春の陣：
Next-Gen Corda (Corda 5) アップデート

齊藤博之（SBI R3 JAPAN）

自己紹介

何か疑問があれば
hiroyuki.saito@sbir3japan.co.jp まで

齊藤 博之（さいとう ひろゆき）

AWS Certified Solutions Architect - Associate
Salesforce Certified Platform Developer – 1
XCS251 - Cryptocurrencies and Blockchain Technologies (Stanford Online)

- SBI R3 Japan：ビジネス推進部長
（SBI Holdings ブロックチェーン推進室兼務）

＜略歴＞東京都出身

ＩＴ⇔金融

学生時代　　：計量経済学（早稲田大学）
証券会社　　：日本の証券会社で個人営業
ITベンチャー：香港でEメールマーケティングのシステム開発
ITベンダー　：インドで不正検知システムのオフショア開発
金融機関　　：日本の銀行で勘定系システム開発（ローン、内為、ATM、ネオバンク）
ITベンチャー：オランダ企業の日本支社でデジタルバンキングシステム開発

2023年SBIへJoin!

アジェンダ

1. ロードマップ
2. Next Gen Corda (Corda5）概要
3. Corda 4とCorda 5の技術比較
    1. 変更点
    2. スケーラビリティ、高可用性、運用コスト削減、 Open Core戦略
4. ワーカー、仮想ノード
5. CPIのビルド・デプロイ、仮想ノード作成・登録
6. デモ（CSDE + Combined Worker）
7. 参考情報（Kubernetes + minikubeでクラスタ環境を作る場合の注意点）
8. Appendix

1. ロードマップ（日付はすべて２０２３年）

ベータ2
3/31

ベータ2
3月末

ベータ3
5/3

ベータ3
5/3

発表イベント
6/14

GA（一般公開）
６月末

5.X
顧客移行
11月末

顧客移行
11月末

2. Next-Gen Corda (Corda 5) 概要

- C4→C5において概念レベルでの変化はない
  - Flow, Transaction, State etc...
  - Notary, Node

- 商用運用ニーズに応えるためにインフラ技術を全面刷新
  - スケーラビリティ
  - 高可用性
  - 運用コスト削減

- Rest化とマイクロサービス化に伴って、SDKとしてのAPIは一新される

2. Next-Gen Corda (Corda 5) 概要

規制ビジネスにおける大規模展開。

・Corda 4がベース

✓ オープン
✓ 信頼
✓ 永続的

なデジタルエコノミーの実現。

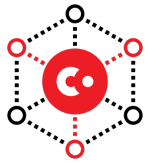
・Cordaアプリ間の相互運用
・パブリックチェーンとの接続

オープンコア。1つのコードベース。

3. Corda 4とCorda 5の技術比較

3.1. 主な変更点：Corda 4とCorda 5の比較

統合

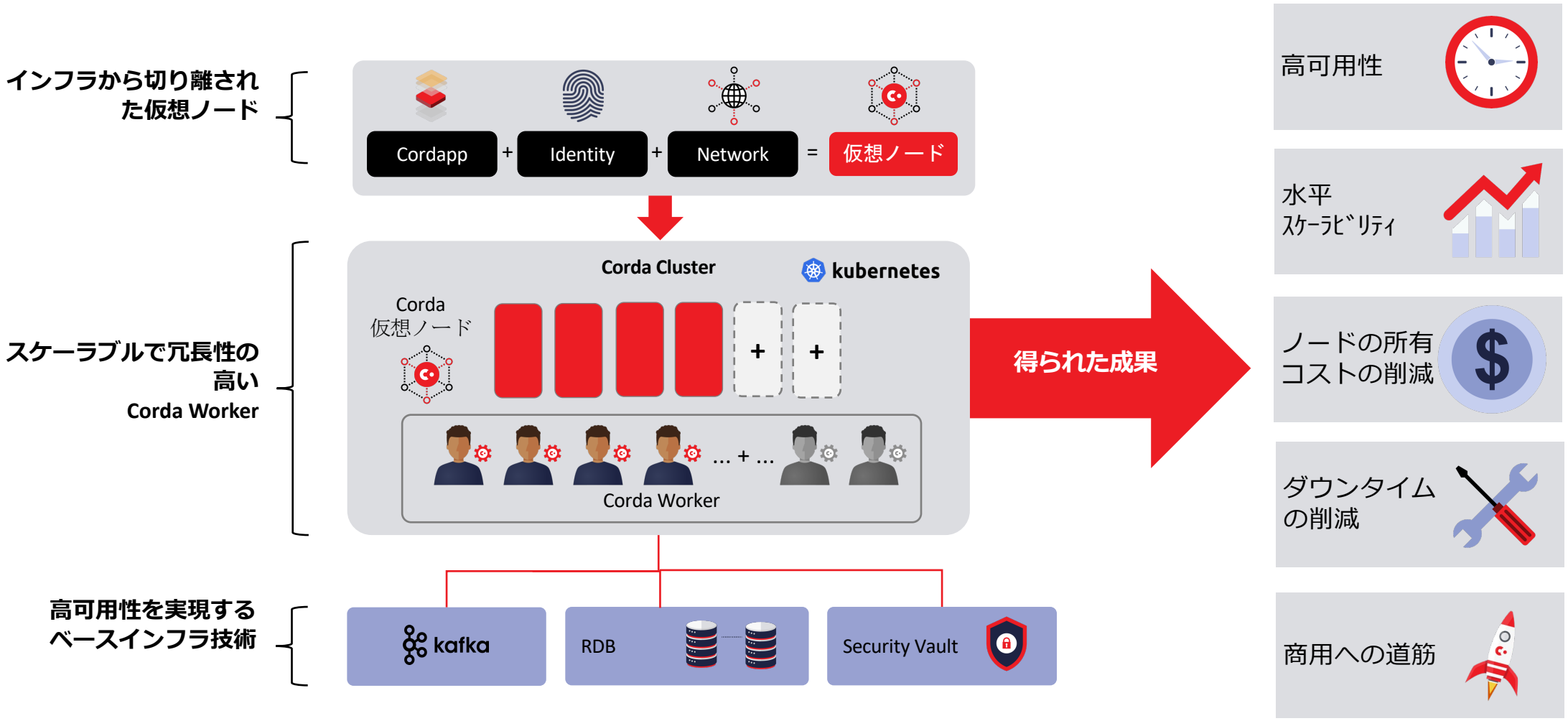


アプリネットワーク

オープンソースが核

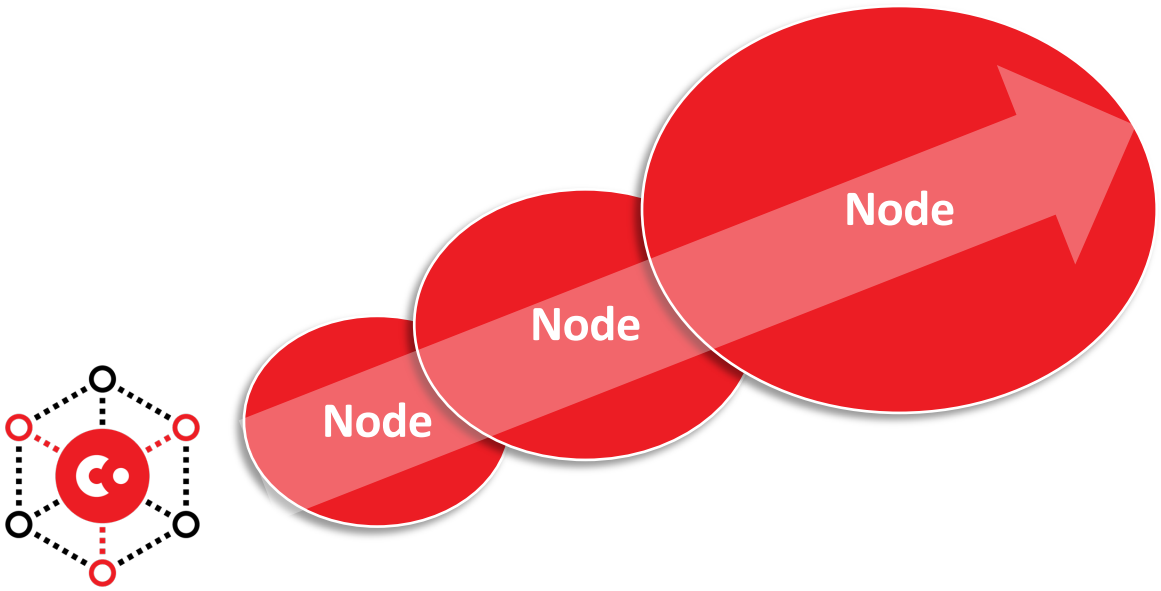
[Corda 4]		[Corda 5]	
Artemisの活用	→	RESTful APIでの実装	開発期間の短縮
複数アプリケーションで台帳を共有	→	個別アプリケーション／ネットワーク専用の台帳を仮想ノードが管理	プライベートで独立したネットワークの実現
オープンソースの提供は限定的	→	Open Core 戦略の採用	自由に構築し、必要に応じて技術サポートを確保

