WebPerformer-NX

WebPerformer-NX
NX開発における自動テストの導入
参考資料

Canon キヤノン ITソリューションズ株式会社

目次

1. 本資料の目的

本資料の対象は、WebPerformer-NX（以降、WP-NX）で開発したアプリに対し
自動テストを活用するための方法をご紹介します。

WP-NXでのアプリ開発を進める際にご活用ください。

◆ 本資料の対象読者
- WP-NXサービスに関する基本的な知識を有していること。
- 一般的なシステム開発経験、もしくは同等の知識を有していること。

◆注意事項
- WP-NXバージョン 3.2.0時点での情報です。
- 本資料は自動テストを使用した開発の一例をご紹介するものであり、
  この方法で開発を進めなければならない、というわけではございません。

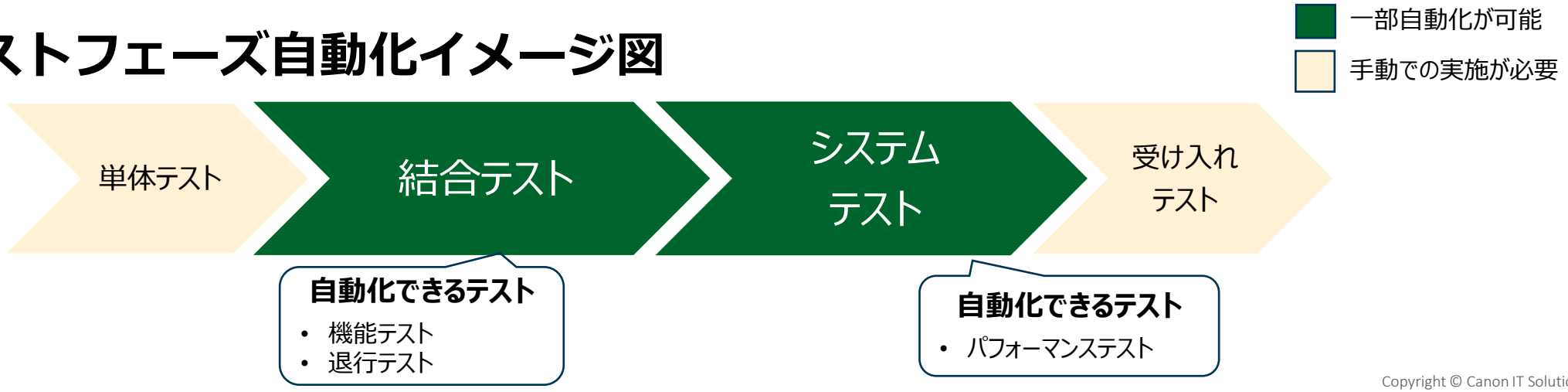
2. WP-NXに適したテスト自動化フェーズの考え方

WP-NXはローコード開発ツールであり、一般的なコードベースの開発とは異なる特徴を持っています。
このため、ソースコードの差分管理や、Git連携によるCI/CDパイプラインの自動実行には制約があります。
ただし、外部ツールを活用することで、テストフェーズの一部を自動化することは可能です。

WP-NXのようなローコードツールでは、部品単位のロジックよりも画面連携やワークフロー全体の整合性が
不具合の主要因となりやすい為、自動化を組み入れるテストフェーズは、
「結合テスト」、「システムテスト」を推奨します。
本資料では、上記フェーズ内の**「機能テスト」、「退行テスト」、「パフォーマンステスト」**
の自動テストについて取り上げてご紹介します。

※本資料における定義
- 結合テスト・・・機能テスト・退行テスト、外部システム連携を含むフェーズ
- システムテスト・・・パフォーマンステスト・セキュリティテスト・ユーザビリティテストを含むフェーズ

■テストフェーズ自動化イメージ図



3. 自動テスト導入のメリット･デメリット

自動テストは、品質向上や開発効率の改善に大きく貢献する一方で、導入や保守に一定のコストがかかるなどの課題も存在します。導入を検討する際には、こうした利点と注意点の両面を正しく理解し、最大限効果が発揮できるよう導入しましょう。

■導入メリット

〇作業効率の大幅改善

- 同一操作の繰り返しテスト負荷軽減
- 頻繁なテストが可能となる
- 夜間実行による開発時間の有効活用

〇品質向上効果

- 機能追加･変更によるデグレ早期発見
- 一貫したテスト実行による品質安定化
- 人的ミスによる見落としリスク軽減

■導入デメリット

〇初期コスト

- テストスクリプト作成工数が大きい
- ツール習得･環境構築時間が必要
- 初期投資回収まで時間がかかる

〇運用上の課題

- 仕様変更に伴うスクリプト修正コスト
- テスト環境の維持管理負荷

テスト自動化の対象を明確な基準で選定し、優先順位を付けて範囲を決定することが重要です。
具体的には、以下のような観点を考慮して自動化範囲を決めます。

- **テスト工数が大きい繰り返し操作**（回帰テスト対象）
- **大量のデータ準備が必要な箇所**（大量ユーザ同時接続など）
- **業務上重要な処理**（障害時の影響が大きい箇所）

