Getting started with Corda

https://docs.corda.net/getting-set-up.html

Getting set up

Software requirements

Corda uses industry-standard tools:

- **Oracle JDK 8 JVM** - minimum supported version **8u171**
- **IntelliJ IDEA** - supported versions **2017.x** and **2018.x** (with Kotlin plugin version 1.2.51)
- **Git**

We also use Gradle and Kotlin, but you do not need to install them. A standalone Gradle wrapper is provided, and it will download the correct version of Kotlin.

Please note:

- Corda runs in a JVM. JVM implementations other than Oracle JDK 8 are not actively supported. However, if you do choose to use OpenJDK, you will also need to install OpenJFX
- Applications on Corda (CorDapps) can be written in any language targeting the JVM. However, Corda itself and most of the samples are written in Kotlin. Kotlin is an official Android language, and you can read more about why Kotlin is a strong successor to Java here. If you're unfamiliar with Kotlin, there is an official getting started guide, and a series of Kotlin Koans.
- IntelliJ IDEA is recommended due to the strength of its Kotlin integration.

Following these software recommendations will minimize the number of errors you encounter, and make it easier for others to provide support. However, if you do use other tools, we'd be interested to hear about any issues that arise.

Set-up instructions

The instructions below will allow you to set up a Corda development environment and run a basic CorDapp. If you have any issues, please consult the Troubleshooting page, or reach out on Slack, Stack Overflow or the forums.

The set-up instructions are available for the following platforms:

- Windows (or in video form)
- Mac (or in video form)

Windows

Warning

If you are using a Mac machine, please follow the Mac instructions instead.

Java

1. Visit http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
2. Scroll down to “Java SE Development Kit 8uXXX” (where “XXX” is the latest minor version number)
3. Toggle “Accept License Agreement”
4. Click the download link for jdk-8uXXX-windows-x64.exe (where “XXX” is the latest minor version number)
5. Download and run the executable to install Java (use the default settings)
6. Add Java to the PATH environment variable by following the instructions at https://docs.oracle.com/javase/7/docs/webnotes/install/windows/jdk-installation-windows.html#path
7. Open a new command prompt and run `java -version` to test that Java is installed correctly

Git

1. Visit https://git-scm.com/download/win
2. Click the “64-bit Git for Windows Setup” download link.
3. Download and run the executable to install Git (use the default settings)
4. Open a new command prompt and type `git --version` to test that git is installed correctly

IntelliJ

1. Visit https://www.jetbrains.com/idea/download/download-thanks.html?code=IIC
2. Download and run the executable to install IntelliJ Community Edition (use the default settings)

3. Ensure the Kotlin plugin in Intellij is updated to version 1.2.51

Download a sample project

1. Open a command prompt
2. Clone the CorDapp example repo by running `git clone https://github.com/corda/cordapp-example`
3. Move into the `cordapp-example` folder by running `cd cordapp-example`

Run from the command prompt

1. From the cordapp-example folder, deploy the nodes by running `gradlew deployNodes`
2. Start the nodes by running `call kotlin-source/build/nodes/runnodes.bat`
3. Wait until all the terminal windows display either “Webserver started up in XX.X sec” or “Node for “NodeC” started up and registered in XX.XX sec”
4. Test the CorDapp is running correctly by visiting the front end at http://localhost:10009/web/example/

Run from IntelliJ

1. Open IntelliJ Community Edition
2. On the splash screen, click `Open` (do **not** click `Import Project`) and select the `cordapp-example` folder

Warning

If you click `Import Project` instead of `Open`, the project’s run configurations will be erased!

3. Once the project is open, click `File`, then `Project Structure`. Under `Project SDK:`, set the project SDK by clicking `New...`, clicking `JDK`, and navigating to `C:\\Program Files\\Java\\jdk1.8.0_XXX` (where `XXX` is the latest minor version number). Click “OK”
4. Again under `File` then `Project Structure`, select `Modules`. Click `+`, then `Import Module`, then select the `cordapp-example` folder and click `Open`. Choose to `Import module from external model`, select `Gradle`, click `Next` then `Finish` (leaving the defaults) and `OK`
5. Wait for the indexing to finish (a progress bar will display at the bottom-right of the IntelliJ window until indexing is complete)

6. At the top-right of the screen, to the left of the green `play` arrow, you should see a dropdown. In that dropdown, select `Run Example Cordapp - Kotlin` and click the green `play` arrow.
7. Wait until the run windows displays the message `Webserver started up in XX.X sec`
8. Test the CorDapp is running correctly by visiting the front end at http://localhost:10009/web/example/

Mac

Warning

If you are using a Windows machine, please follow the Windows instructions instead.

Java

1. Visit http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
2. Scroll down to “Java SE Development Kit 8uXXX” (where “XXX” is the latest minor version number)
3. Toggle “Accept License Agreement”
4. Click the download link for jdk-8uXXX-macosx-x64.dmg (where “XXX” is the latest minor version number)
5. Download and run the executable to install Java (use the default settings)
6. Open a new terminal window and run `java -version` to test that Java is installed correctly

IntelliJ

1. Visit https://www.jetbrains.com/idea/download/download-thanks.html?platform=mac&code=IIC
2. Download and run the executable to install IntelliJ Community Edition (use the default settings)
3. Ensure the Kotlin plugin in Intellij is updated to version 1.2.51

Download a sample project

1. Open a terminal
2. Clone the CorDapp example repo by running `git clone https://github.com/corda/cordapp-example`

3. Move into the `cordapp-example` folder by running `cd cordapp-example`

Run from the terminal

1. From the cordapp-example folder, deploy the nodes by running `./gradlew deployNodes`
2. Start the nodes by running `kotlin-source/build/nodes/runnodes`. Do not click while 8 additional terminal windows start up.
3. Wait until all the terminal windows display either “Webserver started up in XX.X sec” or “Node for “NodeC” started up and registered in XX.XX sec”
4. Test the CorDapp is running correctly by visiting the front end at http://localhost:10009/web/example/

Run from IntelliJ

1. Open IntelliJ Community Edition
2. On the splash screen, click `Open` (do **not** click `Import Project`) and select the `cordapp-example` folder

Warning

If you click `Import Project` instead of `Open`, the project’s run configurations will be erased!