3.1. 商用顧客の声を元にした新しいアーキテクチャー



3.2. スケーラビリティ

Corda 4：垂直方向のスケーリング

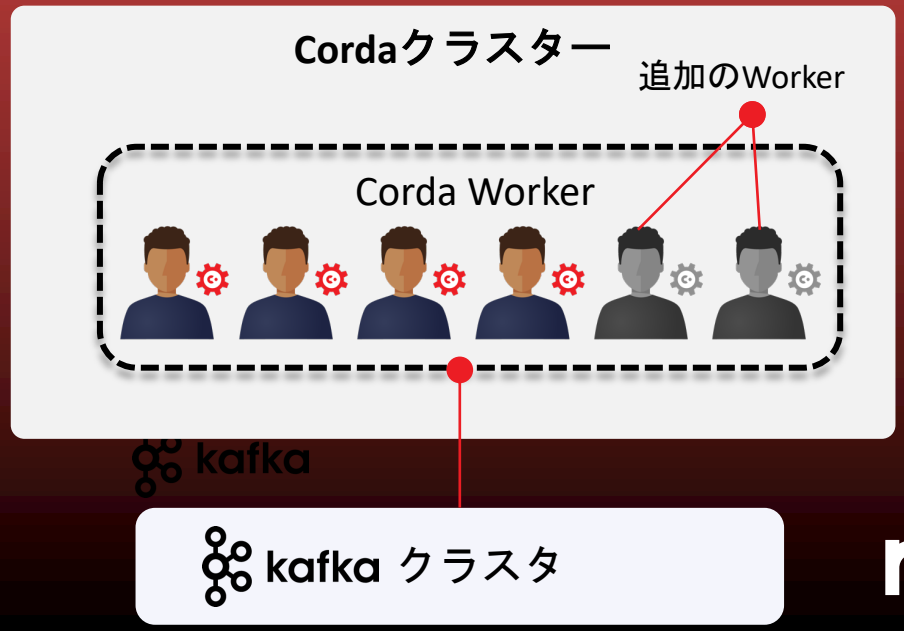


大きなノード => 大きなJVM（プロセッサ＋メモリ）

１ノード１JVMの制約により
- 性能向上には、JVMを拡張しかない

Next-gen Corda (Corda 5)

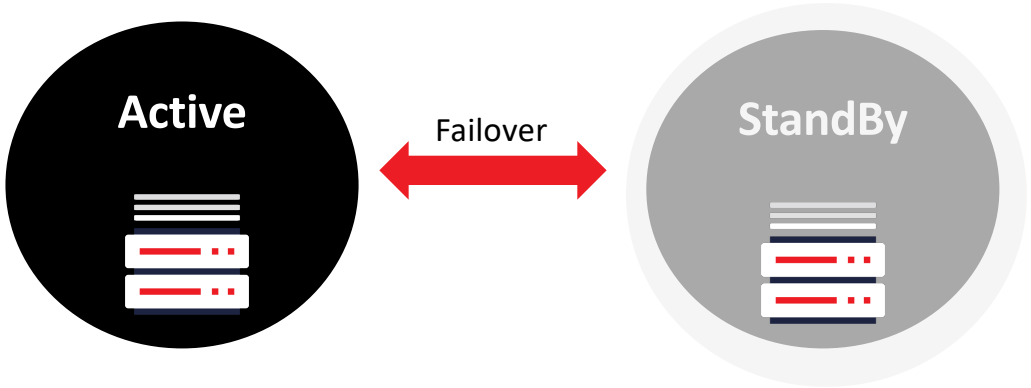
- Event Driven化しスループットを向上
- Workerの追加によるパフォーマンス向上
- 低負荷／無負荷時にはスケールダウン
- Docker /Kubernetes 技術に依存



3.2. 高可用性

Corda 4：

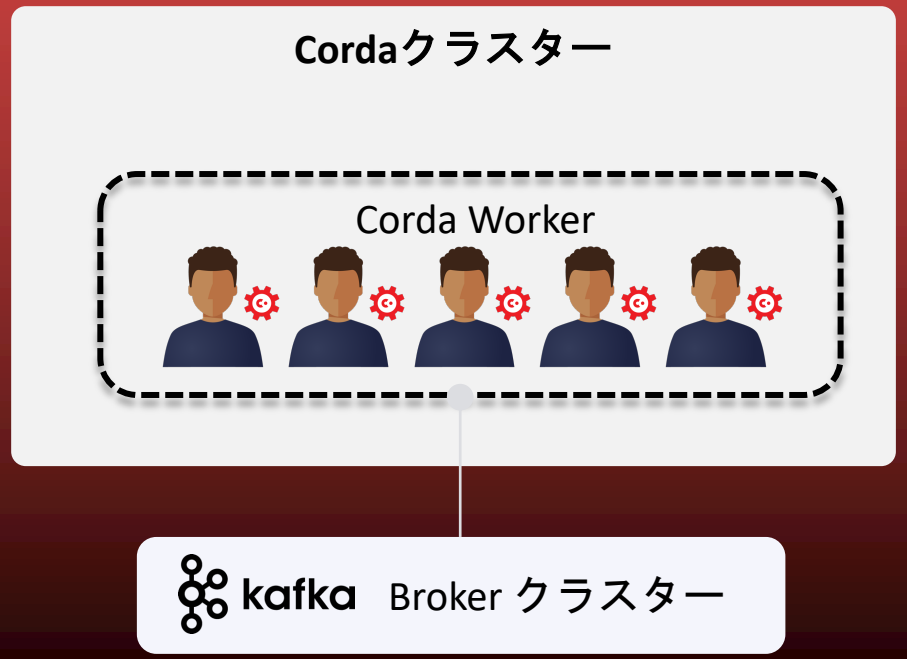
アクティブノード＋スタンバイノード



１ノード１JVMの制約により
- ホット/ホット構成は不可
- アクティブ/アクティブなフェイルオーバーは不可
- スタンバイノードは可能だが、ホット/コールド（アクティブ/スタンバイ）構成のみ
- 停止時間ゼロのフェイルオーバーは不可

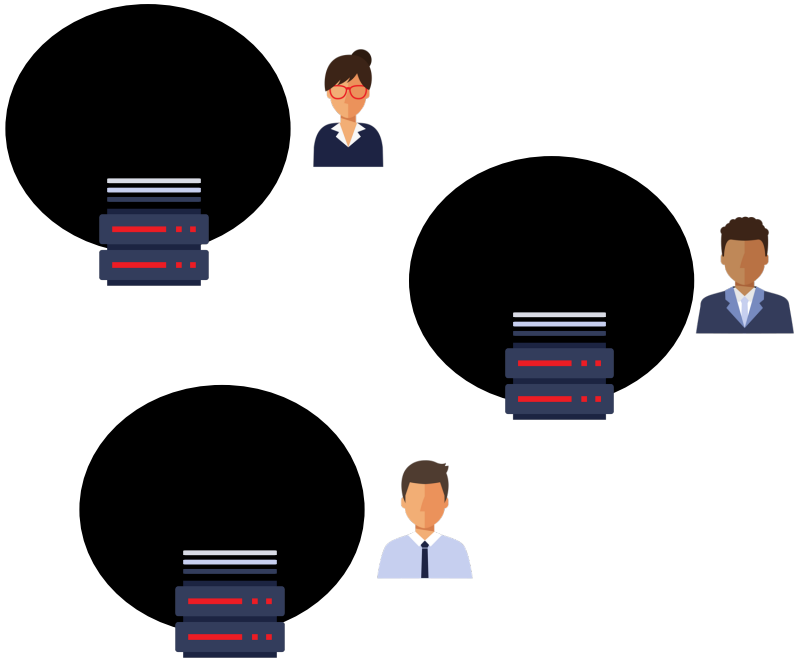
Next-gen Corda (Corda 5)

- **冗長構成のステートレス・ワーカが基盤に**
- **Flow、Transactionが単一ワーカの障害に縛られなくなる。**



3.2. 運用コスト削減

Corda 4: ノードごとのJVM



- １ノード１JVMプロセスが立ち上がっている
- （処理の有無と無関係に）常にプロセスが立ち上がっている必要がある

Next Gen Corda : マルチテナント

- クラスタ概念の導入
- クラスタ毎にひとつ又は複数の（仮想）ノード
- 仮想ノードは、アプリケーションネットワークに属するIdentity
- アプリケーションネットワークのメンバーシップは、MGM（メンバーシップグループマネージャ、それ自体も仮想ノード）で管理
- Cordaクラスタはネットワークで制限されない。別ネットワークに属する仮想ノードにも対応

