IBM API Connect

IBM API Connect
OAuth構成概要と構成例

2017/06/16
日本アイ・ビー・エム株式会社
クラウド・ソフトウェア

Disclaimer

当資料の位置づけ

- 当資料は、IBM API Connect での過去のOAuth機能の実装例を参考に構成方法および考慮点をまとめたものです。
- API Connect V5.0.6を前提としています。

注意事項

- 当資料に含まれる情報は可能な限り正確を期しておりますが、当資料に記載された内容に関して何ら保証するものではありません。ここでの記載内容はあくまでも支援情報であり、使用者の責任において取扱われるものとし、資料の内容によって受けたいかなる損害に関して一切の保証をいたしません。
- 製品の新しいリリース、修正などによって動作／仕様が変わる可能性がありますので、必ずマニュアル等で最新の情報をご確認ください。

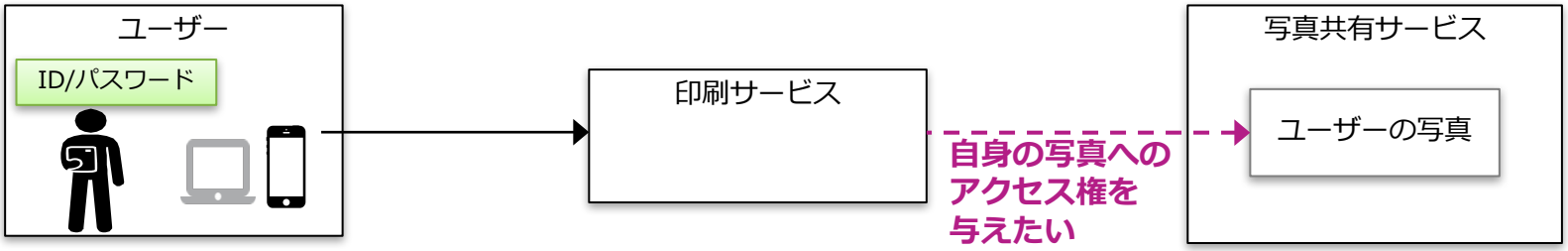
目次

OAuth2.0概要

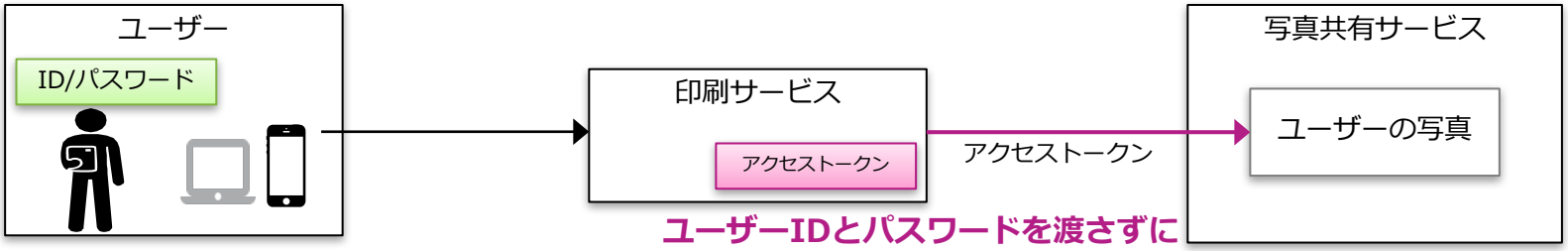
OAuth 2.0概要

❑ OAuth 2.0とは

- サードパーティーアプリケーションによるHTTPサービスへの限定的なアクセスを可能にする認可フレームワーク
- トークンによるAPIアクセス認可の標準仕様
- 現在のバージョンはOAuth 2.0(RFC 6749, RFC 6750)
- OAuth利用の例
  - ユーザーが、印刷サービスに対して、写真共有サービス上に保管されているユーザーの写真へのアクセス権を与えることを想定する

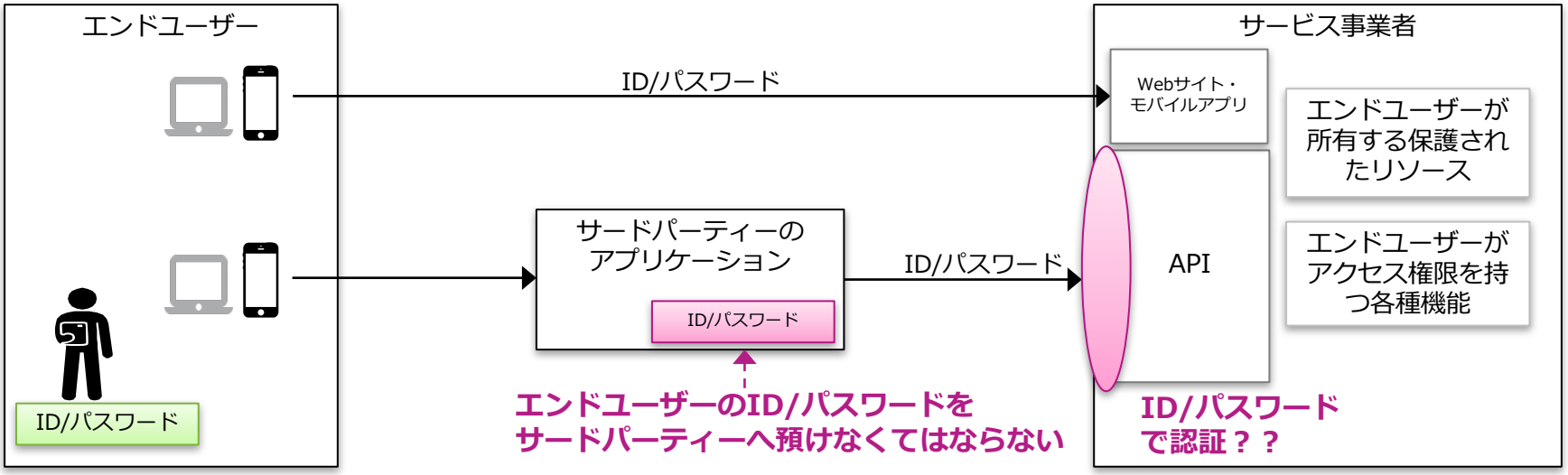


  - OAuthでは、ユーザーIDとパスワードを印刷サービスに与えるかわりに、印刷サービスへのアクセス権限委譲用クレデンシャル (アクセストークン) を使ってアクセスさせる



OAuthが必要となる背景(1)

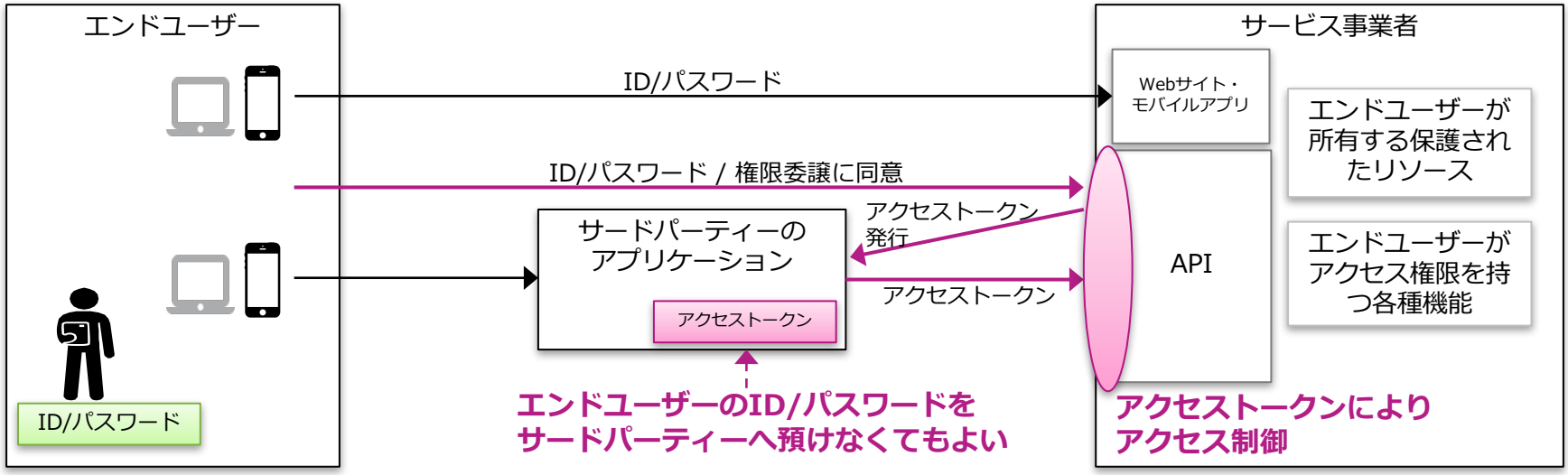
- サードパーティーのアプリケーションへの権限付与における課題
  - エンドユーザーが所有する保護されたリソースやエンドユーザーがアクセス権限をもつサービス事業者の各種機能に対して、サードパーティーのアプリケーションにアクセスさせる場合の課題
    - エンドユーザーのクレデンシャル(ID/パスワード)をサードパーティーのアプリケーションに預けなくてはならないため、クレデンシャルの不正利用や情報漏洩の危険性がある
    - サードパーティーのアプリケーションからのアクセス範囲を制限できないため、過剰な権限が与えられてしまう
    - サードパーティーのアプリケーションのアクセスの有効期間を制御できない
    - サービス事業者は、アクセスするアプリケーションを制御できず、どのアプリケーションからアクセスが行われているのか把握できない



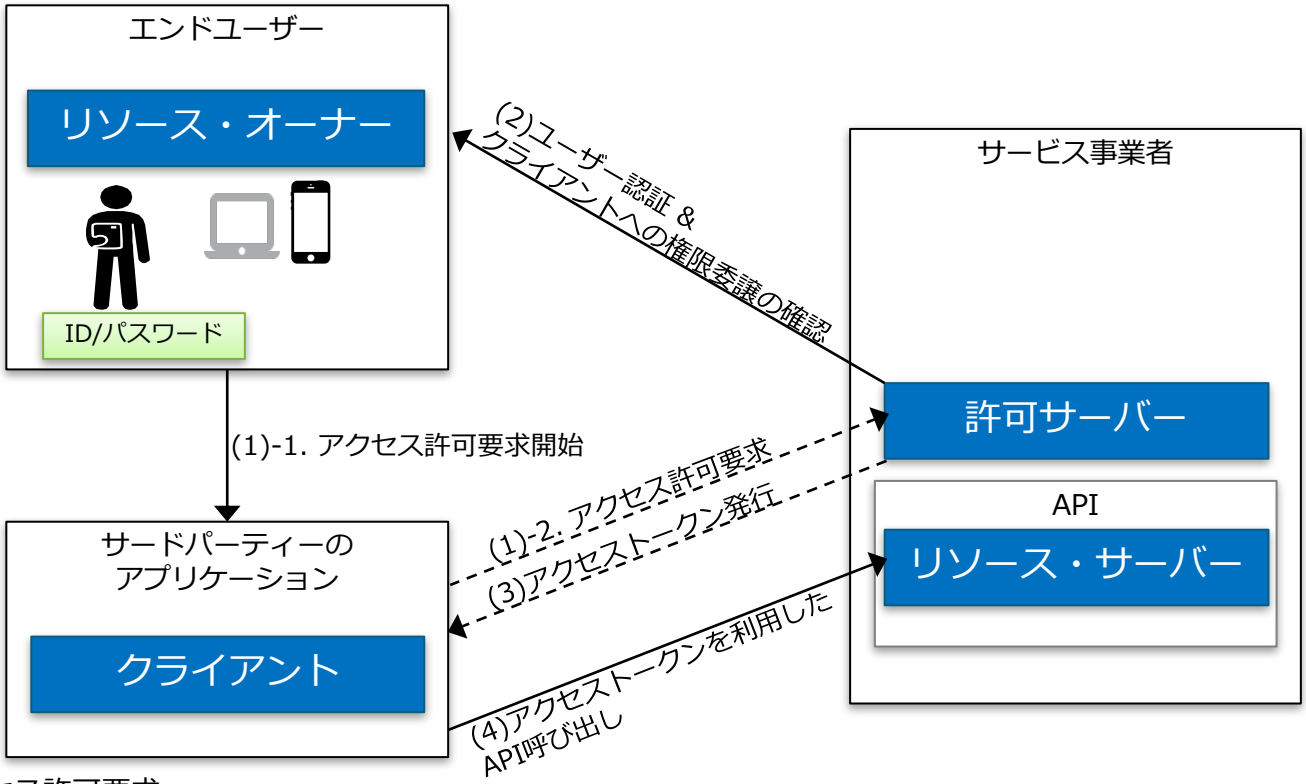
OAuthが必要となる背景(2)

❑ OAuthによるサードパーティーのアプリケーションへの権限付与

- エンドユーザーのクレデンシャル(ID/パスワード）をサードパーティーのアプリケーションへ渡す代わりに、サービス事業者のアクセス権限委譲用クレデンシャル（アクセストークン）を使ってアクセスさせる
  - エンドユーザーのクレデンシャル(ID/パスワード)をサードパーティーのアプリケーションに預けなくてもよい
  - サードパーティーのアプリケーションへ委譲する権限の範囲を制限できる
  - サードパーティーのアプリケーションへアクセスが許可される期間を制限できる
  - サービス事業者は、アクセスするアプリケーションを制御でき、どのアプリケーションからアクセスが行われているのか把握できる



OAuthの登場人物と処理の流れ



(1) アクセス許可要求
　　リソース・オーナーが、リソースへのアクセス許可要求を開始する
　　クライアントは、許可サーバーに対してアクセス許可要求を行う
(2) ユーザー認証 & クライアントへの権限委譲の確認
　　許可サーバーは、リソース・オーナーの認証を行い、クライアントへの権限委譲の確認を行う
(3) アクセストークン発行
　　許可サーバーは、クライアントに対してアクセストークンを発行する
(4) アクセストークンを利用したAPI呼び出し
　　クライアントはアクセストークンを利用して、リソースにアクセスする