3. Once the project is open, click `File`, then `Project Structure`. Under `Project SDK:`, set the project SDK by clicking `New...`, clicking `JDK`, and navigating to `C:\\Program Files\\Java\\jdk1.8.0_XXX` (where `XXX` is the latest minor version number). Click “OK”
4. Again under `File` then `Project Structure`, select `Modules`. Click `+`, then `Import Module`, then select the `cordapp-example` folder and click `Open`. Choose to `Import module from external model`, select `Gradle`, click `Next` then `Finish` (leaving the defaults) and `OK`
5. Wait for the indexing to finish (a progress bar will display at the bottom-right of the IntelliJ window until indexing is complete)
6. At the top-right of the screen, to the left of the green `play` arrow, you should see a dropdown. In that dropdown, select `Run Example Cordapp - Kotlin` and click the green `play` arrow.
7. Wait until the run windows displays the message `Webserver started up in XX.X sec`

8. Test the CorDapp is running correctly by visiting the front end at `http://localhost:10009/web/example/

Corda source code

The Corda platform source code is available here:

https://github.com/corda/corda.git

A CorDapp template that you can use as the basis for your own CorDapps is available in both Java and Kotlin versions:

https://github.com/corda/cordapp-template-java.git

https://github.com/corda/cordapp-template-kotlin.git

And a list of simple sample CorDapps for you to explore basic concepts is available here:

https://www.corda.net/samples/

You can clone these repos to your local machine by running the command `git clone [repo URL]`.

Next steps

The best way to check that everything is working fine is by taking a deeper look at the example CorDapp.

Next, you should read through Corda Key Concepts to understand how Corda works.

By then, you'll be ready to start writing your own CorDapps. Learn how to do this in the Hello, World tutorial. You may want to refer to the API documentation, the flow cookbook and the samples along the way.

If you encounter any issues, please see the Troubleshooting page, or ask on Stack Overflow or via our Slack channels.

Next Previous

https://docs.corda.net/tutorial-cordapp.html

The example CorDapp

Contents

The example CorDapp allows nodes to agree IOUs with each other, as long as they obey the following contract rules:

- The IOU's value is strictly positive
- A node is not trying to issue an IOU to itself

We will deploy and run the CorDapp on four test nodes:

- **Notary**, which hosts a validating notary service
- **PartyA**
- **PartyB**
- **PartyC**

Because data is only propagated on a need-to-know basis, any IOUs agreed between PartyA and PartyB become "shared facts" between PartyA and PartyB only. PartyC won't be aware of these IOUs.

Downloading the example CorDapp

Start by downloading the example CorDapp from GitHub:

- Set up your machine by following the quickstart guide
- Clone the example CorDapp from the cordapp-example repository using the following command: `git clone https://github.com/corda/cordapp-example`
- Change directories to the freshly cloned repo: `cd cordapp-example`

Opening the example CorDapp in IntelliJ

Let's open the example CorDapp in IntelliJ IDEA:

- Open IntelliJ
- A splash screen will appear. Click `open`, navigate to the folder where you cloned the `cordapp-example`, and click `OK`
- Once the project is open, click `File`, then `Project Structure`. Under `Project SDK:`, set the project SDK by clicking `New...`, clicking `JDK`, and navigating to `C:\Program Files\Java\jdk1.8.0_XXX` (where `XXX` is the latest minor version number). Click `OK`
- Again under `File` then `Project Structure`, select `Modules`. Click `+`, then `Import Module`, then select the `cordapp-example` folder and click `Open`.

Choose to `Import module from external model`, select `Gradle`, click `Next` then `Finish` (leaving the defaults) and `OK`

- Gradle will now download all the project dependencies and perform some indexing. This usually takes a minute or so

Project structure

The example CorDapp has the following structure:

```
.
├── config
│   ├── dev
│   │   └── log4j2.xml
│   └── test
│       └── log4j2.xml
├── doc
│   └── example_flow.plantuml
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── lib
│   ├── README.txt
│   └── quasar.jar
├── java-source
│   └── ...
├── kotlin-source
│   ├── build.gradle
│   └── src
│       ├── main
│       │   ├── kotlin
│       │   │   └── com
│       │   │       └── example
│       │   │           ├── api
│       │   │           │   └── ExampleApi.kt
│       │   │           ├── client
│       │   │           │   └── ExampleClientRPC.kt
│       │   │           ├── contract
│       │   │           │   └── IOUContract.kt
│       │   │           ├── flow
│       │   │           │   └── ExampleFlow.kt
│       │   │           ├── model
│       │   │           │   └── IOU.kt
│       │   │           ├── plugin
│       │   │           │   └── ExamplePlugin.kt
│       │   │           ├── schema
│       │   │           │   └── IOUSchema.kt
│       │   │           └── state
│       │   │               └── IOUState.kt
│       │   └── resources
│       │       ├── META-INF
│       │       │   └── services
│       │       │       └── net.corda.webserver.services.WebServerPluginRegistry
│       │       ├── certificates
│       │       │   ├── readme.txt
│       │       │   ├── sslkeystore.jks
│       │       │   └── truststore.jks
│       │       └── exampleWeb
│       │           ├── index.html
│       │           └── js
│       │                  └── angular-module.js
│       └── test
│           └── kotlin
│               └── com
│                   └── example
│                       ├── Main.kt
│                       ├── contract
│                       │   └── IOUContractTests.kt
│                       └── flow
│                           └── IOUFlowTests.kt
├── .gitignore
├── LICENCE
├── README.md
├── TRADEMARK
├── build.gradle
├── gradle.properties
├── gradlew
├── gradlew.bat
└── settings.gradle
```

The key files and directories are as follows:

- The **root directory** contains some gradle files, a README and a LICENSE
- **config** contains log4j configs
- **gradle** contains the gradle wrapper, which allows the use of Gradle without installing it yourself and worrying about which version is required
- **lib** contains the Quasar jar which rewrites our CorDapp's flows to be checkpointable
- **kotlin-source** contains the source code for the example CorDapp written in Kotlin
  - **kotlin-source/src/main/kotlin** contains the source code for the example CorDapp
  - **kotlin-source/src/main/resources** contains the certificate store, some static web content to be served by the nodes and the WebServerPluginRegistry file
  - **kotlin-source/src/test/kotlin** contains unit tests for the contracts and flows, and the driver to run the nodes via IntelliJ
- **java-source** contains the same source code, but written in Java. CorDapps can be developed in any language targeting the JVM

Running the example CorDapp

There are two ways to run the example CorDapp:

- Via the terminal
- Via IntelliJ

Both approaches will create a set of test nodes, install the CorDapp on these nodes, and then run the nodes. You can read more about how we generate nodes here.

Running the example CorDapp from the terminal

Building the example CorDapp

- Open a terminal window in the `cordapp-example` directory
- Build the test nodes with our CorDapp using the following command:
  - Unix/Mac OSX: `./gradlew deployNodes`
  - Windows: `gradlew.bat deployNodes`

  This will automatically build four nodes with our CorDapp already installed

Note

CorDapps can be written in any language targeting the JVM. In our case, we've provided the example source in both Kotlin (`/kotlin-source/src`) and Java (`/java-source/src`). Since both sets of source files are functionally identical, we will refer to the Kotlin version throughout the documentation.