3.2. Open Core戦略の採用

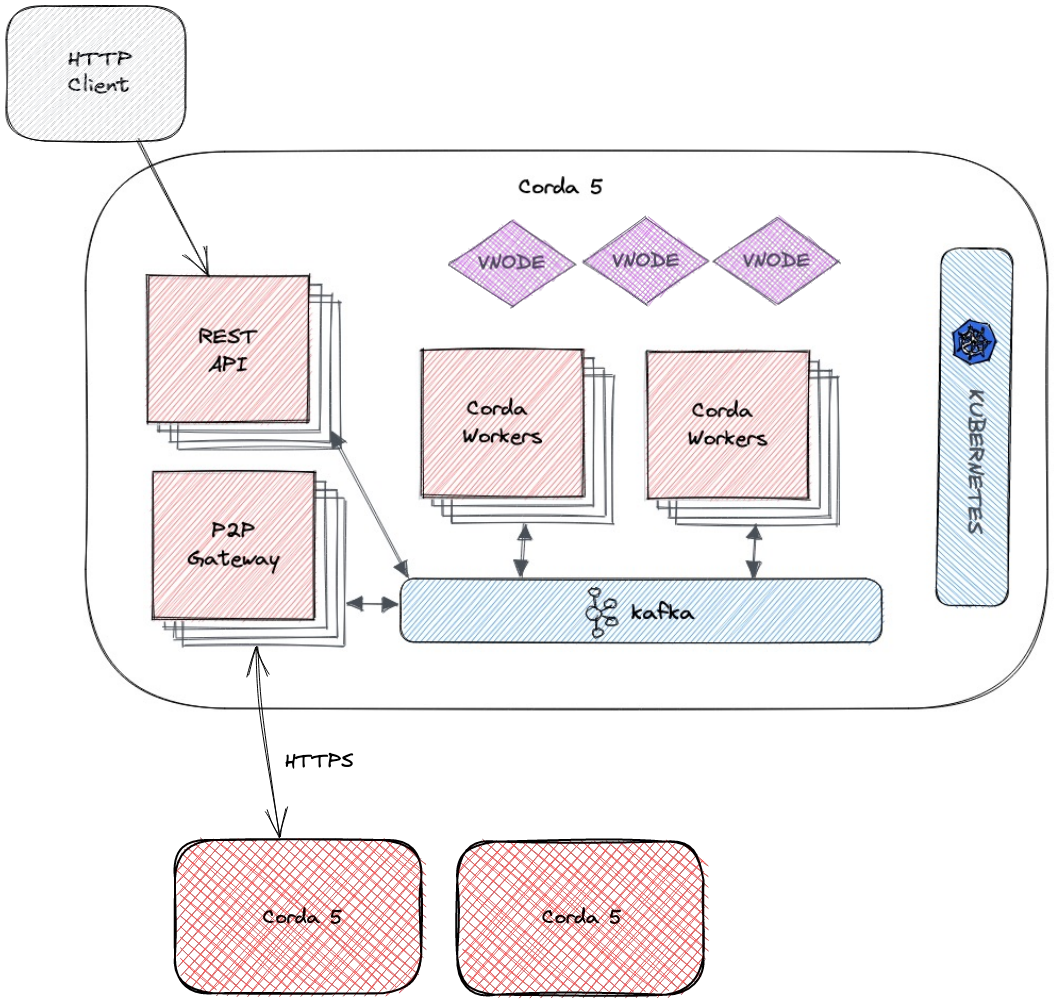
- 「コア」コードは二つのエディション間で共有

特徴・サービス	Community Edition	Engerprise Editon
ライセンス	Apache2ライセンス※	Enterpriseソフトウェアライセンス
セキュリティアップデート	おおよそ3〜6ヶ月に1回	パッチリリースで可能な限り早く
バグフィックス	おおよそ3〜6ヶ月に1回	パッチリリースで可能な限り早く
機能アップデート	おおよそ3ヶ月に1回	おおよそ3ヶ月に1回
サポート	24x5. Sev. 2に限定	24x7.Sev. 1を含む。迅速な対応
プロフェッショナルサービス	なし	利用可能
追加実装	なし	各種RDB対応/各種セキュリティ実装台帳アーカイブ機能他

※Apache2ライセンスに関して
第三者が更新・コピー・配布可能／“Corda”はR3社が商標登録済み

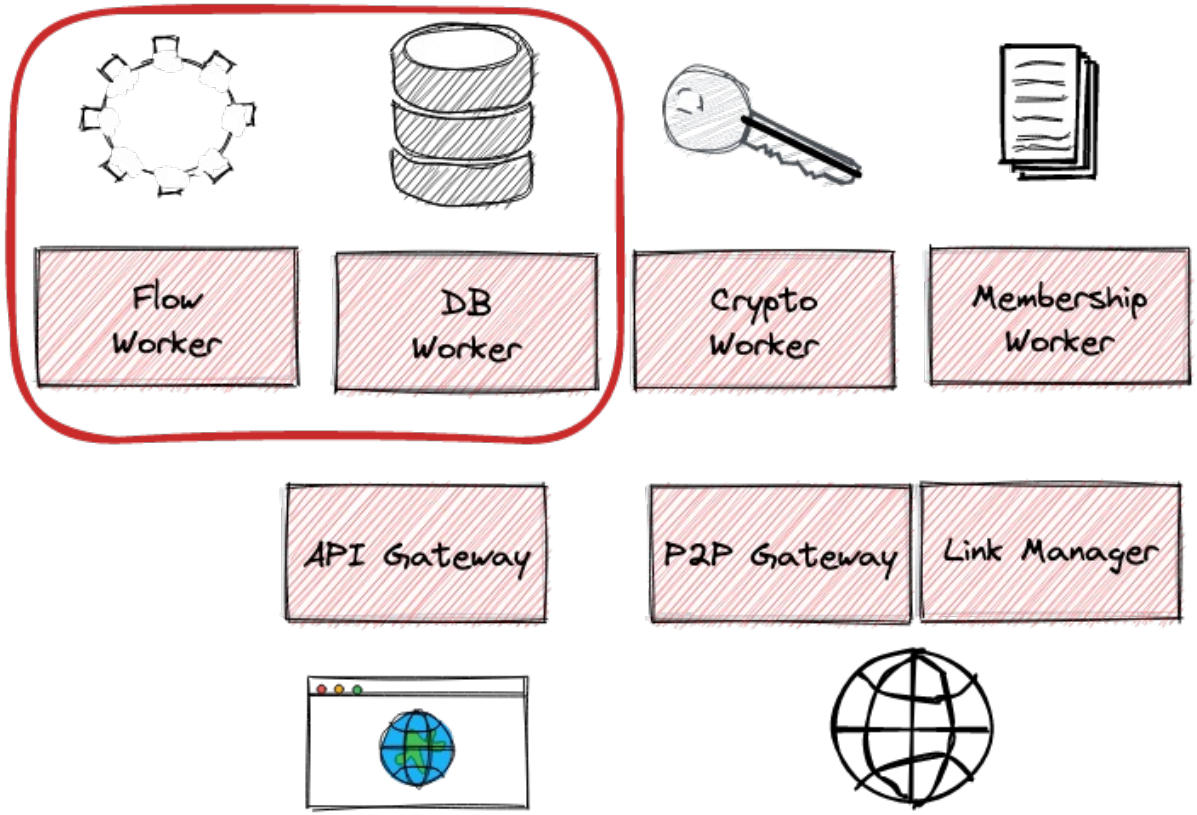
4. Next-Gen Corda：
ワーカー、仮想ノード

4. Next-Gen Corda：クラスタ



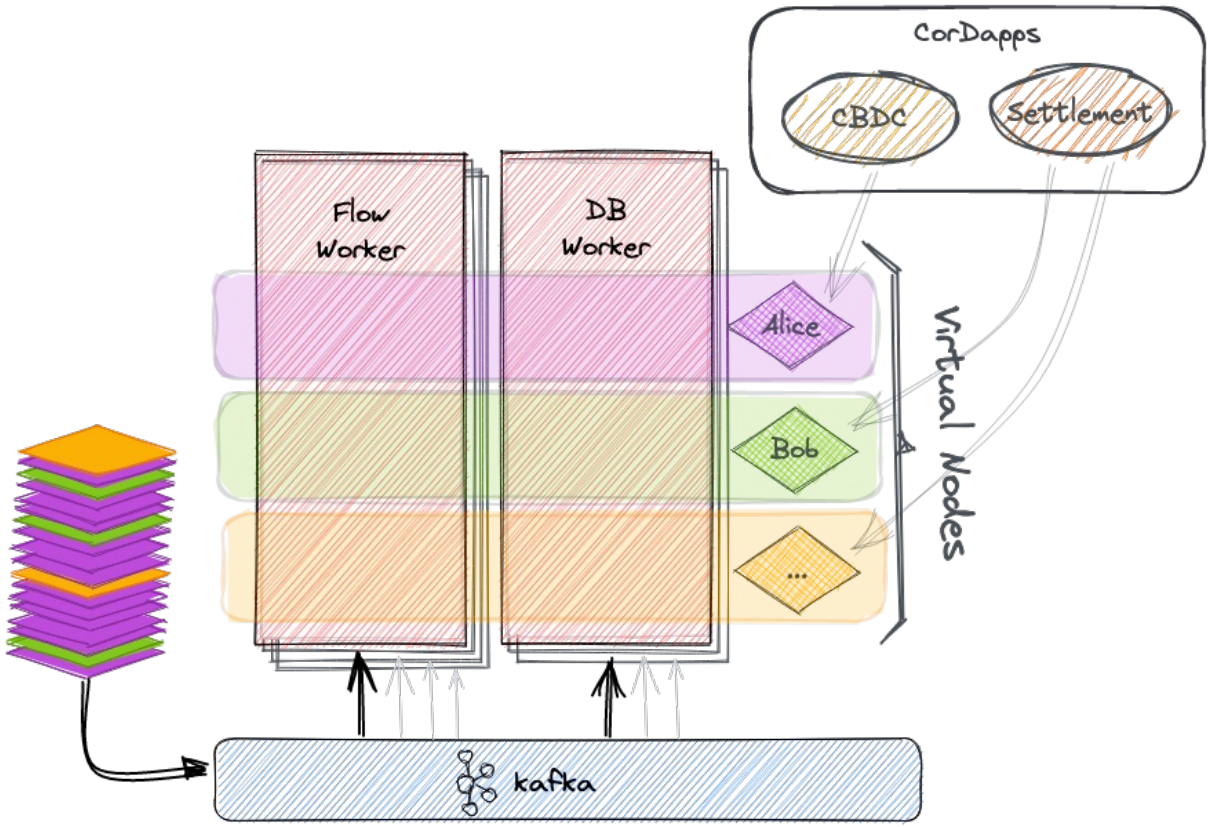
- 分散アーキテクチャをKafkaメッセージバスを中心に構成
- ステートレスなCordaワーカークラスタをホット/ホットで構成
- Cordaノードは仮想化されており、物理的なJVMやプロセスに縛られない
- P2P通信はHTTPS
- クラスタとのやりとり（Flowの開始、ノードの作成、等々）はREST API