RFC6749の日本語訳では、 Authorization serverは「認可サーバー」、 Authorization Codeは「認可コード」と記述されていますが、本製品の製品マニュアル KnowledgeCenterの日本語訳では、「許可サーバー」、「許可コード」と訳されており、本資料中でもその記述を踏襲します。あらかじめご了承ください。

OAuth構成要素 (1/2)

OAuthの詳細についてはRFC 6749を参照

- リソース・オーナー（Resource Owner）
  - 保護されたリソースへのアクセスを許可するエンティティー
  - 人間の場合はエンド・ユーザーに相当
- クライアント（Client）
  - リソース・オーナーの許可を得て、リソース・オーナーの代理として保護されたリソースに対するリクエストを行うアプリケーション
  - アクセス・トークンを使用して保護されたリソースにアクセスする
- 許可サーバー（Authorization Server）
  - リソース・オーナーの認証とリソース・オーナーからの許可取得が成功した後、アクセス・トークンをクライアントに発行するサーバー
  - リソース・サーバーと同一サーバーの場合と異なるサーバーの場合がある
- リソース・サーバー（Resource Server）
  - 保護されたリソースをホストし、保護されたリソースへのリクエストを受理してレスポンスを返すサーバー

RFC6749の日本語訳では、 Authorization serverは「認可サーバー」、 Authorization Codeは「認可コード」と記述されていますが、本製品の製品マニュアルKnowledgeCenterの日本語訳では、「許可サーバー」、「許可コード」と訳されており、本資料中でもその記述を踏襲します。あらかじめご了承ください。

OAuth構成要素 (2/2)

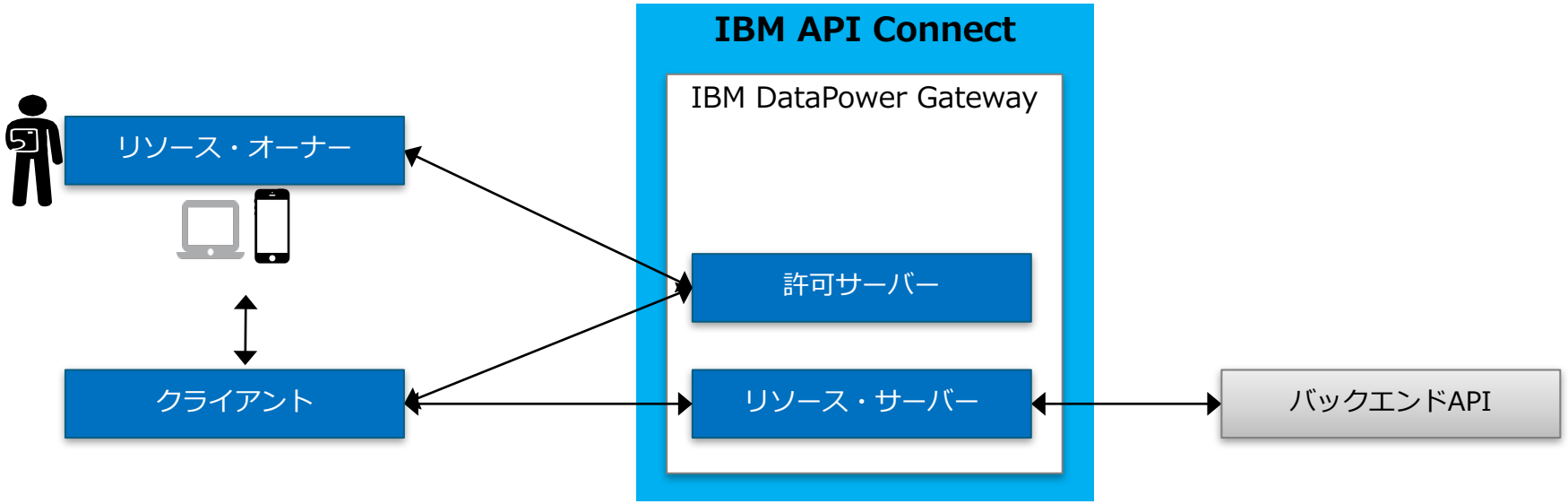
OAuthの詳細についてはRFC 6749を参照

- アクセス・トークン
  - リソース・オーナーの保護リソースにアクセスするためのクレデンシャル
  - OAuthクライアントが保護リソースへアクセスする際にリソース・サーバーに送信する
- リフレッシュ・トークン
  - アクセス・トークンを取得するためのクレデンシャル
  - アクセス・トークンの期限が切れた場合に、許可サーバーから新規アクセス・トークンを要求したり、同一スコープまたはより狭いスコープを持つアクセス・トークンを要求するために使用
  - リソース・サーバーには送信されない

API ConnectにおけるOAuthサポート

API ConnectにおけるOAuthのサポートの概要

- API ConnectはOAuth 2.0をサポート
- OAuth 2.0におけるAPI Connectの役割
  - 「許可サーバー（認可サーバー）」および「リソース・サーバー」の役割を担う
  - バックエンドAPIへのアクセスを、リソース・サーバーがプロキシーして保護する



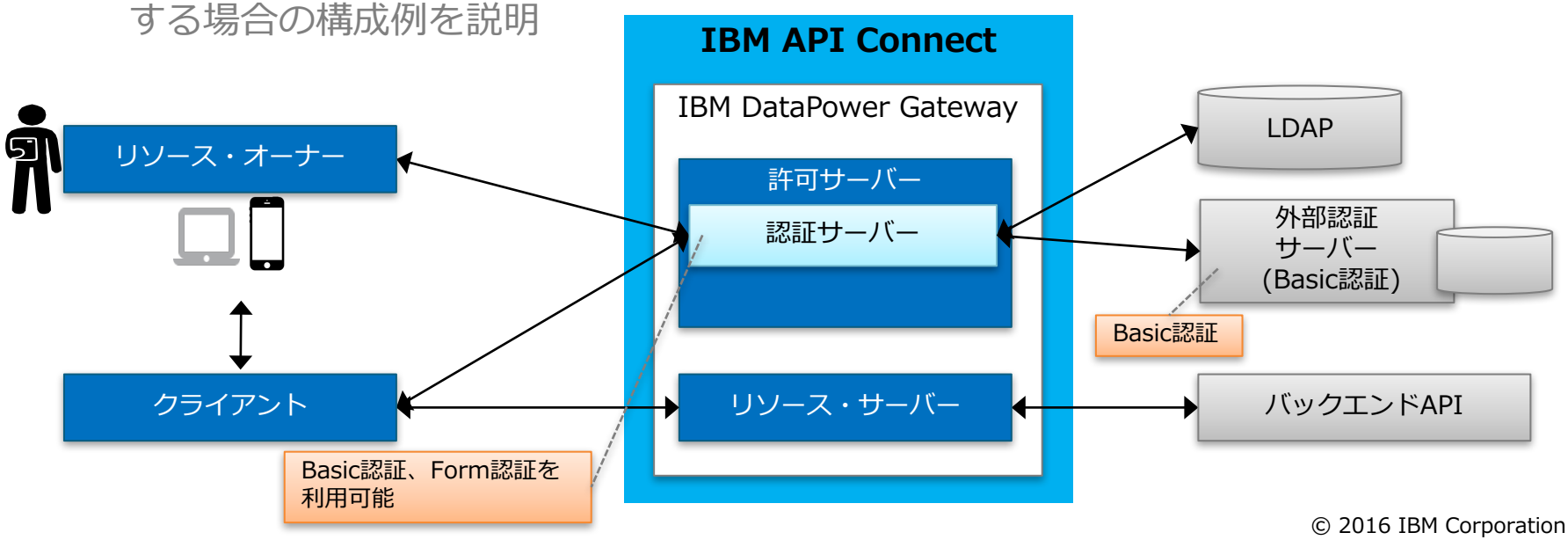
API ConnectでのOAuthの認証方法

- API Connectでは、OAuth 2.0の許可サーバーを実装する場合の認証方法を選択可能
  - 許可サーバーの設定で認証方法を指定
  - 認証方法は大きく2種類から選択可能
    - 方法1 API Connectで認証を行う構成
    - 方法2 外部認証サーバーで認証を行う構成

API ConnectでのOAuthの認証 方法1

❑ API Connectで認証を行う構成

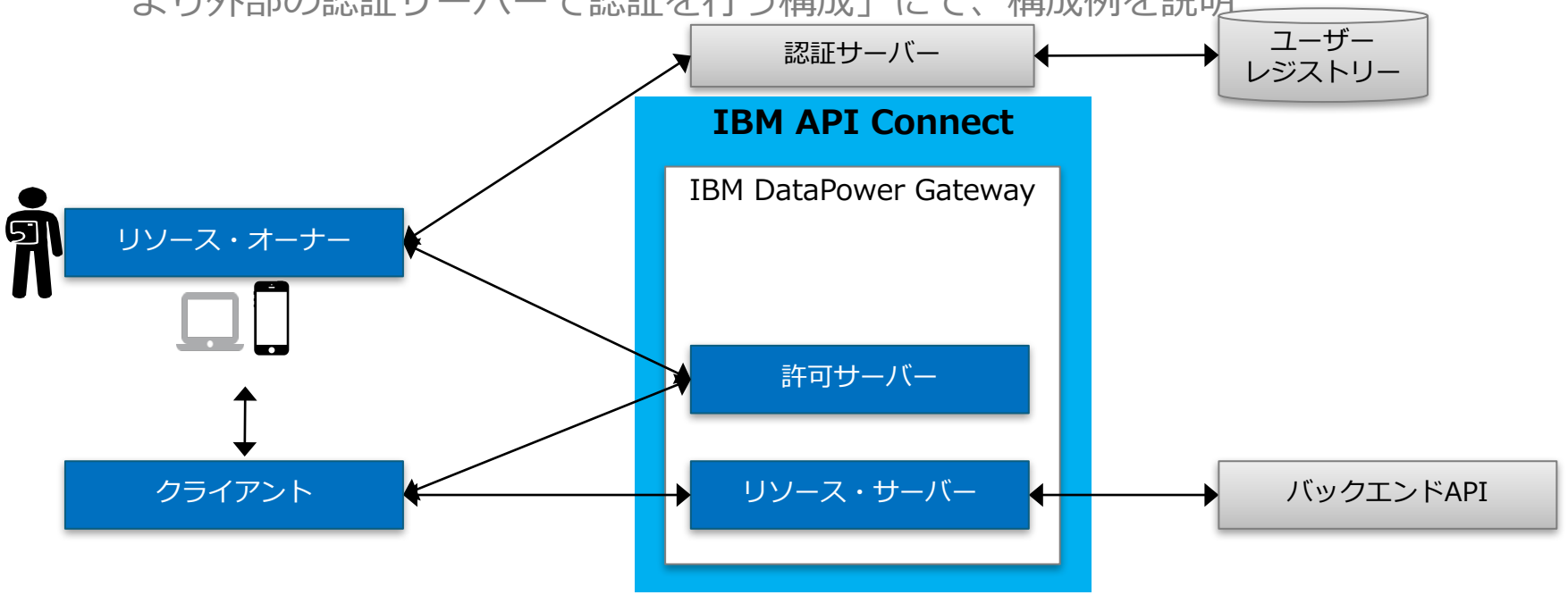
- API Connectの機能を利用して、DataPower Gatewayで認証を行う
- Basic認証、Form認証を利用可能
- ユーザーレジストリーとして、LDAP、Basic認証が構成されたサーバーのURLを利用可能
- 認証画面やリソース許可画面は製品提供の機能（または製品機能のカスタマイズ）を利用するが、認証については、LDAP、Basic認証のサーバーを利用する場合にこの構成を選択
- 後述の「構成例1. 外部認証サーバーを利用したOAuth認証 」「1-1. 外部サーバーとの連携によりAPI Connectで認証を行う構成」にて、Basic認証のサーバーを利用する場合の構成例を説明



API ConnectでのOAuthの認証 方法2

❑ 外部認証サーバーで認証を行う構成

- 外部の認証サーバーにリダイレクトを行い認証処理を任せることが可能
- ID/パスワードによる認証のみでなく、認証方法を柔軟にカスタマイズ可能
- API Connectで指定されるフローの仕様に沿って、外部認証サーバー側のカスタマイズを行う必要あり
- 認証画面やリソース許可画面、および認証ロジック等を外部で柔軟に構築したい場合にこの構成を選択
- 後述の「構成例1. 外部認証サーバーを利用したOAuth認証」「1-2. リダイレクトにより外部の認証サーバーで認証を行う構成」にて、構成例を説明