- After the build finishes, you will see the generated nodes in the `kotlin-source/build/nodes` folder
  - There will be a folder for each generated node, plus a `runnodes` shell script (or batch file on Windows) to run all the nodes simultaneously
  - Each node in the `nodes` folder has the following structure:

```
. nodeName
├── corda.jar                // The Corda node runtime.
├── corda-webserver.jar      // The node development webserver.
├── node.conf                // The node configuration file.
└── cordapps                 // The node's CorDapps.
```

Running the example CorDapp

Start the nodes by running the following command from the root of the `cordapp-example` folder:

- Unix/Mac OSX: `kotlin-source/build/nodes/runnodes`
- Windows: `call kotlin-source\build\nodes\runnodes.bat`

Warning

On Unix/Mac OSX, do not click/change focus until all seven additional terminal windows have opened, or some nodes may fail to start.

For each node, the `runnodes` script creates a node tab/window:

```
   ______               __
  / ____/     _________/ /___ _
 / /     __  / ___/ __  / __ `/         Top tip: never say "oops", instead
/ /___  /_/ / /  / /_/ / /_/ /          always say "Ah, Interesting!"
\____/     /_/   \__,_/\__,_/

--- Corda Open Source corda-3.0 (4157c25) -------------------------------------------
---


Logs can be found in                    : /Users/joeldudley/Desktop/cordapp-
example/kotlin-source/build/nodes/PartyA/logs
Database connection url is              : jdbc:h2:tcp://localhost:59472/node
Incoming connection address             : localhost:10007
Listening on port                       : 10007
Loaded CorDapps                         : corda-finance-corda-3.0, cordapp-example-
0.1, corda-core-corda-3.0
Node for "PartyA" started up and registered in 38.59 sec


Welcome to the Corda interactive shell.
Useful commands include 'help' to see what is available, and 'bye' to shut down the
node.

Fri Mar 02 17:34:02 GMT 2018>>>
```

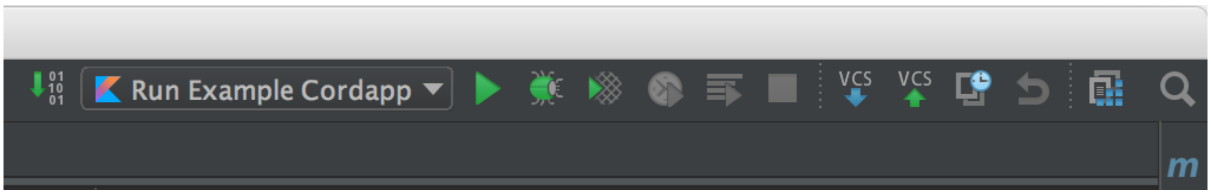
For every node except the notary, the script also creates a webserver terminal tab/window:

```
Logs can be found in /Users/username/Desktop/cordapp-example/kotlin-
source/build/nodes/PartyA/logs/web
Starting as webserver: localhost:10009
Webserver started up in 42.02 sec
```

It usually takes around 60 seconds for the nodes to finish starting up. To ensure that all the nodes are running, you can query the 'status' end-point located at `http://localhost:[port]/api/status` (e.g. `http://localhost:10009/api/status` for `PartyA`).

Running the example CorDapp from IntelliJ

- Select the `Run Example CorDapp - Kotlin` run configuration from the drop-down menu at the top right-hand side of the IDE
- Click the green arrow to start the nodes:



- To stop the nodes, press the red square button at the top right-hand side of the IDE, next to the run configurations

Interacting with the example CorDapp

Via HTTP

The nodes' webservers run locally on the following ports:

- PartyA: `localhost:10009`
- PartyB: `localhost:10012`
- PartyC: `localhost:10015`

These ports are defined in each node's node.conf file under `kotlin-source/build/nodes/NodeX/node.conf`.

Each node webserver exposes the following endpoints:

- `/api/example/me`
- `/api/example/peers`
- `/api/example/ious`
- `/api/example/create-iou` with parameters `iouValue` and `partyName` which is CN name of a node

There is also a web front-end served from `/web/example`.

Warning

The content in `/web/example` is only available for demonstration purposes and does not implement anti-XSS, anti-XSRF or other security techniques. Do not use this code in production.

Creating an IOU via the endpoint

An IOU can be created by sending a PUT request to the `/api/example/create-iou` endpoint directly, or by using the the web form served from `/web/example`.

To create an IOU between PartyA and PartyB, run the following command from the command line:

```
curl -X PUT 'http://localhost:10009/api/example/create-
iou?iouValue=1&partyName=O=PartyB,L=New%20York,C=US'
```

Note that both PartyA’s port number (`10009`) and PartyB are referenced in the PUT request path. This command instructs PartyA to agree an IOU with PartyB. Once the process is complete, both nodes will have a signed, notarised copy of the IOU. PartyC will not.

Submitting an IOU via the web front-end

To create an IOU between PartyA and PartyB, navigate to `/web/example`, click the “create IOU” button at the top-left of the page, and enter the IOU details into the web-form. The IOU must have a positive value. For example:

```
Counterparty: Select from list
Value (Int):   5
```

And click submit. Upon clicking submit, the modal dialogue will close, and the nodes will agree the IOU.

Checking the output

Assuming all went well, you can view the newly-created IOU by accessing the vault of PartyA or PartyB:

Via the HTTP API:

- PartyA’s vault: Navigate to http://localhost:10009/api/example/ious
- PartyB’s vault: Navigate to http://localhost:10012/api/example/ious

Via web/example:

- PartyA: Navigate to http://localhost:10009/web/example and hit the “refresh” button
- PartyB: Navigate to http://localhost:10012/web/example and hit the “refresh” button

The vault and web front-end of PartyC (at `localhost:10015`) will not display any IOUs. This is because PartyC was not involved in this transaction.

Via the interactive shell (terminal only)

Nodes started via the terminal will display an interactive shell:

```
Welcome to the Corda interactive shell.
Useful commands include 'help' to see what is available, and 'bye' to shut down the
node.

Fri Jul 07 16:36:29 BST 2017>>>
```

Type `flow list` in the shell to see a list of the flows that your node can run. In our case, this will return the following list:

```
com.example.flow.ExampleFlow$Initiator
net.corda.core.flows.ContractUpgradeFlow$Initiator
net.corda.core.flows.ContractUpgradeFlow$Initiator
net.corda.finance.flows.CashExitFlow
net.corda.finance.flows.CashIssueAndPaymentFlow
net.corda.finance.flows.CashIssueFlow
net.corda.finance.flows.CashPaymentFlow
```

Creating an IOU via the interactive shell

We can create a new IOU using the `ExampleFlow$Initiator` flow. For example, from the interactive shell of PartyA, you can agree an IOU of 50 with PartyB by running `flow start ExampleFlow$Initiator iouValue: 50, otherParty: "O=PartyB,L=New York,C=US"`.

