

WebPerformer-NX

複数名で同一アプリケーションを開発、変更管理を行うための参考資料

目次

1. 本資料について
2. NX標準機能での実現可能なリソース
3. アプリケーションの複数名での開発、変更管理
 1. リポジトリ機能使用の流れ
 2. 開発体制のイメージ
 3. 役割毎、運用イメージ
 4. 役割毎、運用イメージ詳細
4. データベースの変更管理
5. ファイルの変更管理
6. バッチの複数名での開発、変更管理

1. 本資料について

本資料は「複数名で同一アプリケーションを開発、変更管理を行う手順」を解説しています。

◆ 本資料で扱う事例

- 開発規模
 - 作成する画面数：10
 - 作成するバッチ数：4
 - チーム構成：管理者1名、開発者4名
- 登場人物の役割
 - 管理者：開発環境／実行環境の定義情報、及びDBの管理を担当
 - 開発者：開発を担当

※本事例では「管理者」と「開発者」は別の人物が担当していますが、
複数名で開発を行う際の構成として、「管理者」が「開発者」を兼務することも可能です。

◆ 前提

- WebPerformer-NX バージョン 3.0.0時点での情報です。

2. NX標準機能での実現可能なリソース

複数名で同一アプリケーションを開発、変更管理を行うにあたり、NX で実現可能なリソースは下記の通り

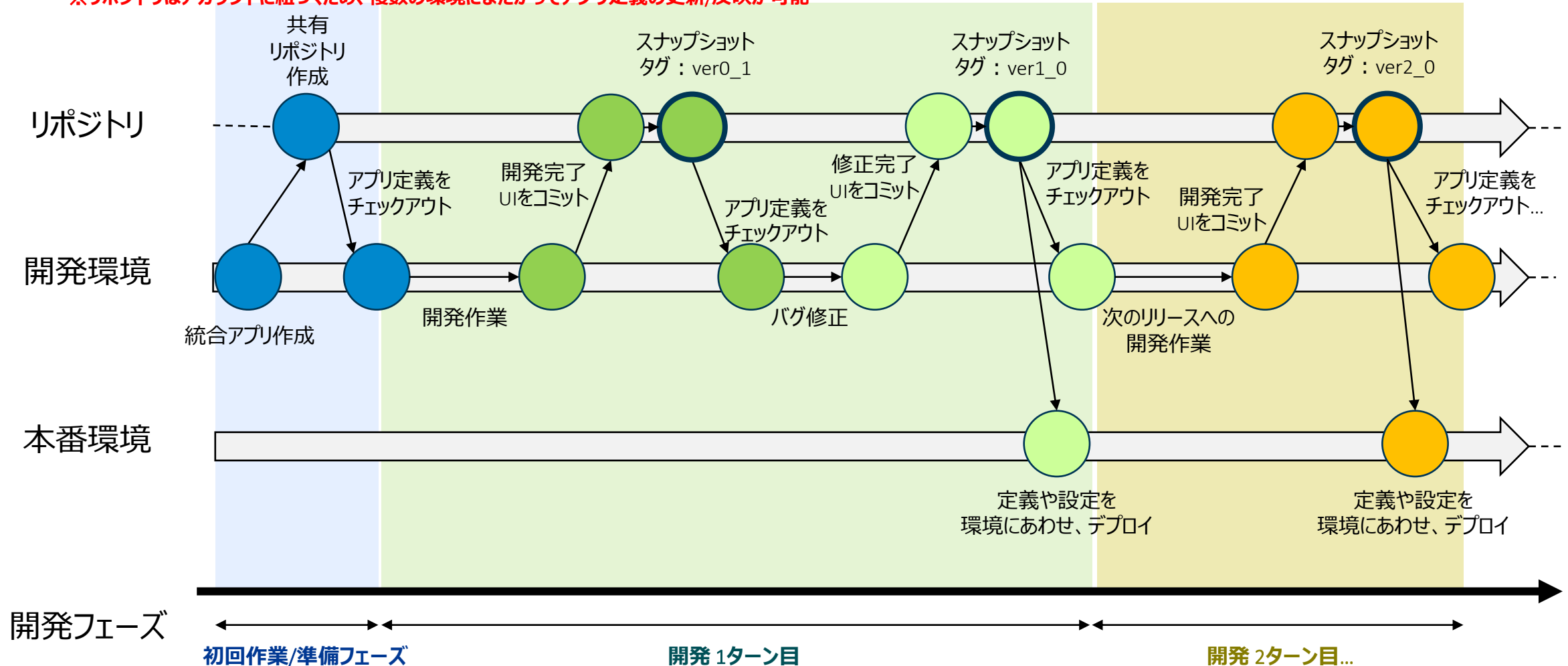
リソースの種類	リソース	利用可否	実現方法
アプリケーション	UI	○	リポジトリ機能を使用する ※P.4～17 をご確認ください
	ワークフロー	○	
	REST API	○	
	グローバル関数	○	
	定数	○	
	アプリケーション設定	○	
データベース	テーブル定義	×	外部サービスを利用して管理する ※P.18～20 をご確認ください
	マスタデータ	×	
ファイル	ファイル	×	外部サービスを利用して管理する ※P.21、22 をご確認ください
バッチ	ジョブ定義	×	外部サービスを利用して管理する ※P.23～25 をご確認ください

