Preliminary Notes

The majority of this laboratory was inspired by the implementation of JusticeChain [1].

The blockchain ecosystem includes two major components: smart contract (chaincode) development and operations (infrastructure). We will learn how to create the chaincode to implement the system B4S QUC.

Contrarily to previous laboratories, you will take a look at the code and experiment with our system while you navigate through this guide. To do so, please clone the repository at, using the command: git clone https://github.com/hyperledger-labs/university-course, and navigate to support/Lab05 This lab uses carbon.sh to create some figures.

1 B4S QUC System

¡ Figure 1 illustrates the different actors in this system.

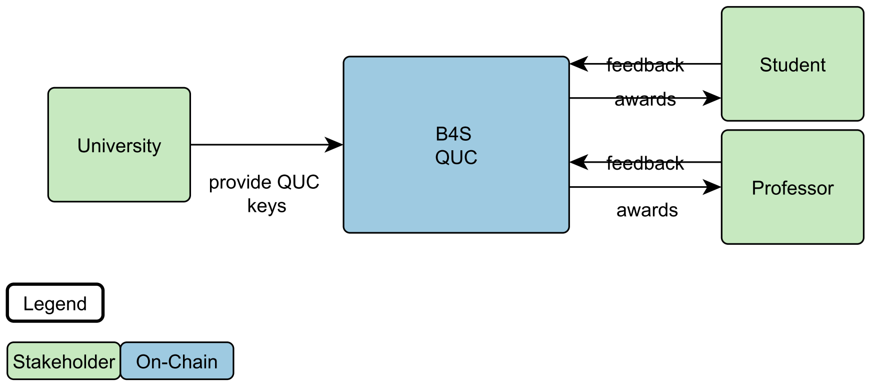


Figure 1: Actors in a B4S QUC blockchain

However, since Fabric can only hold a limited number of peer nodes, and for privacy purposes, both students and professors use proxies in the communications. In other words, students interact with B4S QUC via the students union of their university, while professors do the same via their department, as illustrated in Figure 2.

The students union and department web applications are not hosted by the blockchain (we call those components off-chain components). Therefore, the students and professors are blockchain users that use the students' union peer node and the professors' department peer node.

Next, we focus on each component of our system, namely the chaincode, the off-chain components (application, user interface), and the infrastructure.

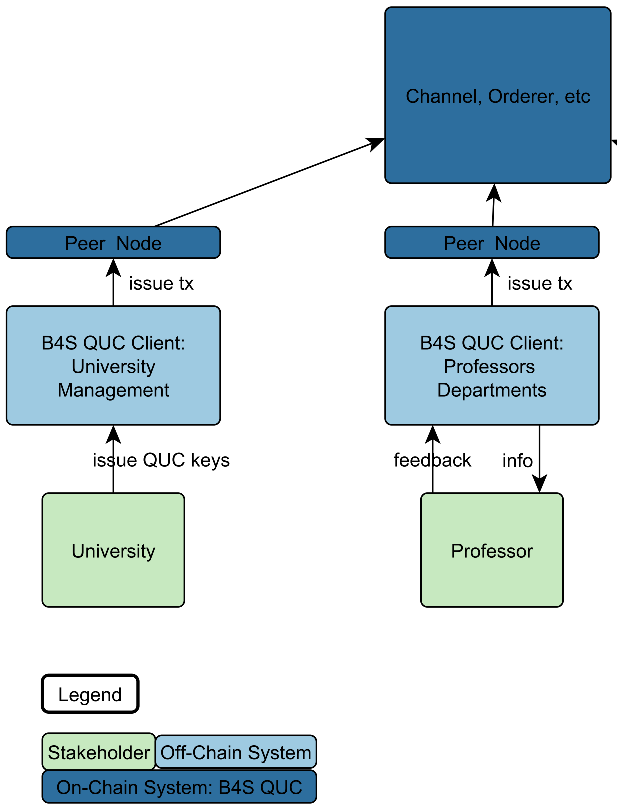


Figure 2: Blockchain and Blockchain Client components in a B4S QUC blockchain

1.1 States and transactions

A state is composed of the stored attributes. Let's analyze our use case in a little more detail. B4S QUC participants such as students and professors from Técnico Lisboa use transactions to achieve their business objectives.

A Fabric state is implemented as a key/value pair. The value encodes the object properties in a format that captures the objects' multiple properties, typically JSON. The ledger database can support advanced query operations against these properties, which helps sophisticated object retrieval. A key can be simple (formed by, for example, the id of a feedback item) or composed (the concatenation of the feedback item id + student id). A key, therefore, identifies a state unequivocally. We model a state like this in the following manner:

```
OBJECT KEY
FIELD: VALUE
FIELD: VALUE
FIELD: VALUE
...
FIELD: VALUE
```

Let's examine the an example of a feedback item:

```
FeedbackItem 011a8b02-e9b7-4023-9be6-4814abf5664c
Creator: Student
Course ID: 197b067b-44fc-43b1-8f8c-1658de7ec185 0808f130-1ba8-4712-991a-250080e2e44c
FeedbackParameter#1: 5
FeedbackParameter#2: 9
FeedbackParameter#3: 2
Produced QUC: null
Current State: Completed
```

The creator of a feedback item may be a student or a professor. Each feedback item is uniquely identified by a Universally unique identifier v4 (UUIDv4) string. Essentially, a UUID is a mechanism to identify information composed of a 128-bit string. The probability of finding a duplicate within 103 trillion version-4 UUIDs is one in a billion. FeedbackParameters are the QUC parameters, which we omit for simplicity (an example would be: Professor_Arrives_In_Time). The produced QUC attribute refers to the QUC in which that feedback item participated. As the feedback hasn't been processed, no QUC was generated with that feedback item.