This will print out the following progress steps:

```
✓    Generating transaction based on new IOU.
✓    Verifying contract constraints.
✓    Signing transaction with our private key.
✓    Gathering the counterparty's signature.
    ✓    Collecting signatures from counterparties.
    ✓    Verifying collected signatures.
✓    Obtaining notary signature and recording transaction.
    ✓    Requesting signature by notary service
            Requesting signature by Notary service
            Validating response from Notary service
    ✓    Broadcasting transaction to participants
✓    Done
```

Checking the output

We can also issue RPC operations to the node via the interactive shell. Type `run` to see the full list of available operations.

You can see the newly-created IOU by running `run vaultQuery contractStateType: com.example.state.IOUState`.

As before, the interactive shell of PartyC will not display any IOUs.

Via the h2 web console

You can connect directly to your node’s database to see its stored states, transactions and attachments. To do so, please follow the instructions in Node database.

Using the example RPC client

`/src/main/kotlin-source/com/example/client/ExampleClientRPC.kt` defines a simple RPC client that connects to a node, logs any existing IOUs and listens for any future IOUs. If you haven’t created any IOUs when you first connect to one of the nodes, the client will simply log any future IOUs that are agreed.

Running the client via IntelliJ

Run the ‘Run Example RPC Client’ run configuration. By default, this run configuration is configured to connect to PartyA. You can edit the run configuration to connect on a different port.

Running the client via the command line

Run the following gradle task:

```
./gradlew runExampleClientRPCKotlin
```

This will connect the RPC client to PartyA and log their past and future IOU activity.

You can close the application using `ctrl+C`.

For more information on the client RPC interface and how to build an RPC client application, see:

- Client RPC documentation
- Client RPC tutorial

Running nodes across machines

The nodes can be configured to communicate as a network even when distributed across several machines:

- Deploy the nodes as usual:
  - Unix/Mac OSX: `./gradlew deployNodes`
  - Windows: `gradlew.bat deployNodes`
- Navigate to the build folder (`kotlin-source/build/nodes`)
- For each node, open its `node.conf` file and change `localhost` in its `p2pAddress` to the IP address of the machine where the node will be run (e.g. `p2pAddress="10.18.0.166:10007"`)
- These changes require new node-info files to be distributed amongst the nodes. Use the network bootstrapper tool (see Network Bootstrapper) to update the files and have them distributed locally:

  ```
  java -jar network-bootstrapper.jar kotlin-source/build/nodes
  ```
- Move the node folders to their individual machines (e.g. using a USB key). It is important that none of the nodes - including the notary - end up on more than one machine. Each computer should also have a copy of `runnodes` and `runnodes.bat`.

  For example, you may end up with the following layout:

  - Machine 1: `Notary`, `PartyA`, `runnodes`, `runnodes.bat`
  - Machine 2: `PartyB`, `PartyC`, `runnodes`, `runnodes.bat`
- After starting each node, the nodes will be able to see one another and agree IOUs among themselves

Warning

The bootstrapper must be run **after** the `node.conf` files have been modified, but **before** the nodes are distributed across machines. Otherwise, the nodes will not be able to communicate.

Note

If you are using H2 and wish to use the same `h2port` value for two or more nodes, you must only assign them that value after the nodes have been moved to their individual machines. The initial bootstrapping process requires access to the nodes' databases and if two nodes share the same H2 port, the process will fail.

Testing your CorDapp

Corda provides several frameworks for writing unit and integration tests for CorDapps.

Contract tests

You can run the CorDapp's contract tests by running the `Run Contract Tests - Kotlin` run configuration.

Flow tests

You can run the CorDapp's flow tests by running the `Run Flow Tests - Kotlin` run configuration.

Integration tests

You can run the CorDapp's integration tests by running the `Run Integration Tests - Kotlin` run configuration.

Debugging your CorDapp

See Debugging a CorDapp.

Next Previous

Letter of Credit

https://github.com/corda/LetterOfCredit/blob/release/script.md

```
318 lines (179 sloc)  14.5 KB
```

Demo script

Before we start

Run the nodes and bring up the network map by following the instructions in the README.

What are we going to show you?

We'll be showing you a demo that will highlight the key benefits of blockchain solutions:

- The guarantee that nodes see the exact same version of shared data
- The ability to transact without intermediaries
- The ability to exchange value

But it will also highlight the unique features of the Corda platform, features that address the challenges presented by other blockchain platforms. Corda uniquely allows:

- Privacy...
- Without foregoing a single global network...
- Of well-know counter-parties...
- With instant transaction finality...
- And the ability to model complex workflows like delivery vs. payment (DVP)

How are we going to show it?

We will showcase the Corda platform using a real world example, a letter of credit scenario. This is one of the first use-cases going into production on the Corda platform.

What is a letter of credit? Well, suppose we have a buyer and a seller who want to enter into a trade. However, the buyer doesn't want to send payment first, and the seller doesn't want to send the goods first. They're in a stand-off.

This stand-off can be addressed using a letter of credit, which avoids the inherent trust issues in buyer/seller relationships:

- The seller issues a purchase order to the buyer
- The buyer takes this purchase order to their bank, and the bank arranges a letter of credit with the seller's bank
- The letter of credit dictates that, instead of the buyer paying the seller directly:

    - The seller's bank pays the seller
    - Then the buyer's bank pays the seller's bank
    - Then the buyer pays the buyer's bank

Running the demo

The network map

Key points introduced: Single global network, well-known counterparties via network map

Points reinforced: N/A

Here is a view of the Corda network. There are four nodes that are of special interest to us:

- Lok Ma Exporters, a seller based in China
- Shenzen State Bank, the seller's bank, also based in China
- Analog Importers, a buyer based in Liverpool
- First Bank of London, the buyer's bank, based in London

All these nodes exist on the same global network. They can transact with one another directly, and with any other nodes on the global network. There are no islands, and no "trapped" assets.

Each node is made aware of the other nodes on the network using a signed, dynamic document called the network map. The network map links nodes to legal names and addresses.

Before joining the network, each node undergoes a KYC process before being issued a certificate. This means that:

- Nodes know exactly which legal entity they are transacting with
- Communication between nodes is encrypted using transport-layer security (TLS)

Issuing the purchase order

Key points introduced: Flexible data model, need-to-know data distribution

Points reinforced: Network map

We can see on the left the series of steps in this letter of credit scenario. We'll go through them one-by-one. We start by issuing an purchase order from the seller to the buyer.

Here's the seller's view of the ledger. There's nothing here yet. Let's start the process by creating an purchase order.

<Click 'Create Purchase Order'>

These are all the fields that are included in this application's representation of an purchase order. The Corda data model is completely flexible. We can have any fields we like, of any type.

For the purposes of the demo, we'll auto-complete most fields.

<Click 'Autocomplete fields'>

However, we still need to pick the buyer. Using the network map, it's very easy to look up the well-known counterparty we want to transact with by name.