3-1. 複数名開発：リポジトリ機能使用の流れ

リポジトリは複数名でのアプリ開発を進めるために、アプリケーションを共有する機能

リポジトリと開発用アプリケーション定義/本番環境用のアプリケーション定義の関係性とフローのイメージは下記の通り

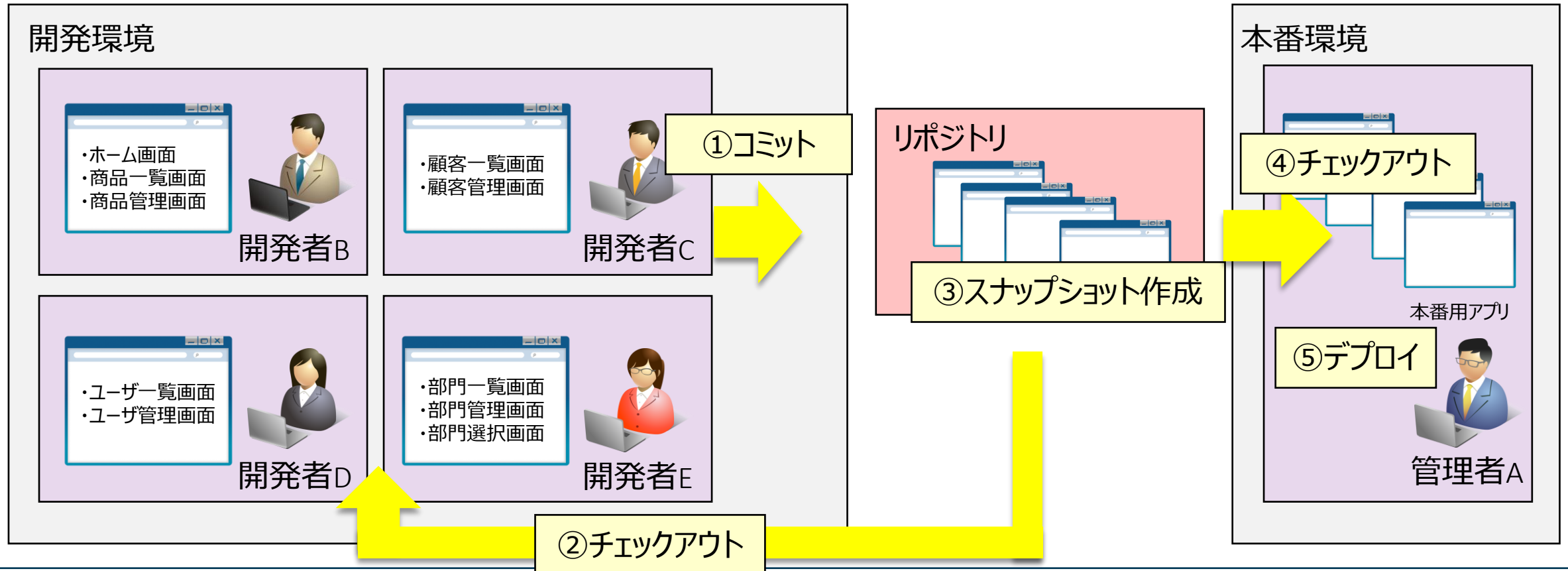
※リポジトリはアカウントに紐づくため、複数の環境にまたがってアプリ定義の更新/反映が可能



3-2. 複数名開発：開発体制のイメージ

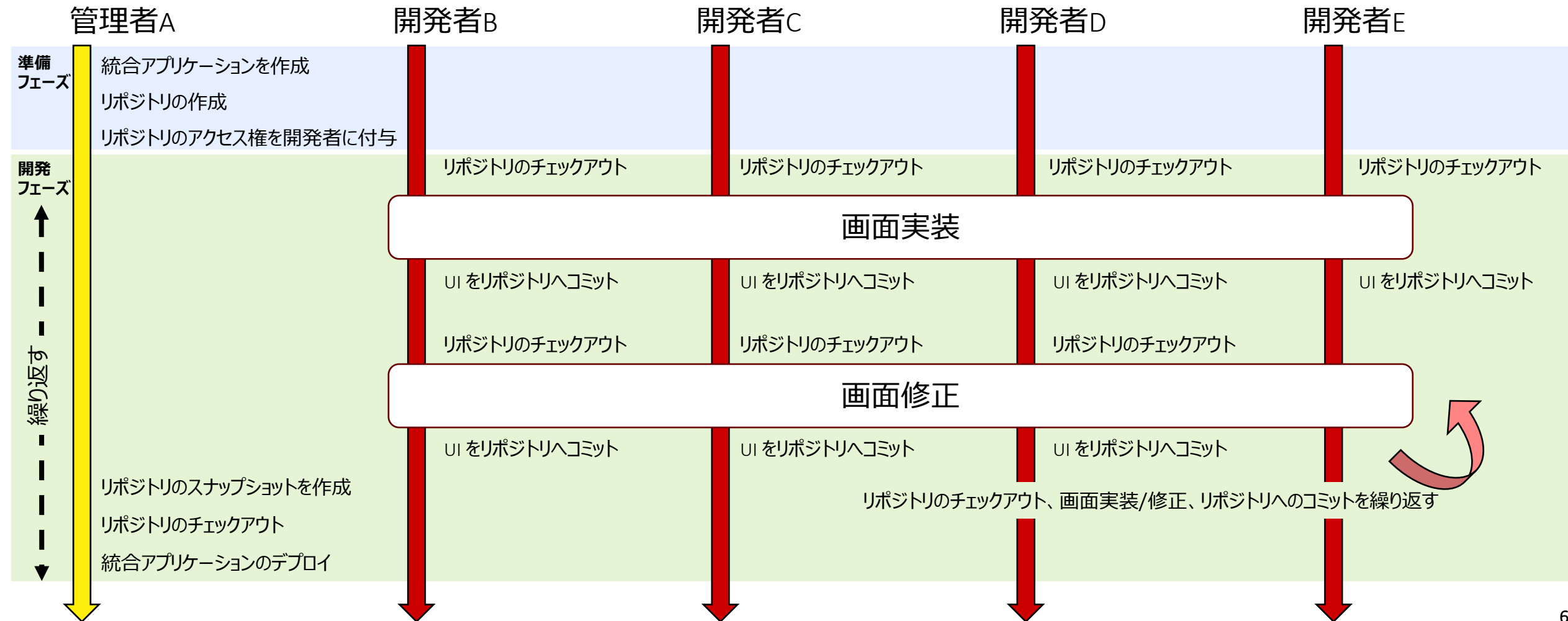
アプリケーション開発では、開発者は画面ごとに実装単位を分けて開発することを推奨
それぞれの担当画面を完成させ、リポジトリにコミット、最後に、本番アプリとして統合しデプロイする
リポジトリ機能を使用することで、過去の変更箇所を確認することや、特定時点の内容に戻すことが可能

WebPerformer - NX



3-3. 複数名開発：役割毎、運用イメージ

リポジトリ機能を利用した開発における、定義情報やそのデプロイに関する運用イメージは下記の通り
※前提として、アカウント登録は完了している状態



3-4. 複数名開発：役割毎、運用イメージ詳細

管理者A：準備フェーズ

1. 統合アプリケーションを作成する

統合アプリケーションを作成

リポジトリの作成

リポジトリのアクセス権を開発者に付与

①アプリケーションの「作成」ボタンを押下し「新規」を選択した後
必要事項を入力し「作成」ボタンを押下



アプリ作成

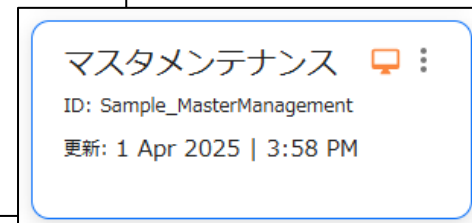
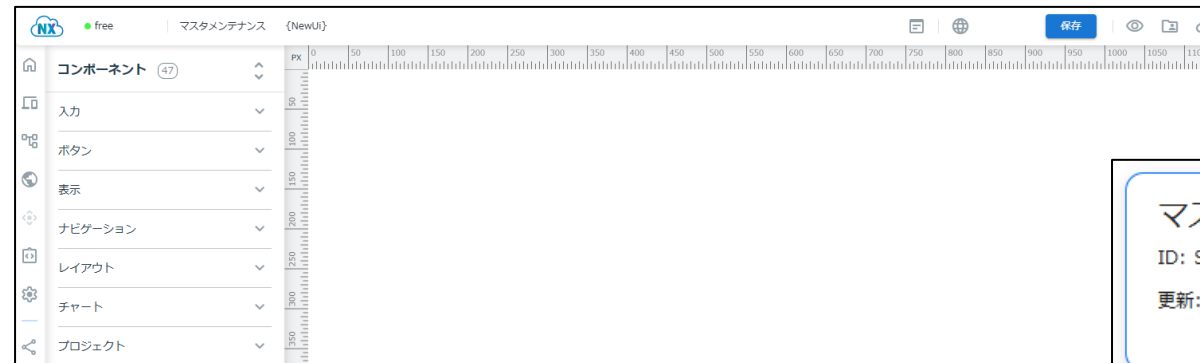
DESKTOP MOBILE REST API