4. Next-Gen Corda：ワーカー



- ワーカーで関心の分離を実現
  - 例：DBワーカーだけがデータベースの読み書きができる
  - 例：Cryptoワーカーだけが秘密鍵にアクセスして取引に署名できる
- 開発・テスト用に全ワーカーを１つのJVMプロセスにまとめることも可能
- Next-Gen CordaのCordappは引き続きJARファイルなので、ホストプロセスのJAVAランタイムで実行

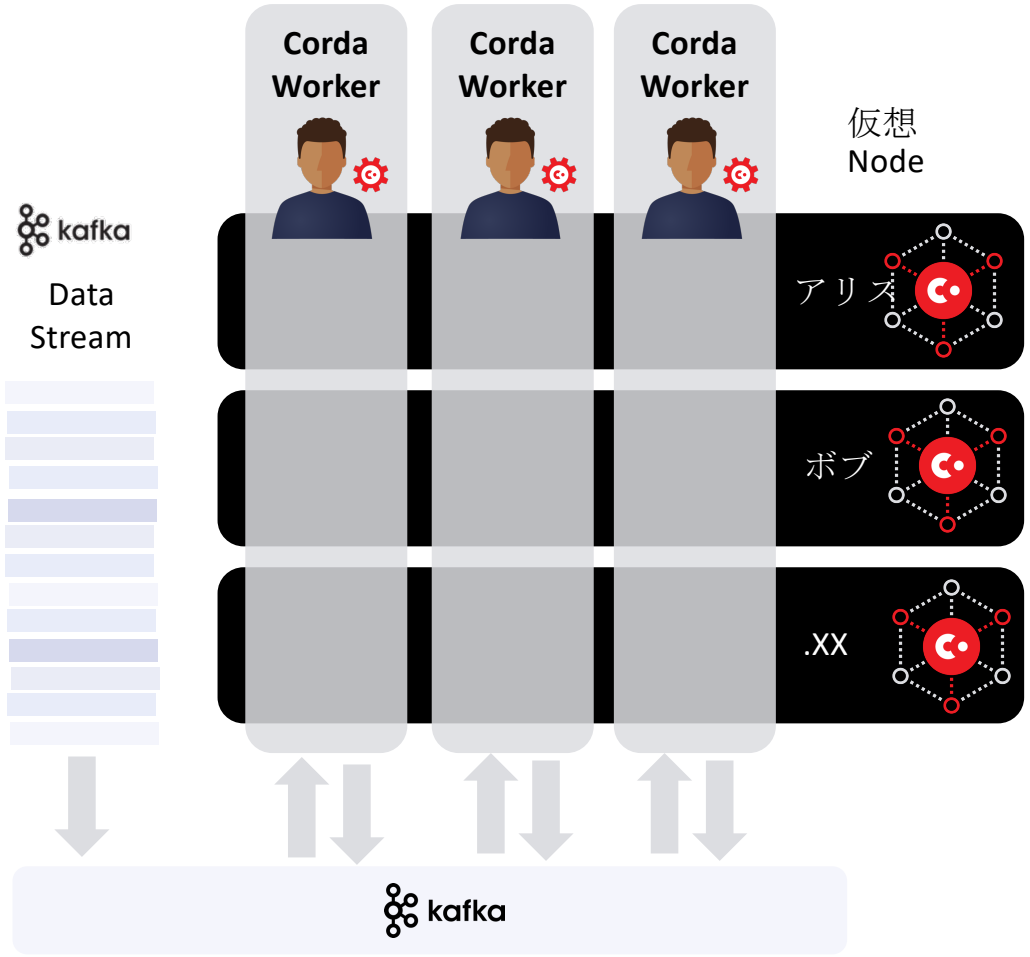
4. Next-Gen Corda ; CorDapps、仮想ノード、ワーカー



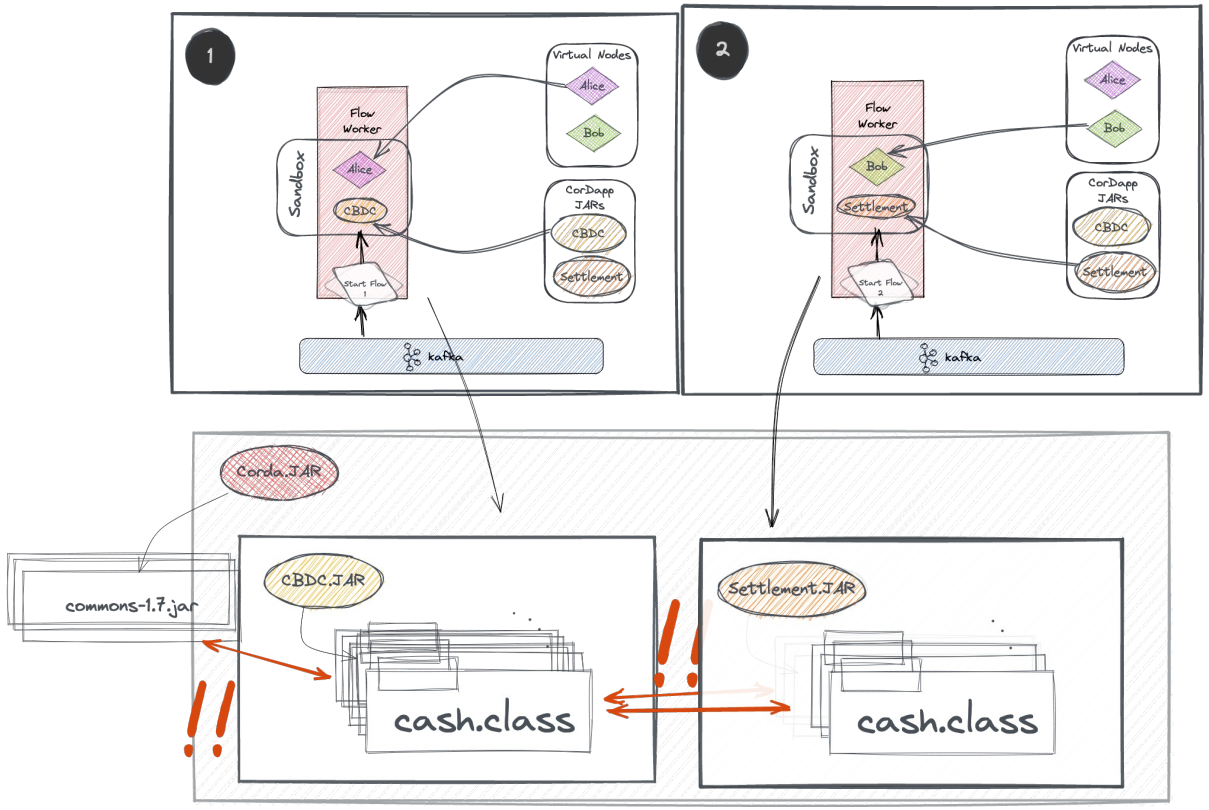
- 仮想化されたノード（仮想ノード）
- 仮想ノードは、単一のJVMプロセスではなく、実行コンテクスト
- 仮想ノードとは、アプリケーションとしてのIdentity、つまりCorDappのこと
- ワーカーは、メッセージバスに置かれたタスクを、仮想ノードを区別せずピックアップ
- 複数のワーカーインスタンスを実行し並行処理
- クラスタは、特定の仮想ノードではなく、全仮想ノード全体の負荷を分散

4. Next-Gen Corda：仮想ノード（とゼロリソース化）

- Corda 5では、IDは（論理的な）仮想ノードと関連付けられている
- Corda Workerは**任意の**仮想ノードの処理を実行できる
- WorkerとCorDapps、CorDapps同士を分離するためにサンドボックス技術が利用される。
  (Java Security Manager/OSGi bundle hooks)
- 「ID」と「計算資源」の関係を断ち切ることで、リソースをID間で共有することが可能になる。