We can differentiate on four types of feedback: feedback from students towards a course, feedback from students towards the teaching staff; feedback from professors toward the course; feedback from the students representative towards courses, and teaching staff. However, for illustration and brevity, we focus on the feedback from a student towards a course. Several feedback items are used to produce a QUC score, which refers to a course and its teaching staff:

```
QUC c38eea64-f494-49d0-8245-3966f02d8e4a
University: University 0177545f-2730-496d-8971-6204ff804979
Teaching Staff: Professor d023fff3-1613-47db-a4ab-b91925ae1a36, Professor 90ea3765-13
Course ID: Course b143ef55-64b7-41b5-b91b-0b8161f98ef6

Course QUC Scores:
    QUC Score#1: 8
    QUC Score#2: 4
    QUC Score#3: 8
    [...]
Teaching Staff QUC Scores:
    [...]
General Score: 6

Current State: Created
```

An example of a state for a course is:

```
Course b143ef55-64b7-41b5-b91b-0b8161f98ef6
University: University 0177545f-2730-496d-8971-6204ff804979
Teaching Staff: Professor d023fff3-1613-47db-a4ab-b91925ae1a36, Professor 90ea3765-13
Students: [...]
Metadata: [...]
Current State: Created
```

A course is taught by teaching staff, which later can also provide feedback from the course. Students allow a student to prove that they were enrolled in that course, thus providing feedback.

An example of student representation is:

```
Student eefe83c5-555b-4804-a027-182afd6957df
University: University 0177545f-2730-496d-8971-6204ff804979
```

Students unions track their student's ID, allowing them to trace the real identity in case of legal dispute.

An example of professor representation is:

```
Professor 43009bef-7c5d-4000-8861-4b2c7805a4e5
University: University 0177545f-2730-496d-8971-6204ff804979
Courses: Course b143ef55-64b7-41b5-b91b-0b8161f98ef6
QUCs: [...]]
```

QUCs are the list of QUC items that refer to that professor.

Hyperledger Fabric distributed ledger contains a world state (value of all objects) and a blockchain that records all transactions that resulted in the current world state. Fabric only accepts a transaction as valid, given that it collected enough endorser signatures.

Feedback items are then conceptual objects of value, modeled as states, whose lifecycle transitions are described by transactions. An effective analysis of states and transactions is an essential starting point for a successful implementation.

We can represent the life cycle of a feedback item using a state transition diagram:

A feedback item is created by either a student or a professor via a createFeedback transaction, issued via the students union or professors department, respectively. The course is taught during the semester. After it is completed, a fillFeedback transaction can be issued, marking feedback as completed. Finally, when QUC scores are being produced, processFeedback is called, marking the feedback as processed (accounting for producing a QUC score).

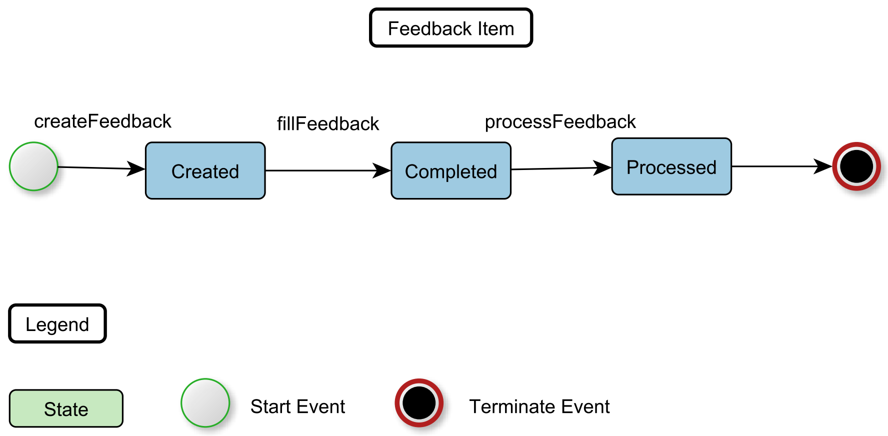


Figure 3: State transition diagram for the feedback item lifecycle. States are changed via transactions, issued by different actors.

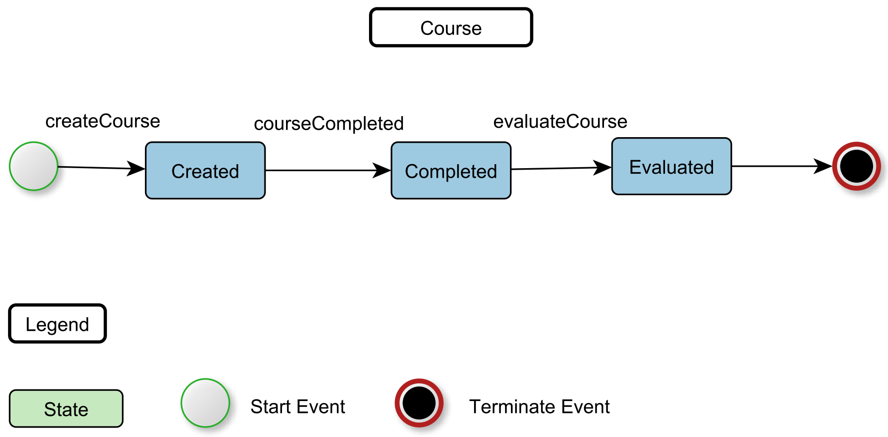


Figure 4: State transition diagram for the course lifecycle.

We can represent the life cycle of a course using a state transition diagram:

The lifecycle of a course comprises its creation by the university or a professor (createCourse), its normal operation and its finishing (courseCompleted), and its evaluation (evaluateCourse). The university calls evaluateCourse when it is deemed to be finished. At that time, feedback items are processed, and a QUC score is calculated for both the course and its teaching staff. The state diagram describes how feedback items and courses change over time and how specific transactions govern the life cycle transitions.