ID
Sample_MasterManagement

ラベル
マスタメンテナンス

キャンセル 作成

②UIエディタが白紙の状態が開かれ、アプリケーション作成完了
アプリケーション一覧画面にも作成したアプリが追加されている



リポジトリのスナップショットを作成

リポジトリのチェックアウト

統合アプリケーションのデプロイ

3-4. 複数名開発：役割毎、運用イメージ詳細

管理者A：準備フェーズ

2. 共有用のリポジトリを作成する

統合アプリケーションを作成

リポジトリの作成

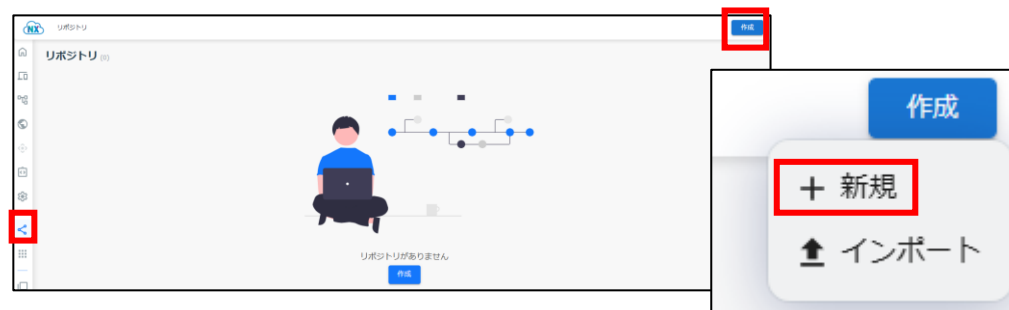
リポジトリのアクセス権を開発者に付与

リポジトリのスナップショットを作成

リポジトリのチェックアウト

統合アプリケーションのデプロイ

- ① サイドメニュー[リポジトリ]を選択して、リポジトリ一覧画面へ遷移後、リポジトリの「作成」ボタンを押下し「新規」を選択



- ② ラベルを入力し、先ほど作成した統合アプリケーションを選択し、「作成」ボタンを押下

リポジトリ追加

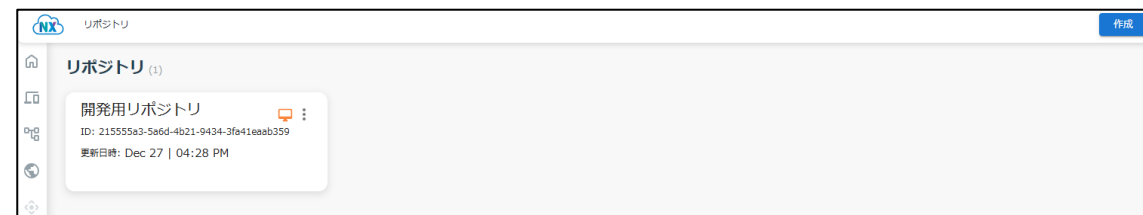
ラベル*
開発用リポジトリ

アプリ*
マスタメンテナンス

コメント

キャンセル 作成

作成したリポジトリが追加される



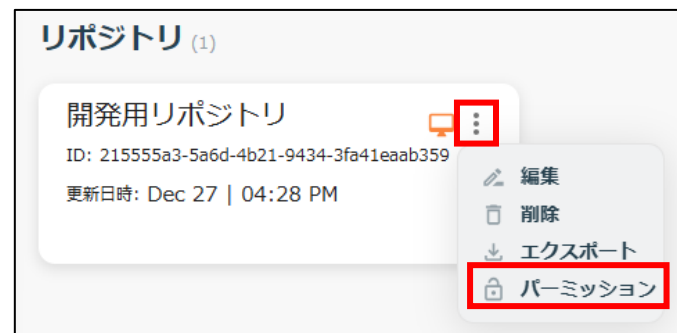
3-4. 複数名開発：役割毎、運用イメージ詳細

管理者A：準備フェーズ

3. リポジトリのアクセス権を開発者に付与する

統合アプリケーションを作成
リポジトリの作成
リポジトリのアクセス権を開発者に付与

①作成したリポジトリの縦三点リーダーを押下し、「パーミッション」ボタンを押下



②「追加」ボタンを押下し、開発者のニックネームと Eメールを入力して「作成」ボタンを押下



リポジトリのスナップショットを作成
リポジトリのチェックアウト
統合アプリケーションのデプロイ

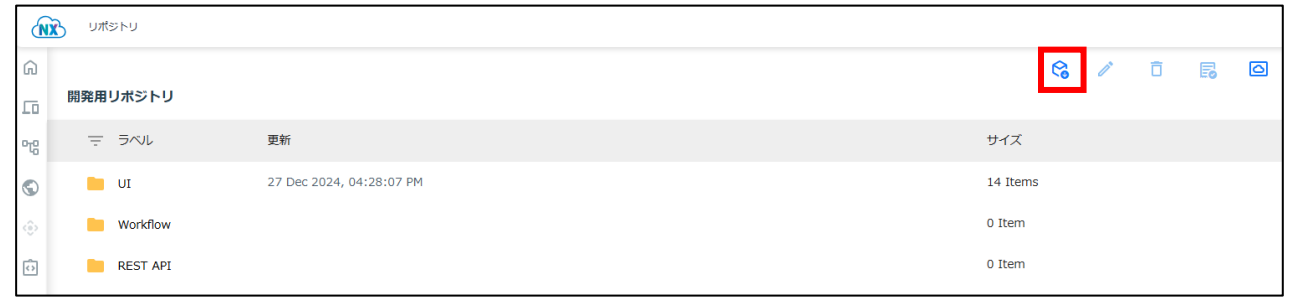
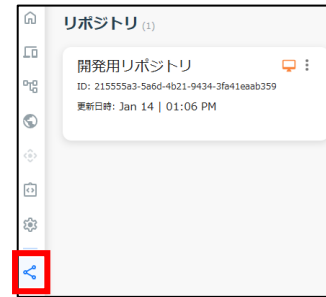
3-4. 複数名開発：役割毎、運用イメージ詳細