4. Next-Gen Corda：サンドボックス

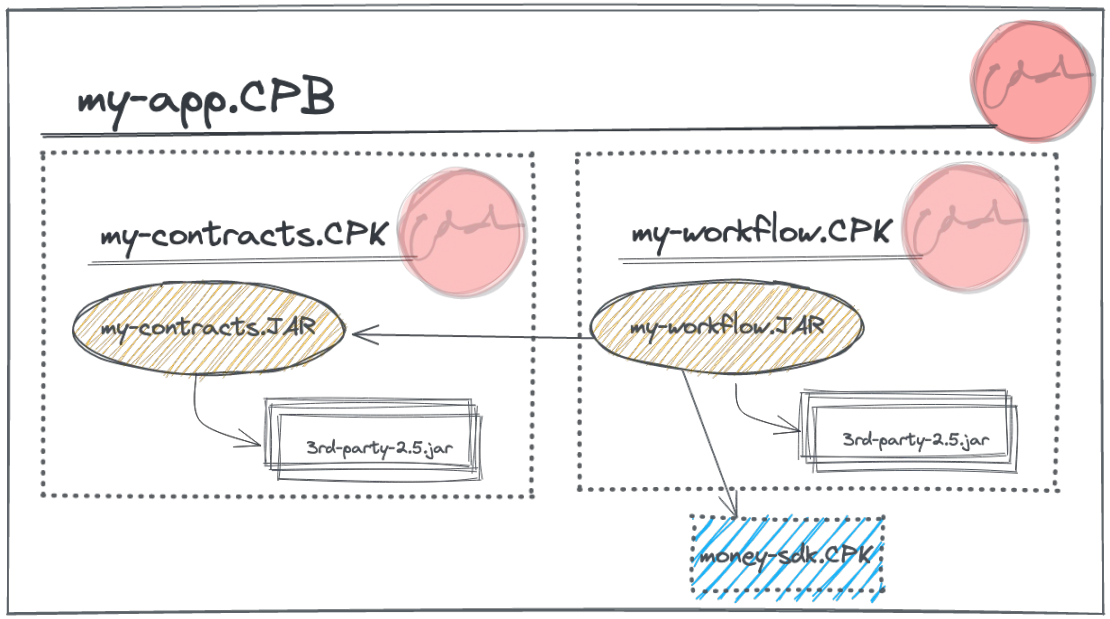


- Next-Gen Cordaでも、CordaプラットフォームとCorDappはJARで、ホストプロセスのJAVAランタイムで実行
- Corda 4では、以下の競合が発生し得た
  - CordaホストとCorDappが同じ外部ライブラリを同時実行
  - 複数のCorDappが同じ外部ライブラリを同時実行
  - 複数のCorDappが同じクラスを同時実行
- Next-Gen Cordaでは、サンドボックス導入により、上記の競合は発生しない

5. Next-Gen Corda：CorDappsのビルド、配布、インストール方法

(Corda 5 パッケージング)

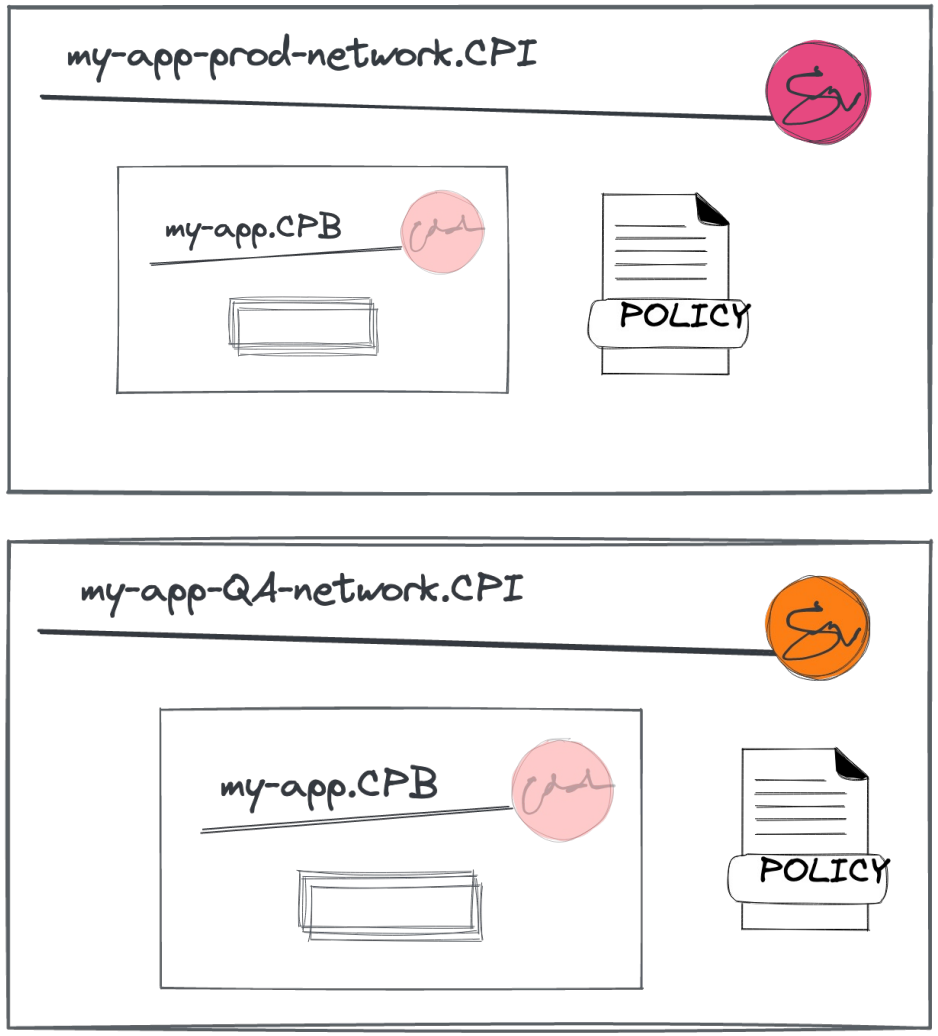
5. Next-Gen Corda： CorDappビルド



- Next-Gen Cordaでも、CorDappは、基本、コントラクトとフローのJARファイルで構成
- コントラクトJARファイルとフローのJARファイルを別々に、依存ファイルと共にバンドル化して、CPKファイルを作成（Cordaパッケージ）
- コントラクトのCPKファイル、フローのCPKファイル、外部ライブラリのCPKファイル等、必要なファイルをすべて結合して、CPBファイルを作成(Cordaパッケージバンドル)

