - `POST /channels/{channelName}/endorse`
  - NOTE: Same url as endorsing a transaction. Same concerns apply about RPC style.
  - optional query string: peers=[i,j,k]
    - Use to target specific peers.
  - input: chaincode input
  - output: endorsement result
- Instantiate new chaincode (end to end) [C]
  - `POST /channels/{channelName}/chaincodes`
  - input: chaincode id
  - output: chaincode result
- Instantiate updated chaincode (end to end) [C]
  - `PUT /channels/{channelName}/chaincodes`
  - input: chaincode id
  - output: chaincode result
- Instantiate new chaincode with the channel's orderers [C]
  - `POST /channels/{channelName}/chaincodes`
  - NOTE: Same as instantiate end to end, action will be determined by payload.
  - input: chaincode id, endorsement results
  - output: chaincode result
- Instantiate updated chaincode with the channel's orderers [C]
  - `PUT /channels/{channelName}/chaincodes`
  - NOTE: Same as instantiate update end to end, action will be determined by payload.
  - input: chaincode id, endorsement results
  - output: chaincode result

Management API

- Retrieve all known channels. [B,C,D]
  - `GET /channels`
  - Related SDK function: *Client.queryChannels(peer)*
  - input: *none*
  - output: array of channel names
- Retrieve information about a named channel. [B,C,D]
  - `GET /channels/{channelName}`
  - Related SDK function: *Client.queryChainInfo(name, peers)* and maybe *Chain.initialize()*

  - input: *none*
  - output: Detailed Channel information
- Create a new named channel [D]
  - `POST /channels/{channelName}`
  - Related SDK function: *Client.createChannel(request)*
  - input: Detailed channel information
  - output: Result
- Update a named channel [D]
  - `PUT /channels/{channelName}`
  - Related SDK function: *Client.updateChain()* Not implemented yet.
  - input: Detailed channel information
  - output: Result
- Join a Peer to a channel [D]
  - `POST /channels/{channelName}/peers`
  - input: peer url
  - output: Result
- *Remove a Peer from a channel [D]*
  - *NOTE: this is not implementable at the moment with Fabric 1.0*
- *NOTE: Orderers to use will be configured in the REST server configuration not over the REST API.*

Query API

- Query by resource - transaction [A,B,C,D]
  - `GET /channels/{channelName}/transactions/{transactionID}`
  - `input: none`
  - `output: transaction entry`
- Query by resource - block [B,D]
  - `GET /channels/{channelName}/blocks`
  - `query string - one of:`
    - `blockID=id`
    - `blockHash=hash`
  - *Chain.queryBlock(blockNumber) or Chain.queryBlockByHash(blockHash)*
  - input: *none*
  - output: block information
- Query by resource - chaincode [B,C,D]
  - `Installed (on peer)`
    - `GET /chaincodes/{id}?peers=[i,j,k]`
  - `Instantiated (in channel)`
    - `GET /channels/{channelName}/chaincodes`
    - `GET /channels/{channelName}/chaincodes/{id}`

  - Related SDK function: *Chain.queryInstalledChaincodes(peer) or Chain.queryInstantiatedChaincodes()*
  - `input: none`
  - `output: chaincode information`
- Query by ledger [B, D]
  - GET /channels/`{channelName}/ledger`
  - `query string - one of:`
    - `chaincode=name`
    - `blockID=id`
    - `blockHash=hash`
    - `transactionID=id`
  - `input: Any`
  - `output: Identifier String and byte array containing the result`
- Query by database [B]
  - Pass a query to the database used to contain the world state on the peer.
  - Rich queries not supported by SDK yet.

Error Handling

What errors are expected and how should they be handled?

Error	**How to handle it**
Connection, client to REST server	Standard processing of loopback, passport, or the client stack depending upon where the failure occurs.
Connection, REST server to Peer	Return a 5xx error. Log the problem in the REST server. This is likely to be a server configuration problem or a peer being offline.
Connection, REST server to Orderer	Return a 5xx error. Log the problem in the REST server. This is likely to be a server configuration problem or the orderer being offline.
Endorsement, all requests failed	Return a 4xx error and log in the REST server
Endorsement, some requests failed	If enough pass to meet endorsement policy then it should not be treated as a failure.

	If less than enough to meet endorsement policy then return a 4xx with endorsement responses and log it in the rest server. ***What if some or all of the failures are connection failures?***
Ordering failed	Return a 5xx, log the problem in the REST server.
Input validation failures	Return a 4xx and log in the REST server

HTTP response codes

- **200** OK => Use for all requests processed cleanly
- **201** Created => Not used
- **202** Accepted => Use this one when asynchronous options are added
- **204** No Content => Use when validly no content is returned
- **400** Bad Request => Use when missing fields or badly formatted data
- **401** Unauthorized => Should be returned when authorization fails
- **404** Not Found => Return when a resource identified in the URL does not exist.
- **408** Request Timeout => Should be returned by the loopback server comms layer. ??? Where is the server side timeout configured?
- **409** Conflict => Use when the orderer says no ???