The QUC score is created when after a course is deemed complete and after the deadline for the QUCs filling is achieved.

The relationship between all objects stored on the ledger is as follows:

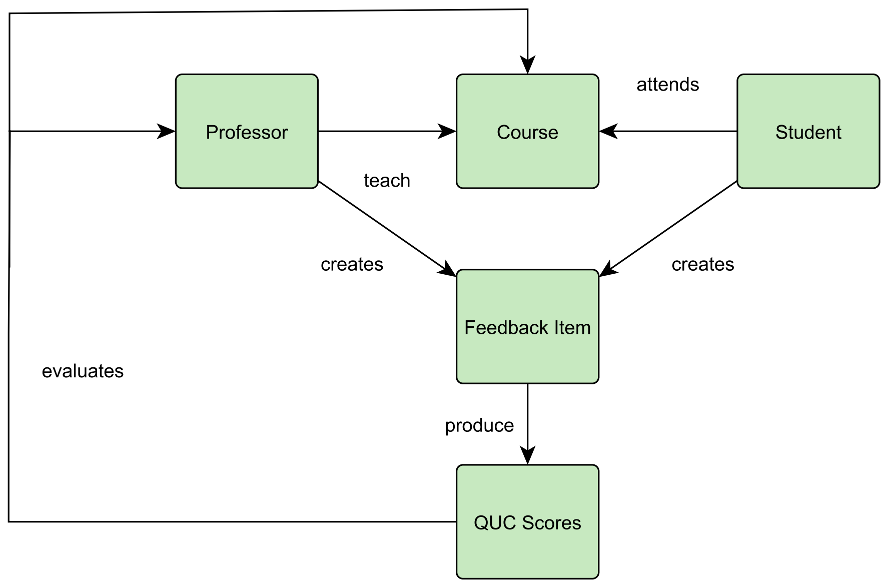


Figure 5: Mental model of the concepts involved in B4S QUC

In Hyperledger Fabric, smart contracts implement transaction logic that transition objects between their different states. The different states of feedback items, courses, etc., are actually held in the ledger world state, as illustrated in the next figure:

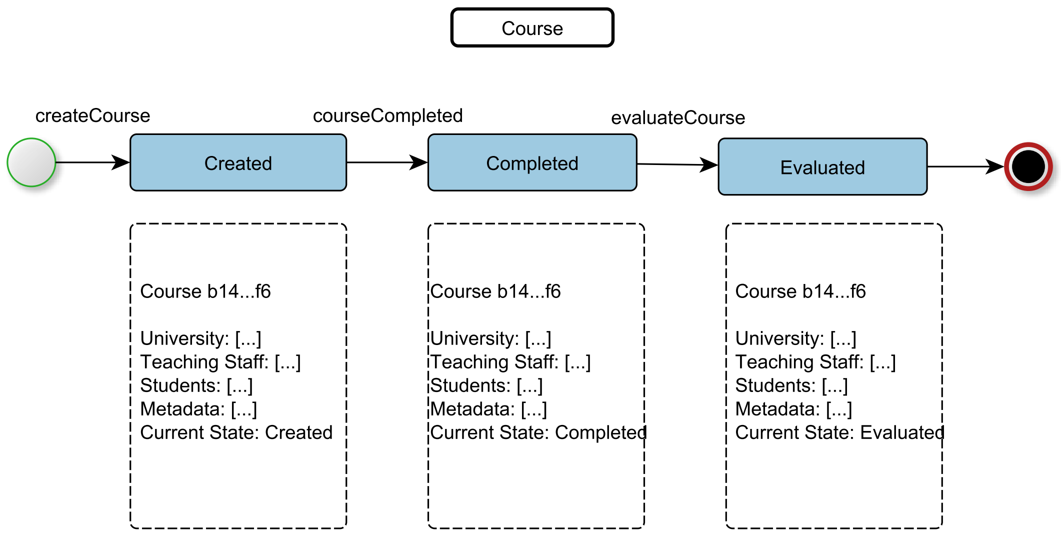


Figure 6: A state representing a course is created as a result of a createCourse transaction. State transitions occur as a result of different transactions.

You're now in a great place to translate these ideas into a smart contract. Don't worry if your programming is a little rusty. We'll provide tips and pointers to understand the code.

1.2 Simplifications

To simplify our use case, we perform some simplifications:

- only consider students' feedback on a course (professors feedback can be implemented in a similar way
- we assume that for QUC calculation purposes, there is only one parameter "feedbackParameters", which represents all the necessary components for feedback
- we use a simplified network, *test-network*, with two organizations: university and students. This network is based on the test-network from fabric-samples[1] [2](see Figure 7.

As an exercise, you might complete some parts of this project, market with `TODO EXERCISE`

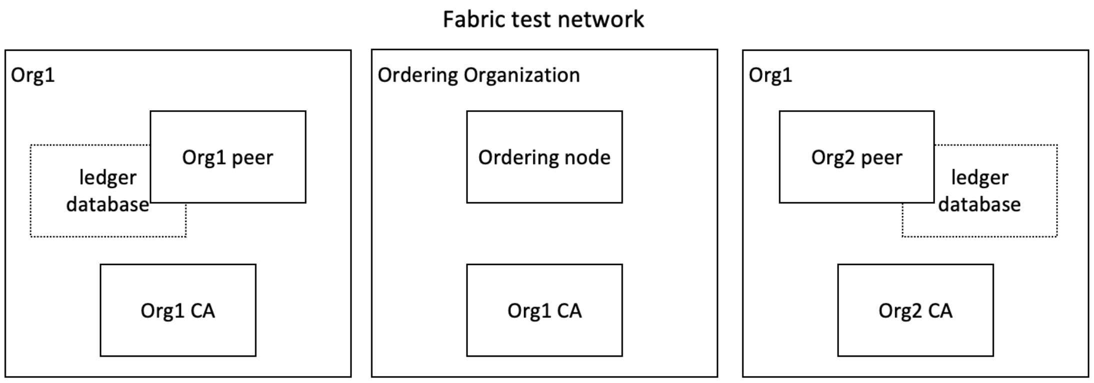


Figure 7: Our network, compressed of Org1, Org2 and an orderer.