4. 自動テスト項目の指針

■結合テストについて

結合テストではアプリ固有機能部分を優先的にテストすることを推奨します。
WP-NX提供機能のテスト実施は任意となりますが必要に応じて実施していただいても構いません。

以下は具体例です。

■テスト**推奨**項目：「アプリ固有機能」
- 所属グループによるパーミッション制御
- アクションに記載した処理（例：バリデーションチェック/画面遷移時の初期表示等）
- コンポーネントのプロパティ設定による制御

■テスト**任意**項目：「WP-NX提供機能」
- 認証機能の基本動作
- 数値項目の入力制御

ポイント

過剰なテストは開発効率を低下させ、不十分なテストは品質リスクを増大させます。
この指針に基づき、テスト範囲を明確にし、「効果的な工数配分と品質保証」を両立することが効果的なテスト戦略のポイントです。

4. 自動テスト項目の指針

■パフォーマンステストについて

WP-NXはサーバレス＋単一DBの特性がある為、
従来のオンプレミス環境で必要だったOSやミドルウェアのチューニング、
サーバリソースのスケール検証、ロードバランサーのスループット確認などは不要です。
これらはクラウド基盤が自動で管理するため、検証は省略することが出来ます。

一方で、サーバレス特有の挙動により、次のような性能テストは実施を推奨します。

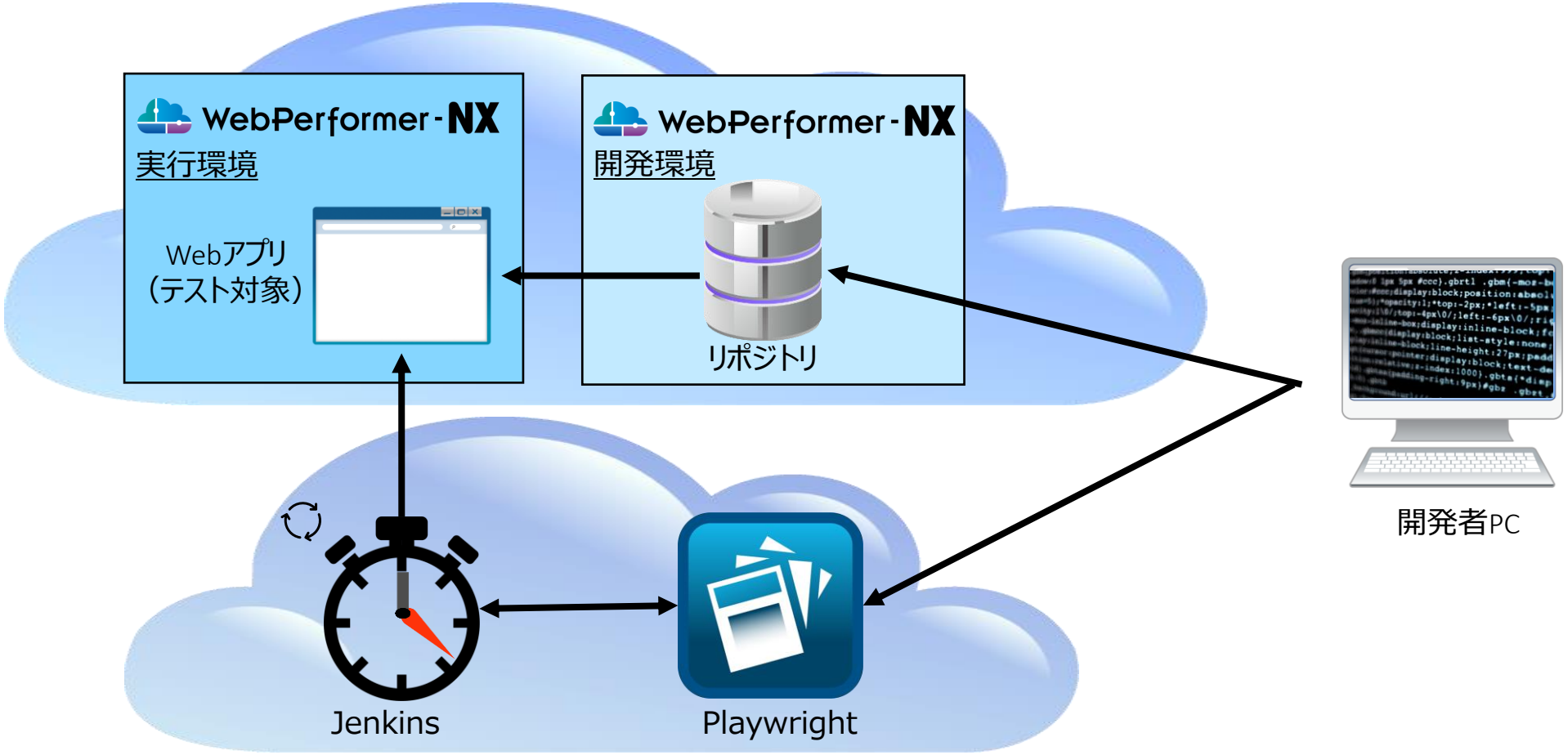
■推奨されるパフォーマンステスト項目

- **同時接続制限の検証**：開発したシステムで大量アクセス時に処理が滞らないか
- **外部サービス連携の性能確認**：単一DBや外部APIの応答速度、接続数制限を検証
- **スケーリング速度の確認**：負荷増加時にどれだけ迅速にスケールできるかを測定

5. 結合テストの自動化

■システム構成図

WP-NXで自動テストを実現する為のシステム構成案を紹介します。(本資料はWebアプリ開発の例です)
開発したデプロイ済みWebアプリに対しJenkinsによるスケジュールを設定し、自動テストを実行する
構成となります。