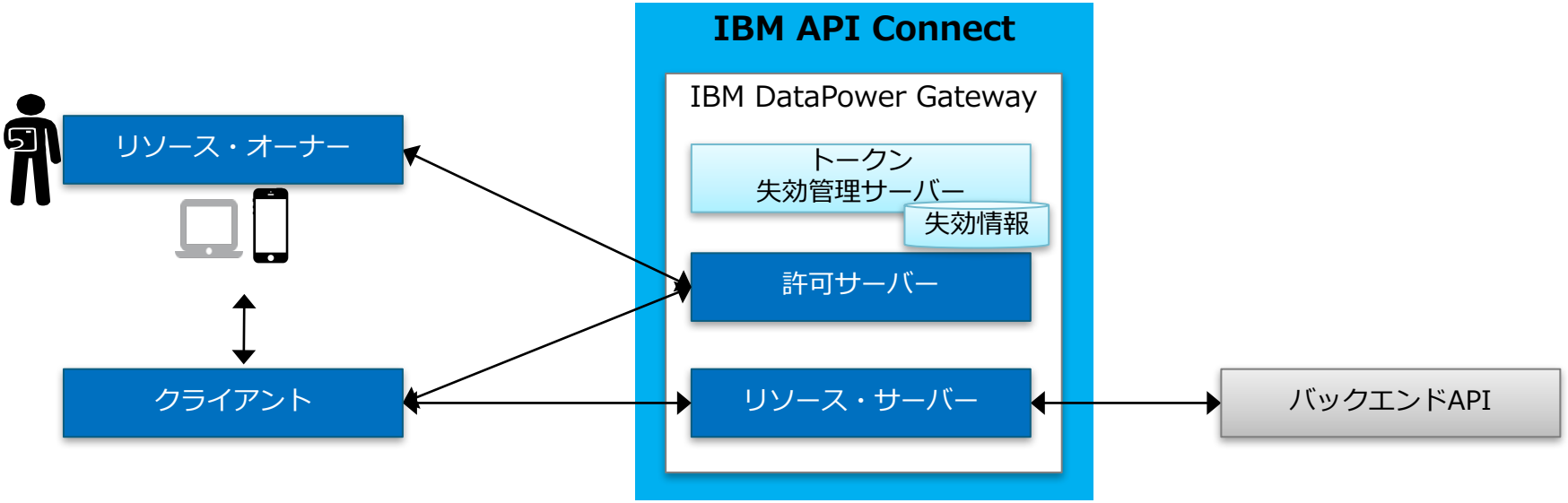
アクセストークンの失効管理方法

- API Connectでは、発行済みのアクセストークンを失効することが可能
  - 許可サーバーの設定で有効にすることで利用可能
  - トークン失効の方法は2種類から選択可能
    - 方法1 DataPower Gatewayを使用
    - 方法2 外部のトークン失効管理サーバーを利用(失効URLの使用)

アクセストークンの失効管理 方法1

❑ DataPower Gatewayを利用したトークン失効管理

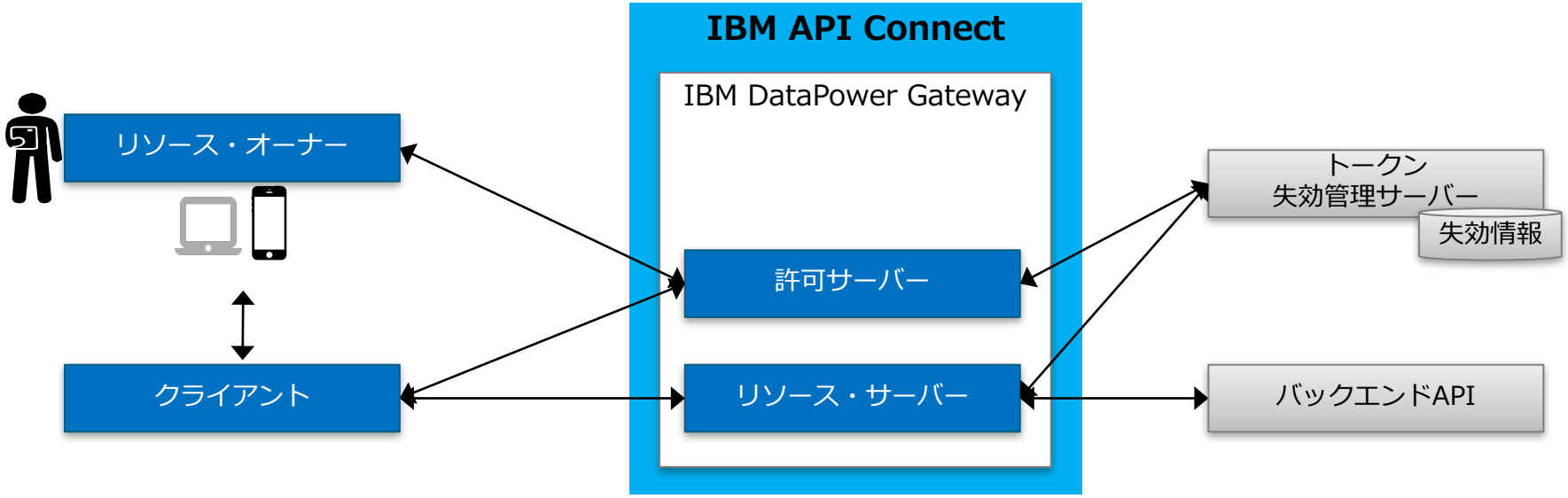
- DataPower Gateway上でトークン失効管理をする設定
- DataPower Gateway上でトークン失効のAPIが提供され、任意のタイミングでトークンの失効要求が可能
- DataPower上で、Quota Enforcement Serverの設定を有効にする必要あり
- 冗長構成をとる場合には、Quota Enforcement Serverのクラスター構成の前提によりDataPower Gatewayが3台以上必要となる
- 後述の「構成例2. トークン失効管理」「2-1. DataPower Gatewayを利用したトークン失効管理」にて、構成例を説明



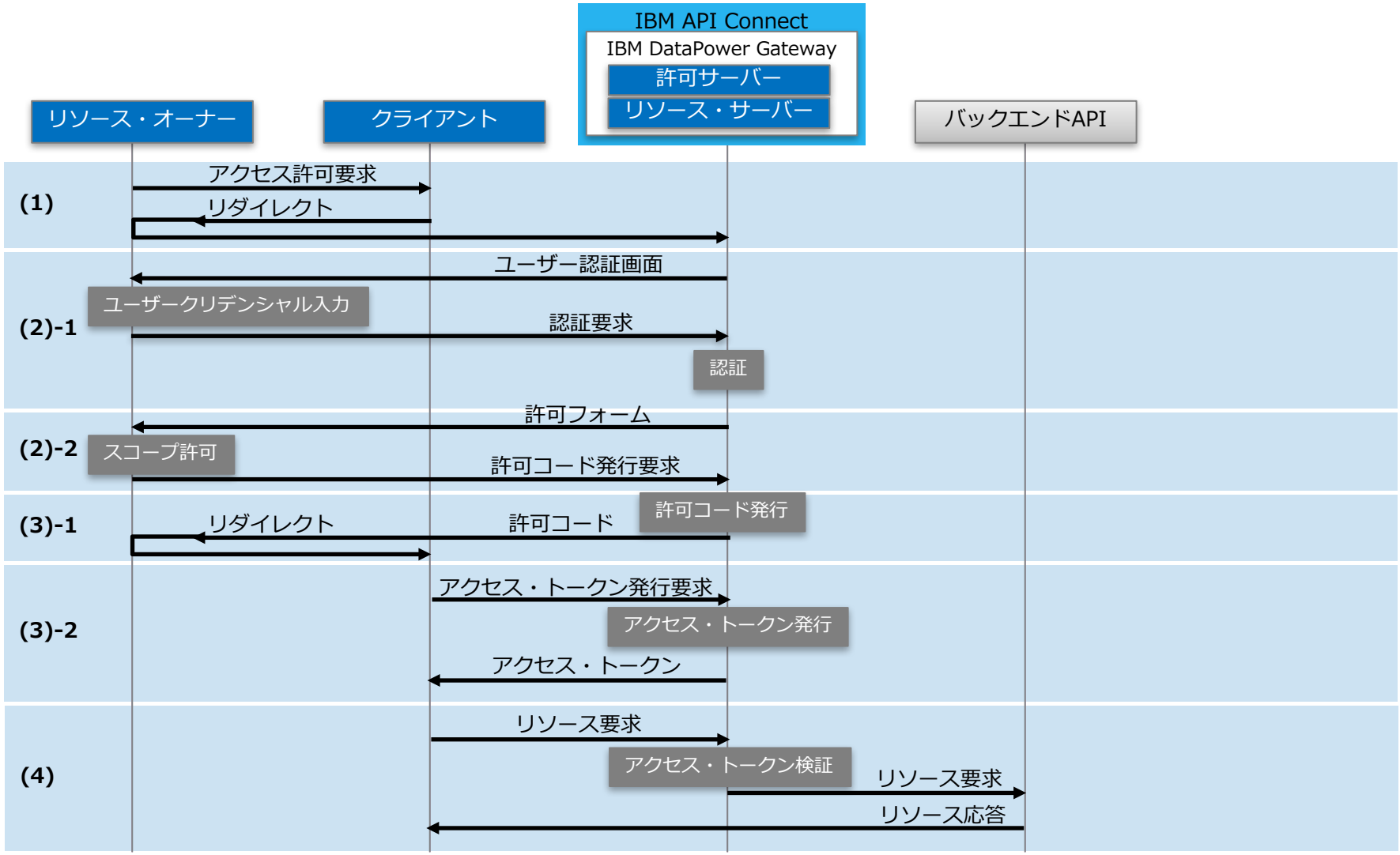
アクセストークンの失効管理 方法2

❑ 外部のサーバーを利用したトークン失効管理（失効URLの利用）

- トークン管理を外部サーバーで実装し、API Connectからのインターフェースを提供する方法
- トークン管理を行う外部サーバーのURLを許可サーバーの失効URLに指定
- API Connectのインターフェース仕様に合わせて、トークン管理を行うアプリケーションやデータベースを別途構築・開発する必要がある
- 後述の「構成例2. トークン失効管理」「2-2. 外部のサーバーを利用したトークン失効管理」にて、構成例を説明



API ConnectによるOAuth 2.0フロー概要 (認可コードグラントタイプ)



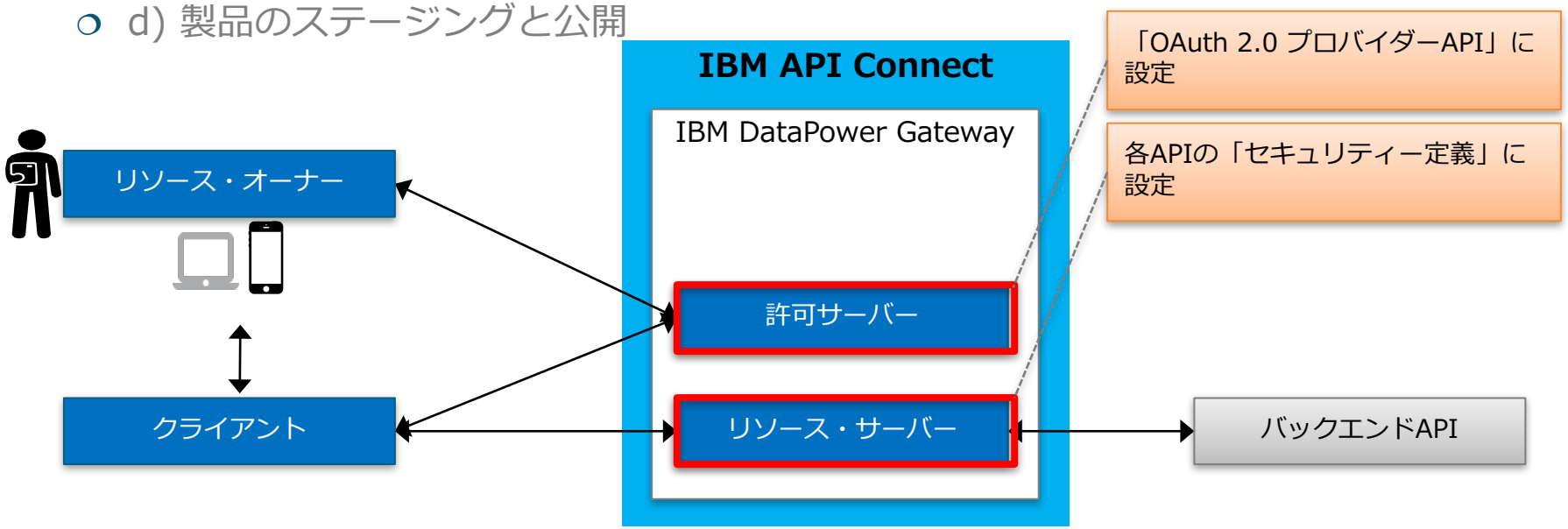
本図は厳密なフローとは異なる場合があります。

API ConnectにおけるOAuth設定

API ConnectにおけるOAuth構成概要

❑ 設定概要

- ❍ 「許可サーバー」の構成と「リソース・サーバー」の構成を行い、製品として公開する
  - a) 許可サーバーの構成
    - 「OAuth 2.0 プロバイダーAPI」に設定
  - b) リソース・サーバーの構成
    - 各APIの「セキュリティー定義」に設定
- ❍ c) 製品の作成
- ❍ d) 製品のステージングと公開



OAuthグラントタイプとAPI Connect定義の関係

❑ API ConnectでのOAuth2.0グラントタイプ

- OAuth2.0の4つのグラントタイプをサポート
- API Connect定義上の名前は、一般的なOAuth 2.0グラントタイプの名前と異なる（関係は以下の表のとおり）
- クライアント・タイプごとに利用可能なグラントタイプが異なる