Swagger definition of the API

- Refer to the working document available here: https://github.com/hyperledger/fabric-sdk-rest/blob/master/fabric-rest.yaml

Loopback Specifics

Connector Required Methods

This connector will wrapper fabric-sdk-node and connect to fabric v1. It will require the datasource to pass the certificate store information needed to create the client and user classes that it will use when connecting to fabric.

The four functions the connector must implement and export are.

Connector Method	Intent
initialize(datasource, callback)	Configure the user credentials and certificate store to use for connections. Need a primary peer to be configured to query it for all channels.

connect()	Validate connection credentials. The SDK establishes it's own connections when needed and so no permanent connection can be made here.
disconnect()	Null op.
ping()	(Optional) Check fabric is still there

It will also need to implement a set of methods to access the channels stored in fabric, including utility functions to create SDK objects required by calls. A likely set of methods will be the following, although the names and number may change during implementation, as needed, the intent is the key thing here.

Connector Method	Intent
setupSecurityContext(externalUser)	Use the authenticated external user to determine the blockchain user credentials to use when connecting to fabric
lookupChannel(channelName)	Get details of the named channel from the peer
runQuery()	Run a query on the primary peer
runTransaction()	Run a transaction on the primary peer
runTransactionEnd2End()	Run, get endorsements, and send to orderer a transaction
getEndorsement(peer)	Get endorsement from the named peer
sendToOrderer(orderer)	Send a transaction with its endorsements to an orderer
deployChaincode()	Deploy chaincode into a channel
updateChannel()	Update the peers/orderers/configuration of a channel

Packaging the Connector

There is naming defined by JIRA work item FAB-156 which proposes naming the connector as an npm installable module named `hfc-rest`. However as the connector will not do any REST itself and is more generic a better alternative should be used. Also the sdk module has been renamed to be `fabric-client` and no longer uses `hfc`.

Loopback has a naming convention where connectors are loopback-connector-*NAME* and they are referenced in the datasources.json file by just the *NAME*. Therefore we propose using `loopback-connector-fabric` instead.

For npm install create a `package.json` with the following contents. The contents will be refined during implementation.

```
{
  "name": "loopback-connector-fabric",
  "version": "1.0.0",
```

```
  "description": "LoopBack Connector for Hyperledger Fabric v1.0",
  "engines": {
    "node": ">=4"
  },
  "keywords": [
    "StrongLoop",
    "LoopBack",
    "Hyperledger",
    "Fabric",
    "DataSource",
    "Connector"
  ],
  "main": "index.js",
  "dependencies": {
    "loopback-connector": "^3.0.0"
  }
}
```

Where `index.js` includes

```
'use strict';

module.exports = require('./lib/fabricconnector.js');
```

REST provider configuration (loopback datasource)

Aim for configuration to be similar to Hyperledger Composer.

- EITHER
  - Name the fabric user.
  - Secret for the fabric user.
- OR
  - Define the key value store where the user certificates are stored on the loopback server. Used to map external users to fabric users
- Array of Peers to connect to. This is the array of Peers that
- Array of orderers. One or more orderers that the REST server could send the order transaction/chaincode request to.

In file `server/datasources.json`

```
"fabricDataSource": {
  "name": "fabricDataSource",
  "connector": "fabric",
  "peers": ["NOT-SET"],
  "orderers": ["NOT-SET"]
}
```

`TODO detailed configuration`

Hyperledger Composer stores information about the connection profile and the name of the business network to connect to. These relate to Hyperledger Composer's view of the fabric and so are not needed in the rest SDK layer.

Loopback Model Configuration

The loopback model is used to define the external REST API. A skeleton model can be generated from a swagger document using the `lb swagger FILE` command.

Connect the model to the datasource by updating `server/model-config.json`

```
...
"swagger_fabric" : {
  "datasource" : "fabricDataSource",
  "public": true
},
...
```

Packaging the Loopback Model

Potentially package this part with the datasource as an npm module called `fabric-rest` that has a dependency on the `loopback-connector-fabric`.

External References

- REST SDK Source code on github
- REST SDK source on gerrit
- JIRA work item FAB-156
- SDK source on github
- SDK source on gerrit
- Node SDK, reference
- *Fabric v1.0 SDK design doc*
- **This document is out of date** (note added 20 April 2017)
- Node SDK, Read the docs
- Loopback "Building a Connector" documentation
- Fabric use cases

Community Quotes

This section is recording anything from the community chat that might be useful to reference when looking at future design enhancements. TODO: record this information in a more appropriate place.

Source chat.hyperledger.org

User	Date	Related comment	Channel
sajz	2017-3-2 10:22	… “also the SDK is handling the certificate of the Client locally, and if we use our API, we really don’t want to ”break" the identity-proof link by receiving request from the Client through our API on a server and then interacting (grpc) with the chaincode with its certificate or ours (1st case we will handle their certificate but we would rather not, 2nd case we will lose the traceability of whom made the tx“	fabric-sdk

…