Hyperledger Fabric Python SDK

Introduction

- **Name:** Kiv Chen
- **Location:** Liverpool, United Kingdom
- **University:** University of Liverpool
- **Mentor(s):** Dixing Xu, Baohua Yang, Guillaume Cisco, Wang Dong
- **Hyperledger Project:** Support Decentralized Governance for Smart Contracts in Fabric Python SDK
Hyperledger Fabric Python SDK

Project Description: With the introduction of Fabric v2.x, a more decentralized way of chaincode management is implemented. There are several improvements over the previous lifecycle and it requires several changes on the sdk. This project aims to support decentralized governance for smart contracts in fabric python sdk and add features such as private data sharing/verifying and external chaincode launcher. The projects will provide a user-friendly and easy-to-use tool for fabric developers and operators.
Project Objectives:

- Obj 1: Chaincode lifecycle management on sdk-py
- Obj 2: Align with new features of fabric 2.x
- Obj 3: Documentation on using fabric 2.x
Hyperledger Fabric Python SDK

Project Deliverables:

- Deliverable 1: Development of fabric 2.2+ Lifecycle 2.0 full support
- Deliverable 2: Documentation for fabric 2.2+ features
Hyperledger Fabric Python SDK

› Project Execution & Accomplishments:

› Get familiar with Fabric & Fabric SDK
› Fulfill and extend the decentralized support for Fabric 2.x
› Revise some previous APIs according to mentor’s suggestions
Chaincode Lifecycle Step 1 Setup

- Setup needed attributes ...
  - Name/Version
  - Sequence
  - Endorsement Policy
  - Validation configuration

- ... that influence the Fabric “citizens”
  - Client and Peers
  - Channels
  - Chaincode
Chaincode Lifecycle Step 2 Package

- Build a tar file from the source code files and metadata files
- Available to be sent to other organizations

```python
def package(self, source_path, label, dest_path=None, cc_type=CC_TYPE_GOLANG):
    """
    Package chaincode
    
    :param source_path: Path to the chaincode
    :param label: The package label contains a human-readable description of the package
    :param dest_path: Path with file name where package would be stored
    :param cc_type: Language the chaincode is written in (default "golang")
    :return: bytes of the packaged chaincode
    """
    metadata = {
        "path": source_path,
        "type": cc_type,
        "label": label
    }
    tar_bytes = lifecycle_package(package_chaincode(source_path, cc_type), metadata)
    if dest_path:
        with open(dest_path, "wb") as file:
            file.write(tar_bytes)
    return tar_bytes
```
Chaincode Lifecycle Step 3 Install

- Send the packaged code to a peer
- Save the hash value returned

```python
async def install(self, requestor, peers, packaged_cc=None):
    
    # Install chaincode to given peers by requestor role

    # param requestor: User role who issue the request
    # param peers: List of peer name and/or Peer to install
    # param packaged_cc: packaged chaincode
    # return: A dict representation of `InstallChaincodeResult`

    target_peers = self._client.get_target_peers(peers)

    tx_context = create_tx_context(requestor, requestor.cryptoSuite, TXProposalRequest())

    install_args = lp.InstallChaincodeArgs()
    install_args.chaincode_install_package = packaged_cc

    responses, proposal, header = utils.send_proposal(tx_context, target_peers, install_args, LC_INSTALL,
                                                       LIFECYCLE_CC)
    res = await asyncio.gather(*responses)
    return self.parse_proposal_res(res, lp.InstallChaincodeResult)
```
Chaincode Lifecycle Step 4 Approve for Organization