Let's commit this purchase order to the ledger.

<Click 'Create Purchase Order'>

If we look at the bottom of the seller's view, we can see the transaction has already been committed to the ledger.

<Close the seller's modal and open the buyer's modal>

If we open the buyer's view, we can see that they see the exact same transaction and purchase order as the seller, since they were involved in the transaction.

<Close the buyer's modal and open the seller's bank's modal>

However, if we open the view of the other nodes - the buyer's bank or the seller's bank, for example - we can see that they aren't even aware of this transaction, even though they're on the same network. In Corda, ledger updates are distributed to nodes on a need-to-know basis. Because the purchase order only involves the buyer and the seller, only they have seen the transaction.

Applying for a letter of credit

Key points introduced: Immutability, double-spend prevention

Points reinforced: Flexible data model, need-to-know data distribution

<Close the seller's bank's modal and click on the second step>

Going back to the buyer's view, we can apply for a letter of credit. Remember, this is an agreement between the buyer and seller and their respective banks that payment will be made via the banks for trust purposes.

<Click 'Apply for LOC'>

Again, note that the data model is completely flexible. This is the set of fields this application uses for letter of credit applications. They're different from the purchase order fields.

We're going to autocomplete again...

<Click 'Autocomplete fields'>

But note again how we can quickly pick the banks involved using their legal names from the network map.

We finalise the application.

<Click 'Apply'>

The transaction is now committed to the ledger. Looking at the transaction, we can see that:

- The transaction has been signed by the buyer, meaning that:
    - It cannot be modified. Any attempt to modify it now would invalidate the buyer's signature
    - It cannot be repudiated. The buyer's bank can prove that the buyer applied for a letter of credit
- The transaction has also been signed by the notary pool. The signature from the notary pool means that the transaction does not represent a double-spend attempt. Notary pools are services on the network that can use any algorithm to prevent double-spends

Let's try and apply for the same purchase order twice.

<Click 'Apply for LOC', then 'Autocomplete fields', then 'Apply'>

The application will fail because we have already used this purchase order to apply for a letter of credit.

<Close the buyer's modal and open the modal of the seller>

And again, we can see that this ledger update has only been propagated to the parties involved - the buyer and the buyer's bank.

Approving the letter of credit application

Key points introduced: UTXO model

Points reinforced: Immutability and double-spend prevention

<Close the seller's modal and click on the third step>

Now going to the buyer's bank's view, we can approve the letter of credit.

<Click 'Approve'>

Let's take a look at this transaction:

- We have "consumed" the existing letter of credit application and purchase order
- We have created a letter of credit

This is how the Corda ledger is updated. We consume some existing set of states to create a new set of states.

Again, the signatures by the buye's bank and the notary pool make the transaction immutable, irrepudiable, and immune to double-spend attempts.

Shipping the goods

Key points introduced: Shared view of the data, smart contract logic

Points reinforced: UTXO model

<Close the buyer's bank's modal and click on the fourth step>

With the letter of credit in place, the seller can add the bill of lading and ship the goods. The bill of lading is a legal document indicating who has title to the goods shipped by the seller.

<Click 'Create Bill of Lading', then 'Autocomplete fields', then 'Submit'>

The status of the letter of credit has been updated to "LADED". Looking at the transactions view, we can see that the letter of credit with a status of "ISSUED" has been consumed to create the letter of credit with a status of "LADED".

What happens if we try and add the bill of lading twice?

<Click 'Create Bill of Lading', then 'Autocomplete fields', then 'Submit'>

The smart contract logic underlying the letter of credit rejects this attempt to add a bill of lading twice. The requirement for the input letter of credit to have a status of "ISSUED" is not satisfied, since the order is already laded.

Now let's ship the goods.

<Click 'Ship'>

Instantly, the status of the letter of credit is updated to shipped. Every other node who is tracking the letter of credit automatically sees their ledger updated to show the new status. For example, if we open the buyer's modal...

<Close the seller's modal and open the buyer's modal>

We see that the status of the letter of credit has been updated.

Unwinding the letter of credit

Key points introduced: Atomicity, instant finality

Points reinforced: Smart contract logic

<Close the buyer's modal>

It's now time to unwind the letter of credit:

- The seller's bank needs to pay the seller...
- So that the buyer's bank can pay the seller's bank...
- So that the buyer's can pay its bank

Currently, the bill of lading is held by the seller.

<Click 'View' on the bill of lading, then click the sliders in the top-right>

Let's pay the seller in exchange for the bill of lading.

<Click 'Pay Seller'>

The status of the bill of lading is updated to say "BENEFICIARY_PAID".

We can now see that the bill of lading is held by the seller's bank.

<Click 'View' on the bill of lading, then click the sliders in the top-right>

If we look at the transaction, we can see that it is an atomic delivery-versus-payment transaction. The money moved from the seller's bank to the seller at exactly the same time as the bill of lading moved from the seller to the seller's bank. It is impossible for only part of the transaction to go through.

It's also important to note that transaction finality is instant. As soon as the required signatures - including the notary pool's signature - have been applied, the transaction is final. There is no possibility of the transaction being reversed, or a need to wait for a number of "confirmations".

If we try and pay again...

<Click 'Pay Seller'>

Again, the smart contract logic underlying the letter of credit rejects this attempt to pay the seller twice. The requirement for the seller not to have already paid is not satisfied, since the seller has already been paid.

<Close the seller's bank's modal and click on the sixth step>

Now the buyer's bank pays the seller's bank in exchange for the bill of lading.

<Click 'Pay Advising Bank'>

Again, the letter of credit's status is updated.

We can now see that the bill of lading is held by the buyer's bank.

<Click 'View' on the bill of lading, then click the sliders in the top-right, then close the buyer's bank's modal and click on the seventh step>

Finally, the buyer exchanges its cash for the bill of lading, completing the full cycle.

<Click 'Settle'>

Oh...

So, uh... So the buyer doesn't actually have any money. We forgot to give him any cash...

But it might actually be ok.

Transferring cash

Key points introduced: Interoperability between applications

Points reinforced: N/A

In Corda, all applications run side-by-side on a single network. In our case, as well as the letter of credit CorDapp, the bank nodes have a cash CorDapp installed.

These two CorDapps were developed completely in isolation:

- The cash CorDapp was built by a software consultancy called Giant Machines. It is inspired by Project Jasper, a Central Bank of Corda project on Corda
- The letter of credit CorDapp was built by members of the R3 consortium

And yet the cash generated by the cash CorDapp is completely usable as an input into the letter of credit CorDapp. The Corda network is completely free of silos that restrict digital assets to certain business networks or sets of nodes.

Let's issue that cash now.

<Close the buyer's modal, then click 'Powered by Corda', then tell everyone to watch the series of steps on the left-hand side. An extra step will appear>

We'll transfer some cash from the buyer's bank to the buyer so that he can pay off the letter of credit.