開発者B～E：開発フェーズ

4. リポジトリをチェックアウトし、それぞれの開発環境にアプリケーションを作成する

リポジトリのチェックアウト
画面実装
UIをリポジトリへコミット

①サイドメニュー[リポジトリ]を選択して、リポジトリ一覧画面へ遷移後
共有されたリポジトリを開き、チェックアウトアイコンを選択する



②バージョン[最新]を選択し、アプリケーション[新規]を選択して、
ID とラベルを入力し「チェックアウト」ボタンを押下

チェックアウト

バージョン
最新 (highlighted) スナップショット

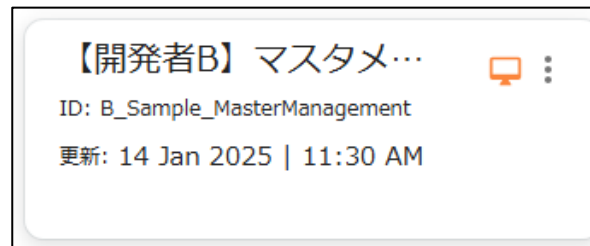
アプリケーション
上書き 新規 (highlighted)

ID *
B_Sample_MasterManagement

ラベル *
【開発者B】 マスタメンテナンス

キャンセル チェックアウト (highlighted)

アプリケーション一覧にチェックアウトしたアプリケーションが作成される
※2回目以降の開発では、アプリケーション[上書き]を選択することで、
最新版のアプリの再取得が可能



チェックアウト後に担当分の画面実装を行う

3-4. 複数名開発：役割毎、運用イメージ詳細

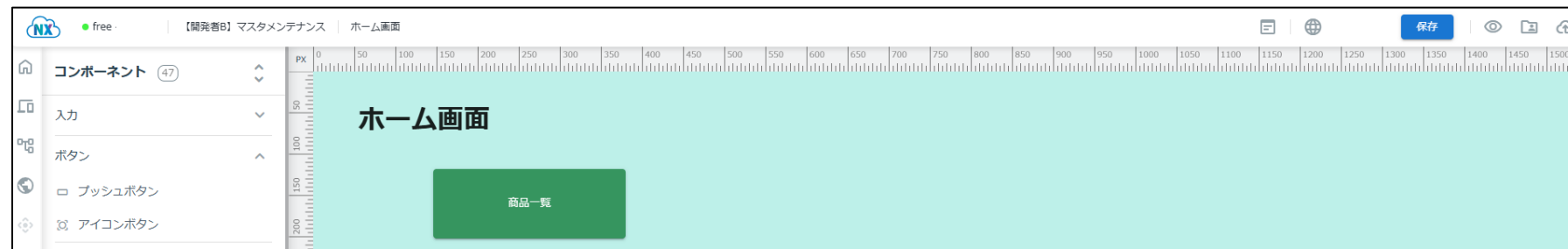
開発者B～E：開発フェーズ

5. UI をリポジトリへコミットする

リポジトリのチェックアウト
画面実装
UIをリポジトリへコミット

開発完了後に、コミットを行う

① UI を開いた状態で、リポジトリアイコンを選択する



②共有されたリポジトリを選択する

リポジトリへのコミット

リポジトリ *

コメント

キャンセル コミット

開いているアプリケーションの定義情報が表示される

リポジトリへのコミット

リポジトリ *

開発用リポジトリ

【開発者B】 マスタメンテナンス

ラベル

- UI
- Workflow
- REST API
- Global Functions
- Constants
- Application Settings

3-4. 複数名開発：役割毎、運用イメージ詳細

開発者B～E：開発フェーズ

5. UI をリポジトリへコミットする

リポジトリのチェックアウト
画面実装
UIをリポジトリへコミット

③UI をクリックし、実装した画面分の UI を選択した状態でコメントを入力し、「コミット」ボタンを押下する

リポジトリへのコミット

リポジトリ*
開発用リポジトリ

【開発者B】 マスタメンテナンス > UI

☐	ID	ラベル	タイプ
☒	UI_Home	ホーム画面	Custom
☒	UI_Product_List	商品一覧画面	Custom
☒	UI_Product_Edit	商品管理画面	Custom
☐	Change_Password_code	Change Password	Auth
☐	Error_code	Error	Auth
☐	MFA_Setting_Step1_code	MFA Setting Step1	Auth

コメント
ホーム画面/商品一覧画面/商品管理画面を新規作成

キャンセル コミット

④リポジトリにコミットした UI が反映されたことを確認

リポジトリ

開発用リポジトリ > UI

☐	ID	ラベル	タイプ	バージョン	担当者	コメント	更新	サイズ
☒	UI_Home	ホーム画面	Custom	AWqtCZTxD0E2xYwZDGQHUI1bkQtB1bgX	developerB@example.com	ホーム画面/商品一覧画面/商品管理画面を新規作成	07 Jan 2025, 03:59:32 PM	6.10KB
☒	UI_Product_Edit	商品管理画面	Custom	LKqS4bqGJPJaYedr9shPDqziYKZe8W	developerB@example.com	ホーム画面/商品一覧画面/商品管理画面を新規作成	07 Jan 2025, 03:59:32 PM	6.31KB
☒	UI_Product_List	商品一覧画面	Custom	ui503gSKldRgzAEIyVfLHM9o7TBv3yWY	developerB@example.com	ホーム画面/商品一覧画面/商品管理画面を新規作成	07 Jan 2025, 03:59:32 PM	7.75KB

3-4. 複数名開発：役割毎、運用イメージ詳細

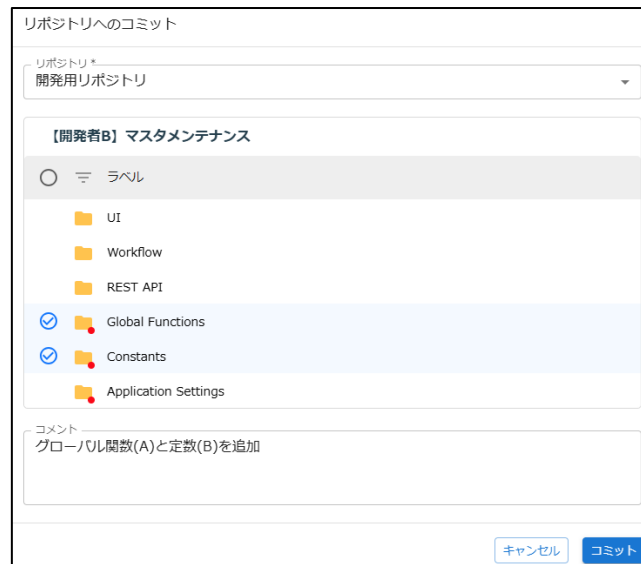
開発者B～E：開発フェーズ

補足：定数やグローバル関数をコミットしたい場合

リポジトリのチェックアウト
画面実装
UIをリポジトリへコミット

定義ファイルはコミットすると、上書きされる
既にコミット済みの定義との差分はマージされず消えてしまうため、注意が必要

① グローバル関数と変数を選択し、コミットボタンを押下



複数人で開発する場合、自分が編集中の定義に他の開発者がコミットしたら、まず編集内容をバックアップする。その後、最新の定義を取り込み、自分の編集内容を適用してコミットする必要がある
※チェックアウトされる内容の詳細については、