1.3 Smart Contracts & Chaincode

Smart contracts are the core of a blockchain. They allow generating new information that is appended to the ledger. Every smart contract has a name that uniquely identifies it within a chaincode. Applications access a particular smart contract within a chaincode using its contract name. Essentially, smart contracts are high-level abstractions, encapsulated by chaincode, made available to developers.

[1]https://hyperledger-fabric.readthedocs.io/en/release-2.2/tutorial/commercial_paper.html
[2]https://github.com/hyperledger/fabric-samples

A chaincode is a generic container for deploying code to a Hyperledger Fabric blockchain network. It groups smart contracts and can also be used for low-level programming of the Fabric network. Thus, smart contracts manage transactions, whereas chaincode governs the management of smart contracts.

In most cases, a chaincode will only have one smart contract defined within it. However, it can make sense to keep related smart contracts together in a single chaincode.

1.3.1 Chaincode organization

In our case, we have one chaincode: *T-QUC*. The T-QUC chaincode manages courses, feedback items, a QUCs, and manages the creation and operations regarding the different actors on the network (universities, students, and professors).

The T-QUC chaincode contains several smart contracts:

- Course + University (Logistics Smart Contract)
- QUC + Feedback Items (QUC Smart Contract)
- Student + Professor (Users Smart Contract)

For simplicity, we focus on the students' feedback (the professors' feedback can be implemented in a very similar fashion).

1.3.2 Developing Chaincode

Chaincode development can be seen as object-oriented programming, with some useful abstractions. Our abstraction regarding the blockchain is implemented through the State and Statelist classes. In short, the blockchain only cares about key-value pairs that compose the blockchain's world state. However, we want to group types of objects (let's say Students, Professors, or Universities) by giving each key a proper identifier (a prefix). The correlated states (let's say the state of Student 1 and Student 2) are further grouped into a Statelist. This allows for easy querying of the world state.

Open *state.js*, inside the directory *ledger-api* with your favorite editor. A state is just a reference to a key on the blockchain that holds a certain value. The key might be simple (single value) or composed (a combination of values, separated by ":"). We support composed keys, supported by the *makeKey* method: [breaklines]javascript

```
this.key = State.makeKey(keyParts);
...
static makeKey(keyParts)  return keyParts.map(part =¿ JSON.stringify(part)).join(':');
```

Furthermore, the State class provides useful methods for serializing and deserializing the information. In other words, the information is passed to the ledger in the JSON format (the value corresponding to a key). It is desirable that when we query or retrieve information from the ledger, we obtain an object from a certain class, not plain JSON.

Now, focus on *statelist.js*. Here is where the magic happens: 1) adding information to the blockchain; 2) updating a state; 3) obtaining a state; 4) querying the blockchain.

For instance, the following method adds a new state to the blockchain. Note that this method is exactly equal to *updateState*, because the primitive *putState* adds or updates a state if the state does not or does already exist, respectively. It is desirable to differentiate them because they are semantically different. Basically, to add a new state, we retrieve the key stored in the State, serialize the data, and call the blockchain API.

[breaklines]javascript async addState(state) let key = this.ctx.stub.createCompositeKey(this.nam
state.getSplitKey()); let data = State.serialize(state); await this.ctx.stub.putState(key, data);

Now, the existing classes (Course, University, QUC, Feedback, Professor, Student) are just specialized states, with their own attributes and methods. For instance, the Course has methods to change an attribute called current state, which can be either Created, Completed, or Evaluated (see Figure 6): [breaklines]javascript const State = require('../../ledger-api/state.js');

const courseState = CREATED: 0, COMPLETED: 1, EVALUATED: 2 ;

//course item class Course extends State

constructor(obj) //Creator ID corresponds to the university ID super(Course.getClass(), [obj.universityId, obj.courseId]); this.currentState = courseState.CREATED; this.courseId = obj.courseId; this.teachingStaff = obj.teachingStaff; this. students = obj.students; Object.assign(this, obj);

setStateToCompleted() this.currentState = courseState.COMPLETED;

[...] static fromBuffer(buffer) return course.deserialize(buffer);

toBuffer() return Buffer.from(JSON.stringify(this));

static deserialize(data) return State.deserializeClass(data, Course);

/** * Factory method to create a course Item */ static createInstance(courseId, universityId, teachingStaff, students) return new Course(courseId, universityId, teachingStaff, students);

static getClass() return "org.b4s.course";

Now, the other crucial piece is the corresponding lists: in the course case, the course-list. Whereas a course is a state, a course-list is a state-list. In the list, we are defining the logic of interacting with the chain, something like a middleware layer in the backend of a centralized webapp. For example, the course-list contains a method *getAllCourses*, which uses the *getQueryResults* method from the state-list: [breaklines]javascript async getAllCourses() let query = "selector": "class": "org.b4s.course"
;

let prepared$_{q}uery = JSON.stringify(query); returnawaitthis.getQueryResults(prepared_{q}uery);$

Finally, we develop our smart contracts on top of these objects

For more information, refer to the commercial paper example[3], as well as fabric-samples[4].