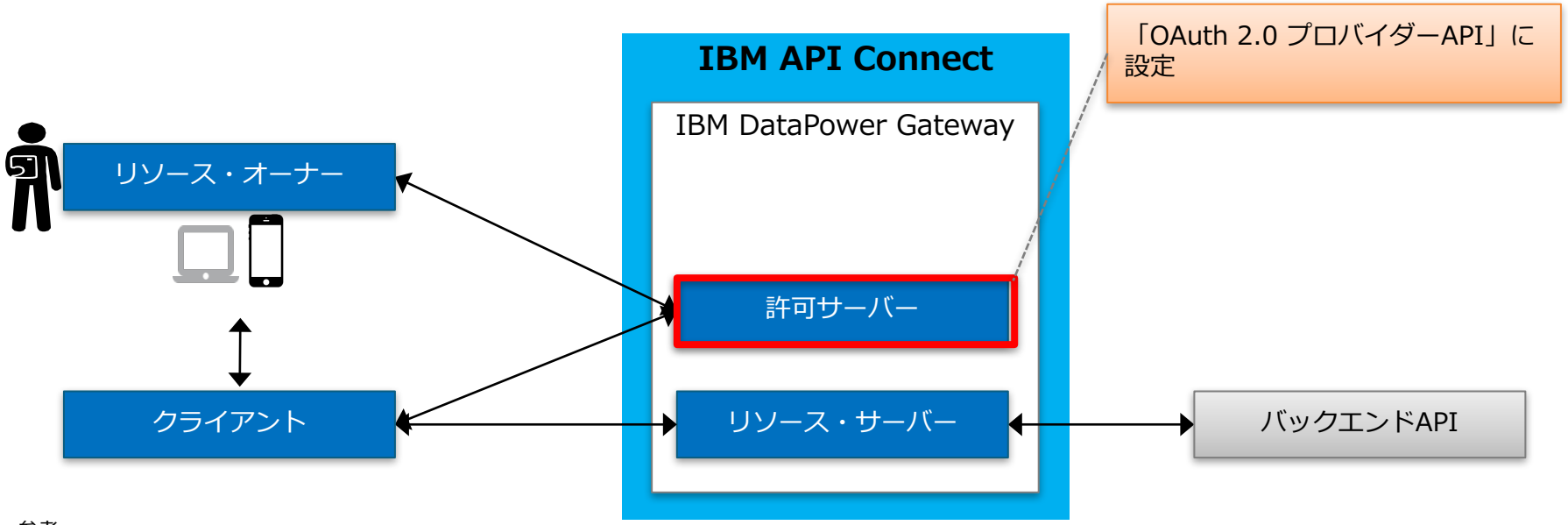
グラントタイプ		クライアント・タイプ	
API Connect OAuthセキュリティ定義	対応するOAuth 2.0 グラントタイプ	公開クライアント	機密クライアント
暗黙	インプリシット (Implicit)	○	
パスワード	リソースオーナーパスワードクレデンシャル (Resource owner password credential)	○	○
アプリケーション	クライアントクレデンシャル (Client credential)		○
アクセス・コード	認可コード (Authorization code)	○	○

- クライアント・タイプについて
  - 公開クライアント(Public client)
    - クライアント・クレデンシャルが漏洩する可能性のあるクライアント
    - ブラウザー上で実行されるJavaScriptアプリケーションやモバイルネイティブアプリケーションなど
  - 機密クライアント(Confidential client)
    - クライアント・クレデンシャルが漏洩する可能性が無いクライアント
    - サーバー・サイドで稼働するWebアプリケーションなど

a) 許可サーバーの構成

❑ 設定概要

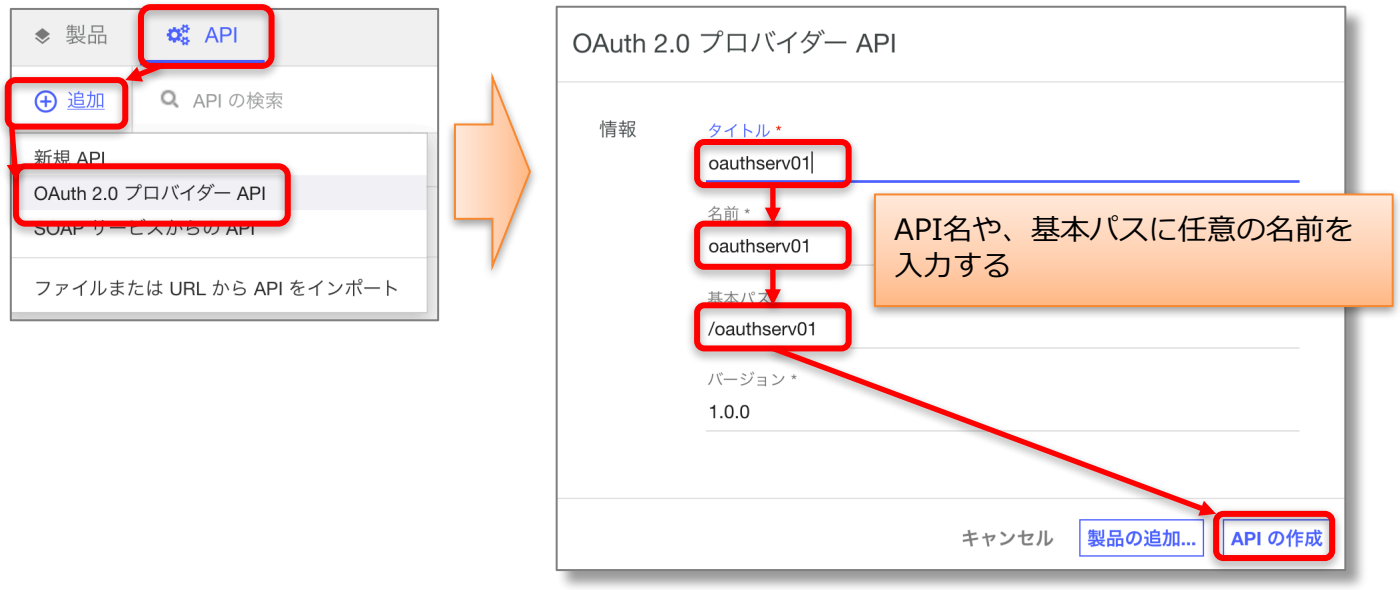
- OAuth許可サーバーとしての振る舞いを、 APIマネージャーまたはAPIデザイナーから「OAuth 2.0 プロバイダーAPI」に設定
  - a)-1. 「OAuth2.0プロバイダーAPI」の作成
  - a)-2. 「OAuth2.0プロバイダーAPI」の設定



<参考>
IBM API Connect Knowledge Center : OAuth プロバイダー API の作成
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/tapim_sec_api_config_scheme_oauth_endpoint.html

a) 許可サーバーの構成
a)-1. OAuth2.0プロバイダーAPIの作成

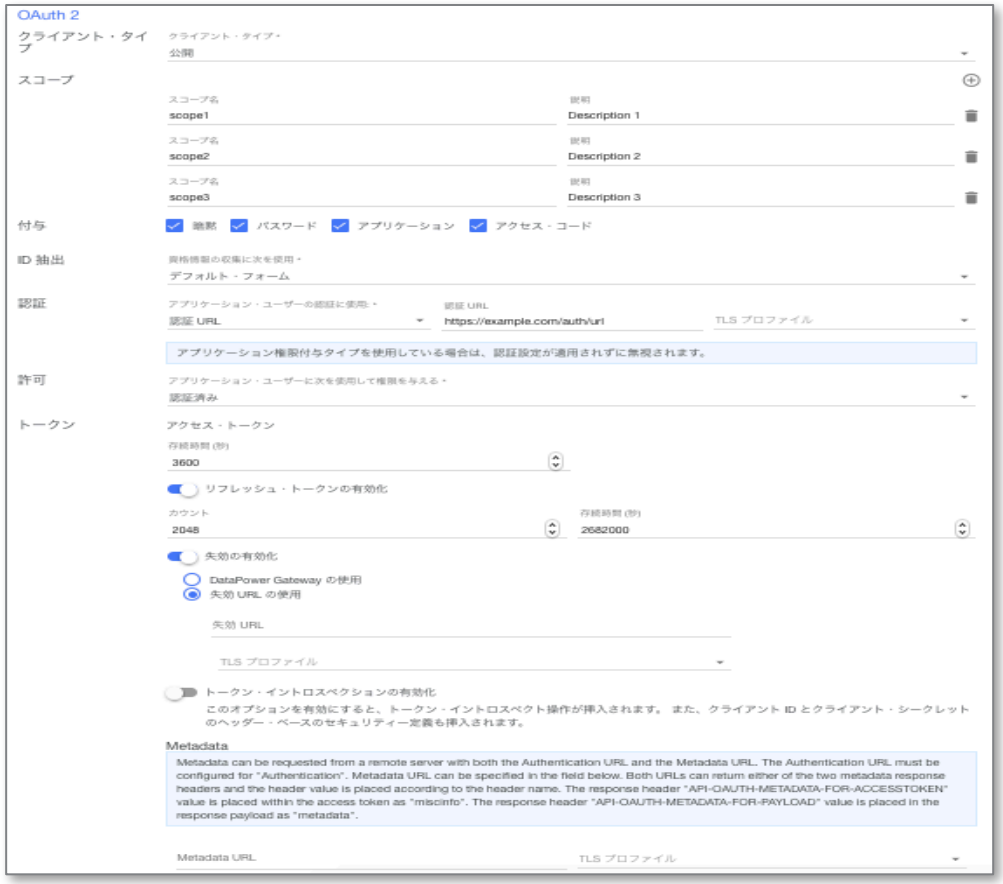
- 「API」タブの「+ 追加」ボタンから「OAuth 2.0 プロバイダーAPI」を選択し、基本パスなどの定義情報を入力してAPIを作成



a) 許可サーバーの構成
a)-2. OAuth2.0プロバイダーAPIの設定

- OAuth 2.0プロバイダーAPIの「設計」タブの「OAuth 2」セクションに設定して保存

※ 各設定項目の説明は後続の説明を参照



設定画面イメージ

a) 許可サーバーの構成
a)-2. OAuth2.0プロバイダーAPIの設定

- OAuth2.0プロバイダーAPIを作成すると、許可サーバーの許可エンドポイント、トークン・エンドポイントのパスが自動的に生成される。
- 「 b) リソース・サーバーの構成」で設定するAPIの「セキュリティ定義」の「許可 URL」と「トークン URL」にこのパスを指定する
  - 許可エンドポイント(/oauth2/authorize)：許可URLに指定
  - トークン・エンドポイント(/oauth2/token)：トークンURLに指定

許可エンドポイント（リソース・サーバーとなるAPIに指定する「許可URL」のパス）

トークン・エンドポイント（リソース・サーバーとなるAPIに指定する「トークンURL」のパス）

a) 許可サーバーの構成
a)-2. OAuth2.0プロバイダーAPIの設定

設定項目(1)

設定項目		説明
クライアント・タイプ		- クライアント・タイプを選択
	□公開(パブリック) □機密(コンフィデンシャル)	
スコープ		- スコープ名を指定
		- ここで指定したスコープ名は、各API定義の「セキュリティ定義」でOAuthを選択するときに指定
付与		- OAuth 2.0の4つのフロー(グラントタイプ)から選択
	□暗黙 □パスワード □アプリケーション □アクセス・コード	- 主に使用されるのは、「アクセス・コード（許可コード）」または「暗黙（インプリシット）」 - クライアント・タイプごとに選択可能な「付与（グラントタイプ）」が異なる - クライアント・タイプに「公開」を選択する場合、「アプリケーション（クライアントクレデンシャル）」は選択不可 - クライアント・タイプに「機密」を選択する場合、「暗黙（インプリシット）」は選択不可 - OAuth 2.0仕様のグラントタイプとの紐付けは、P.22の「OAuthグラントタイプとAPI Connect定義の関係」を参照

a) 許可サーバーの構成
a)-2. OAuth2.0プロバイダーAPIの設定

設定項目(2)

設定項目			説明
ID抽出			- **認証のためのユーザークレデンシャルの抽出方法の選択**
	□デフォルト・フォーム		- デフォルトのフォーム認証を利用する場合に選択
	□基本		- 基本認証を利用する場合に選択
	□カスタム・フォーム		- カスタムHTMLフォームを利用する場合に選択 - カスタム・フォームはAPI Connectで必要なフィールド名などの要件を満たすフォームを作成し、フォームのURLを指定 - 通信にTLS通信を行う場合には、TLSプロファイルを指定する
		カスタム・フォーム	
		TLSプロファイル	
	□リダイレクト		- DataPower Gatewayで認証は行わず、外部の認証サービスを利用する場合に選択 - 「リダイレクト」を選択し、リダイレクト先のURLを指定 - API Connectで指定されるフローの仕様に沿って、外部認証サーバー側のカスタマイズを行う必要あり
		リダイレクトURL	

<参考>
IBM API Connect Knowledge Center：カスタム・サインインフォームの作成
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/task_apionprem_Create_a_custom_login_form.html
IBM API Connect Knowledge Center：リダイレクト URL を介した認証および許可
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/task_apionprem_redirect_form_.html

a) 許可サーバーの構成
a)-2. OAuth2.0プロバイダーAPIの設定

設定項目(3)

設定項目			説明
認証			- **認証のために使用するユーザーレジストリの選択**

	□ユーザー・レジストリー		- 「ID抽出」に「デフォルト・フォーム」「基本」「カスタム・フォーム」を選択した場合：
			  - LDAPサーバーを利用する場合は、「ユーザー・レジストリー」に、レジストリー名を指定
			  - APIマネージャーからLDAPサーバーをレジストリとして登録しておく必要あり
			- 「ID抽出」に「リダイレクト」を選択した場合には、指定できない

		ユーザー・レジストリー	

	□認証URL		- 「ID抽出」に「デフォルト・フォーム」「基本」「カスタム・フォーム」を選択した場合：
			  - 基本認証が構成された外部のサーバーで認証を行う場合には、「認証URL」に、基本認証が構成された外部のサーバーのURLを指定
			  - API Connectは取り出したユーザークレデンシャルをAuthorizationヘッダーに埋め込んで認証URLへGET送信し、200 OK応答を受け取ることで認証成功とする
			  - 外部の認証サーバーとの通信にTLS通信を行う場合には、TLSプロファイルを指定する
			- 「ID抽出」に「リダイレクト」を選択した場合：
			  - リダイレクト先の外部認証サーバーにて認証後に、API Connectから認証確認（確認コードの検証）をするための外部認証サーバーのURLを指定する

		認証URL	
		TLSプロファイル	

<参考>
IBM API Connect Knowledge Center：認証 URL
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/con_auth_url.html

a) 許可サーバーの設定方法
a)-2. OAuth2.0プロバイダーAPIの設定

設定項目(4)