5. 結合テストの自動化

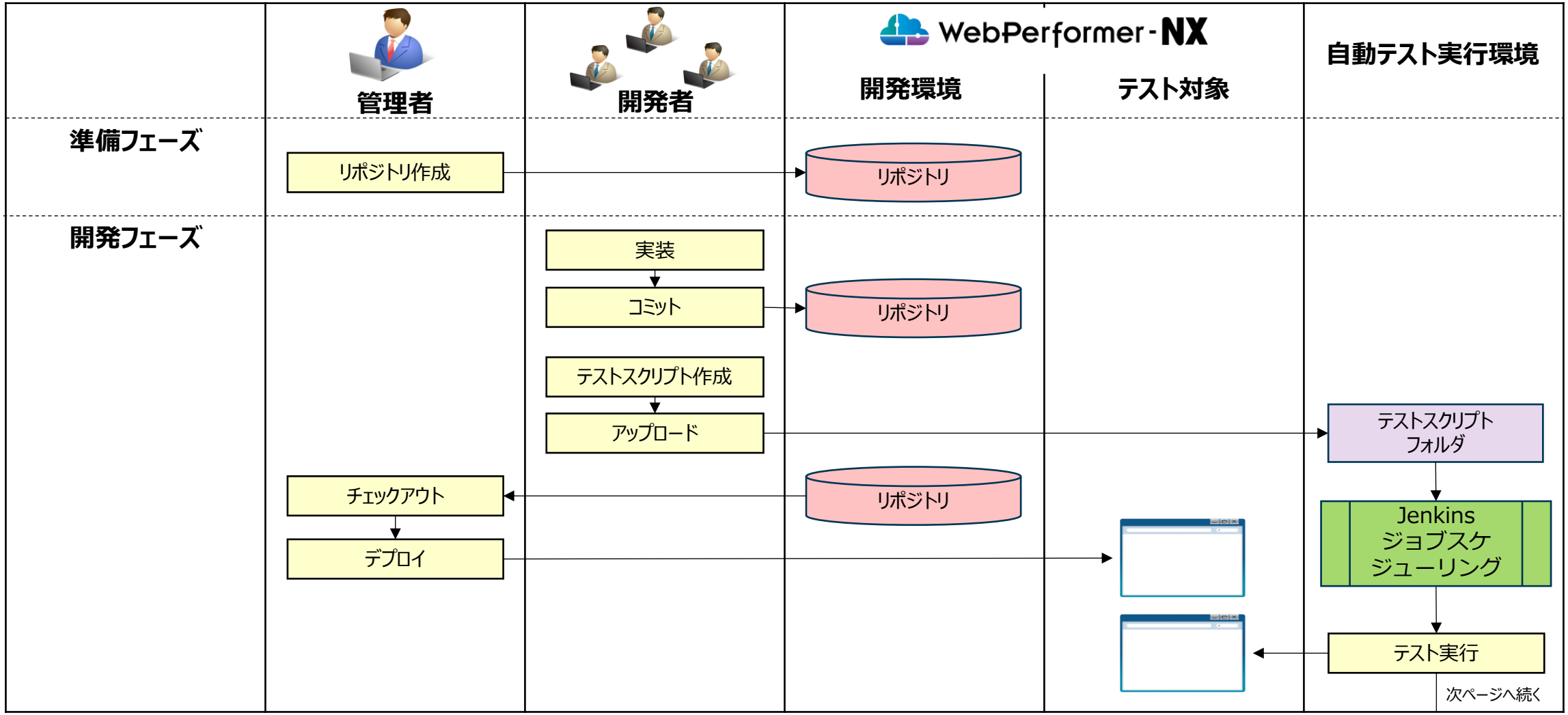
■使用するツール

本資料では自動テスト環境の構築にあたり、以下3つの主要ツールを使用します。
各ツールの特徴とWP-NXとの連携ポイントを詳しく解説します。

ツール名	役割	概要
Playwright	自動テストツール	• Microsoftが開発したWeb UI自動テストフレームワーク • WP-NXで開発したアプリに対して自動テストを実行するために使用します。 【公式サイト】https://playwright.dev/