[3]https://hyperledger-fabric.readthedocs.io/en/release-2.2/tutorial/commercial_paper.html

[4]https://github.com/hyperledger/fabric-samples

1.3.3 Wrap up

Confused? Do not worry. A blockchain is a complex system. For leveraging a blockchain system, an equilibrium between decentralized and centralized components has to be achieved. We can envision B4S as a full-stack system, in which the database is a blockchain. A user interface communicates with a blockchain client, which is a server. This server can act on behalf of an organization, issuing transactions against a peer node that redirects it to a chaincode container. This container holds several smart contracts (user, logistics, quc). Each smart contract is divided into the basic entities (User, Feedback), and useful abstractions help us develop chaincode (State, Statelist). This is illustrated in Figure 8.

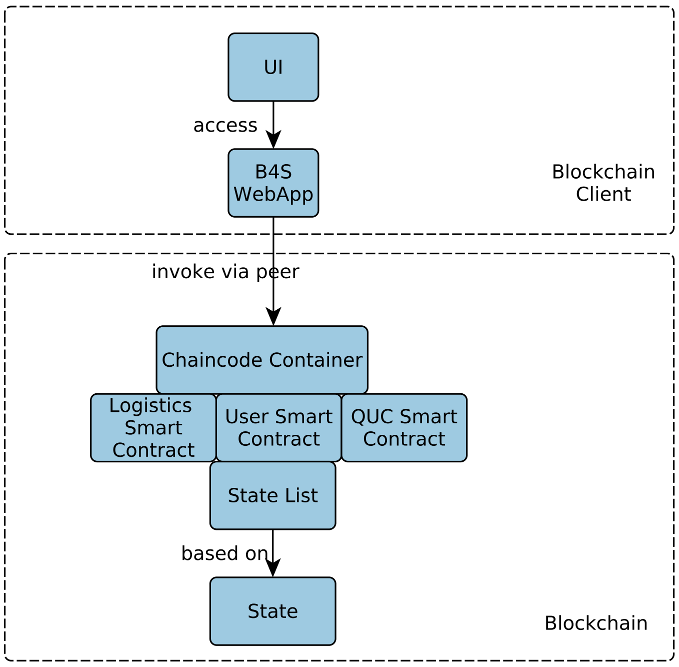


Figure 8: B4S architecture

2 Setting up B4S

2.1 Install prerequisites

This project has several prerequisites. Please refer to the *README.md* for instructions.

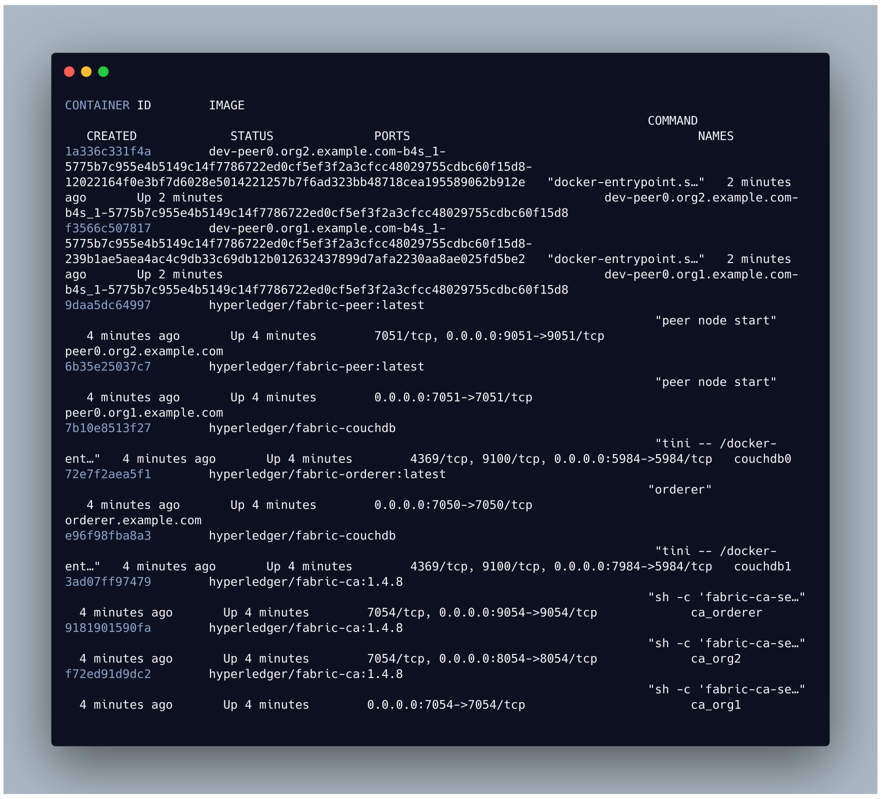
For a straightforward setup, clone the repository at `https://github.com/hyperledger-labs/univ`
Go to the B4S folder by running `cd b4s`. Install the prerequisites by running `./install-prerequesite`
In case of any problem, refer to the *Troubleshooting* section of the *README.md*.

All issues should be reported at https://github.com/hyperledger-labs/university-course/issues, with the label *lab-question*, and an extra lab referring to this lab number.

2.2 Instantiate the network

On the b4s directory, run `./network-starter`. This script calls *network* and *deployCC* scripts, present on the *test-network* directories. Do not worry about the details of these scripts. We will cover the infrastructure in the next lab.

The execution should return no errors. After it is completed, run *docker ps*. Notice that it created a network composed of 2 organizations, 2 peers, 1 orderer, 2 CAs, and 2 CouchDB databases.