設定項目		説明
認可		- スコープの許可を行うためのフォームの提供方法を選択
	□デフォルト・フォーム	- API Connect によって提供されているデフォルト・フォームを使用するには、「デフォルト・フォーム」を選択
	□カスタム･フォーム	- カスタムHTMLフォームを利用する場合に選択 - カスタム・フォームはAPI Connectで必要なフィールド名などの要件を満たすフォームを作成し、フォームのURLを指定 - 通信にTLS通信を行う場合には、TLSプロファイルを指定する
	カスタム・フォーム	
	TLSプロファイル	
	□認証済み	- 認証が完了すれば、スコープの許可が行われたものとして、許可を自動的に与える場合に選択 - リダイレクトURL を使用してユーザーを認証および許可する場合は、「認証済み」を選択する必要あり

<参考>
IBM API Connect Knowledge Center :カスタム許可フォームの作成
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/task_apionprem_create_a_custom_authorization_form.html

a) 許可サーバーの構成
a)-2. OAuth2.0プロバイダーAPIの設定

設定項目(5)

設定項目			説明
トークン			-アクセス・トークン、リフレッシュ・トークンに関する設定を行う
	アクセス・トークン		
		存続時間(秒)	- アクセストークンの存続時間を指定(デフォルト3600秒、最小1秒、最大63244800秒(=2年))
	リフレッシュ・トークンの有効化		- リフレッシュ・トークンを使用する場合に有効にする
		カウント	- リフレッシュ・トークンを要求できる回数（デフォルト2048回、最大 4096回) - カウント数だけリフレッシュトークンを発行すると、それ以上リフレッシュ・トークンによるアクセストークンは再発行ができなくなる - カウントに達した際は、あらためて、スコープの承認から、許可コード、アクセストークンを発行する
		存続時間(秒)	- リフレッシュトークンが有効である秒数 (デフォルト2682000秒(約31日)、最小2秒、最大252979200秒(=8年))

a) 許可サーバーの構成
a)-2. OAuth2.0プロバイダーAPIの設定

設定項目(6)

設定項目			説明
トークン			-アクセス・トークン、リフレッシュ・トークンに関する設定を行う
	失効の有効化		- アクセス・トークンの失効を行う場合に有効にする
		□DataPower Gatewayの使用	- 「ユーザーに許可の表示と取り消しを許可する」を有効にする - DataPower Gateway上でトークン失効管理をする設定 - DataPower上で、Quota Enforcement Serverの設定を有効にする必要あり - トークンの発行状況の表示と失効(削除)のインターフェースが提供される - 以下の2つのパスが有効となる GET /oauth2/issued DELETE /oauth2/issued
		□失効URLの使用	- トークン管理を外部サーバーで実装し、API Connectからのインターフェースを提供する方法 - トークン管理を行う外部サーバーのURLを指定 - 以下のタイミングで、失効URLに対してリクエストメッセージが送信され、トークン管理やトークンの妥当性チェックを実施 - アクセス・トークン発行時 - サービスAPI呼び出し時 - リフレッシュトークン発行時　など
	トークン・イントロスペクションの有効化		- 有効にすると、アクセス・トークンのスコープや有効性など、発行済のアクセス・トークンに関する情報を取得するインターフェースが提供される

<参考>
IBM API Connect Knowledge Center：OAuth 失効 URL
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/ref_oauth_sampleurl.html

a) 許可サーバーの構成
a)-2. OAuth2.0プロバイダーAPIの設定

設定項目(7)

設定項目			説明
トークン			-アクセス・トークン、リフレッシュ・トークンに関する設定を行う
	Metadata		OAuthメタデータに関する設定

		Metadata URL	- アクセストークンまたはアクセストークンをクライアントに発行するときの応答ペイロードに独自のメタデータを含めたい場合で、Metadata URLを利用してメタデータの情報を追加したい場合に指定する
			- メタデータの情報をヘッダーに格納して応答を返すサービスのURLを指定する
			- Metadata URLからは以下のHTTPヘッダーを応答に含めることが可能
			  - API-OAUTH-METADATA-FOR-PAYLOAD
			  - API-OAUTH-METADATA-FOR-ACCESSTOKEN

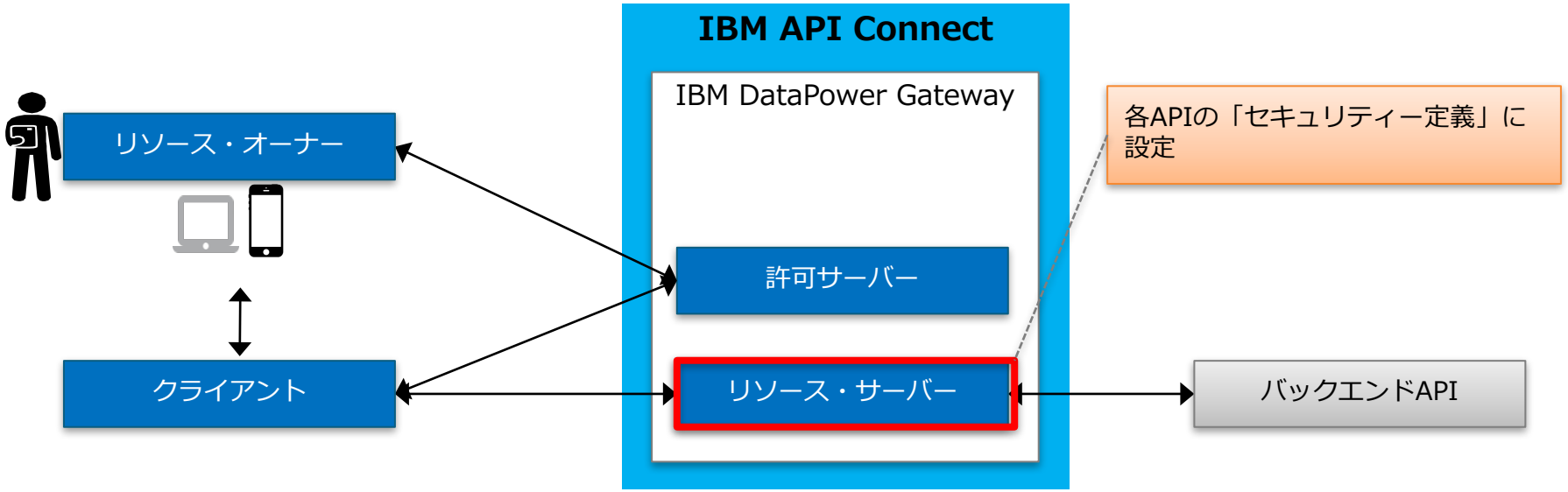
		TLSプロファイル	- Metadata URLとの通信にTLS通信を行う場合には、TLSプロファイルを指定する

<参考>
IBM API Connect Knowledge Center : OAuth メタデータ
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/con_metadata.html

b) リソース・サーバーの構成

❑ 設定概要

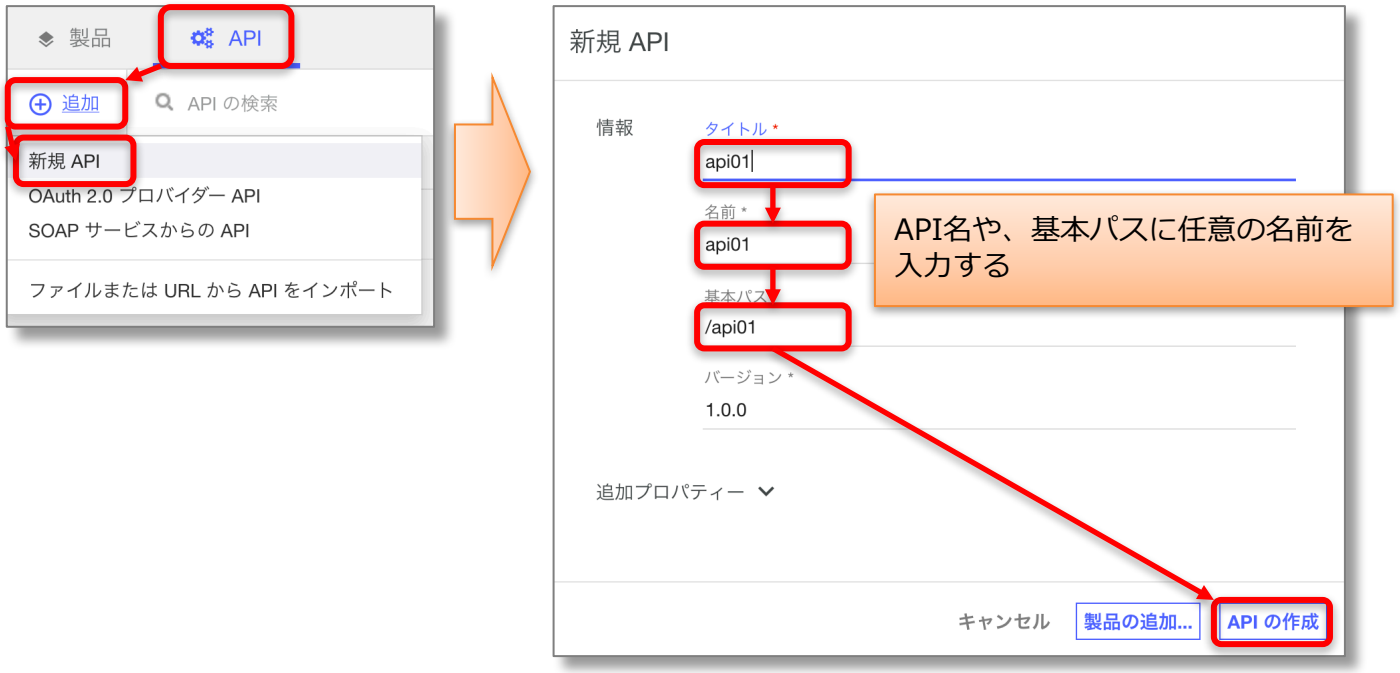
- OAuthリソースサーバーの設定を、 APIマネージャーまたはAPIデザイナーから、APIの「セキュリティー定義」に設定
  - b)-1. APIの作成
  - b)-2. APIの設定
  - b)-3. OAuth用の「セキュリティー定義」設定
  - b)-4. 製品の作成
  - b)-5. 製品にAPIを追加
  - b)-6. 製品のステージングと公開



b) リソース・サーバーの構成
b)-1. APIの作成

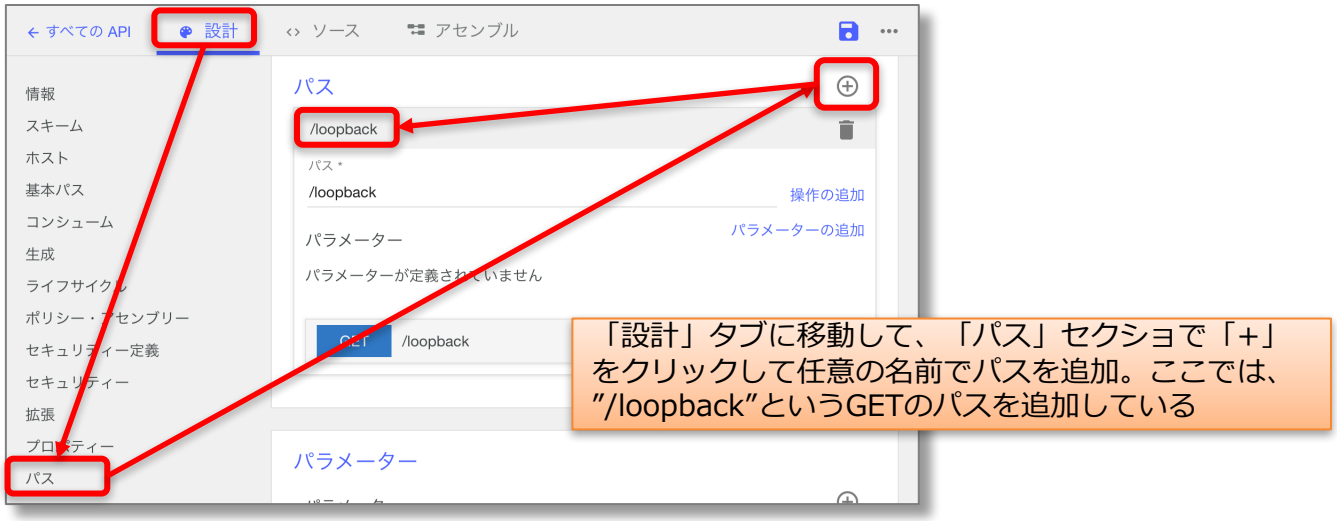
- 「API」タブの「+ 追加」ボタンからAPIを追加し、基本パスなどの定義情報を入力してAPIを作成