マニュアルのリポジトリ章をご確認ください。

ex) 定数の定義のみを選択した状態で、チェックアウトを実施



ラベル	タイプ	バージョン	担当者	コメント	更新	サイズ
Constants	Constants	k5rkfpRVW1b1uaZyqbCE4tO.FPpgVO3	developerD@example.com	LOG_DB を追加	10 Jan 2025, 09:42:35 AM	0.08KB

最新版の定義に対して、編集を行い、コミットを実施する



他の開発者の定義と競合が発生しないよう
密に連携しながら、編集を進めていく

3-4. 複数名開発：役割毎、運用イメージ詳細

開発者B～E：開発フェーズ

6. リポジトリのチェックアウト⇒画面修正⇒コミットを繰り返し行う

① アプリ定義ごとリポジトリをチェックアウトする



② 画面の修正を行う

③ UI をリポジトリへコミットする

開発用リポジトリ > UI > UI_Home								
○ ≡	ID	ラベル	タイプ	バージョン	担当者	コメント	更新	サイズ
☰	UI_Home	ホーム画面	Custom	z3ePS6zJp5zTVNrY51ZKrfeqx7IvTehU	developerC@example.com	顧客一覧画面への画面遷移処理を追加	14 Jan 2025, 01:06:49 PM	8.52KB
☰	UI_Home	ホーム画面	Custom	AWqtcZTzxD0EzxYwZDGQHU1bkQhBIbgX	developerB@example.com	ホーム画面/商品一覧画面/商品管理画面を新規作成	07 Jan 2025, 03:59:32 PM	6.10KB

コメントから修正内容を確認できる
特定のバージョンを無効/削除することも可能
※詳細はマニュアルのリポジトリ章をご確認ください。



①～③
リポジトリのチェックアウト、
画面実装/修正、リポジトリへのコミットを繰り返す

3-4. 複数名開発：役割毎、運用イメージ詳細

管理者A：開発フェーズ

7. リポジトリのスナップショットを作成

統合アプリケーションを作成
リポジトリの作成
リポジトリのアクセス権を開発者に付与

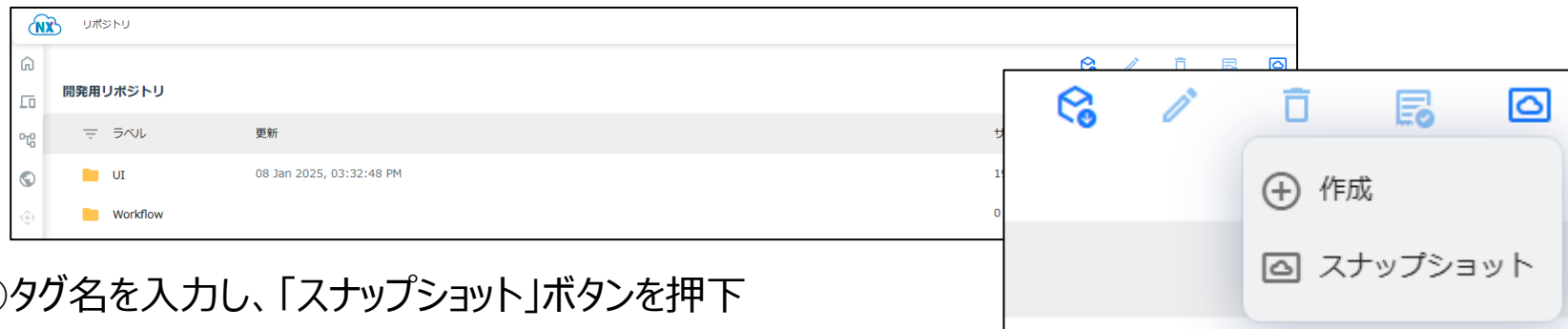
リポジトリのスナップショットを作成
リポジトリのチェックアウト
統合アプリケーションのデプロイ

定義を保存（FIX）したい時点でスナップショットを取得する

①作成したリポジトリをクリックする



②右上のスナップショットアイコンを選択し、「作成」ボタンを押下する



③タグ名を入力し、「スナップショット」ボタンを押下

スナップショット作成

タグ名
ver1_0

キャンセル スナップショット

スナップショットを作成することで、
特定のバージョンのアプリ定義を使用したい場合に、
チェックアウトして取得することが可能

3-4. 複数名開発：役割毎、運用イメージ詳細

管理者A：開発フェーズ

8. 「統合アプリケーションのデプロイ」の一環として、最新のアプリをチェックアウトする

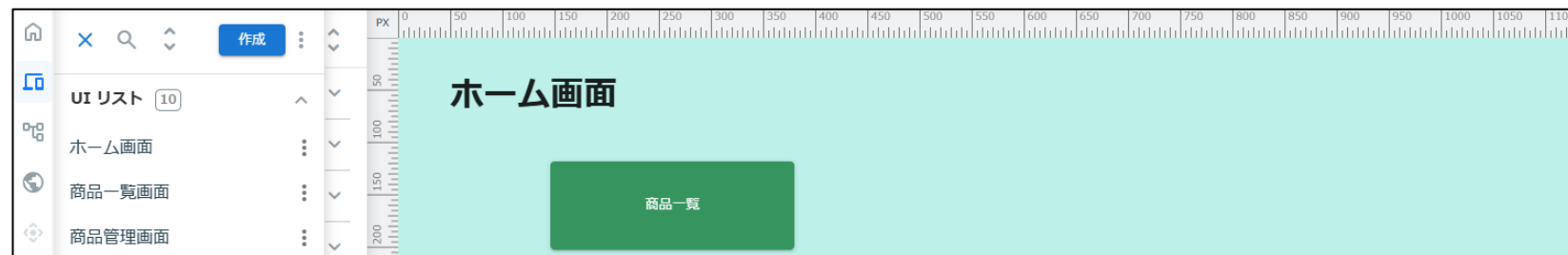
統合アプリケーションを作成
リポジトリの作成
リポジトリのアクセス権を開発者に付与