VSCode	テストスクリプト開発ツール	• Microsoftが開発した無料でオープンソースのコードエディタ • テストスクリプトの開発に使用します。
Jenkins	実行スケジューリングツール	• ソフトウェア開発におけるビルド、テスト、デプロイなどのプロセスを自動化するツール • テスト開始のトリガーを自動化するために使用します。 【公式サイト】https://www.jenkins.io/

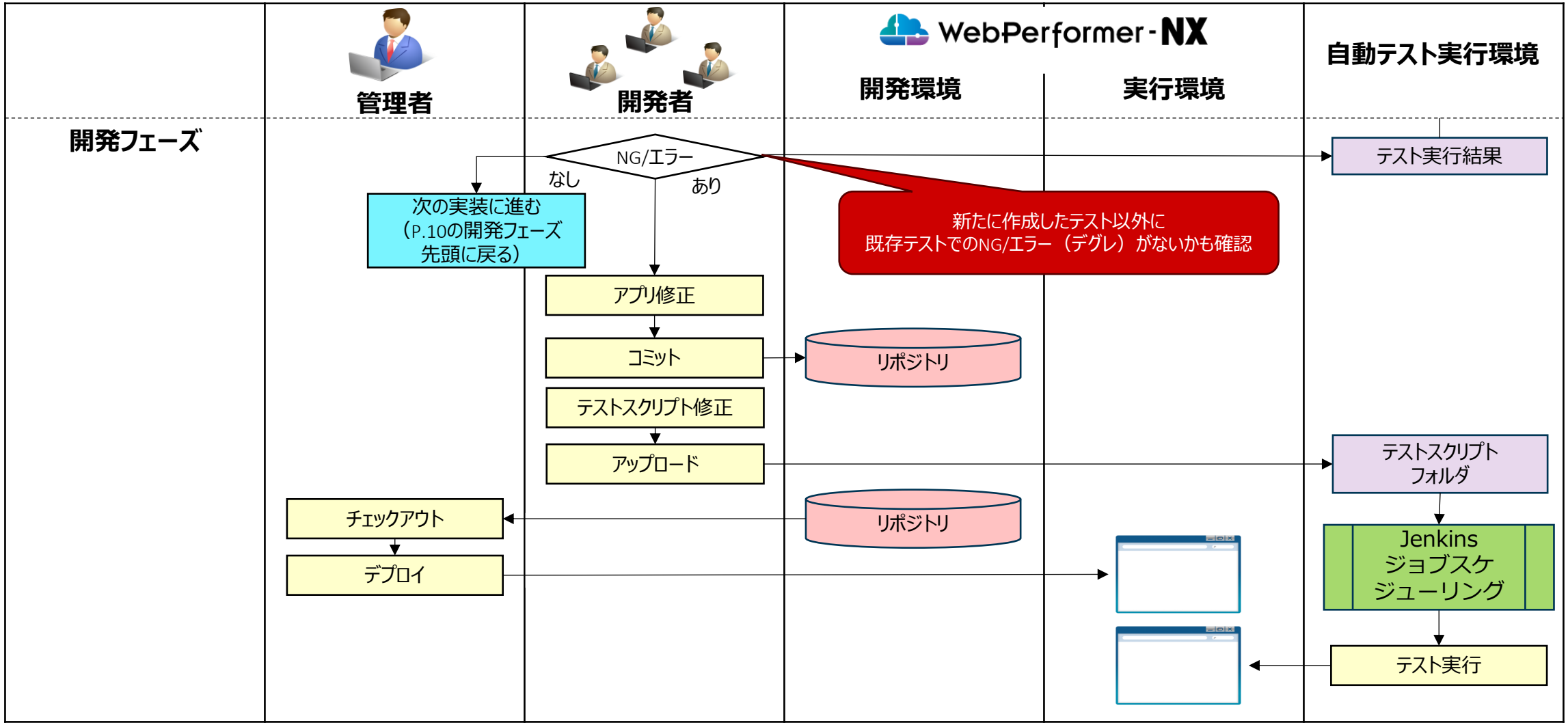
5. 結合テストの自動化

■WebPerformer-NX 開発・運用フロー [1/2]



5. 結合テストの自動化

■WebPerformer-NX 開発・運用フロー [2/2]



