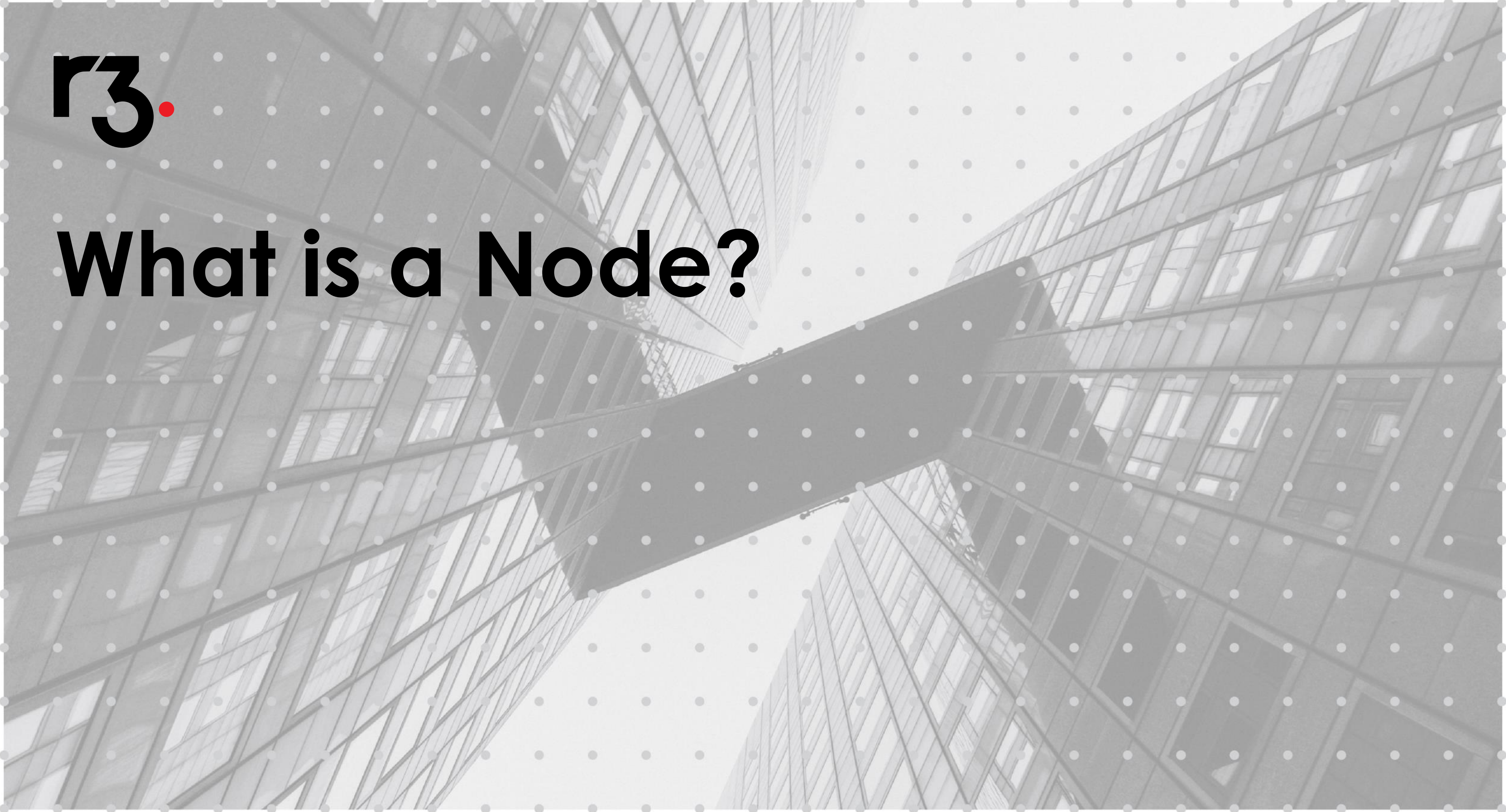
Module 9

The Node

Learning outcomes

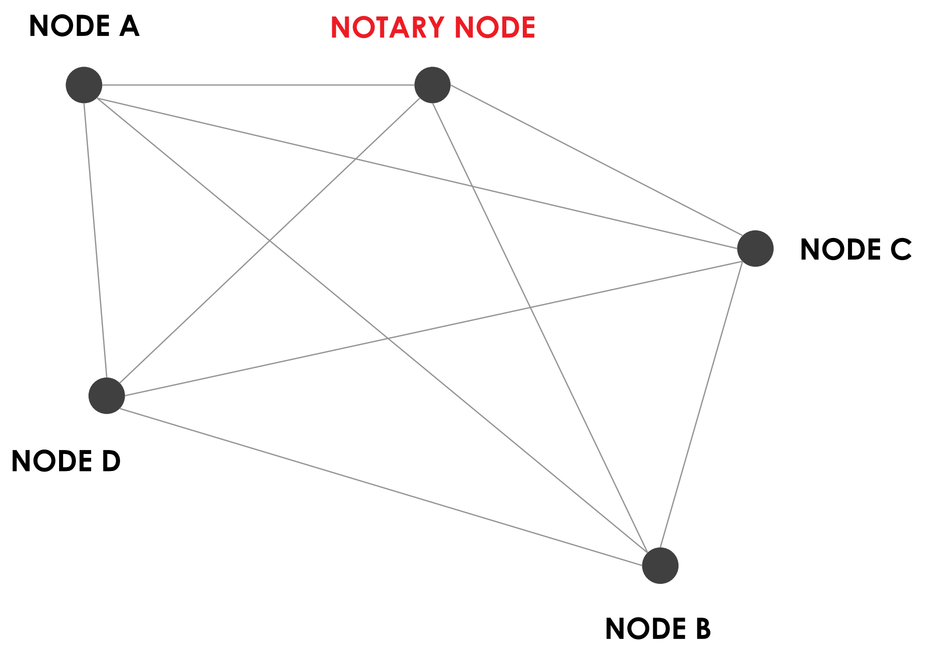
- Understand how the node is structured
- Learn what services are available to the node internally
- Learn what APIs are available to interact with the node
- Learn how to deploy and launch nodes



What is a Node?

Corda networks

- A Corda network is made up of nodes:

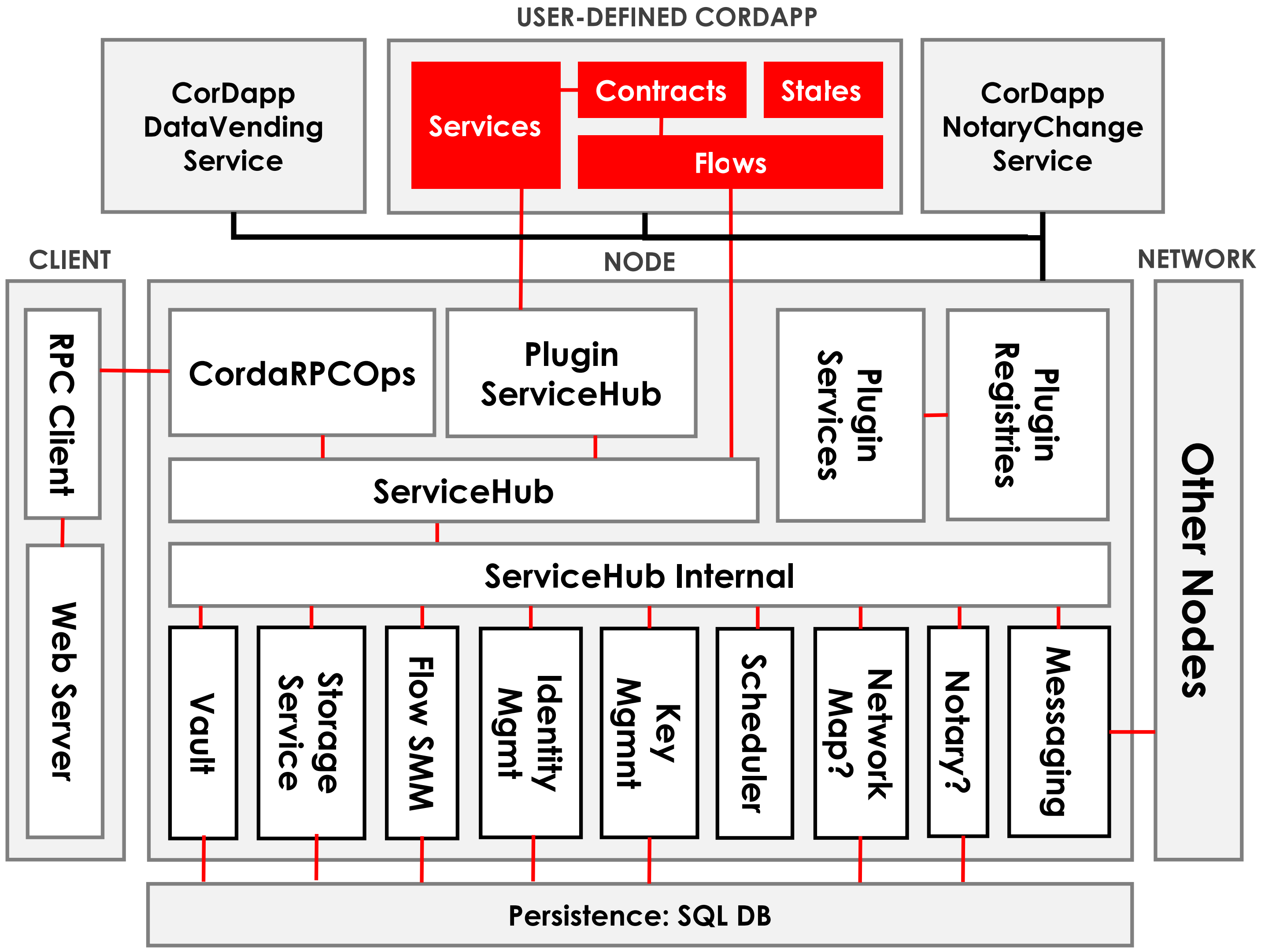


Defining a node

- A node is an instance of the Corda software with a unique identity on the network
- Each node's identity is provided by a certificate signed by a network root authority
- Networks can decide what constitutes a valid identity – legal names, usernames, IP addresses…

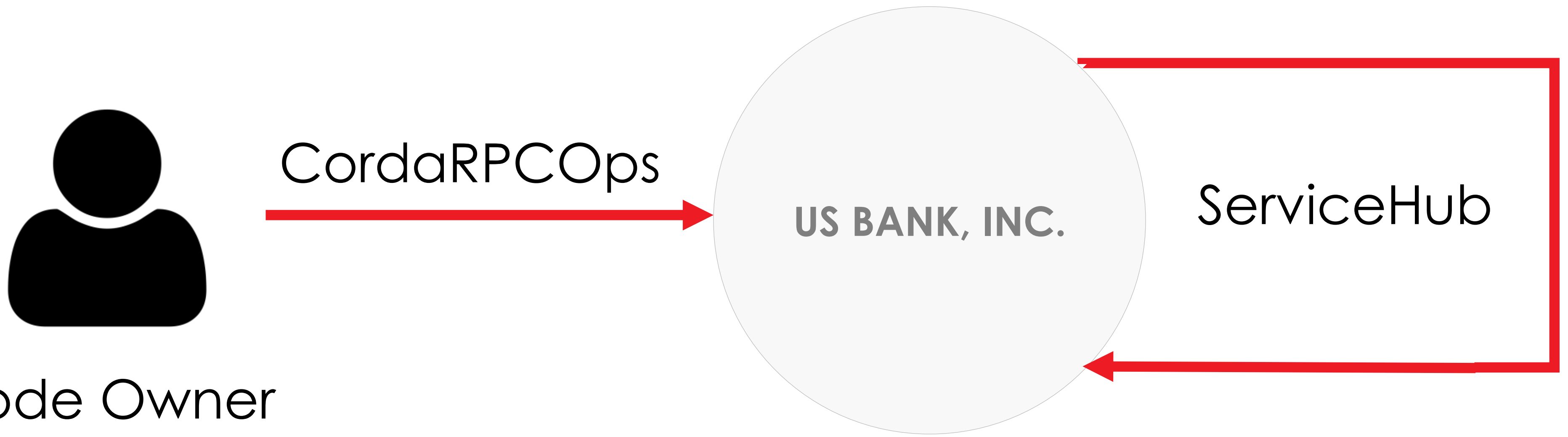
US BANK, INC.
ALICE
141.201.27.48

Node internals



ServiceHub vs. CordaRPCOps

- **CordaRPCOps** defines how the owner interacts with their node
- **ServiceHub** defines how the node accesses its services internally



The vault

- The node stores data in two ways:
  - **The vault**, which represents the node's personal ledger:
    - **Consumed states**, which represent an immutable ledger of the owner's past activity
    - **Unconsumed states**, which represent the owner's "balance"
    - **Attachments**, binary files linked to a transaction
  - **Local storage** for all other data (eg. transactions, metadata) that is needed for the node to interact with the ledger

Default Node Services

The ServiceHub interface

- **ServiceHub** represents the operations available internally to the node:

```
interface ServiceHub {
    val myInfo: NodeInfo
    val identityService: IdentityService
    val attachments: AttachmentStorage
    val vaultService: VaultService
    val keyManagementService: KeyManagementService
    val validatedTransactions: TransactionStorage
    val networkMapCache: NetworkMapCache
    val clock: Clock
```

The ServiceHub

- The key operations provided by the **ServiceHub** can be divided into four types:
  1. Information on the node’s identity and keys
  2. Information on other network nodes
  3. Retrieving the current time and scheduling events
  4. Retrieving or recording data in the vault or local storage

The Observable pattern

- Many **ServiceHub** and **CordaRPCOps** methods return an instance of the **rx.Observable** class