①リポジトリをチェックアウトする



スナップショットを選択すると、
保存している任意のバージョンを選択できる

②10画面がマージされた状態のアプリ定義が完成



リポジトリのスナップショットを作成
リポジトリのチェックアウト
統合アプリケーションのデプロイ

3-4. 複数名開発：役割毎、運用イメージ詳細

管理者A：開発フェーズ

9. 統合アプリケーションをデプロイする

統合アプリケーションを作成
リポジトリの作成
リポジトリのアクセス権を開発者に付与

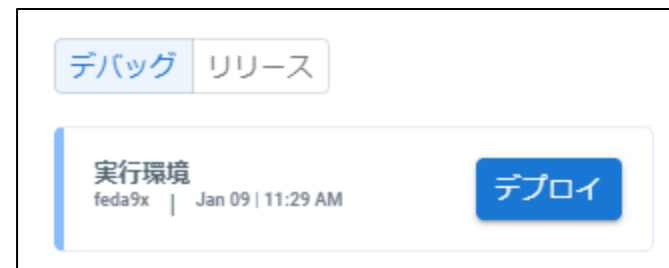
リポジトリのスナップショットを作成
リポジトリのチェックアウト
統合アプリケーションのデプロイ

①デプロイする環境にあわせて、定義修正や設定を行う

②アプリケーションを開いた状態で実行アイコンを選択



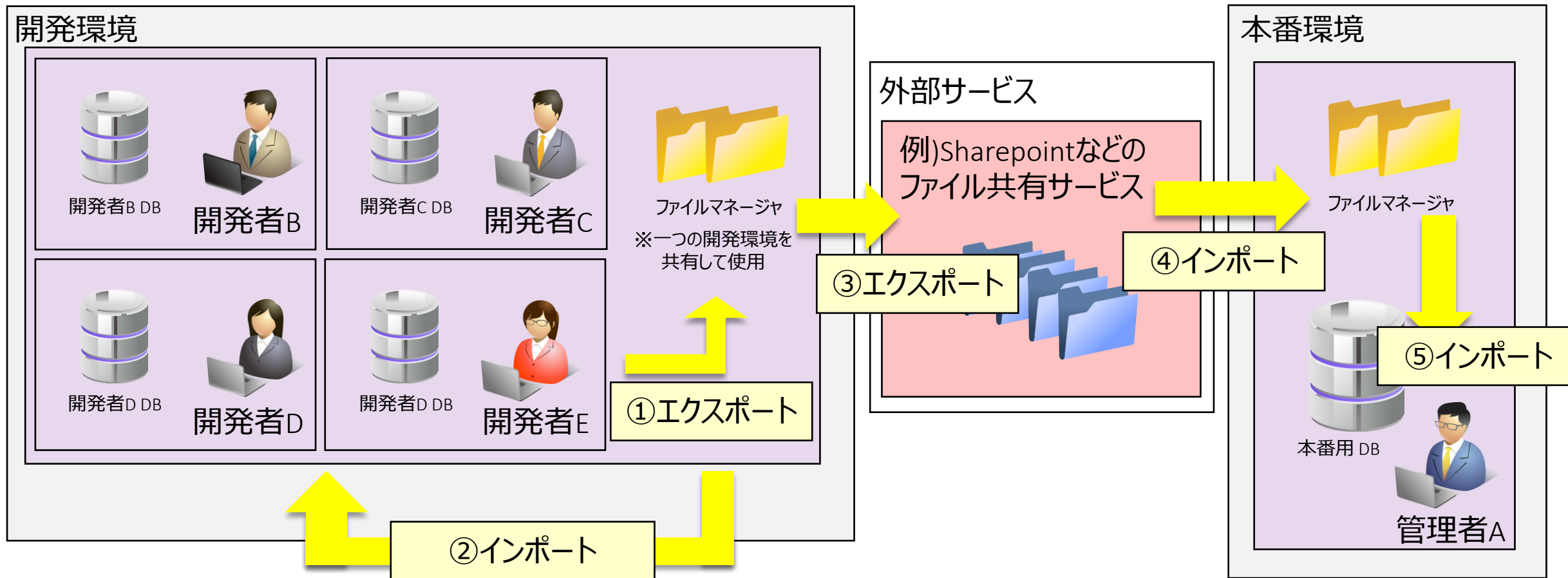
③デプロイ方法を選択し、デプロイを実行する



➡ P.10 開発フェーズを繰り返す

4. データベースの変更管理

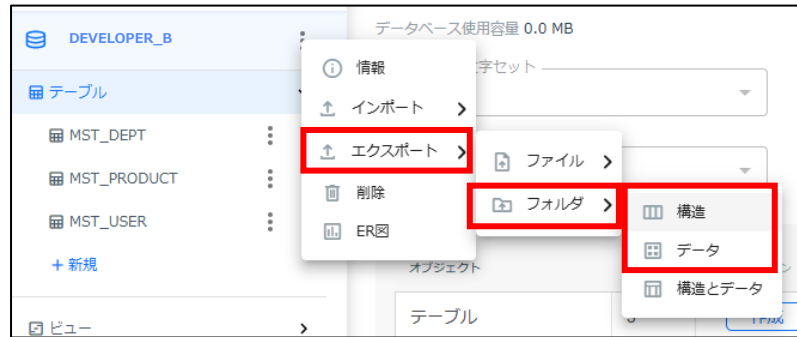
テーブル定義は NX で変更管理できないため、NX外のサービスを使って管理する必要があるため、
テーブル定義/マスタデータを NX のデータベースからエクスポートし、外部サービスで管理する



4. データベースの変更管理

外部サービスを利用した管理方法は下記の通り

① (代表者一人)開発B～E：必要なテーブル、データを準備し、テーブル定義/データをそれぞれエクスポートする



サイドメニュー「データベース」

> DB名の縦三点リーダーをクリックし「エクスポート」

> (テーブル定義)「フォルダ」-「構造」を選択して[エクスポート]

(データ)「フォルダ」-「データ」を選択して[エクスポート]

⇒ 作成したテーブル定義/データがそれぞれ Zipファイル としてファイルマネージャにエクスポートされる

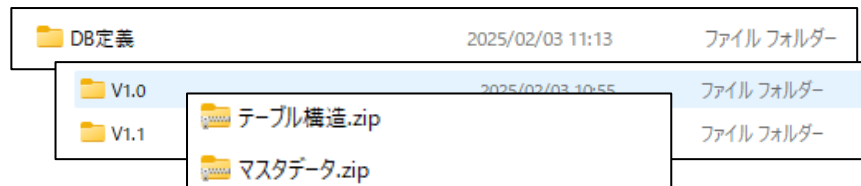
※「構造とデータ」を選択してまとめて[エクスポート]する手順でも問題ありません。



サイドメニュー「ファイル」

> エクスポートしたフォルダを選択した状態でダウンロードアイコンをクリック

②(外部サービス)管理者A：テーブル定義/データをバージョン管理するためのフォルダを作成し、①でエクスポートしたファイルを格納する



変更が発生する度、①と②の作業を繰り返す

※修正箇所が分かるようにコメントを残したい、バージョンの考え方などの運用ルールは、チームごとに検討・管理する必要がある

4. データベースの変更管理

③管理者A/開発者B～E：それぞれの環境に最新のテーブル定義/データを反映する



サイドメニュー「ファイル」

> フォルダアップロードアイコンをクリックし、任意の場所に解凍したZipファイルをアップロード



サイドメニュー「データベース」

> DB名の縦三点リーダーをクリックし「インポート」

> 「フォルダ」を選択して[インポート]