6. パフォーマンステスト自動化の考慮事項

■使用するツール

本資料ではパフォーマンステスト自動化にあたり、以下のツールを使用します。

ツール名	役割	概要
Apache JMeter	自動テストツール	• Apacheが提供しているオープンソースの負荷テスト／性能テストツール • WP-NXで開発したアプリに対して自動テストを実行するために使用します。 【公式サイト】https://jmeter.apache.org/

WP-NXでは、ユーザごとの認証情報や個別データを扱うため、パフォーマンステスト（以降、PT）において特定の設定が必要です。
具体的には**リクエストごとに動的にトークンやユーザ情報を付与する設定が必要**となります。
テスト実施時には、リクエストの値を適切に修正する必要があります。
※JMeterの記録結果には数十～数百件のHTTPリクエストが含まれることが一般的です。

以降はWP-NXのリクエスト仕様と JMeterの設定ポイントをまとめています。

◆ 前提
・JMeterを使用する（V5.6.3）
・JMeterの一般的な知識を有していること
・JMeterの記録コントローラー機能を使って記録したリクエストに対して、修正していく

6. パフォーマンステスト自動化の考慮事項

■WP-NXアプリのPTに必要なリクエスト仕様とJMeterでの設定ポイント

1. Body Dataの修正（各リクエストに共通）

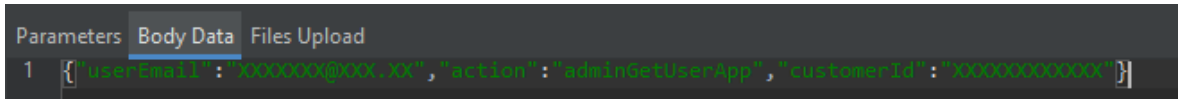
■設定対象：記録コントローラーで記録された HTTP Request（HTTP サンプラー）のうち、
「Server Name or IP」の末尾が「.webperformer.jp」のBody Data

■設定内容：Body Data内に記載があるユーザごとのログインユーザ情報等に、
以下の項目名称に「当てはまるキー」がある場合、変数に置換して Bodyに差し込む

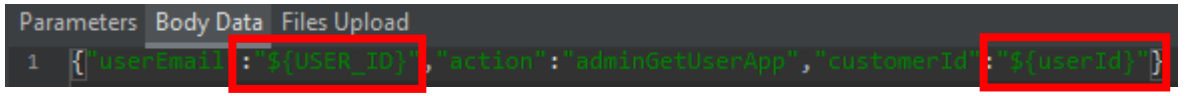
項目名称	値
username	${USER_ID}
userEmail（Email）	${USER_ID}
userId	${userId}
customerID	${userId}
localDate	${currentDateTime}
browserId	${browserId}
idToken	${idToken}
idTokenApp	${idToken}
accesstoken	${accesstoken}

○画像例

［置き換え前］



［置き換え後］



画像の説明
“action”：”adminGetUserApp”のリクエストには
【userEmail】と【customerId】のキーが存在している為
・"userEmail":"${USER_ID}“
・"customerId":"${userId}“
の2か所の修正を実施する

6. パフォーマンステスト自動化の考慮事項

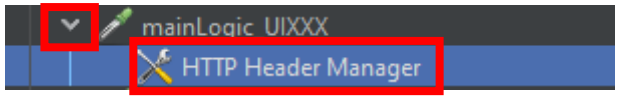
2. HTTP Header Manager の設定

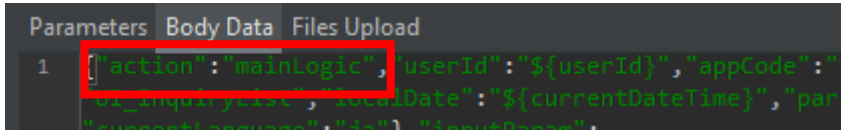
■設定対象：記録コントローラーで記録された HTTP Request（HTTPサンプラー）のうち、
「Server Name or IP」がテスト対象アプリのサーバ（例：*.webperformer.jp）を指すBody Data内に
[“action”:“mainLogic”]の記載があるリクエスト

■設定内容：該当リクエストの「＞」を開いたHTTP Header ManagerにAuthorizationヘッダーの追加

項目名称	値
Authorization	${idToken}

○画像例





6. パフォーマンステスト自動化の考慮事項

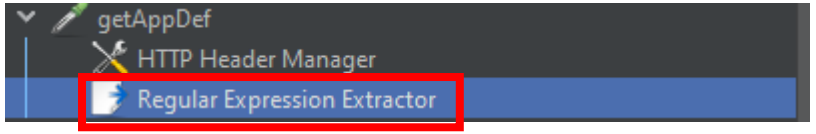
3. 変数の取得方法

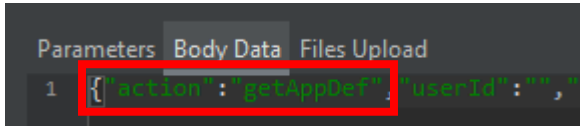
- userId の抽出（ [action:getAppDef]リクエストより取得）