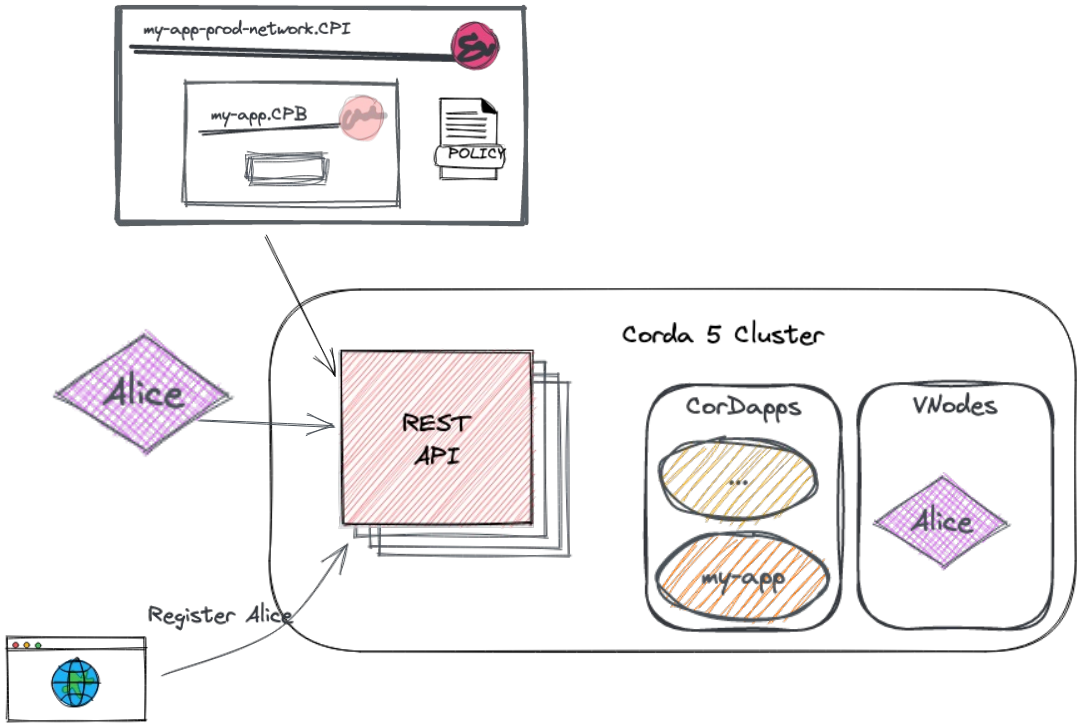
  -> 依存性チェックは、実行時ではなく、ビルド時に実施！

5. Next-Gen Corda： CorDapp配布



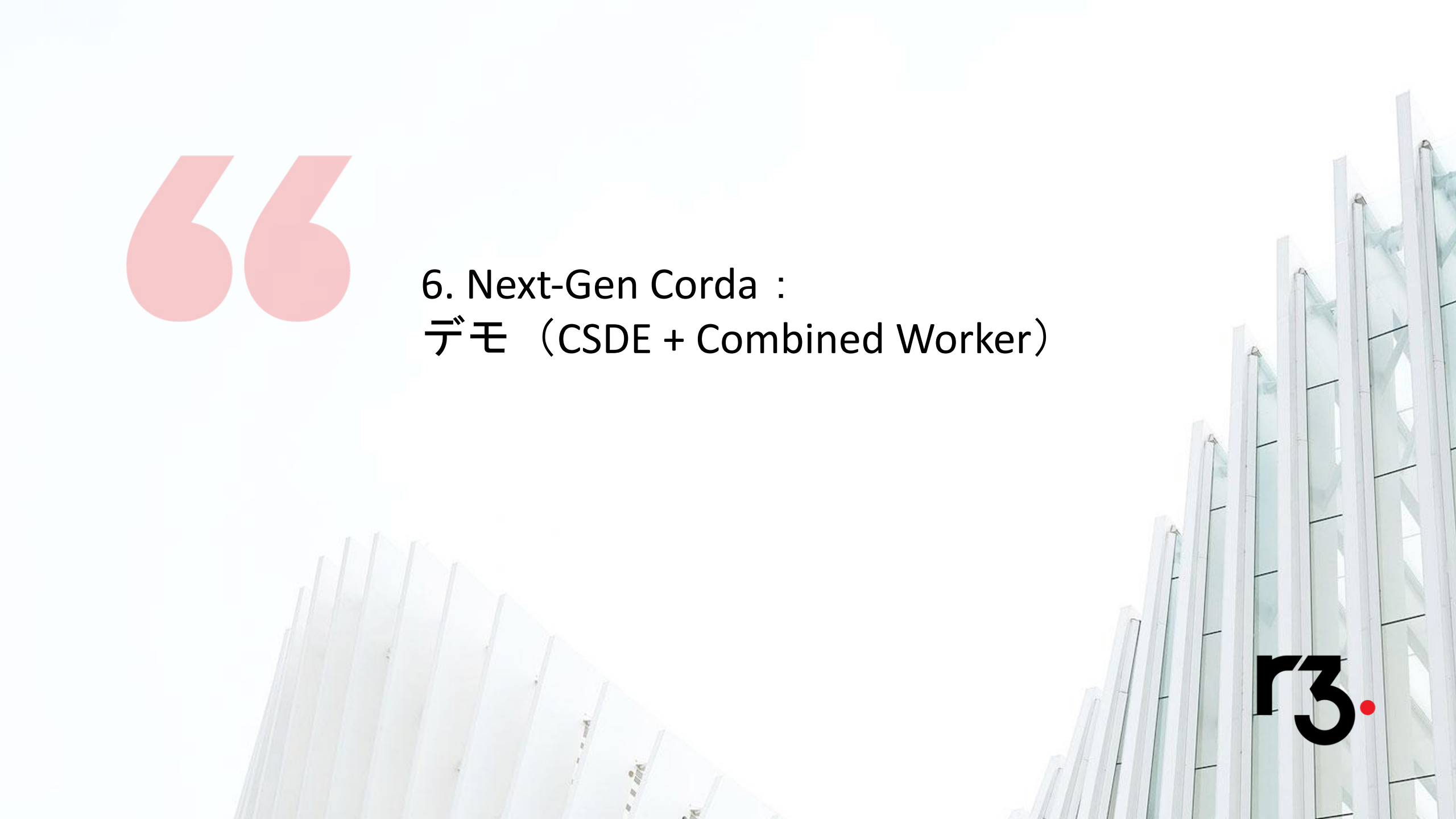
- CPBファイルを、直接、クラスタにインストールすることは出来ない！
- ネットワーク運用者が、CPBを受領し、グループポリシーファイル（特定のネットワークを参照）と紐付け、ネットワークポリシーを定義（MGMを参照）
- CPBファイルとグループポリシーファイルを結合して、CPI (Cordaパッケージインストーラー)を作成、署名
- 同じCPBファイルを、別のグループポリシーファイルに紐づけて、別ネットワーク用のCPIを作成

5. Next-Gen Corda：CorDappインストール



- ネットワーク運用者が、クラスタ運用者に、CPIを配布
- クラスタ運用者が、REST APIを使って、CPIをCordaクラスタにインストール（CPIを利用可能な状態にする）
- クラスタ運用者が、利用可能なCPIを使って、仮想ノードを作成
- 仮想ノードがMGMに登録されると（仮想ノードがMGMに参加依頼を送り、許可されると登録完了）、仮想ノードがネットワーク内で取引が出来る