- The observable pattern is as follows:
  - The observable emits events
  - An observer can subscribe to the observable
  - The observer will then be notified of any emitted events

- The observable pattern allows the node's owner to be automatically notified of updates to things such as the network map, the vault, and in-progress flows

1. Node information – myInfo

- Information on the node is obtained from **ServiceHub.myInfo**, which returns a **NodeInfo** instance:

```
data class NodeInfo(
   val addresses: List<NetworkHostAndPort>,
   val legalIdentities: List<Party>,
   val legalIdentitiesAndCerts: Set<PartyAndCertificate>,
   val platformVersion: Int,
   val serial: Long)
```

- Where:
  - **addresses** are the Artemis MQ address that allows other nodes to send the node messages
  - **legalIdentitiesAndCerts** allows other nodes to identify this node

2. Network information – networkMapCache

- Communication between network nodes is point-to-point (i.e. between named nodes)
- To look up other nodes, the node uses **ServiceHub.networkMapCache**, which implements **NetworkMapCache**
- **NetworkMapCache** tracks four kinds of nodes:
  - **partyNodes**: every node on the network
  - **networkMapNodes**: nodes advertising a network map service
  - **notaryNodes**: nodes advertising a notary service
  - **regulatorNodes**: nodes advertising a regulatory service

2. Network information – networkMapCache

- The **networkMapCache** provides methods for retrieving nodes based on:
  - Name
  - Public key
  - Service advertised

- The **track()** method allows the node to learn about changes to the network map over time:

```
fun track(): DataFeed<List<NodeInfo>, MapChange>
```

2. Network information – Looking Up Nodes

- Some **networkMapCache** methods to look up nodes:

```
fun getNodeByLegalName(principal: CordaX500Name): NodeInfo?

fun getNodeByLegalIdentityKey(identityKey: PublicKey): NodeInfo?

fun getNodeByLegalIdentity(party: AbstractParty): NodeInfo?

fun getNodeByAddress(address: NetworkHostAndPort): NodeInfo?
```

3. Time information – clock

- In Corda, there is no network time. Each node tracks time separately
- Each node tracks time using **ServiceHub.clock**, which embeds an instance of Java's built-in **Java.Clock**
- This clock is used to choose time-stamps for transactions

4. Data-related services – The Vault

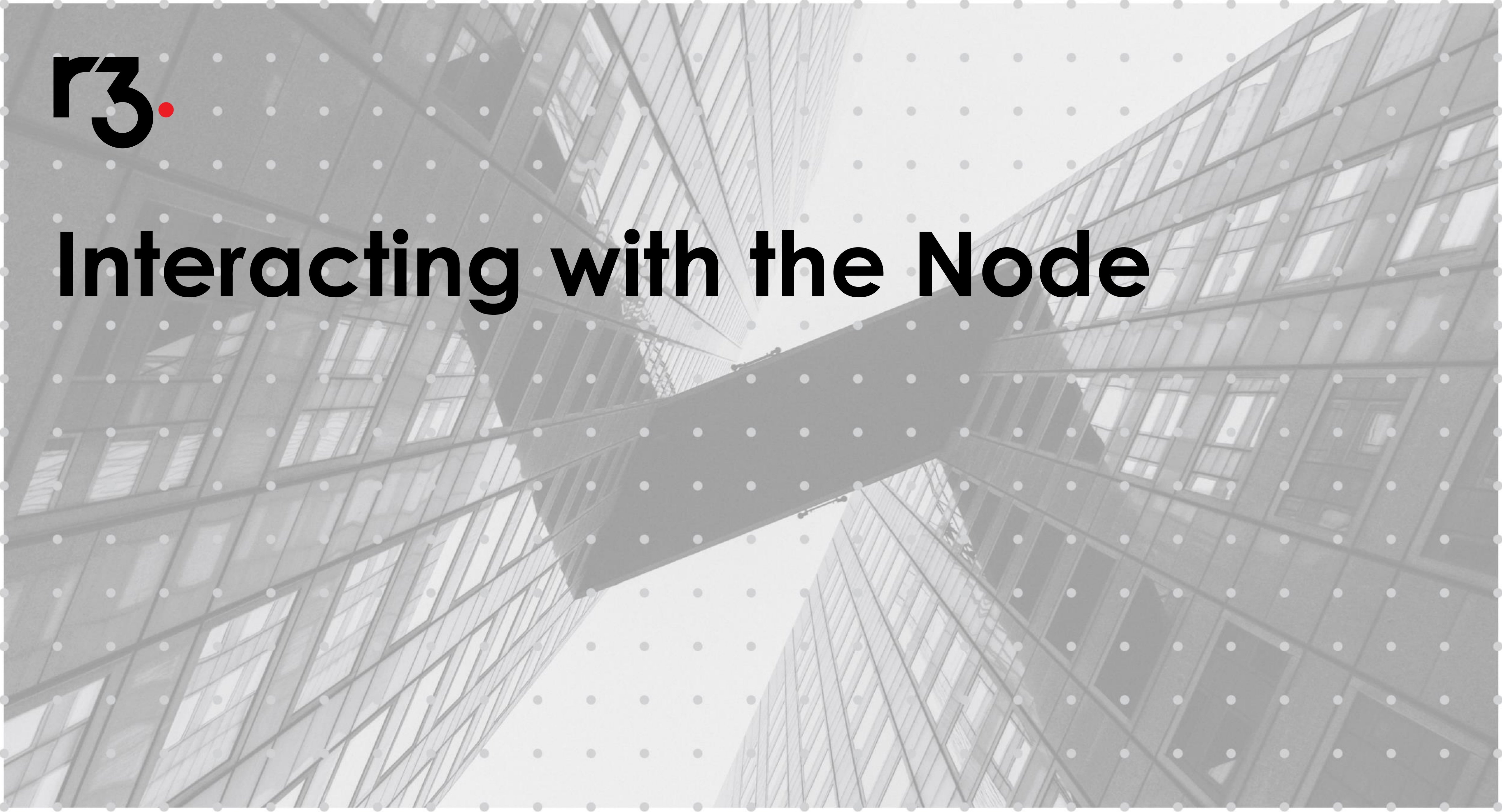
- The vault is accessed via **ServiceHub.vaultService**
- A rich API is available for querying the vault: https://docs.corda.net/api-vault-query.html
- A simple example: here is how we could extract all **IOUState**s:

```
serviceHub
    .vaultService
    .queryBy<IOUState>(queryCriteria)
    .states
```