<Click the seventh step, then click 'Transfer', then transfer USD2mn to Analog Imports>

Unwinding the letter of credit redux

Key points introduced: N/A

Points reinforced: N/A

Finally, we can make the final payment and completely unwind the letter of credit.

<Close the cash issuance modal, then click the eight step, then click 'Settle'>

Finally, the buyer has the bill of lading and owns the goods.

<Click 'View' on the bill of lading, then click the sliders in the top-right>

Recap

This demo has highlighted some of the key benefits of blockchain solutions generally:

- The guarantee that when two parties see the same fact, they see the exact same version of that fact
- The ability for parties to transact without intermediaries
- The ability for parties to exchange value

But it has also highlighted some unique features of the Corda platform, features that address the challenges presented by other blockchain platforms:

- Corda uniquely allows privacy, with data only distributed on a need-to-know basis. For example, when the seller and the buyer originally agreed the purchase order, their banks were not aware of this fact
- And importantly, this privacy was achieved in a single global network. Any node can transact directly with any other node, and assets can move across use-cases
- The nodes on the network all have well-known identities, meaning that you know exactly who you're transacting with
- Transaction finality is instant. There's no need to wait for a number of confirmations
- Corda allows you to model complex workflows easily, like the letter of credit workflow we saw today

Business Network Membership Service (BNMS)

https://github.com/corda/corda-solutions/tree/master/bn-apps/memberships-management

Contents of this article assume the reader's familiarity with the concepts of Business Networks and Business Network Operator. Please see this page for more information on this topic.

Business Network Membership Service aims to solve the following problems:

- On-boarding of new members to a business network.
- Suspension of members from a business network.
- Distribution of a membership list to the business network members.
- Association of a custom metadata with a node's identity.

- Participation in multiple business networks by a single node.

BNMS provides the following API extension points:

- Custom membership metadata. Nodes can associate a custom metadata with their memberships. The metadata might contain such fields as role, address, displayed name, email and etc. The metadata can be different for different business networks. It can be even represented with different classes. Associated metadata gets distributed to other Business Network members as a part of general membership distribution mechanism.
- Custom *Membership Contract* implementations. BNOs can extend from `MembershipContract` and add their custom verification logic (for example to verify their custom metadata evolution).
- Automatic acceptance of memberships requests. BNOs can implement a custom verification code that would be run against the incoming membership requests to determine whether they are eligible for auto-activation. Auto-activated members will be able to start transacting straight-away (otherwise a separate Activation step is required).

Please see the design doc for more information about technical design considerations.

Please see FullBNMSFlowDemo for a detailed how-to-use example.

Structure

BNMS contains implementations of flows, states and contracts to model memberships on a Business Network. BNO and members are required to have both of the CorDapps installed.

BNMS consists of 2 CorDapps:

- `membership-service-contracts-and-states` - contracts and states
- `membership-service` - flows for both BNO and member CorDapps

States

Memberships are represented with a MembershipState. Users can associate a custom metadata with their `MembershipState` via `membershipMetadata` field. `MembershipState` is generic and doesn't enforce any restrictions over the type of the metadata.
`MembershipState` can exist in the following statuses:

- `PENDING` - the very first status for all newly issued memberships. To be able to transact on the Business Network `PENDING`memberships need to be activated first.
- `ACTIVE` - active membership holders can transact on the Business Network.
- `SUSPENDED` - Business Network members can be temporarily suspended by their BNO, for example as a result of a governance action. Suspended members can't transact on the Business Network.

Membership contract

`MembershipState` evolution is curated by MembershipContract. By default `MembershipContract` verifies evolution of `MembershipStates` only and doesn't verify an evolution of memberships metadata, as it's a generic parameter.
Membership metadata evolution can be verified in the following ways:

- In the responding flows, by overriding them at the BNO's side (*off-ledger verification*). Will be introduced in Corda 4.
- By extending the `MembershipContract` (*on-ledger verification*). `MembershipContract` is an `open` class and can be extended by users to add a custom verification logic. A custom contract implementation can be provided to the BNMS via `membershipContractName` configuration property.

Flows

BNMS flows are split into 2 packages: `bno` and `member` (with the flows for BNOs and members respectively).
Flows that can be invoked by members:

- `RequestMembershipFlow` - to request a membership.
- `AmendMembershipMetadataFlow` - to update a membership metadata
- `GetMembershipsFlow` - to pull down a memberships list from a BNO. Members retrieve the full list on the first invocation only. All subsequent updates are delivered via push notifications from the BNO. Memberships cache can be force-refreshed by setting `forceRefresh` of `GetMembershipsFlow` to true. Members that are missing from the Network Map are filtered out from the result list.

Flows that can be invoked by BNO:

- `ActivateMembershipFlow` - to activate a `PENDING` membership.
- `SuspendMembershipFlow` - to suspend an `ACTIVE` membership.

Activation and suspension transactions don't require the member's signature. BNO is eligible to suspend memberships unilaterally, for example as a result of a governance action.

Multiple Business Networks

BNMS provides a support for multiple business networks. Business Networks are uniquely identified by BNO's `Party` object. All flows that assume any member -> BNO interactions take BNO's identity as a mandatory parameter.

Membership Auto Approval

If you don't want to manually activate every membership request and would like to instead automate the process then implement the interface `MembershipAutoAcceptor` and set `membershipAutoAcceptor` property in BNO's configuration file.

Configuration

CorDapp configuration is red from `cordapps/config/membership-service.conf` file with a fallback to `membership-service.conf`on the CorDapp's classpath.

Member configuration

```
// whitelist of accpted BNOs. Attempt to communicate to not whitelisted BNO would
result into an exception
bnoWhitelist = ["O=BNO,L=New York,C=US", "O=BNO,L=London,C=GB"]

// Name of the contract to validate membership transactions with.
// Defaults to "net.corda.businessnetworks.membership.states.MembershipContract"
if not specified
membershipContractName = "com.app.MyMembershipContract"
```

BNO configuration

```
// Name of the Notary
notaryName = "O=Notary,L=Longon,C=GB"

// Name of the contract to validate membership transactions with.
// Defaults to "net.corda.businessnetworks.membership.states.MembershipContract"
if not specified
membershipContractName = "com.app.MyMembershipContract"

// Name of the class that implements MembershipAutoAcceptor interface. Optional
parameter.
membershipAutoAcceptor = "com.app.MyMembershipAutoAcceptor"
```

Designing your flows for Business Networks

As Business Networks is an *application level* concept, memberships have to be verified manually inside the flows of your CorDapp. *Corda does not perform any membership checks by itself*.