⇒ ①のテーブル定義/データをそれぞれインポートすることが可能

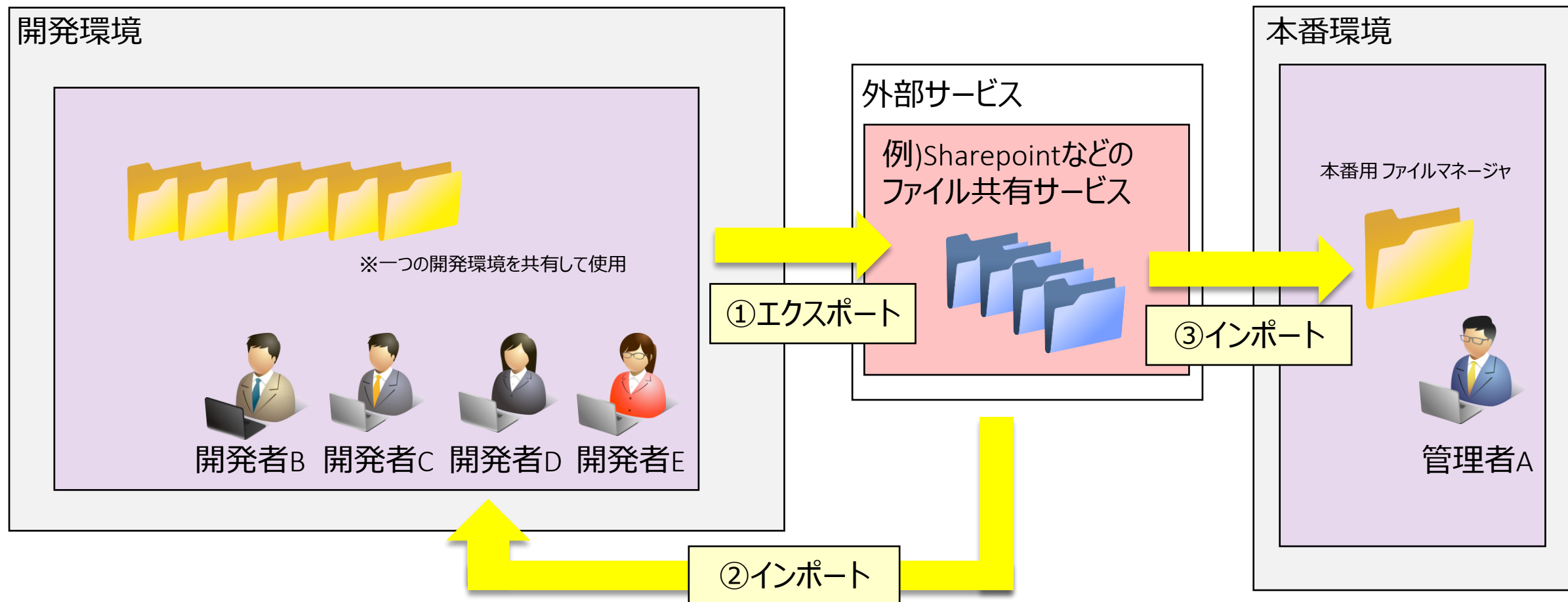


一度テーブル定義/データを作成した後に変更分を反映する場合、
場面に応じて下記の対応を実施するとよい

- ・登録したデータを事前にエクスポートしておき、テーブル定義を削除してからインポートする
- ・手動でテーブル定義/データを編集する

5. ファイルの変更管理

ファイルは NX で変更管理できないため、NX 外のサービスを使って管理する必要があるため、NX のファイルマネージャからフォルダ構成毎エクスポートし、外部サービスで管理する



5. ファイルの変更管理

外部サービスを利用した管理方法は下記の通り

- ① (代表者一人)開発B～E：必要なフォルダ、ファイルを準備し、ファイルマネージャからそれぞれエクスポートする



サイドメニュー「ファイル」

> 保存したいフォルダを選択した状態でダウンロードアイコンをクリック

⇒ 作成したテーブル定義/データがそれぞれ Zipファイル としてエクスポートされる

Rootフォルダに必要なファイルを格納する場合は、選択した状態でダウンロードアイコンを選択して保存する

- ②(外部サービス)管理者A：ファイルをバージョン管理するためのフォルダを作成し、①でエクスポートしたファイルを格納する

📁 ファイル構成	2025/02/13 16:37	ファイル フォルダー	
📁 V1.0	2025/02/03 10:55	ファイル フォルダー	
📁 V1.1	2025/02/03 11:13	ファイル フォルダー	
📁 Sampleフォルダ.zip	2025/02/12 17:15	圧縮 (zip 形式) フォ...	2,679 KB
📁 Rootフォルダ.zip	2025/02/12 17:21	圧縮 (zip 形式) フォ...	387 KB