4. Data-related services – Storage

- The node’s local storage is only accessed using:
  - **validatedTransactions**, which provides methods to access the node’s stored transactions

Interacting with the Node



Interacting with the node

- A node's owner interacts with the node using the **CordaRPCOps** interface
- The key operations in **CordaRPCOps** are:
  - **startTrackedFlowDynamic** to start flows
  - **vaultQueryBy** to access the vault
  - Methods to check for, upload and open attachments

Starting flows

- You command a node to start a flow using:

```
fun <T : Any> startTrackedFlowDynamic(
    logicType: Class<out FlowLogic<T>>, vararg args: Any?)
: FlowHandle<T>
```

- This method returns a **FlowProgressHandle**:

```
data class FlowProgressHandle<A>(
    val id: StateMachineRunId,
    val progress: Observable<String>,
    val returnValue: ListenableFuture<A>)
```

- Where:
  - **progress** is an **Observable** allowing you to track the flow's progress
  - **returnValue** is a **ListenableFuture** representing the asynchronous result of running the flow

Accessing the vault

- The node owner accesses the contents of the vault using:

```
fun vaultQueryBy(
   criteria: QueryCriteria,
   paging: PageSpecification,
   sorting: Sort): Vault.Page<T>
```

- This returns a page of the states matching the vault query criteria

Attachments

- CordaRPCOps defines three methods for interacting with attachments:

```
fun attachmentExists(id: SecureHash): Boolean

fun openAttachment(id: SecureHash): InputStream

fun uploadAttachment(jar: InputStream): SecureHash
```

Implementation

Implementation

- One implementation of a node is **AbstractNode** (see **net.corda.node.internal.AbstractNode**)
- **AbstractNode**'s **ServiceHub** is initialized in its **start()** method
- The **AbstractNode** class is itself abstract, and subclasses must provide implementations for (at least):
  - val log: Logger
  - val networkMapAddress: SingleMessageRecipient?
  - val serverThread: AffinityExecutor
  - fun makeMessagingService(): MessagingServiceInternal
  - fun startMessagingService(rpcOps: RPCOps)

Implementation – Recording transactions

AbstractNode records transactions using **ServiceHubInternal.recordTransactions(**notifyVault: Boolean, txs: Iterable<SignedTransaction>**)**

This will:

1. Records the transaction in local storage
2. Notify the vault

Implementation

- Corda is agnostic about how the available RPC operations are implemented

- One implementation is **net.corda.node.internal.CordaRPCOpsImpl**

- In **CordaRPCOpsImpl**, transactions and attachments are stored as blobs in key-value mappings – see:
  - DBTransactionStorage
  - NodeAttachmentService

The node in summary

- Users interact with a node using a set of RPC operations
- Internally, a node uses its ServiceHub to retrieve and record data, and to start flows
- **AbstractNode** is one implementation of the node architecture, but other implementations are possible