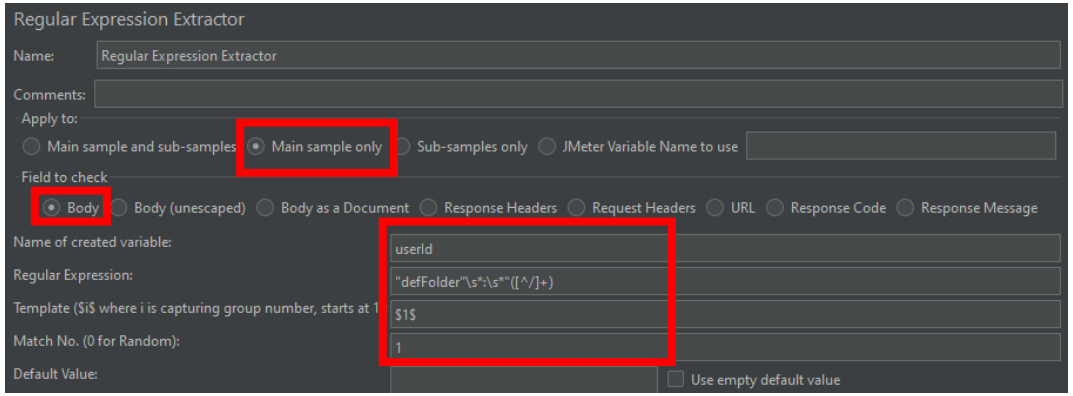
■設定対象：記録コントローラーで記録された HTTP Request（HTTPサンプラー）のうち、
Body Data内に[“action”:“getAppDef”]の記載があるリクエスト
■設定内容：該当リクエスト直下にRegular Expression Extractorを追加し下記の値を設定

項目名称	値
Apply to	Main sample only
Field to check	Body
Name of created variable	userId
Regular Expression	"defFolder"¥s*:¥s*"([^/]+)
Template	1
Match No	1

○画像例





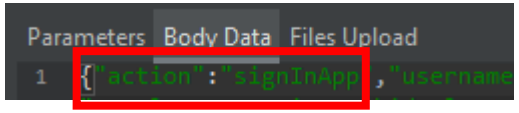


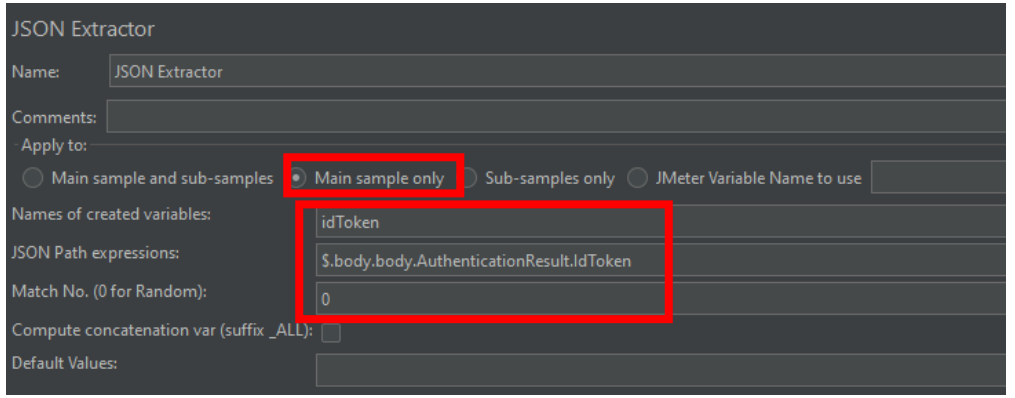
6. パフォーマンステスト自動化の考慮事項

- idToken の抽出（ [action: signInApp]リクエストより取得）
  - ■設定対象：記録コントローラーで記録された HTTP Request（HTTPサンプラー）のうち、Body Data内に[“action”:“signInApp”]の記載があるリクエスト
  - ■設定内容：該当リクエスト直下にJSON Extractorを追加し下記の値を設定

項目名称	値
Apply to	Main sample only
Name of created variables	idToken
JSON Path expressions	$.body.body.AuthenticationResult.IdToken
Match No	0