- Send a “approve chaincode definition for organization” chaincode lifecycle transaction to one peer in our organization
- Commit the transaction (send to orderer)

```python
async def approve_for_my_org(self, requestor, peers, channel, cc_version, package_id, signature_policy=None,
                             channel_config_policy=None, init_required=False, sequence=1, collections_config=None,
                             endorsement_plugin=None, validation_plugin=None, wait_for_event=True,
                             wait_for_event_timeout=DEFAULT_WAIT_FOR_EVENT_TIMEOUT):

    # Approve chaincode definition for current org
    ...
    return await self.chaincode_definition_operation(requestor, peers, channel, cc_version, package_id=package_id,
                                                     signature_policy=signature_policy,
                                                     channel_config_policy=channel_config_policy,
                                                     init_required=init_required, sequence=sequence,
                                                     collections_config=collections_config,
                                                     endorsement_plugin=endorsement_plugin,
                                                     validation_plugin=validation_plugin,
                                                     wait_for_event=wait_for_event,
                                                     wait_for_event_timeout=wait_for_event_timeout)
```
Chaincode Lifecycle Step 5 Commit

- Send a “commit definition chaincode for channel” chaincode lifecycle transaction to enough organizations
- Commit the transaction (send to orderer)
Chaincode Lifecycle Step 6 Init

- Invoke the chaincode

```python
async def initialize_chaincode(self):
    """
    Test initialising chaincode
    """
    logger.info("E2E: Chaincode initialisation start")
    chaincode = Chaincode(self.client, CC_NAME)
    org = "org1.example.com"
    org_admin = self.client.get_user(org, "Admin")
    args = ['a', '200', 'b', '300']
    res = await chaincode.invoke(org_admin, self.channel_name, ['peer0.' + org, 'peer1.' + org], args, fcn="Init", is_init=True, wait_for_event=True)
    self.assertEqual("", res, res)
    time.sleep(2)
    logger.info("E2E: Chaincode initialisation done")
```
Hyperledger Fabric Python SDK

```python
// Copyright the Hyperledger Fabric contributors. All rights reserved.

syntax = "proto3";

option java_package = "org.hyperledger.fabric.protos.peer.lifecycle";
option go_package = "github.com/hyperledger/fabric/protos/go(peer/lifecycle";

package lifecycle;
import "hfc/protos/peer/collection.proto";

// InstallChaincodeArgs is the message used as the argument to
// lifecycle.InstallChaincode.
message InstallChaincodeArgs {
  bytes chaincode_install_package = 1; // This should be a marshaled lifecycle.Chaincode
}

// InstallChaincode is the message returned by
// lifecycle.InstallChaincode.
message InstallChaincodeResult {
  string package_id = 1;
  string label = 2;
}

// QueryInstalledChaincodeArgs is the message used as arguments
// lifecycle.QueryInstalledChaincode.
message QueryInstalledChaincodeArgs {
  string package_id = 1;
}

// QueryInstalledChaincodeResult is the message returned by
// lifecycle.QueryInstalledChaincode. It returns a list of installed
// chaincodes, including a map of channel name to chaincode name and version
// pairs of chaincode definitions that reference this chaincode package.
message QueryInstalledChaincodeResult {
  message InstalledChaincode {
    string package_id = 1;
    string label = 2;
    map<string, References> references = 3;
  }
  message References {
    repeated Chaincode chaincodes = 1;
  }
  message Chaincode {
    string name = 1;
    string version = 2;
  }
}

repeated InstalledChaincode installed_chaincodes = 1;
```
Hyperledger Fabric Python SDK

Recommendations for future work:

- Add more chaincode examples
- More detailed documentation
- Track and support latest features of Fabric
Hyperledger Fabric Python SDK

Project Output or Results:

Code Available at:

https://github.com/hyperledger/fabric-sdk-py

Project Link:

https://wiki.hyperledger.org/display/INTERN/Support+Decentralized+Governance+for+Smart+Contracts+in+Fabric+Python+SDK
Insights Gained:

- Communication
  - Managing feedbacks, deliveries and expectations

- Programming Skills
  - The thing about and not about the written code

Take Aways:

- Documentation is important!
- The open-source workflow and paradigm
THANK YOU!