Practical

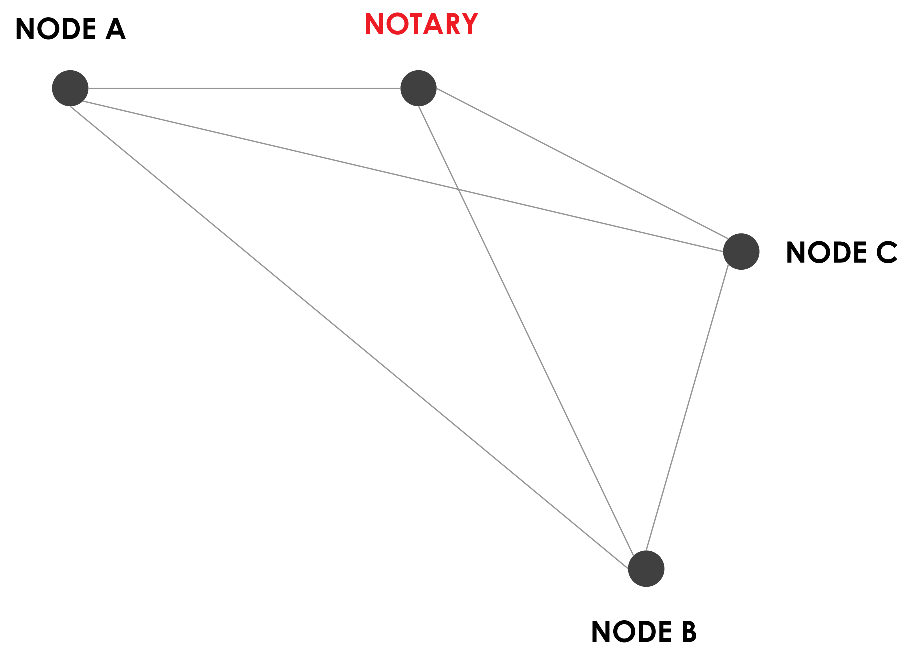
Designing the Network

- We are now going to move beyond the test DSLs and make this real:
  - We'll define the nodes on our network
  - We'll deploy (i.e. build) the nodes we've defined
  - We'll run the nodes

Step 1 – Defining the Nodes

Network Layout

- We will use the following network design:



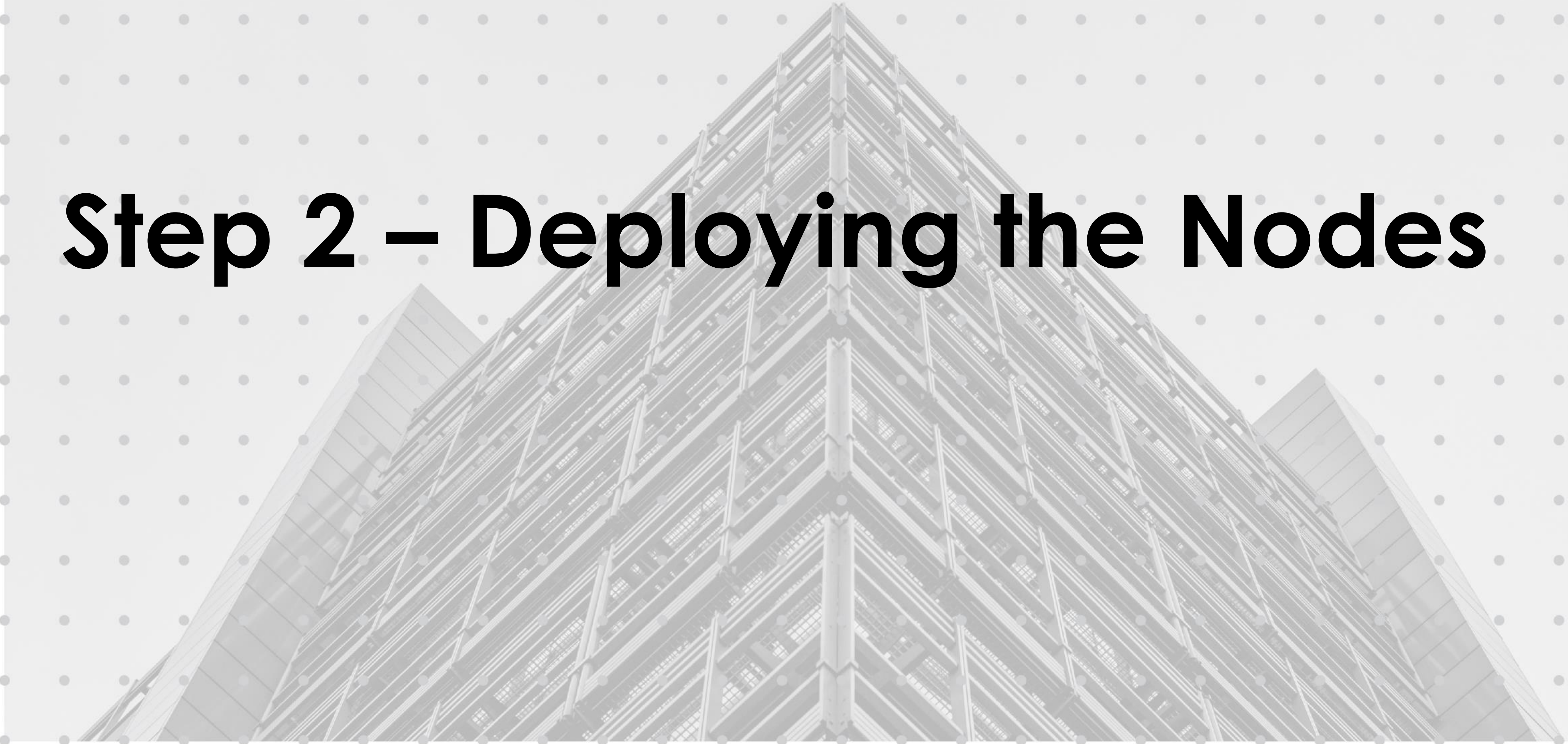
The DeployNodes Task

- Nodes are built using the Gradle build system
- Gradle builds are organized into tasks
- Nodes are created using the **deployNodes** task:

```
        node {
                name "O=Network Map Service,L=London,C=GB"
advertisedServices = ["corda.notary.validating"]
                p2pPort 10002
                rpcPort 10003
                cordapps = []
        }
```

- **deployNodes** extends the Cordform plugin:
  – http://jcenter.bintray.com/net/corda/plugins/cordformation/

Step 2 – Deploying the Nodes



Deploying the Nodes

- The **deployNodes** task for the IOU CorDapp is defined in build.gradle

- The build.gradle file shows that our network is made up of 4 nodes:
  - The Network Map Service, who runs the notary and the network map
  - Three regular nodes:
    - ParticipantA
    - ParticipantB
    - ParticipantC

- Running the **deployNodes** task will use these configs to build actual nodes

Deploying the Nodes - Instructions

Goal	Deploy the nodes
Where?	The command line
Steps	1. From the command line, move to the root of your CorDapp directory 2. Deploy the nodes: • Unix: **./gradlew deployNodes** • Windows: **gradlew deployNodes** 3. Navigate to the build/nodes folder to view the deployed nodes
Key Docs	N/A

Step 3 – Running the Nodes

The runnodes Script

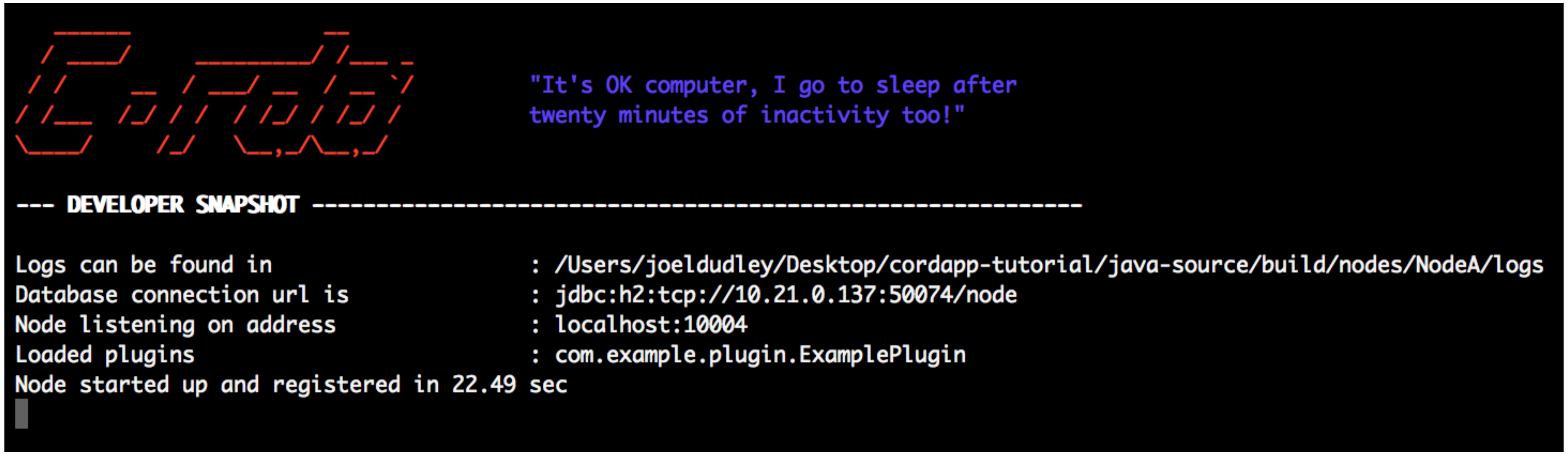
- The deployed nodes are found in the **build/nodes** folder
- Each node folder contains a **corda.jar** file of our IOU CorDapp
- We can use the **runnodes** script to start up all the nodes