※必要に応じて、「設計」タブや「アセンブル」タブでAPIの設定を行う



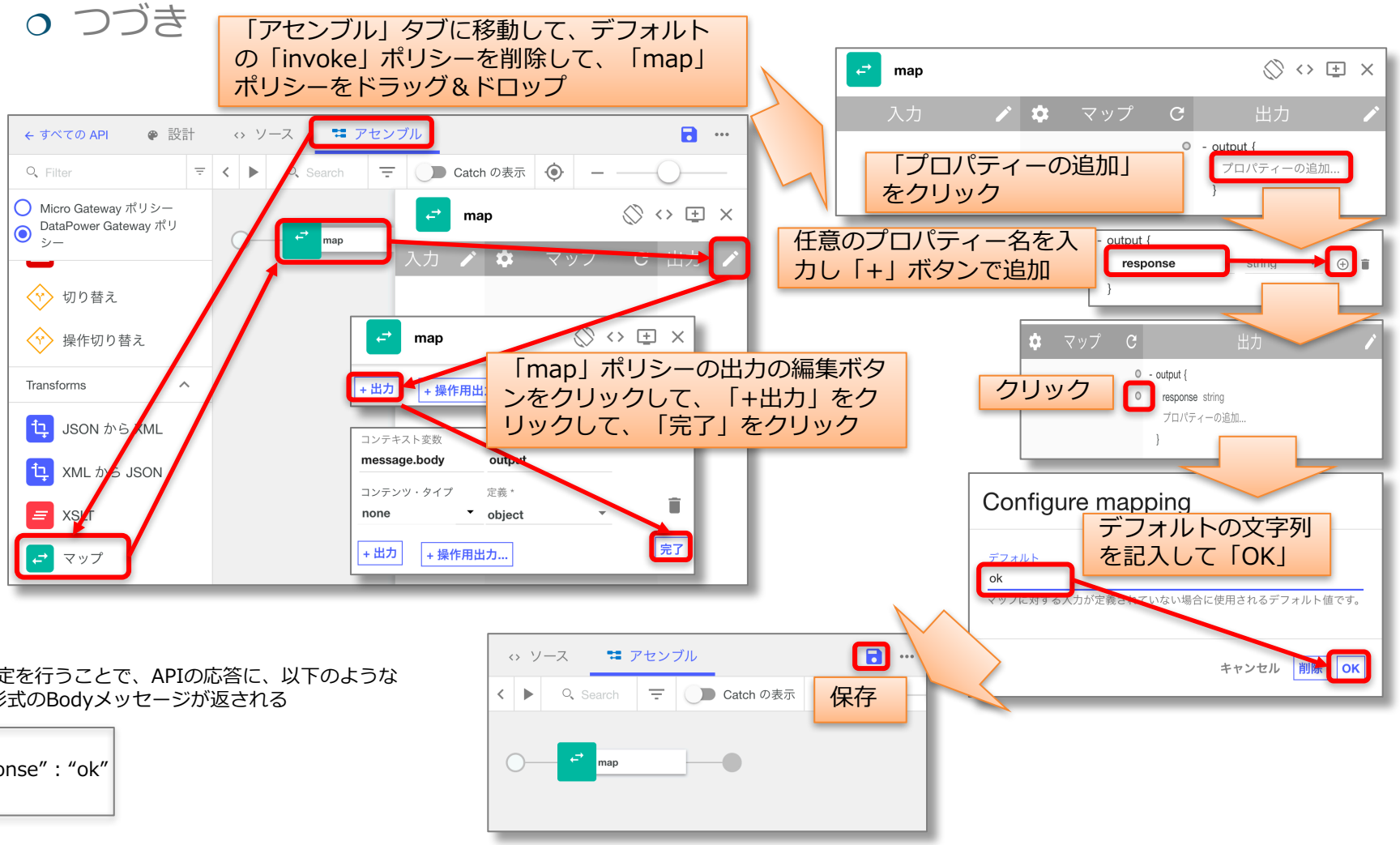
b) リソース・サーバーの構成
b)-2. APIの設定

- 必要に応じて、「設計」タブや「アセンブル」タブでAPIの設定を行う。API Connectのゲートウェイで折り返すサンプルAPIを定義するには、以下のように設定する。



b) リソース・サーバーの構成
b)-2. APIの設定

- つづき



※
この設定を行うことで、APIの応答に、以下のようなJSON形式のBodyメッセージが返される

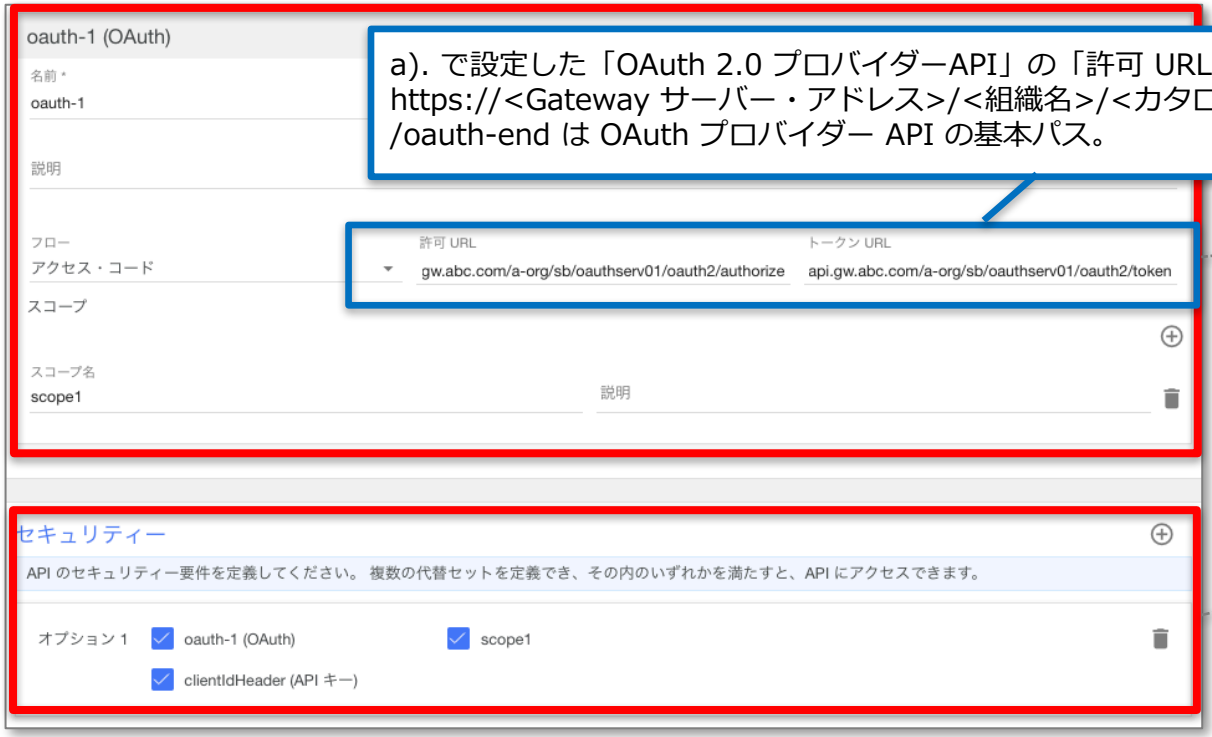
```
{
“response” : “ok”
}
```

b) リソース・サーバーの構成
b)-3. OAuth用の「セキュリティー定義」設定

- APIの「設計」タブの「セキュリティー定義」セクションに「OAuth」セキュリティー定義を追加し、「セキュリティー」セクションでチェックを追加して保存

※ 各設定項目の説明は後続の説明を参照

設定画面イメージ



a). で設定した「OAuth 2.0 プロバイダーAPI」の「許可 URL」 と「トークン URL」 のパスの前に、https://<Gateway サーバー・アドレス>/<組織名>/<カタログ名>/oauth-end を付加したもの。/oauth-end は OAuth プロバイダー API の基本パス。

「セキュリティー定義」セクションにOAuthのセキュリティ定義を追加する

リソース・サーバーとしての振る舞いを、利用するOAuth許可サーバーに合わせて設定する

「セキュリティー」セクションで追加したOAuthのセキュリティー定義にチェックを入れる

b) リソース・サーバーの構成
b)-3. OAuth用の「セキュリティー定義」設定

設定項目

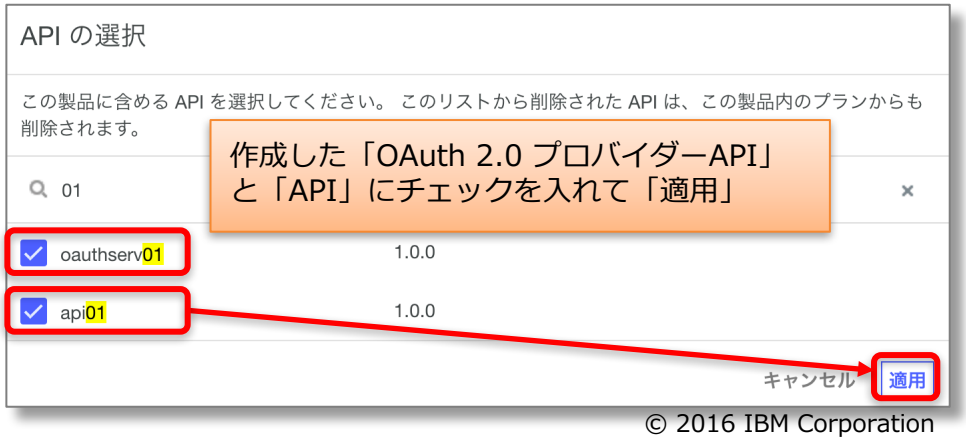
設定項目		説明
フロー	□暗黙 □パスワード □アプリケーション □アクセス・コード	- 使用する OAuth セキュリティー定義のグラントタイプを選択 - OAuth 2.0仕様のグラントタイプとの紐付けは、P.22の「OAuthグラントタイプとAPI Connect定義の関係」を参照
許可URL		- 「アクセス・コード」および「暗黙フロー」の場合: 「許可 URL」フィールドで、OAuth プロバイダー API の許可パスの URL を指定 - 許可パスは以下を指定 [Gatewayのアドレス]/[組織のパス・セグメント]/[カタログのパス・セグメント]/[Oauth プロバイダーAPIの基本パス]/oauth2/authorize
トークンURL		- 「パスワード」「アクセス・コード」および「アプリケーション」のフローの場合：「トークン URL」フィールドで、OAuth プロバイダー API のトークン・パスの URL を指定 - トークン・パスは以下を指定 [Gatewayのアドレス]/[組織のパス・セグメント]/[カタログのパス・セグメント]/[OAuth プロバイダーAPIの基本パス]/oauth2/token
スコープ		- スコープ名を指定する

b) リソース・サーバーの構成
b)-4. 製品の作成

- 「製品」タブの「+ 追加」ボタンから製品を追加し、タイトルなどの定義情報を入力して製品を作成

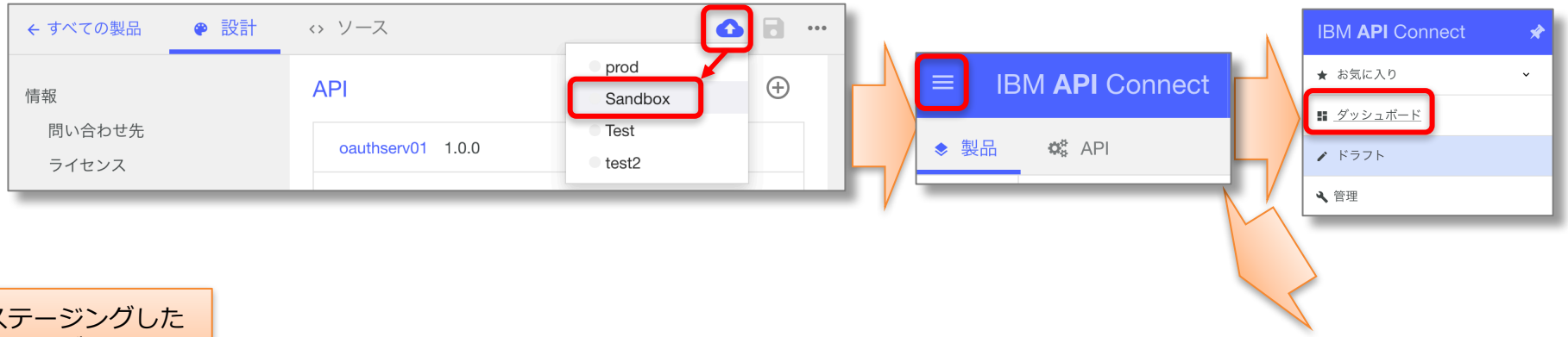
b) リソース・サーバーの構成
b)-5. 製品にAPIを追加

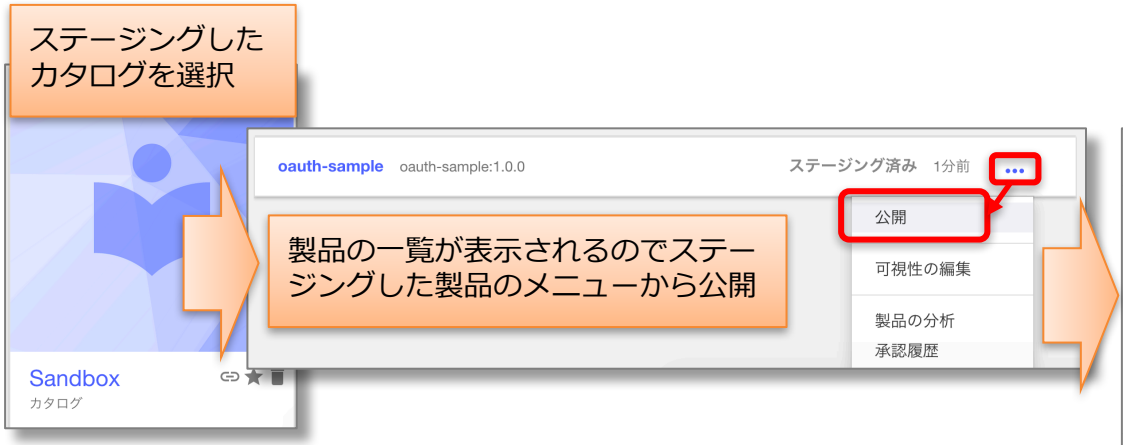
- 「設計」タブの「API」セクションにて、作成した「OAuth 2.0 プロバイダーAPI」と「API」を製品に追加して保存



b) リソース・サーバーの構成
b)-6. 製品のステージングと公開

- 製品の「設計」タブの右上のボタンをクリックして、ステージング先のカタログを選択してステージングを行い、ステージング先のカタログで製品を公開する





構成例

1. 外部認証サーバーを利用したOAuth認証
- 1-1. 外部認証サーバーとの連携によりAPI Connectで認証を行う構成
- 1-2. リダイレクトにより外部認証サーバーで認証を行う構成
- 1-Tips. OAuth認証とログのカスタマイズ