Logic for a counterparty's membership verification can be found in `BusinessNetworkAwareInitiatedFlow`:

```
/**
 * Extend from this class if you are a business network member and you don't want
to be checking yourself whether
 * the initiating party is also a member. Your code (inside
onCounterpartyMembershipVerified) will be called only after
 * that check is performed. If the initiating party is not a member an exception
is thrown.
 */
abstract class BusinessNetworkAwareInitiatedFlow<out T>(protected val flowSession
: FlowSession) : FlowLogic<T>() {
    @Suspendable
    override fun call(): T {
        verifyMembership(flowSession.counterparty)
        return onOtherPartyMembershipVerified()
    }

    /**
     * Will be called once counterpart's membership is successfully verified
     */
    @Suspendable
    abstract fun onOtherPartyMembershipVerified() : T

    /**
     * Identity of the BNO to verify counterpart's membership against
     */
    abstract fun bnoIdentity() : Party

    @Suspendable
    private fun verifyMembership(initiator : Party) {
        // Memberships list contains valid active memberships only. So we need to
just make sure that the membership exists.
        subFlow(GetMembershipsFlow(bnoIdentity()))[initiator] ?: throw
NotAMemberException(initiator)
    }
}
```

When verifying a membership it's important to make sure that:

- The membership actually exist
- The membership is active
- The membership is verified by the expected contract

The easiest way of making your flow "*business network aware*" is to extend from `BusinessNetworkAwareInitiatedFlow`. Otherwise a counterparty's membership verification would have to be performed manually.
It's important to keep in mind that applications which are designed for Business Networks should be taking the list of available peers from their membership service instead of the Network Map (for example to populate a ComboBox in the UI).

Please note that during development you can take an advantage of extending from `BusinessNetworkOperatorFlowLogic`and `BusinessNetworkOperatorInitiatedFlow` if you are developing custom flows for BNO.

Project Ubin Phase 2 – Corda

https://github.com/project-ubin/ubin-corda

This repository contains the source code and test scripts for the Corda prototype in Project Ubin Phase 2.

Ubin Phase 2 is a collaborative design and rapid prototyping project, exploring the use of Distributed Ledger Technologies (DLT) for Real-Time Gross Settlement.

- Read the **Project Ubin Phase 2 Report** here.
- For more detailed documentation, refer to the Technical Reports: Overview, Corda and Testing.

All CorDapp code is in Kotlin and uses custom Corda libraries which are based on Corda v1.0 with additonal fixes for Project Ubin use cases. The libraries are hosted in the following artifactory: https://ci-artifactory.corda.r3cev.com/artifactory/ubin

A copy of the libraries are also stored in the repository below: https://github.com/project-ubin/ubin-corda-core.git

Additional notes:

- An external service (mock RTGS service) is to be deployed for Pledge and Redeem functions. It can be found in the `ubin-ext-service`
- A common UI can be found in the `ubin-ui` repository

A. Pre-Requisites

You will need the following installed on your machine before you can start:

- JDK 8 installed and available on your path.
- Latest version of IntelliJ IDEA (note the community edition is free)
- h2 web console (download the "platform-independent zip")
- Git

For more detailed information, see the getting set up page on the Corda docsite.

B. Getting Started

To get started, clone this repository with:

```
git clone https://github.com/project-ubin/ubin-corda.git
```

C. Build CorDapp

1. Go to newly created folder

```
cd ubin-corda
```

2. Build CorDapp with Gradle

```
$ ./gradlew clean build deployNodes
```

3. CorDapp can be found at:

```
ubin-corda/build/nodes/<anyofthefolder>/plugins/
```

```
# Example:
ubin-corda/build/nodes/MASRegulator/plugins/
```

D. Running the network locally:

```
$ cd build/nodes
$ ./runnodes
```

All the nodes (defined in the deployNodes gradle task in the project root) will start in console. From this point the CorDapp can be controlled via web APIs or websites hosted by the nodes.

Set Up New Network

Note: Following steps have been tested in Ubuntu 16.04 LTS

A. Pre-Requisites

You will need the following components set up/installed:

- 15 Ubuntu (Xenial - LTS 16.04) VMs (11 banks, 1 MAS Central Bank node, 1 MAS as Regulator node, 1 Network Map, 1 Notary) with minimum specifications of 1 core, 3.5 GB RAM
- JDK 8 installed and available on your path.
- Git

B. Network setup step:

1. Set up network map

2. Set up notary

3. Set up bank nodes

4. Set up Ubin external service (MEPS+ mock service)

The script `configure.sh` from `ubin-corda-deployment` repository takes in 5 input parameters:

- Node type (value: networkmap, notary, node)
- Virtual machine (VM) username
- Network Map Name
- Notary type (default value is "nonValidating")
- Network Map IP address

1. Set Up Network Map

1. SSH to Network Map virtual machine.

2. Clone `ubin-corda-deployment` repository

```
$ git clone https://github.com/project-ubin/ubin-corda-deployment.git
```

3. Determine network map node name (e.q. Network Map)

4. Get virtual machine IP address with following command:

```
$ hostname -i
```

5. Execute `configure.sh` script with:

```
sudo ./configure.sh networkmap <<VM Username>> <<Network Map Name>> nonValidating
<<Network Map IP>>
```

Example:

```
# Network map name is "Network Map"
# Network map IP address is "10.0.0.47"
# VM username is "dltusr"

chmod +x configure.sh
sudo ./configure.sh networkmap dltusr "Network Map" nonValidating 10.0.0.47
```

Note: Network map IP address is required in the set up of notary node and additional bank nodes

2. Set Up Notary

1. SSH to Notary virtual machine.

2. Clone `ubin-corda-deployment` repository

```
$ git clone https://github.com/project-ubin/ubin-corda-deployment.git
```

3. Determine Notary node name (e.g. Notary)

4. Get network map IP Address from the previous step (network map setup).

5. Execute `configure.sh` script with:

```
sudo ./configure.sh networkmap <<VM Username>> <<Network Map Name>> nonValidating
<<Network Map IP>>>
```

Example:

```
# Notary name is "Notary"
# Network map IP address is "10.0.0.47"
# VM username is "dltusr"
$ chmod +x configure.sh
$ sudo ./configure.sh notary dltusr "Notary" nonValidating 10.0.0.47
```

3. Set Up Bank Nodes

1. SSH to bank nodes virtual machine.

2. Clone `ubin-corda-deployment` repository

```
$ git clone https://github.com/project-ubin/ubin-corda-deployment.git
```

3. Determine bank node name (usually the bank SWIFT code).

4. Get network map IP Address from the previous step (network map setup).

5. Go to `ubin-corda-deployment` directory.
6. Verify `config.properties` to ensure `ApproveRedeemURI` is configured with the Central Bank domain name.

```
ApproveRedeemURI=http://<<host>>:9001/meps/redeem
```

Note: Replace host with Central Bank virtual machine host/domain name.