○画像例



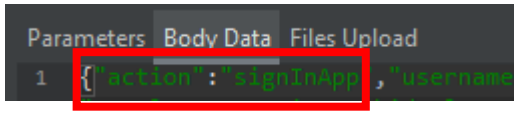


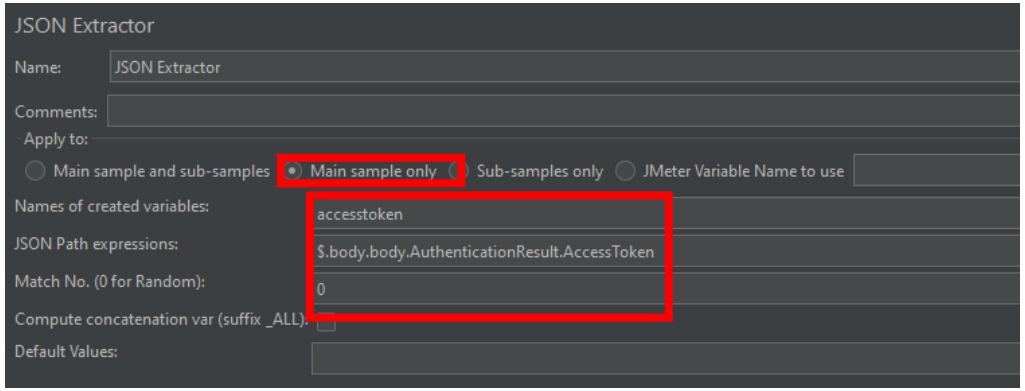
6. パフォーマンステスト自動化の考慮事項

- accessToken の抽出（ [action: signInApp]リクエストより取得）
  - ■設定対象：記録コントローラーで記録された HTTP Request（HTTPサンプラー）のうち、Body Data内に[“action”:“signInApp”]の記載があるリクエスト
  - ■設定内容： 該当リクエスト直下にJSON Extractorを追加し下記の値を設定

項目名称	値
Apply to	Main sample only
Name of created variables	AccessToken
JSON Path expressions	$.body.body.AuthenticationResult. AccessToken
Match No	0

○画像例





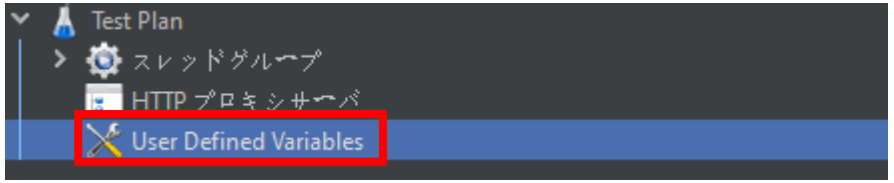
6. パフォーマンステスト自動化の考慮事項

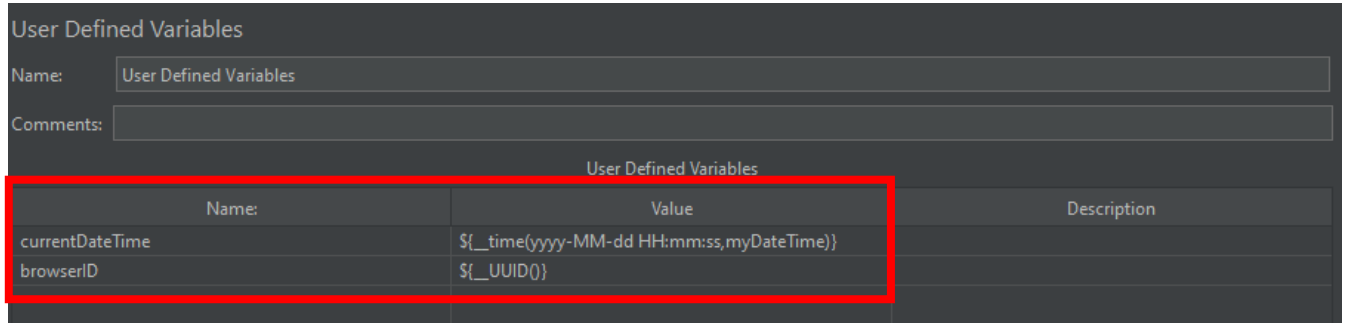
4. その他の変数設定

■設定対象：Test Plan
■設定内容：Test Plan直下にUser Defined Variablesを追加し下記を設定

項目名称	値
currentDateTime	${__time(yyyy-MM-dd HH:mm:ss,myDateTime)}
browserID	${__UUID()}