2. トークン失効管理
- 2-1. DataPower Gatewayを利用したトークン失効管理
- 2-2. 外部のサーバーを利用したトークン失効管理

3. DataPower Gatewayでのフロント処理のカスタマイズ追加

構成例1. 外部認証サーバーを利用したOAuth認証

1-1. 外部認証サーバーとの連携によりAPI Connectで認証を行う構成

1-2. リダイレクトにより外部認証サーバーで認証を行う構成

- この構成を実現する動機と実現方法
- 実装概要図
- トランザクション・フロー
- API Connect 設定のポイント

1-Tips. OAuth認証とログのカスタマイズ

1-1. 外部認証サーバーとの連携によりAPI Connectで認証を行う構成

❑ この構成を実現する動機

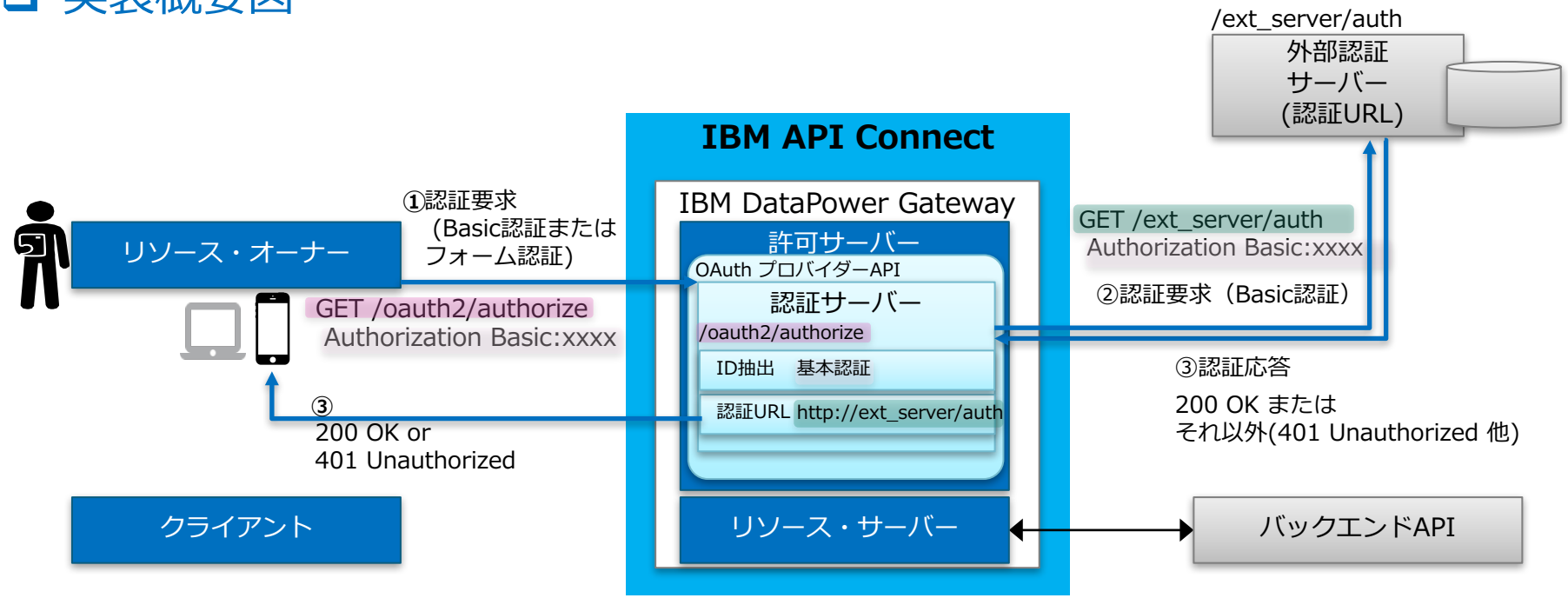
- OAuthの実装で製品提供の機能を利用した認証画面やリソース許可画面を利用するが、外部認証サーバーと連携して認証する
  - 既存のユーザー・リポジトリと認証の仕組みを使用したい
  - 認証画面やリソース許可画面は、製品機能（または製品機能のカスタマイズ）を使用したい

❑ IBM API Connect での実現方法

- API Connectの許可サーバーの「OAuth 2.0プロバイダーAPI」設定で対応
  - 「ID抽出」項目で、認証方法 (基本、デフォルト・フォーム、カスタム・フォームのいずれか）を選択
  - 「認証」項目で、「認証URL」を選択し、外部認証サーバーのURLを指定
- 外部認証サーバーでは、Basic認証を構成
  - リソース・オーナーが認証要求時に指定したID、パスワードは、API Connectから外部認証サーバーへ、Basic認証に置き換えられて送信
  - 外部認証サーバーは、認証OKの場合は、HTTP 200を戻し、認証エラーの場合はそれ以外のコードを戻す
- API Connectでは、認証URLでHTTP 200が戻されれば認証OKとし処理を継続し、それ以外の場合は認証エラーと応答

1-1. 外部認証サーバーとの連携によりAPI Connectで認証を行う構成

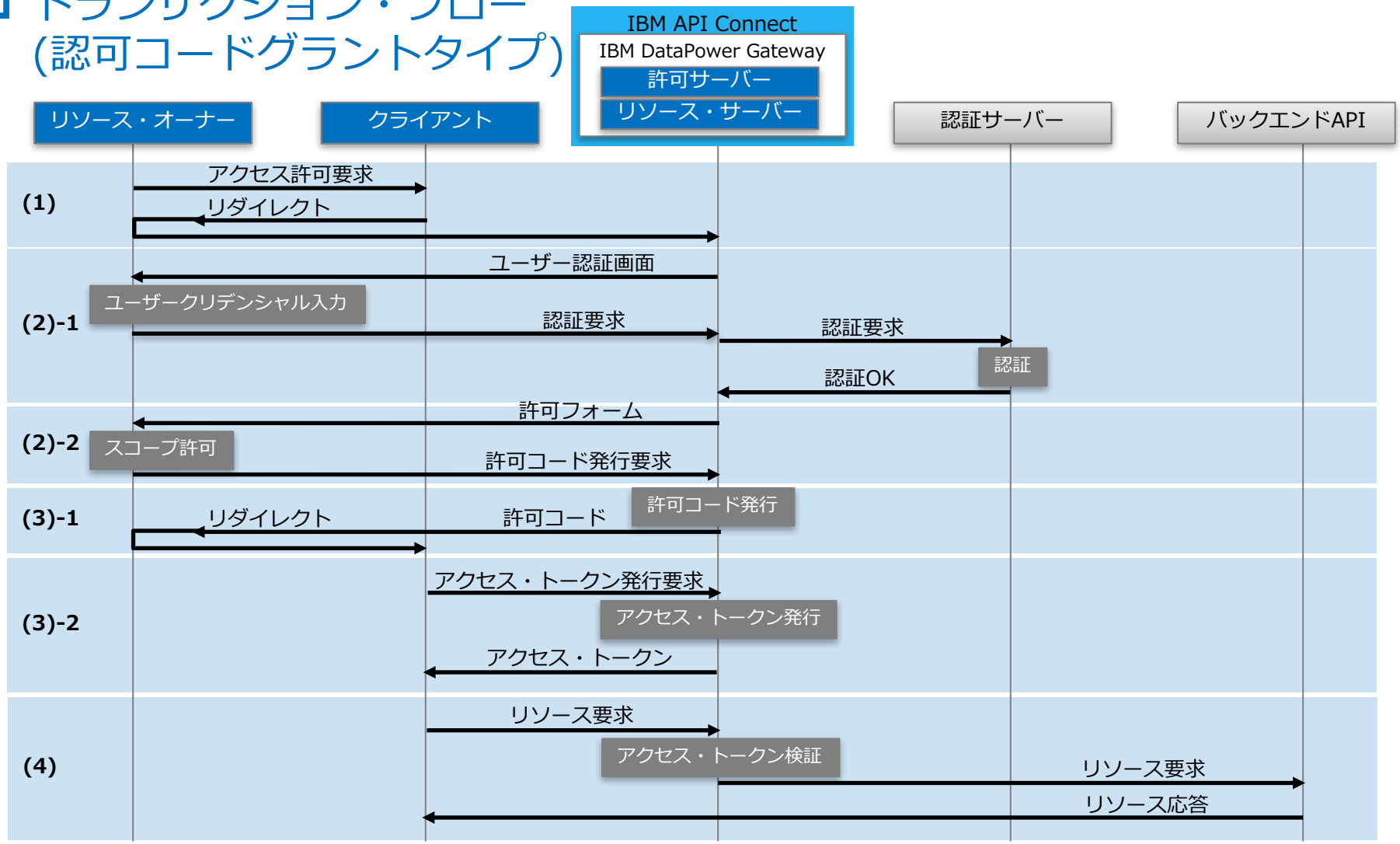
❑ 実装概要図



P.47 トランザクション・フローの(2)-1の認証部分の処理概要
① リソース・オーナーからの認証要求
② 認証URL(外部認証サーバー)に対して、受信したユーザーID、パスワードでBasic認証を実行
③ 認証が成功すれば、正常応答(200 OK)が戻され、リクエスターにも正常応答(200 OK)が戻される。認証エラーになった場合は、認証エラー応答(401 Unauthorized)が戻される

1-1. 外部認証サーバーとの連携によりAPI Connectで認証を行う構成

❑ トランザクション・フロー (認可コードグラントタイプ)



本図は厳密なフローとは異なる場合があります。

1-1. 外部認証サーバーとの連携によりAPI Connectで認証を行う構成

❑ API Connect 設定のポイント

❍ API Connectの「OAuth 2.0 プロバイダーAPI」に以下を設定

項目	設定値	備考
ID抽出	「基本」「デフォルト・フォーム」「カスタム・フォーム」のいずれかを選択	
認証	「認証URL」を選択し、認証URLを指定	- 「認証URL」に、基本認証が構成された外部のサーバーのURLを指定 - API Connectは取り出したユーザークレデンシャルをAuthorizationヘッダーに埋め込んで認証URLへGET送信し、200 OK応答を受け取ることで認証成功とする
認可	「デフォルト・フォーム」「カスタム・フォーム」「認証済み」のいずれかを選択	- 「デフォルト・フォーム」「カスタム・フォーム」を選択した場合：認証URLからの応答で認証OKとなった場合に、スコープの承認画面（デフォルト・フォームまたはカスタム・フォーム）が表示され、リソース・オーナーがスコープを承認することで許可コードが発行される - 「認証済み」を選択した場合：認証が完了すれば、スコープ承認されたものとして処理され、許可コードが発行される

設定例

構成例1. 外部認証サーバーを利用したOAuth認証

1-1. 外部認証サーバーとの連携によりAPI Connectで認証を行う構成

1-2. リダイレクトにより外部認証サーバーで認証を行う構成

- この構成を実現する動機と実現方法
- 実装概要図
- トランザクション・フロー
- API Connect 設定のポイント

1-Tips. OAuth認証とログのカスタマイズ

1-2. リダイレクトにより外部認証サーバーで認証を行う構成

❑ この構成を実現する動機

- リソース・オーナーの認証とリソースに対するアクセス許可を独自に実装したい。
  - 既存のユーザー・リポジトリを使用したい
  - 独自のユーザー・インターフェース (画面デザインや遷移) を使用したい
  - 独自の認証・許可ロジックを使用したい

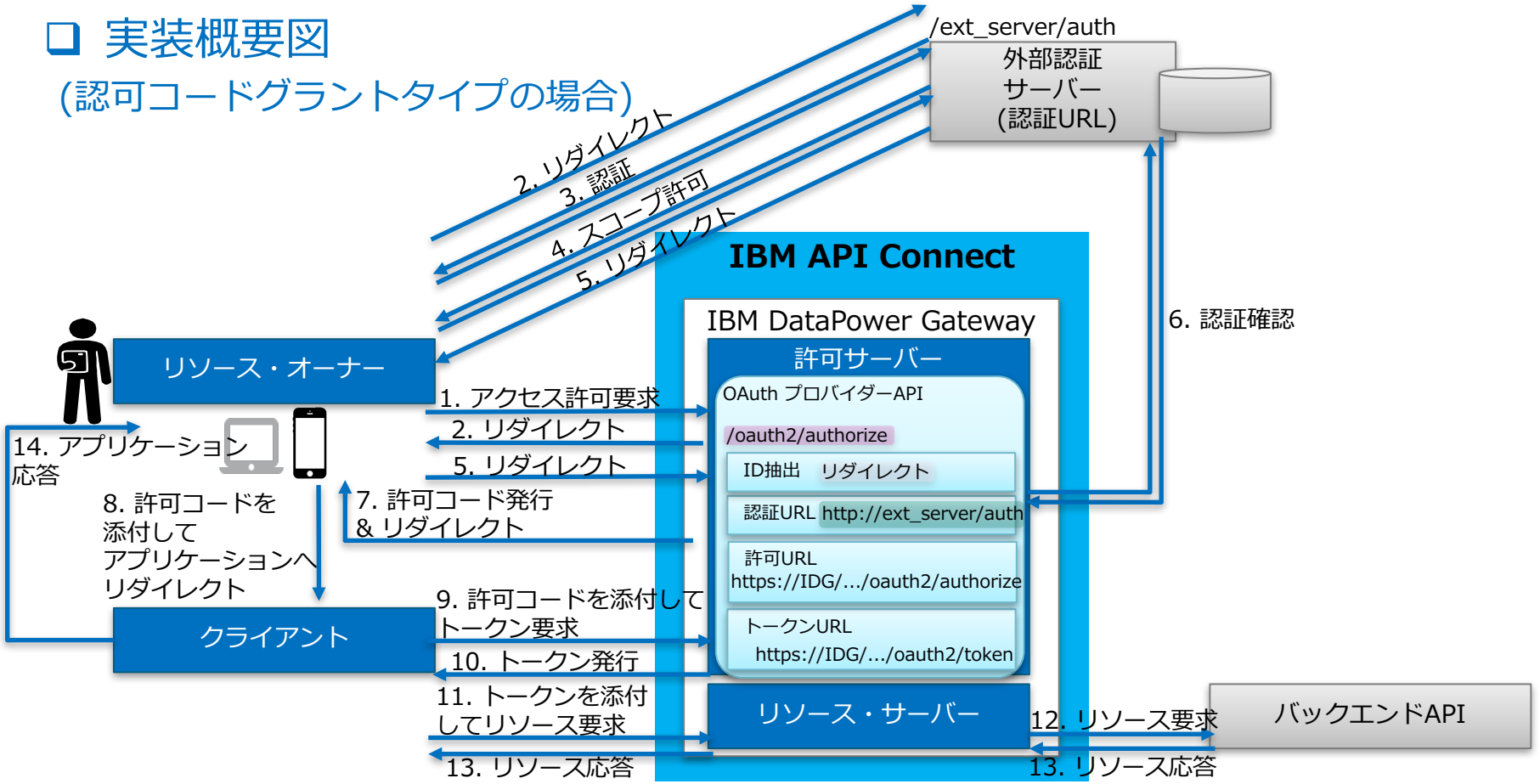
❑ IBM API Connect での実現方法

- API呼び出し時のユーザー認証に外部認証サーバーを使用する
  - この外部認証サーバーを独自に実装する。
  - API Connect 「OAuth 2.0 プロバイダー API」で、「ID 抽出」に「リダイレクト」を選択する。
- クライアントからのAPI呼び出しに先だって、外部認証サーバーへリダイレクトさせて認証ページを表示させる。
  - API Connect 「OAuth 2.0 プロバイダー API」で、「リダイレクト URL」に認証情報収集 (認証ページ表示) と認証を実施する URL を設定する。
- 認証ページでリソース・オーナーに認証情報を入力してもらうことで認証を実施し、続いてスコープに対する認可を実施する。
  - API Connect 「OAuth 2.0 プロバイダー API」で「認証」に「認証 URL」を選択して、リダイレクト先から認証済みユーザーと確認コードを受け取って認証確認を行う API の URL を設定する。