6. Next-Gen Corda：
デモ（CSDE + Combined Worker）

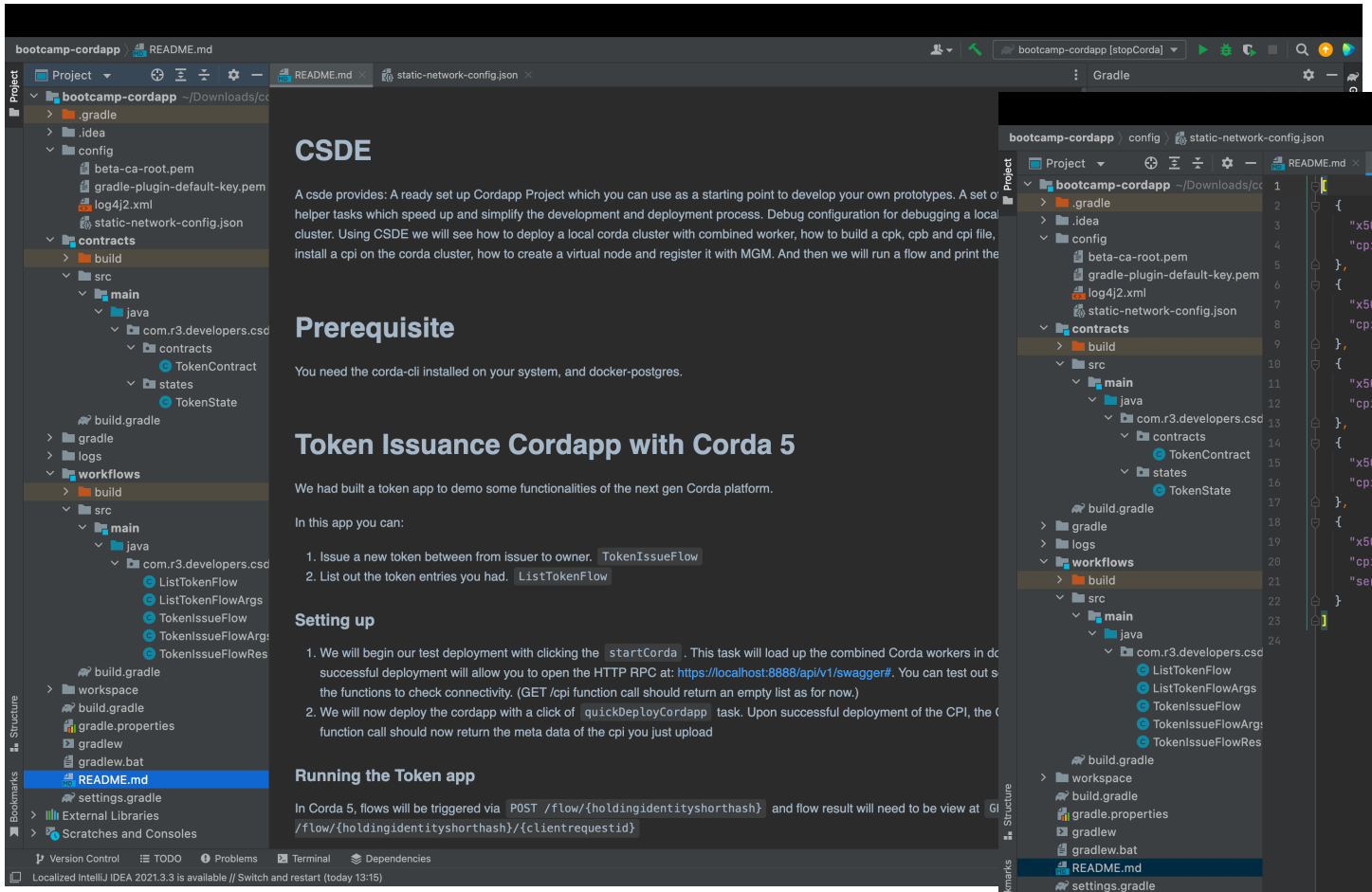


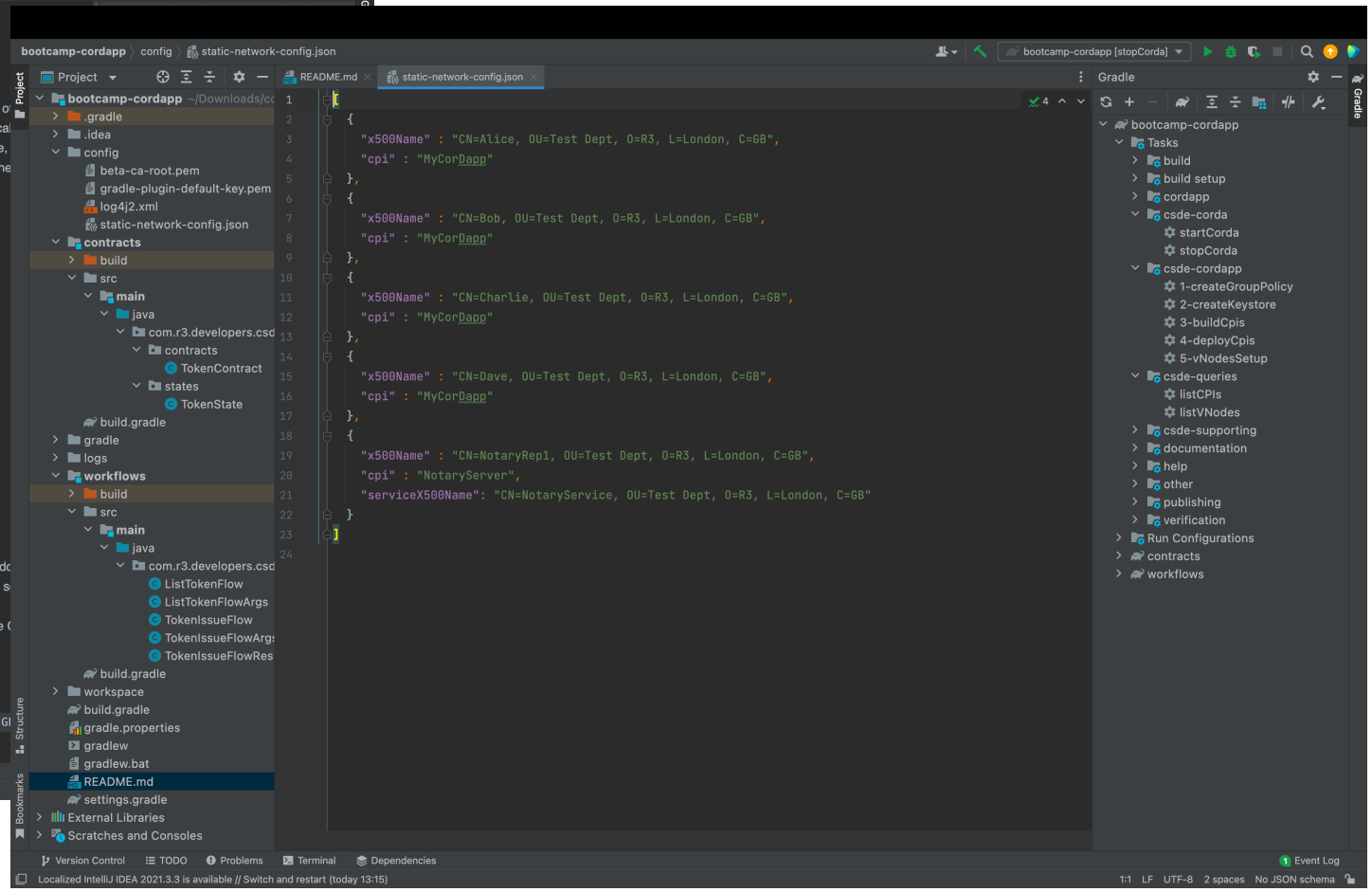
6. デモ：CSDEとサンプルCordappの紹介

- 本日は、CSDEを利用して、Combined Workerを使った静的ネットワークで、デモを実施させて頂きます。
（Kubernetesを利用したローカルクラスタへのデプロイについては、巻末に参考情報を記載しました。）
- CSDE （Corda Standard Development Environment）は、開発とテストに利用する環境です。
- CSDEでは、主要タスクがGradleタスクとして利用できる為、迅速なプロトタイプ開発とデプロイが可能です。
- CSDEの動作には、Corda CLIのインストールとDocker Postgresが必要です。

- 本デモでは、トークン発行のサンプルCordappプロジェクトを利用します。
https://github.com/corda/corda5-samples/tree/bootcamp-cordapp/java-samples/bootcamp-cordapp
- トークン発行Cordappの登場人物は、Alice、Bob、Charlie、Dave、Notaryです。
- ネットワーク参加者は、クラスタ環境ではGroupPolicyで定義するが、CSDEではstatic-network-config.jsonで定義。
- TokenIssueFlowで、Aliceがトークンを発行し、Bobに移転します。
- 上記の取引の結果として、Issuer（発行者）がAlice、Owner（所有者）がBobとして、Stateが作成されます。
- ListTokenFlowで、Alice、Bob、Charlieそれぞれの、トークンの保有状況を確認します。
- Corda 4と同様、アプリケーションのソースコードは、contractとworkflowフォルダに収録されています。
contractフォルダにはcontractとstates、workflowフォルダにはflowのコードファイルが格納されています。

6. デモ：CSDEとサンプルCordappの紹介



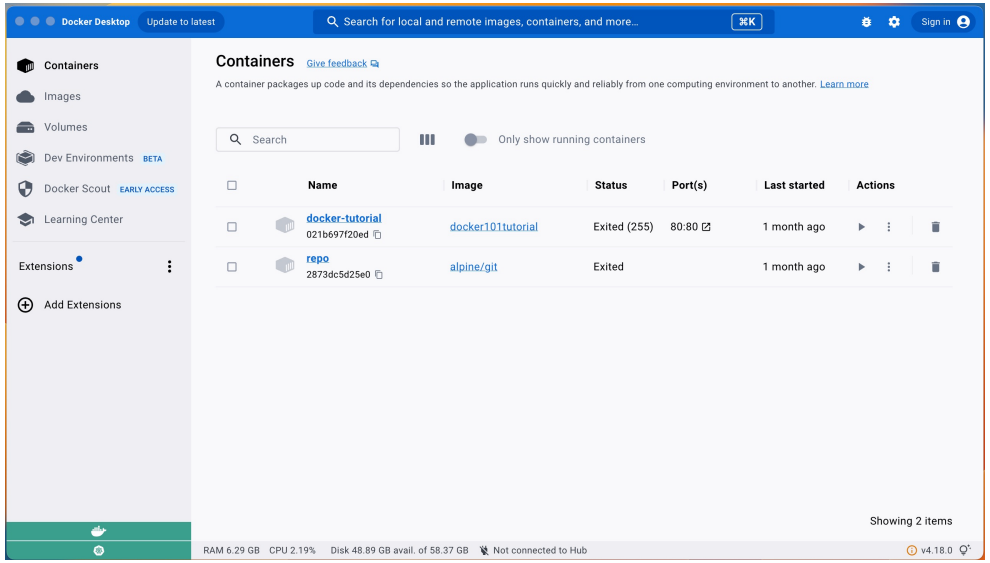


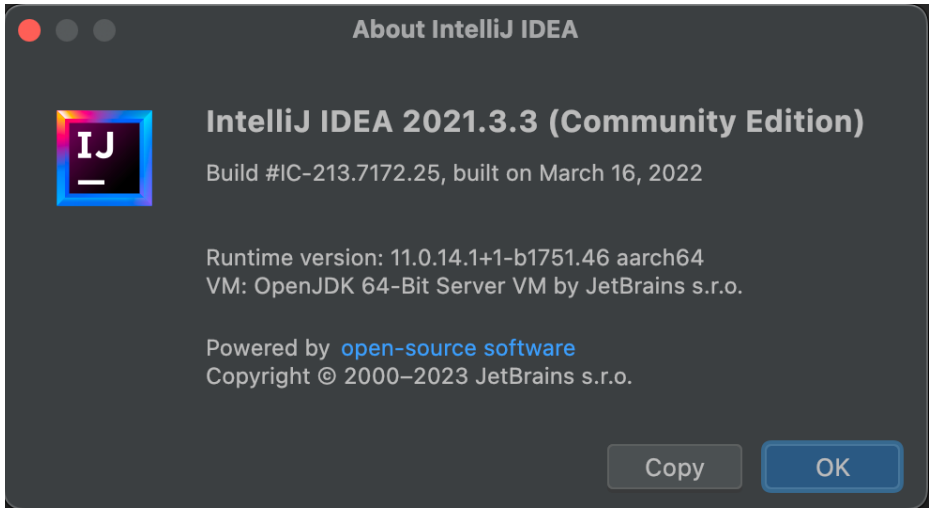
6. デモ：事前準備

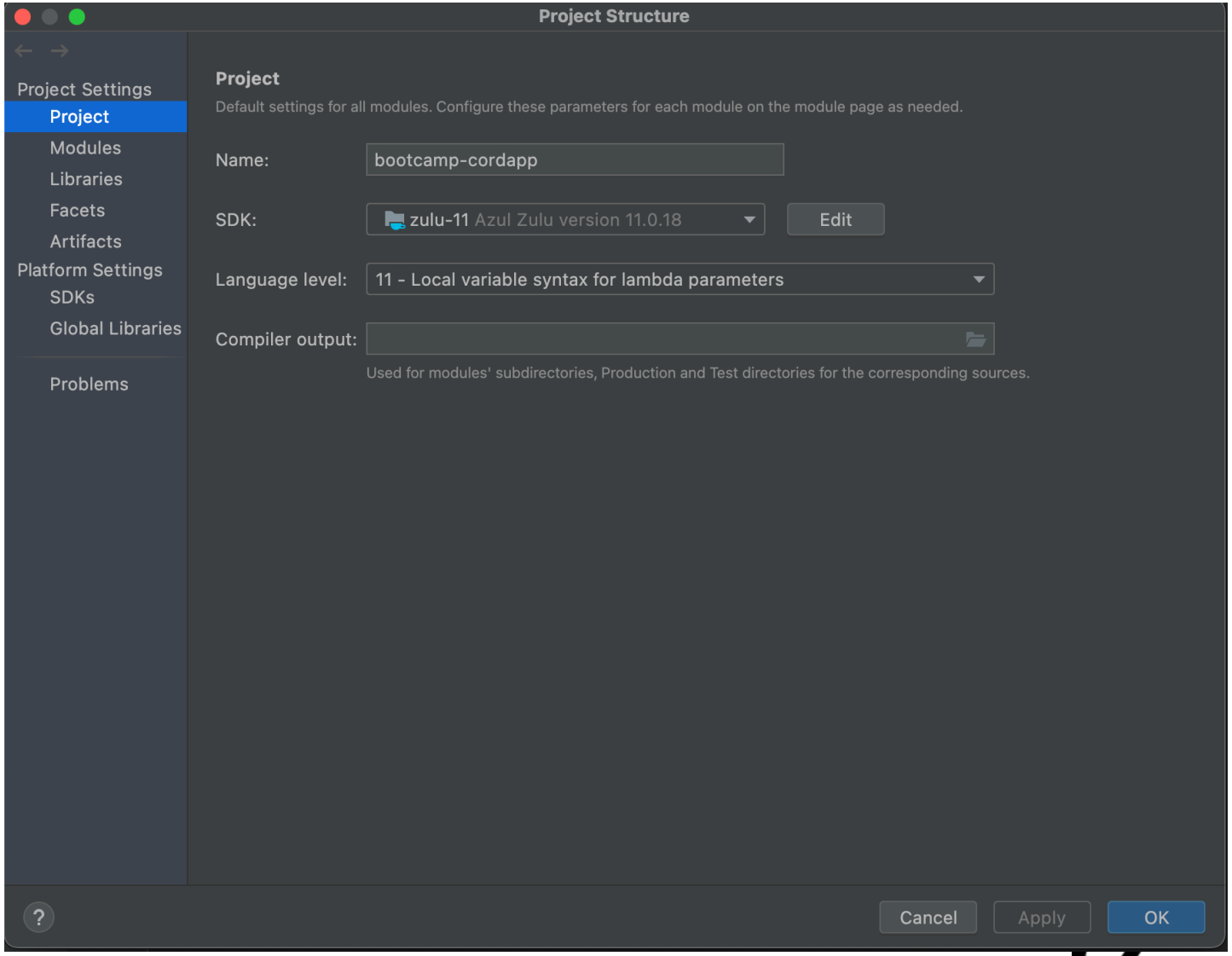
- Docker desktopを、あらかじめ起動しておきます。
- IDEはIntelliJのバージョン「~v2021.X.Y community edition」の利用が推奨されています。
- Javaは、Azul JDK 11の利用が前提となっているため、IntelliJの設定（FileメニューのProject Structure -> Project SettingのProject）で、SDKに「Azul JDK 11」が設定されていることを確認してください。