7. Execute `configure.sh` script with:

```
$ sudo ./configure.sh notary <<VM Username>> <<Notary Name>> nonValidating
<<Network Map IP>>
```

Example:

```
# Nodename name is "BankA"
# Network map IP address is "10.0.0.47"
# VM username is "dltusr"

$ chmod +x configure.sh
$ sudo ./configure.sh node dltusr "BankA" nonValidating 10.0.0.47
```

Note: do not name the Corda node with a name containing "node".

4. Setup Ubin External Service

Ubin external service should be set up in the `Central Bank` virtual machine. This is a mock service of the current RTGS system, MEPS+.

Build

1. Clone the repository locally

```
$ git clone https://github.com/project-ubin/ubin-ext-service.git
```

2. Go to newly created folder

```
$ cd ubin-ext-service
```

3. Build project using gradle

```
$ ./gradlew build
```

4. Build artifact can be found at

```
build/libs/ubin-ext-service-0.0.1-SNAPSHOT.jar
```

Start External Service

1. Update the `application.properties` file

```
ubin-ext-service/application.properties
```

With Corda configurations:

```
PledgeURI=http://<host>:9001/api/fund/pledge
RedeemURI=http://<host>:9001/api/fund/redeem
Dlt=Corda
```

Note:

- Replace host with Central Bank host/domain name.
- `RedeemURI` is not used in Corda, it is just a placeholder.

2. Copy built JAR artifact and properties files to the Central Bank VM

```
ubin-ext-service/build/libs/ubin-ext-service-0.0.1-SNAPSHOT.jar
ubin-ext-service/application.properties
```

Note: Ensure both files are in the same directory

3. From Central Bank VM, start the mock service application

```
$ java -jar -Dspring.config.location=application.properties -Dserver.port=9001
ubin-ext-service-0.0.1-SNAPSHOT.jar
```

Central Deployment and Server Admin

A. Getting Started

1. Go to Node 0 Virtual Machine

2. From the home directory, clone the `ubin-corda-deployment` repository

```
$ git clone https://github.com/project-ubin/ubin-corda-deployment.git
```

3. Update configuration variable with target environment detail

```
username='username'
host_prefix='cordavm'
host_suffix='.example.com'
notary_host='notary.example.com'
networkmap_host='networkmap.example.com'
nm_db_url='jdbc:h2:tcp://networkmap.example.com:11000/node'
nm_db_user='sa'
```

Note:

- Populate username with VM username - This script assumes all vm in the network has the same username
- This script assumes VMs hostname is in following format:
- `  <host_prefix><number><host_suffix>`
- Number in hostname is assumed to be in consecutive order.
- `#Example:`
-

```
• node1.example.com
• node2.example.com
• ...
• node13.example.com
•
• #host_prefix: node
• #host_suffx: .example.com
```

B. CorDapp Deployment

1. Copy CorDapp JARs into VM Node 0 into the following directory with SCP/FTP:

```
/home/<username>/ubin-corda-deployment/plugin
```

Note: Replace `username` with your VM username
2. Log in to VM Node 0 using SSH

3. Go to `ubin-corda-deployment` directory
4. Execute manage.sh to deploy CorDapp to all nodes in the network except the Notary and the Network Map. The script assumes that the nodes are named sequentially (e.g. 0 to 12):

```
$ ./manage.sh deploy 0 12
```

This step does the following:

- Delete everything in /app/corda/plugins
- Copy all plugins file from deployment node (VM Node 0) to the target node's /app/corda/plugins folder
- Restart Corda and webserver in the node
- Repeat for selected Nodes

Note: 0 and 12 in Step 4 represents the range of nodes. If you only require deployment on nodes 2-4, change the parameters to "deploy 2 4".

C. Restart All Nodes

1. Log in to VM Node 0 using SSH

2. Go to `ubin-corda-deployment` directory
3. Execute `manage.sh` to restart all nodes in the network (e.g. Node 0 - Node 12):

```
$ ./manage.sh restart 0 12
```

Note: 0 and 12 in Step 3 represents the range of nodes. If you only require deployment on nodes 2-4, change the parameters to "deploy 2 4".

D. Stop All Nodes

1. Log in to VM Node 0 using SSH

2. Go to `ubin-corda-deployment` directory
3. Execute `manage.sh` to stop all nodes in the network (e.g. Node 0 - Node 12):

```
$ ./manage.sh stop 0 12
```

Note: 0 and 12 in Step 3 represents the range of nodes. If you only require deployment on nodes 2-4, change the parameters to "deploy 2 4".

E. Clear All Data in Vault

1. Log in to VM Node 0 using SSH

2. Check that you are in `home` directory
3. Go to `ubin-corda-deployment`
4. Stop all Corda nodes (node 0 to 12) using:

```
$ ./manage.sh stop 0 12
```

5. Clear all data in network (node 0 to 12) - note that the data is *not recoverable*:

```
$ ./manage.sh reset 0 12
```

6. Restart all nodes 0 to 12:

```
$ ./manage.sh restart 0 12
```

F. Update Node Names

1. Stop the Corda server and webserver

2. Go to `/app/corda`
3. Update `node.conf` with command:

```
$ vi node.conf
```

4. Change `myLegalName` in "O" key :

```
"O=BOTKSGSX, L=Singapore, C=Singapore"
```

5. Delete the `certificates` folder. The certificates will be regenerated automatically upon Corda server start up
6. To purge the old entries in the Network Map, login to the h2 DB of the Network Map (nm0) and delete the old entries in `NODE_NETWORK_MAP_NODES` table
Note:

As of Corda v1.0, 'CN' / Common Name field in the X500 is no longer supported. It is used only for services identity. In addition, words such as "node" is blacklisted in CordaX500Name and therefore should not be used.

G. Database Admin

(Based on instructions in https://docs.corda.net/node-database.html)

1. Install h2 client locally and run h2.sh (Unix) or h2.bat (Windows)

2. Enter the JDBC URL with format:

```
jdbc:h2:tcp://<<CORDA NODE HOST>>:<<H2 DATABASE PORT>>/node

# Example:
jdbc:h2:tcp://<host>:11000/node
```

Note: Replace host with the node virtual machine host/domain name

3. Username and password is as per default 4. Connect to browse h2 DB

H. Individual Node CorDapp Deployment

1. Copy new/updated CorDapp JARs to the following directory in the Corda Node VM:

```
/app/corda/plugins
```

2. Restart Corda nodes:

```
$ sudo systemctl restart corda
$ sudo systemctl restart corda-webserver
```

Note:

- Replace "restart" with "start" or "stop" to simply start/stop the services
- Replace "restart" with "status" to view the latest logs
- Log location: `/app/corda/logs`
- Once the CorDapp JARs are uploaded and replaced, the Corda server (service) must be restarted

Test Scripts

Postman is the main testing tool used for this prototype. The Postman collection and environments are located in the test-scripts folder in this repository. The API definitions can be found in the Technical Report repository.

License