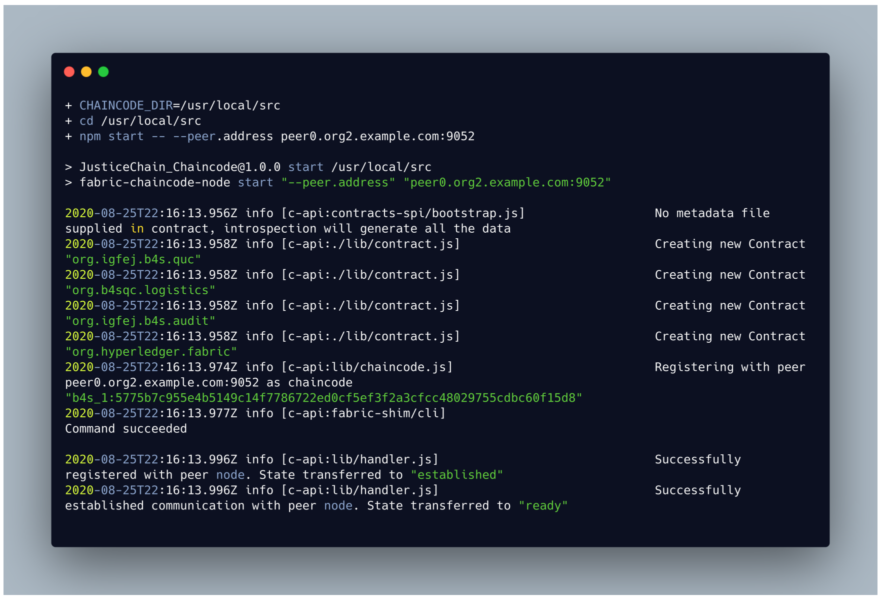
```
CONTAINER ID        IMAGE
                                                                                 COMMAND
   CREATED             STATUS              PORTS                                        NAMES
1a336c331f4a        dev-peer0.org2.example.com-b4s_1-
5775b7c955e4b5149c14f7786722ed0cf5ef3f2a3cfcc48029755cdbc60f15d8-
12022164f0e3bf7d6028e5014221257b7f6ad323bb48718cea195589062b912e   "docker-entrypoint.s…"   2 minutes
ago       Up 2 minutes                                                       dev-peer0.org2.example.com-
b4s_1-5775b7c955e4b5149c14f7786722ed0cf5ef3f2a3cfcc48029755cdbc60f15d8
f3566c507817        dev-peer0.org1.example.com-b4s_1-
5775b7c955e4b5149c14f7786722ed0cf5ef3f2a3cfcc48029755cdbc60f15d8-
239b1ae5aea4ac4c9db33c69db12b012632437899d7afa2230aa8ae025fd5be2   "docker-entrypoint.s…"   2 minutes
ago       Up 2 minutes                                                       dev-peer0.org1.example.com-
b4s_1-5775b7c955e4b5149c14f7786722ed0cf5ef3f2a3cfcc48029755cdbc60f15d8
9daa5dc64997        hyperledger/fabric-peer:latest
                                                                                  "peer node start"
   4 minutes ago       Up 4 minutes        7051/tcp, 0.0.0.0:9051->9051/tcp
peer0.org2.example.com
6b35e25037c7        hyperledger/fabric-peer:latest
                                                                                  "peer node start"
   4 minutes ago       Up 4 minutes        0.0.0.0:7051->7051/tcp
peer0.org1.example.com
7b10e8513f27        hyperledger/fabric-couchdb
                                                                                  "tini -- /docker-
ent…"   4 minutes ago       Up 4 minutes        4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp   couchdb0
72e7f2aea5f1        hyperledger/fabric-orderer:latest
                                                                                 "orderer"
   4 minutes ago       Up 4 minutes        0.0.0.0:7050->7050/tcp
orderer.example.com
e96f98fba8a3        hyperledger/fabric-couchdb
                                                                                  "tini -- /docker-
ent…"   4 minutes ago       Up 4 minutes        4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp   couchdb1
3ad07ff97479        hyperledger/fabric-ca:1.4.8
                                                                                 "sh -c 'fabric-ca-se…"
  4 minutes ago       Up 4 minutes        7054/tcp, 0.0.0.0:9054->9054/tcp             ca_orderer
9181901590fa        hyperledger/fabric-ca:1.4.8
                                                                                 "sh -c 'fabric-ca-se…"
  4 minutes ago       Up 4 minutes        7054/tcp, 0.0.0.0:8054->8054/tcp             ca_org2
f72ed91d9dc2        hyperledger/fabric-ca:1.4.8
                                                                                 "sh -c 'fabric-ca-se…"
  4 minutes ago       Up 4 minutes        0.0.0.0:7054->7054/tcp                       ca_org1
```

Figure 9: Result of running the *docker ps* command, showing the created infrastructure

2.3 Interact with the network

Congratulations! You have a working Hyperledger Fabric v2.2 blockchain running on your computer. Now, we shall interact with it.

Start by examining the first chaincode container, by running *docker logs CONTAINER_ID*. In my case, I should run *docker logs 1a336c331f4a*, as illustrated in Figure 9.



```
+ CHAINCODE_DIR=/usr/local/src
+ cd /usr/local/src
+ npm start -- --peer.address peer0.org2.example.com:9052

> JusticeChain_Chaincode@1.0.0 start /usr/local/src
> fabric-chaincode-node start "--peer.address" "peer0.org2.example.com:9052"

2020-08-25T22:16:13.956Z info [c-api:contracts-spi/bootstrap.js]                 No metadata file
supplied in contract, introspection will generate all the data
2020-08-25T22:16:13.958Z info [c-api:./lib/contract.js]                          Creating new Contract
"org.igfej.b4s.quc"
2020-08-25T22:16:13.958Z info [c-api:./lib/contract.js]                          Creating new Contract
"org.b4sqc.logistics"
2020-08-25T22:16:13.958Z info [c-api:./lib/contract.js]                          Creating new Contract
"org.igfej.b4s.audit"
2020-08-25T22:16:13.958Z info [c-api:./lib/contract.js]                          Creating new Contract
"org.hyperledger.fabric"
2020-08-25T22:16:13.974Z info [c-api:lib/chaincode.js]                           Registering with peer
peer0.org2.example.com:9052 as chaincode
"b4s_1:5775b7c955e4b5149c14f7786722ed0cf5ef3f2a3cfcc48029755cdbc60f15d8"
2020-08-25T22:16:13.977Z info [c-api:fabric-shim/cli]
Command succeeded

2020-08-25T22:16:13.996Z info [c-api:lib/handler.js]                             Successfully
registered with peer node. State transferred to "established"
2020-08-25T22:16:13.996Z info [c-api:lib/handler.js]                             Successfully
established communication with peer node. State transferred to "ready"
```

Figure 10: Result of running the *docker logs 1a336c331f4a* command.