○画像例



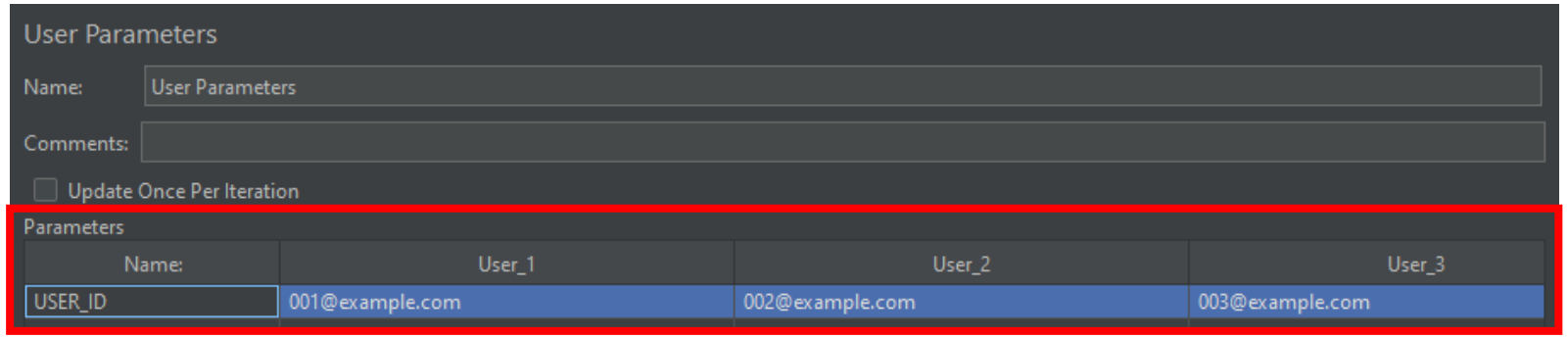


6. パフォーマンステスト自動化の考慮事項

■設定対象：Thread Group
■設定内容：Thread Group直下にUser Parametersを設定しPTにて想定ユーザ数のメールアドレスを
下記のように設定
※User ParametrersのUSER_ID（メールアドレス）は“ダミー値”として動作する為、
実在するものである必要はありません。また、ユーザマネージャに存在している必要もありません。
ただし、メールアドレスを入力/通知/検索等で実機能のキーにするシナリオでは、妥当な値を使用してください。

項目名称	値
USER_ID	（例）001@example.com,002@example.com,003,,,

○画像例（想定ユーザ数が3名の場合）



7.パフォーマンステストのためのシナリオ例と負荷設定例

本ページでは、PTを具体的にイメージできるよう、“テストシナリオ例” を提示します。

提示するシナリオは、あくまで PT設計の出発点となる参考例であり、最終的な画面遷移や操作内容、負荷設定はシステム要件に応じて適宜調整してください。

●シナリオ例

タイムライン	画面操作
[00:00]	サインイン
[00:10]	パスワード入力
[00:20]	A画面 –「新規作成」ボタンクリック
[00:30]	B画面 -「はじめる」ボタンクリック
[00:40]	C画面 – 情報入力して「次へ」ボタンクリック
[00:50]	D画面-情報入力して「次へ」ボタンクリック
[01:00]	E画面 -情報入力して「次へ」ボタンクリック
[01:10]	F画面-情報入力して「保存」ボタンクリック
[01:20]	G画面 -「ユーザ」ボタンクリック
[01:30]	プロフィール画面 -「サインアウト」ボタンクリック

7. パフォーマンステストのためのシナリオ例と負荷設定例

●同時接続と測定時間

本ページでは、P20で示した「PTシナリオ例」を実際に負荷試験へ落とし込む際に使用する、
Number of Threads（同時接続数）・Ramp-up（立ち上げ時間）・Duration（測定時間）の
設定例を提示します。これらはあくまで参考値であり、実際の要件に応じて適宜調整してください。

項目名称	設定例
Number of Threads(users)	135
Ramp-up period(seconds)	720
Duration(seconds)	1800

※上記設定は、１スレッド毎に５秒間隔で処理を実施し、135同時ユーザを12分で立ち上げ、
合計30分間処理を続けるテストの一例です。

※同時リクエスト制限とPT実施時の注意事項

WP-NX では、同一IPアドレスからのリクエストに対して下記の制限があります。

- 同一IPからの同時リクエスト数は、最大 2,000 リクエスト／分で制限されている
- 大量リクエストを意図的に発生させるテスト（例：高負荷・高並列アクセス試験）を行う場合、
  Web Application Firewall（WAF）によりブロックされる可能性がある

そのため、同一IPからの同時リクエスト数を増加させるPTを実施する場合は、事前に制限緩和の相談・申請が
必要となります。（制限緩和については、事前にQAサイトまたは営業担当を通じてご相談・お問い合わせいただくようお願い致します。）

7. パフォーマンステストのためのシナリオ例と負荷設定例

●まとめ

PT設計時にはJMeterの設定に加え、本資料の6章に記載した「設定ポイント」を併せて参照していただくことで、WP-NX特有のリクエスト仕様に適合した正しいPT実装が可能になります。

以上を踏まえ、実際のプロジェクトにおいては、WP-NX 特有のリクエスト仕様を確実に反映させるため、
本資料の「設定ポイント」と「シナリオ例」を併せて参照しつつ、
システム要件に応じたパフォーマンステストの設計および自動化の検討を進めていただければ幸いです。