- インストールと事前準備が完了している想定で、以下のステップでデモを実施します。
  ステップ１. ローカルクラスタにCombined Workerをデプロイ
  （Combined Workerは、開発・テスト用に、複数のWorkerを結合したもの）
  ステップ２. CPK/CPB/CPIファイルをビルド、クラスタにCPIをインストール、仮想ノードを作成しMGMに登録
  ステップ３. フローを実行

6. デモ：CSDEとサンプルCordappの紹介







6. デモ：ステップ１（ローカルクラスタにCombined Workerをデプロイ）

- ステップ１では、ローカルクラスタにCombined Workerをデプロイします。
- 右側のGradeタスクバーで、startCordaというGradleタスクをダブルクリックします。
- Combined WorkerとPostgresドライバがHOMEディレクトにダウンロードされ、ローカルJVMでCombined Workerプロセスがポート8888で実行されます。
- Combined Workerは、デフォルトで、Kafkaメッセージバスではなく、Postgresを、モックのメッセージバスとして利用します。通常の開発・テストでは、Kafkaは不要です。

6. デモ：ステップ２（ビルド、デプロイ、仮想ノード作成・登録）

- ステップ２では、CPK/CPB/CPIファイルをビルド、クラスタにCPIをインストール、仮想ノードを作成しMGMに登録します。
- 右側のGradeタスクバーで、5-vNodesSetupというGradleタスクをダブルクリックすると、グループポリシーとキーストアの作成、CPIのビルドとデプロイ、仮想ノードの作成・登録が完了します（ひとつひとつ実行頂いても構いません）。

[グループポリシーの作成（1-createGroupPolicy）、キーストアの作成（2-createKeystore）、CPIのビルド（3-buildCpis）]

- CPIファイルは、workflowディレクトリに作成されます。
- GroupPolicy.jsonファイルには、特定のネットワークに関する情報が収録されています。
- このネットワークの参加者（Alice、Bob、Charlie）は、configフォルダのdev-net.jsonファイルで確認できます。
- CPIファイルの作成は、内部的には、Corda CLIツールが使われています。

[CPIのデプロイ（4-deployCPIs）]

- 内部的には、HTTP REST APIのひとつ、/cpi GET APIをコールして、CPIをネットワークにアップロードしています。
- 右側のGradeタスクバーで、listCPIsというGradleタスクをダブルクリックして、ネットワークで利用可能なCPIを調べることが出来ます。

[仮想ノードの作成・登録（5-vNodesSetup）]

- 内部的には、HTTP REST APIをコールして、仮想ノードを作成し、MGMに登録しています。
- 仮想ノードからMGMへのネットワーク参加依頼が承認されると、仮想ノードは、ネットワークでの取引が可能になります。
- クラスタが立ち上がり、CPIがクラスタにアップロードされ、各Identity（Alice/Bob/Charlie）に仮想ノードが作成されます。

6. デモ：ステップ３（1/2、トークンの作成と移転）

- ステップ３では、フローを実行し、トークンを作成し、移転します。

- まず、HTTP RPC APIを使って、AliceからBobに対し、トークン発行Flowを実行します。
- Swagger API画面で、HTTP POST /flow/{holdingidentifyshorthash}を使います。
- 右側のGradeタスクバーで、listVNodesタスクをクリックして、Aliceの仮想IDを取得して、holdingidentifyshorthashに入力します。
- Request BodyはTokenIssueFlowソースコード下部のサンプルをコピペ＆金額だけ変更して、Executeボタンをクリックして、実行します。

- 次に、フローの状況を確認します。
- Swagger API画面の、HTTP GET /flow/{holdingidentifyshorthash}/{clientrequestId}で、holdingidentifyshorthashにAliceの仮想ID、clientrequestidにRequest Bodyに入力したclientrequestIdを入力して、Executeボタンをクリックすると、flowStatusが完了（COMPLETED）となっていることが確認出来ます。

6. デモ：ステップ３（2/2、トークンの状態確認）

- 最後に、各主体のデータベースに収録されたトークンの状態を調べます。

- まずは、Alice。
- Swagger API画面の、HTTP POST /flow/{holdingidentifyshorthash}で、holdingidentifyshorthashにAliceの仮想ID、Request BodyはListIssueFlowソースコード下部のサンプルをコピペして、Executeボタンをクリックして、トークン発行一覧を出力します。
- 次に、Swagger API画面の、HTTP GET /flow/{holdingidentifyshorthash}/{clientrequestId}で、holdingidentifyshorthashにAliceの仮想ID、clientrequestIdにRequest Bodyに入力したclientrequestIdを入力して、Executeボタンをクリックして、上記のトークン発行一覧を取得します。
- Aliceが発行者、Bobが所有者であることが確認できます。

- 同様の方法でBobを確認すると、Bobのデータベースにも、Aliceと同じ情報が収録されていることがわかります。

- 同様の方法でCharlieを確認すると、Charlieのデータベースでは、トークンの状態は空であることがわかります。
- つまり、同じクラスタに存在しても、Charlieは、AliceとBobの間の取引が発生していたことを知りません。

6. 参考情報（Kubernetes + minikubeでクラスタ環境を作る場合の注意点）

- コマンド「minikube start --cpus=6 --memory=8G 」でkubernetesクラスタを実行するので、それに合わせて、Dockerのリソース設定で、CPUsを６以上、メモリを８GB以上、割り当ててください。

- Windowsで、ワーカーの起動が異常に遅い場合（４０分から１時間）、*.wslconfigの設定で、メモリを16 GB以上を割り当ててみてください。また、プロセッサ数は、デフォルトでは８ですが、６にした方が起動が早くなるようです（デフォルトは、WindowsとWSLでプロセッサ数を共有して８、この場合、WSLに動的にメモリを割り当てる処理が発生、あらかじめ、.wslconfigでプロセッサ数を６と固定値にする事でメモリ割り当ての処理が無くなる分、早くなると想像）。
- プロセッサ数は、Windowsタスクマネージャーの CPU項目の論理プロセッサから、WSLだとlscpuコマンドで確認できます。

- R3ドキュメントの指定に従い、バージョン3.5以下のDockerを利用した場合、Windowsでは、ワーカーがなかなか立ち上がらないことが多いようです。WSL2とDocker ~v3.5.xの組み合わせでは、メモリ解放に問題があり、ワーカーに割り当てるメモリが利用可能でないことが原因と思われます。Dockerを再起動し、Dockerに占有されるメモリが少ない状態で、再度、ワーカーの立ち上げを実行してください。Dockerのバージョンアップで問題が解決するか検証中です。

  WSL2 + Docker causes severe memory leaks in vmmem process, consuming all my machine's physical memory #8725

  https://github.com/microsoft/WSL/issues/8725

7. Appendix： Corda 5.0 GA の制約事項

- Corda 4からCorda 5へのインラインアップグレードはありません。
  →移行が必要です。

- ハードウェアセキュリティモジュール（HSM）はサポート対象外です。
  →Corda 5.0 Enterprise GAでは、Hashicorp Vaultの利用を想定しています。
  →将来のバージョンでHSMが利用可能になる予定です。

- Corda 4 Tokens SDKはありません。
  →fungible states/tokenが利用可能です。

- Corda 4 Accounts SDKはありません。
  →マルチテナンシーは仮想ノードで対応します。

- Corda 4 Confidential Identityはありません。
  →匿名化された仮想ノードでの対応を想定しています。