You shall obtain a similar output:

To interact with the students-union peer, you need to interact with peer0 from organization 1. In contrast, if you want to interact with the university peer, you need to interact with peer0 from organization 2. To get information about the channel in which the peers are connected, run `docker exec peer0.org1.example.com peer channel getinfo -c mychannel`.

Another way is to load the configuration environment variables. To do this, from the *b4s* folder, run (also see section *Interacting with the network*, from the README.md:

bash source organization/university/university.sh

Right now, you can issue commands on behalf of the university peer.

2.4 Issuing Transactions

You can see the result of the execution by analyzing the docker container logs: *docker logs 1a336c331f4a*, and *docker logs peer0.org1.example.com*.

Let us issue a transaction against the QUC smart contract. For that, we need information regarding our certificates, in which the ABSOLUTE_PATH variable is the absolute path on your local machine to the lab (careful copying the code, because the buffer might contain invalid characters:

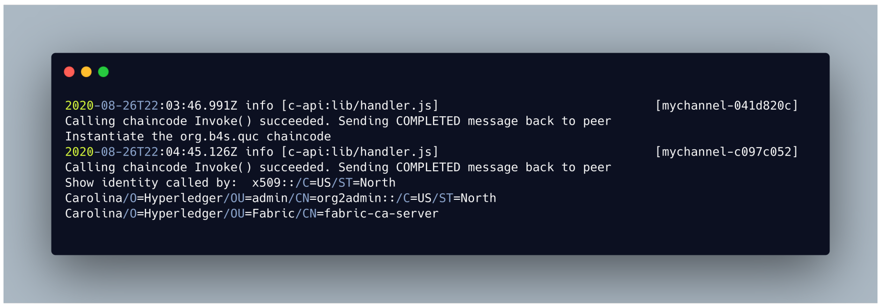
[breaklines, breakafter=test-network mathescape, linenos, numbersep=5pt, numbersep=5pt, xleftmargin=0pt]bash peer chaincode invoke -o localhost:7050 –ordererTLSHostnameO
orderer.example.com –tls true –cafile ABSOLUTE$_PATH/test-network/organizations/ordererOrga$
$cert.pem-Cmychannel-nb4s--peerAddresseslocalhost:7051--tlsRootCertFilesABSOLUTE_PA$
$network/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/c$

$-peerAddresseslocalhost : 9051--tlsRootCertFilesABSOLUTE_PATH/test-network/organizatio$
$c'"function" : "org.b4s.quc : start", "Args" : []' - -waitForEvent$

For instance, the full command would be: [breaklines, breakafter=test-network mathescape, linenos, numbersep=5pt, numbersep=5pt, xleftmargin=0pt]bash peer chaincode invoke -o localhost:7050 –ordererTLSHostnameOverride orderer.example.com –
tls true –cafile /home/rafael/Projects/university-course/support/Lab05/test-network/organizations/ord
cert.pem -C mychannel -n b4s –peerAddresses localhost:7051 –tlsRootCertFiles /home/rafael/Projec
course/support/Lab05/test-network/organizations/peerOrganizations/org1.example.com/peers/peer0
–peerAddresses localhost:9051 –tlsRootCertFiles /home/rafael/Projects/university-course/support/L
network/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.c
-c '"function":"org.b4s.quc:start", "Args":[]' –waitForEvent

Try, and issue another transaction: [breaklines, breakafter=test-network mathescape, linenos, numbersep=5pt, numbersep=5pt, xleftmargin=0pt]bash peer chaincode invoke -o localhost:7050 –ordererTLSHostnameOverride orderer.example.com –tls true –cafile
/home/rafael/Projects/university-course/support/Lab05/test-network/organizations/ordererOrganizat
cert.pem -C mychannel -n b4s –peerAddresses localhost:7051 –tlsRootCertFiles /home/rafael/Projec
course/support/Lab05/test-network/organizations/peerOrganizations/org1.example.com/peers/peer0
–peerAddresses localhost:9051 –tlsRootCertFiles /home/rafael/Projects/university-course/support/L
network/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.c
-c '"function":"org.b4s.quc:showIdentity", "Args":[]' –waitForEvent

What did you notice? This transaction obtains and returns information about the party calling the transaction. Inspect the logs: *docker logs 1a336c331f4a*. You can now see that the transaction was processed successfully.



```
2020-08-26T22:03:46.991Z info [c-api:lib/handler.js]                          [mychannel-041d820c]
Calling chaincode Invoke() succeeded. Sending COMPLETED message back to peer
Instantiate the org.b4s.quc chaincode
2020-08-26T22:04:45.126Z info [c-api:lib/handler.js]                          [mychannel-c097c052]
Calling chaincode Invoke() succeeded. Sending COMPLETED message back to peer
Show identity called by:  x509::/C=US/ST=North
Carolina/O=Hyperledger/OU=admin/CN=org2admin::/C=US/ST=North
Carolina/O=Hyperledger/OU=Fabric/CN=fabric-ca-server
```