The runnodes Script - Instructions

Goal	Run the nodes
Where?	The command line
Steps	1. On the command line, navigate to the build/nodes folder to view the deployed nodes 2. Execute the **runnodes** script, using: • Unix: **sh runnodes** • Windows: **runnodes** 3. All the nodes will start up 4. There should be seven terminal windows in all
Key Docs	N/A

The Node Terminal Window

- The first four terminal windows show information on each node:



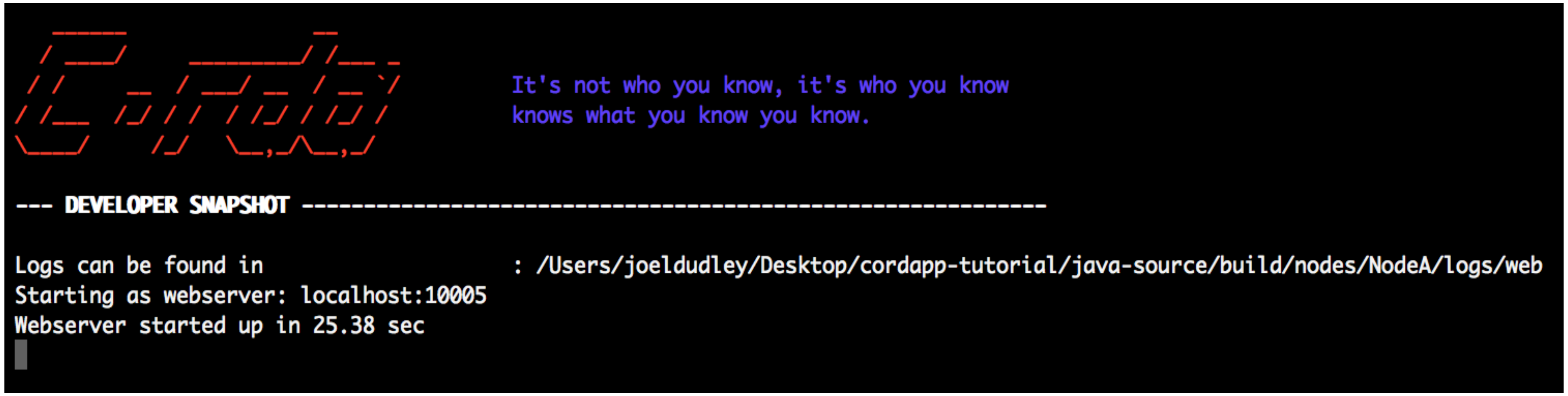
```
   ______               __
  / ____/     _________/ /___ _
 / /     __  / ___/ __  / __ `/         "It's OK computer, I go to sleep after
/ /___  /_/ / /  / /_/ / /_/ /          twenty minutes of inactivity too!"
\____/     /_/   \__,_/\__,_/

--- DEVELOPER SNAPSHOT ------------------------------------------------------------

Logs can be found in                    : /Users/joeldudley/Desktop/cordapp-tutorial/java-source/build/nodes/NodeA/logs
Database connection url is              : jdbc:h2:tcp://10.21.0.137:50074/node
Node listening on address               : localhost:10004
Loaded plugins                          : com.example.plugin.ExamplePlugin
Node started up and registered in 22.49 sec
```

The Webserver Terminal Window

- The remaining four windows show each node’s webserver:



```
   ______               __
  / ____/     _________/ /___ _
 / /     __  / ___/ __  / __ `/         It's not who you know, it's who you know
/ /___  /_/ / /  / /_/ / /_/ /          knows what you know you know.
\____/     /_/   \__,_/\__,_/

--- DEVELOPER SNAPSHOT ------------------------------------------------------------

Logs can be found in                    : /Users/joeldudley/Desktop/cordapp-tutorial/java-source/build/nodes/NodeA/logs/web
Starting as webserver: localhost:10005
Webserver started up in 25.38 sec
```

Checkpoint – Progress So Far

Our progress so far

- Our nodes are up and running
- But we still don't have a way to interact with them
- For that, we'll be building a simple REST API