ファイルをバージョン管理するためのフォルダを準備し、フォルダ構成ごとファイルを格納する



変更が発生する度、①と②の作業を繰り返す
※バックアップの手法や運用ルールは、チームごとに検討・管理する必要がある

- ③管理者A/開発者B～E：それぞれの環境に最新のファイル構成を反映する



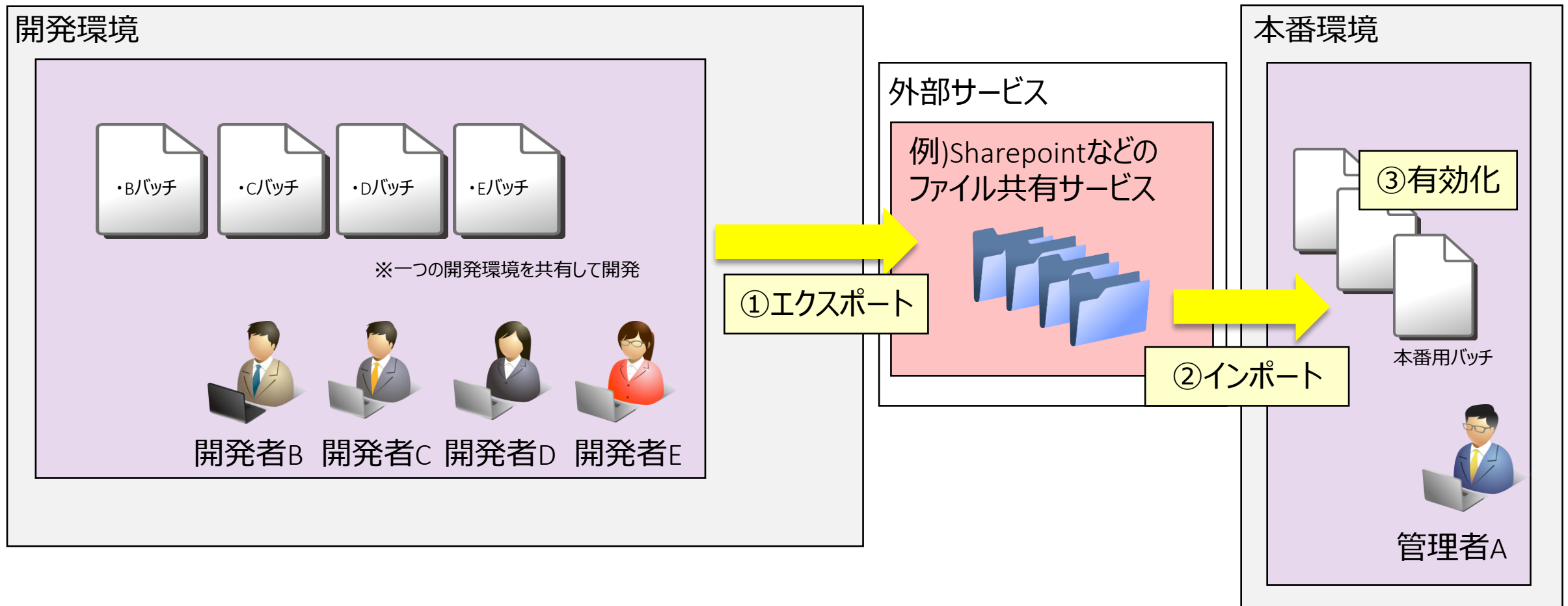
サイドメニュー「ファイル」

> フォルダアップロードアイコンを選択して、解凍したZipファイルをアップロード

> Rootフォルダにファイルをアップロードする場合はファイルアップロードアイコンを選択して、解凍したファイルをアップロード

6. バッチの複数名での開発、変更管理

ジョブ定義（バッチ）はリポジトリ機能で取り扱いできないため、外部サービスにて管理する。
それぞれで担当するジョブ定義を完成させたら、バッチをエクスポートし、
本番環境にインポートした後、本番環境実行するために定義を整え、実行する。

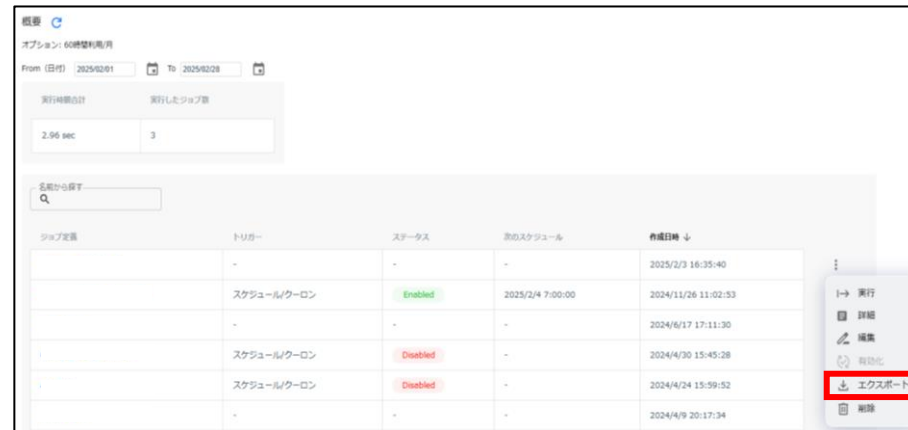


6. バッチの複数名での開発、変更管理

外部サービスを利用した管理方法は下記の通り

開発完了後に、エクスポートを行う

①開発者B～E：ジョブ定義をそれぞれエクスポートする



サイドメニュー「バッチ」

> 対象のジョブ定義バッチの縦三点リーダーをクリックし「エクスポート」

⇒ 作成したジョブ定義が zipファイル としてエクスポートされる

②開発者B～E：ジョブ定義をバージョン管理するためのフォルダを作成し、ファイルを格納する

ジョブ定義	2025/02/03 11:13	ファイル フォルダ
V1.0	2025/02/03 10:55	ファイル フォルダ
V1.1	2025/02/03 11:13	ファイル フォルダ

名前	更新日時	種類	サイズ
メール送信バッチ.zip	2025/02/03 17:14	圧縮 (zip 形式) フォ...	1 KB

エクスポートした zipファイルをフォルダに格納する



定義を保存（FIX） したい時点で、①と②の作業を繰り返す

※修正箇所が分かるようにコメントを残したい、バージョンの考え方などの運用ルールは、チームごとに検討・管理する必要がある

③管理者A：本番環境に最新のジョブ定義をインポートする



サイドメニュー「バッチ」

> 画面右上の作成ボタンをクリックし「インポート」

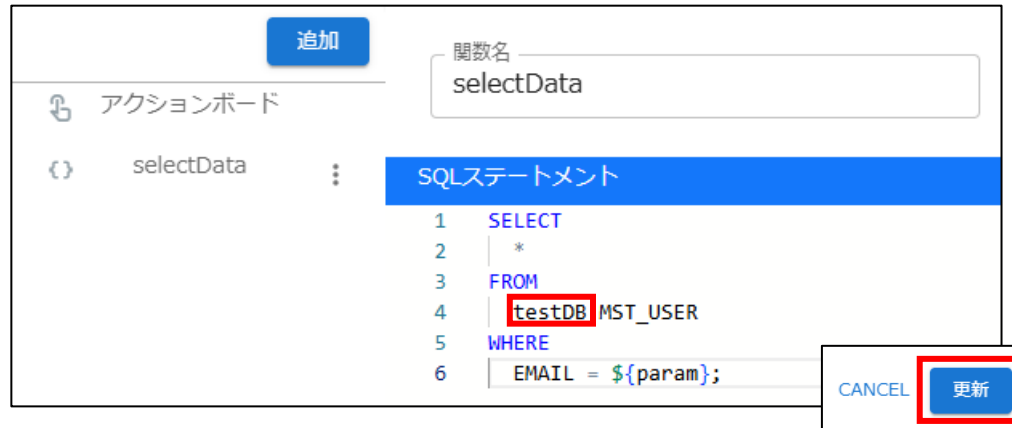
⇒ ジョブ定義を取り込むことが可能となる

6. バッチの複数名での開発、変更管理

④管理者A：本番環境にて実行するため、インポートしたジョブ定義を整える



サイドメニュー「バッチ」
＞インポートしたジョブ定義を開き、編集ボタンを選択



SQL関数を開き、
接頭辞として指定しているデータベース名を本番環境用のデータベース名に書き換える
⇒ 定義している SQL関数分、変更作業を実施する
⇒ 変更が完了したら、画面右下の更新ボタンを押下し、保存する

⑤管理者A：ジョブ定義を有効にし、実行状態とする



サイドメニュー「バッチ」
＞ジョブ定義バッチの縦三点リーダーをクリックし「有効化」
⇒ ジョブ定義が有効状態となり、スケジュールされた時間ごとにバッチが実行される