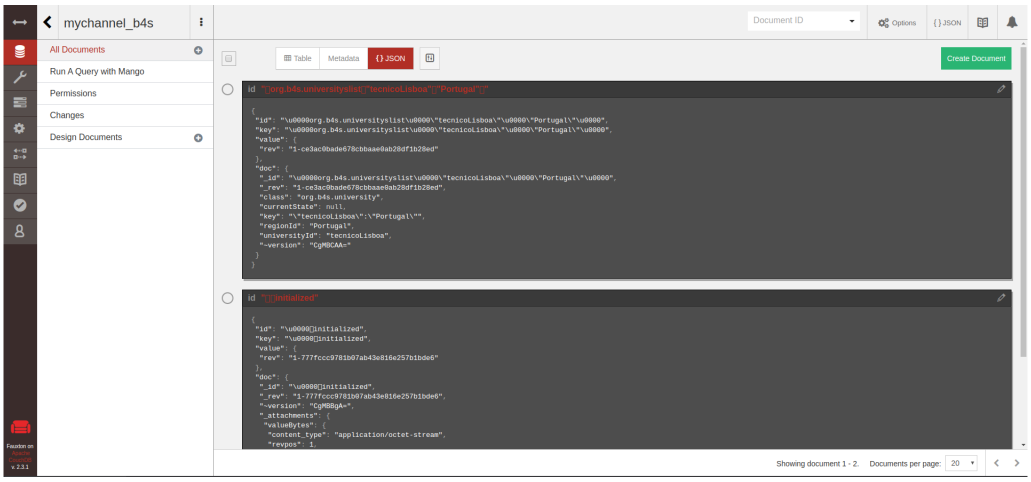
Figure 11: Logs of one of the chaincode containers, after issuing *showIdentity*

Finally, let us create a university! Run: [breaklines, breakafter=test-network mathescape, linenos, numbersep=5pt, numbersep=5pt, xleftmargin=0pt]bash peer chaincode invoke -o localhost:7050 –ordererTLSHostnameOverride orderer.example.com –
tls true –cafile /home/rafael/Projects/university-course/support/Lab05/test-network/organizations/ord
cert.pem -C mychannel -n b4s –peerAddresses localhost:7051 –tlsRootCertFiles /home/rafael/Projec
course/support/Lab05/test-network/organizations/peerOrganizations/org1.example.com/peers/peer0
–peerAddresses localhost:9051 –tlsRootCertFiles /home/rafael/Projects/university-course/support/L
network/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.c

-c '"function":"org.b4s.logistics:createUniversity", "Args":["tecnicoLisboa", "Portugal"]' –waitForEvent

Observe the output and the chaincode logs. You just created your first asset on a blockchain. You can check it on the CouchDB user interface, which holds the world state. To access it, on your browser, go to: bash http://localhost:5984/$_{u}tils/database/mychannel_{b}4s/_{a}ll_{d}ocs$

You should see the created object:



```
id "org.b4s.universityslist"tecnicoLisboa""Portugal""
{
 "id": "\u0000org.b4s.universityslist\u0000\"tecnicoLisboa\"\u0000\"Portugal\"\u0000",
 "key": "\u0000org.b4s.universityslist\u0000\"tecnicoLisboa\"\u0000\"Portugal\"\u0000",
 "value": {
  "rev": "1-ce3ac0bade678cbbaae0ab28df1b28ed"
 },
 "doc": {
  "_id": "\u0000org.b4s.universityslist\u0000\"tecnicoLisboa\"\u0000\"Portugal\"\u0000",
  "_rev": "1-ce3ac0bade678cbbaae0ab28df1b28ed",
  "class": "org.b4s.university",
  "currentState": null,
  "key": "\"tecnicoLisboa\":\"Portugal\"",
  "regionId": "Portugal",
  "universityId": "tecnicoLisboa",
  "~version": "CgMBCAA="
 }
}

id "initialized"
{
 "id": "\u0000initialized",
 "key": "\u0000initialized",
 "value": {
  "rev": "1-777fccc9781b07ab43e816e257b1bde6"
 },
 "doc": {
  "_id": "\u0000initialized",
  "_rev": "1-777fccc9781b07ab43e816e257b1bde6",
  "~version": "CgMBBgA=",
  "_attachments": {
   "valueBytes": {
    "content_type": "application/octet-stream",
    "revpos": 1,
```

Figure 12: CouchDB user interface, b4s channel

2.5 Clean Up

In the next class, we will explore in more detail the network topology, operations (start the network, monitor the network), the smart contract lifecycle in Fabric v2.x, and off-chain blockchain components (this is, the blockchain client application and user interfaces that allows end-users to interact with the blockchain). This will help you understand Fabric as a whole.

Do not forget the network, which is running and consuming resources! To delete the network, go to the b4s folder, and run `./network-clean`.

3 Exercises

How to provide extra differentiation for teaching quality?

One of the critiques to QUCs is that a professor's score depends on his or her popularity. How to decrease this risk?

What is the scalability of B4S QUC, given that Hyperledger Fabric can only contain a limited number of peer nodes and orgs?

When is a transaction considered valid in this ecosystem?

Suppose that the T-QUC chaincode is split into T-QUC Management and Actor Management chaincodes. What are the pros and cons of this approach?

To get information about the channel in which the peers are connected, run docker exec peer0.org1.example.com peer channel getinfo -c mychannel. What do you see?

Analyze the chaincode container logs, as well as the peer logs. What do you see?

Instead of the chaincode receiving a unique Id as a parameter, we could generate it on the smart contract itself (e.g., let uniqueId = uuidv4()). This way, the issuer may not tamper or control the generated IDs. Why is this a bad idea?

References

[1] R. Belchior, A. Vasconcelos, and M. Correia. Towards Secure, Decentralized, and Automatic Audits with Blockchain. In *European Conference on Information Systems*, 2020.