1-2. リダイレクトにより外部認証サーバーで認証を行う構成

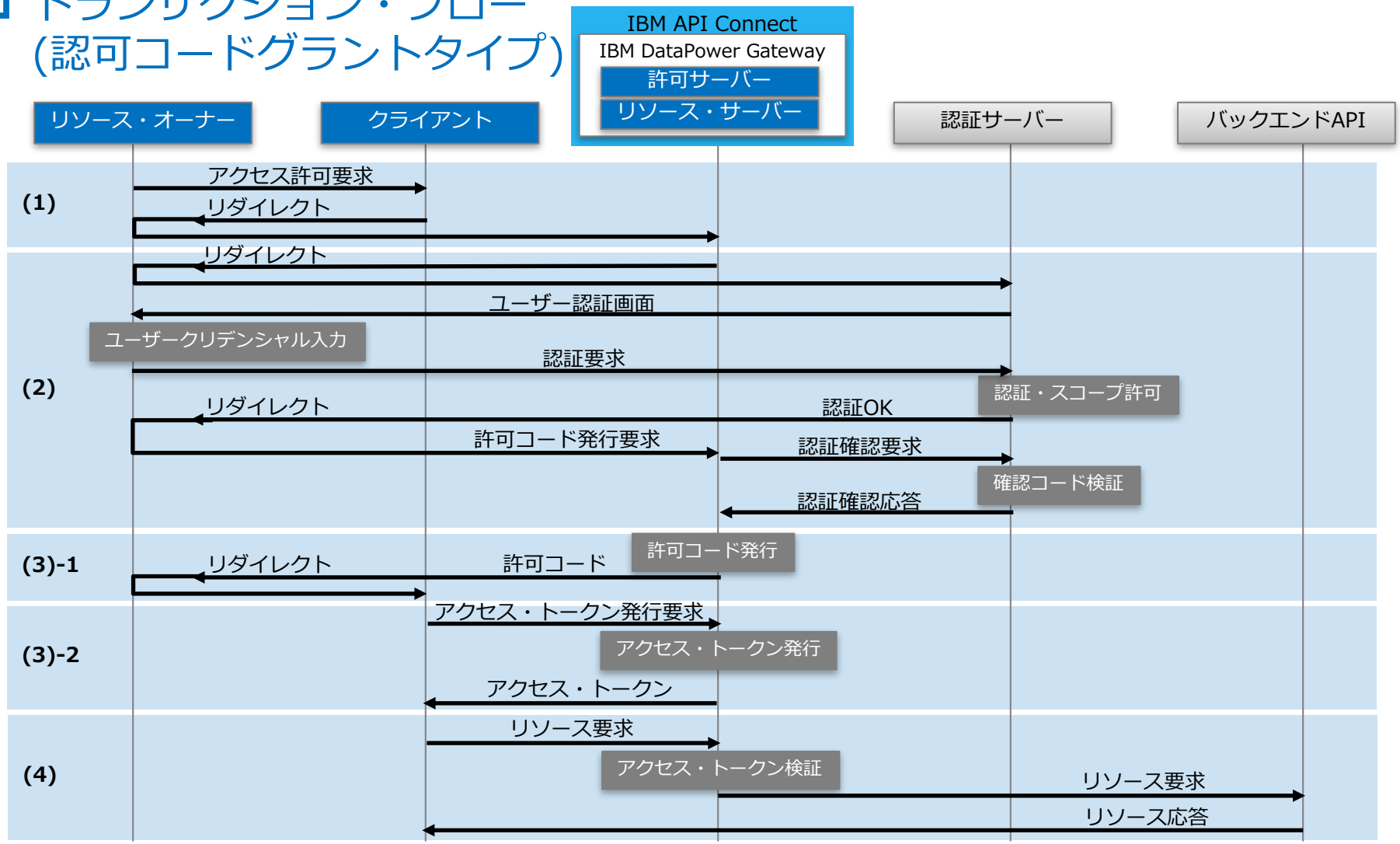
❑ 実装概要図

(認可コードグラントタイプの場合)



1-2. リダイレクトにより外部認証サーバーで認証を行う構成

❑ トランザクション・フロー (認可コードグラントタイプ)



本図は厳密なフローとは異なる場合があります。

1-2. リダイレクトにより外部認証サーバーで認証を行う構成

❑ API Connect 設定のポイント

❍ API Connectの「OAuth 2.0 プロバイダーAPI」に以下を設定

項目	設定値	備考
ID抽出	「リダイレクト」を選択し、リダイレクト先のURLを指定	- API Connectで指定されるフローの仕様に沿って、外部認証サーバー側のカスタマイズを行う必要あり
認証	「認証URL」を選択し、認証URLを指定	- リダイレクト先の外部認証サーバーにて認証後に、API Connectから認証確認（確認コードの検証）をするための外部認証サーバーのURLを指定する
認可	「認証済み」を選択	- 外部認証サーバーでユーザー認証とスコープ許可の両方を実施する - API Connect では認証済みリクエストは許可もされているものとみなす

1-2. リダイレクトにより外部認証サーバーで認証を行う構成サンプルコード

A. ユーザー認証と確認コード検証 サンプル・コード

B. OAuth クライアント・アプリケーション サンプル・コード

A. ユーザー認証と確認コード検証 サンプル・コード

今回はサンプルのため、ユーザー認証画面、認証、ユーザー名と確認コードの検証を同一アプリケーションの別パスに実装しています。3つの処理をそれぞれ別アプリケーション、別サーバーに配置することも可能です。

routes/redirect.js (抜粋)

```
var express = require('express');
var url = require('url');
var http = require('http');
var router = express.Router();

ユーザー認証画面
router.get('/', function(req, res) {
  var query = url.parse(req.url).query;
  res.render('redirect/index', {query: query});
})

認証
router.get('/collect_form', function(req, res) {
  var req1 = url.parse(req.url, true).query;
  var request_url_encoded = req1.query.slice(13);
// ここで認証処理を行う。今回はテストのため、任意のユーザー名とパスワードを受け入れる (= 検査しない)。
  var request_url = decodeURIComponent(request_url_encoded) + '&username=' + req1.user_name + '&confirmation=1234';
// 今回はテストのため、確認コードは固定値 1234 を使用。
  res.redirect(request_url);
})

ユーザー名と確認コードの検証
router.get('/verify', function(req, res) {
  var req_encoded_credential = req.get('Authorization').slice(6);
  req_decoded_credential = new Buffer(req_encoded_credential, 'base64').toString();
  var element = req_decoded_credential.split(":");
if (element[0] == 'Bob Garcia' && element[1] == '1234') {
    res.status(200).send('Verification succeeded.');
  } else {
    res.status(401).send('Verification failed.');
  }
 })

module.exports = router;
```

リクエストからクエリー・パラメーターを抽出し、そのパラメーターを付加して認証データ取得画面へ遷移させる。

認証データ取得画面からの POST リクエストを受け付けて認証を行う。認証 OK ならば、'/' へのリクエストに付加されていたクエリー・パラメーターに、認証データ取得画面で入力されたユーザー名と、任意に採番した確認コードを追加して original-url へ遷移させる。

認証データ取得画面で入力されたユーザー名と、確認コードの検証を行う。今回はユーザー名、確認コードとも固定値を使用。また今回は許可ロジックを実装せずに応答コード 200 を返しているため、認証済み=許可済み となる。

B. OAuth クライアント・アプリケーション サンプル・コード

```
var express = require('express');
var router = express.Router();
var https = require('https');
var url = require('url');

router.get('/', function(req, res, next) {
var code = url.parse(req.url, true).query.code;
customAgent = new https.Agent({ rejectUnauthorized : false });
  var token_request_options = {
    host : '9.188.124.xx',
    path : '/org/sb/oauth-end/oauth2/token?grant_type=authorization_code&redirect_uri=http://9.188.124.65:3001/&code=' + code,
    method : 'POST',
    agent : customAgent
  }
  var token_request = https.request(token_request_options, function(token_response) {
    var token_data = '';
    var token = '';

    token_response.on('data', function(chunk) {
      token_data += chunk;
    });

    token_response.on('end', function() {
      token = JSON.parse(token_data).access_token;
      var api_request_options = {
        host : '9.188.124.xx',
        path : '/org/sb/branches-onprem/details',
        method : 'GET',
        agent : customAgent
      }

      var api_request = https.request(api_request_options, function(api_response) {
        var api_data = '';
        api_response.on('data', function(chunk) {
          api_data += chunk;
        });
        api_response.on('end', function() {
          console.log('api_data = ' + api_data);
          res.render('index', { title: 'Express', api_data: api_data });
        });
      });
      api_request.setHeader('Authorization', 'Bearer ' + token);
      api_request.setHeader('X-IBM-Client-Id', '68c15e49-2af8-4feb-a58a-e35d5069cf25');
      api_request.end();
    });
  });

  token_request.setHeader('Authorization', 'Basic ' + new Buffer('68c15e49-2af8-4feb-a58a-
e35d5069cf25:M2iF2yV3dV1kG3fT1iW6vU6hN6eY5mR1jB6kC4tH8tI8pA1mU6').toString('base64'));
  token_request.end();
});

module.exports = router;
```

クエリ・パラメーターで受け取った許可コードを添付して、トークン URL へアクセス・トークンを要求する。

トークン URL からの応答で受け取ったアクセス・トークンを添付して、API エンドポイントへリクエストを発行する。

構成例1. 外部認証サーバーを利用したOAuth認証

1-1. 外部認証サーバーとの連携によりAPI Connectで認証を行う構成
1-2. リダイレクトにより外部認証サーバーで認証を行う構成

- この構成を実現する動機と実現方法
- 実装概要図
- トランザクション・フロー
- API Connect 設定のポイント

1-Tips. OAuth認証とログのカスタマイズ

1-Tips. OAuth認証とログのカスタマイズ

❑ OAuth認証時の応答によるアクセストークンのカスタマイズ

- 構成例1-1、1-2の両ケースにおいて、OAuthの認証でアクセストークン内の情報をカスタマイズ可能

- カスタマイズ可能な項目
  - ユーザー・クレデンシャル
  - OAuthメタデータ

- ユーザー・クレデンシャルのカスタマイズ
  - 認証時のユーザー・クレデンシャルを上書きし、アクセストークン内のリソース・オーナー名を変更可能
  - 「OAuth2.0プロバイダーAPI」に設定した認証URLからの応答ヘッダーに、以下のヘッダー名で値をセットすることで、カスタマイズ可能
    - ヘッダー名：API-Authenticated-Credential

    （ヘッダーが指定されないデフォルトの挙動は、認証時に使用されたユーザー名がそのままアクセス･トークン内のリソース・オーナー名としてセットされる）

1-Tips. OAuth認証とログのカスタマイズ

❑ OAuth認証時の応答によるアクセストークンのカスタマイズ

- OAuthメタデータのカスタマイズ
  - アクセストークンまたはアクセストークンをクライアントに発行するときの応答ペイロードに独自のメタデータを含めることが可能
  - 「OAuth2.0プロバイダーAPI」設定の認証URLまたはメタデータURLを設定して、認証URLまたはメタデータURLからの応答ヘッダーに以下のヘッダーを追加し値を設定することでカスタマイズ可能
  - ヘッダー名
    - API-OAUTH-METADATA-FOR-ACCESSTOKEN
      - » アクセストークン内にメタデータを含める
    - API-OAUTH-METADATA-FOR-PAYLOAD
      - » アクセストークン応答電文内にメタデータを含める
  - 認証URLおよびメタデータURLの設定で考慮点があるため、下部リンクのKnowledge Centerを参照のこと

<参考>
IBM API Connect Knowledge Center：OAuth メタデータ
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/con_metadata.html

1-Tips. OAuth認証とログのカスタマイズ

❑ OAuth認証時の応答によるアクセストークンのカスタマイズ

- カスタマイズした、ユーザー・クレデンシャル、OAuthメタデータの参照方法
  - APIの処理の中で、リソース・オーナー名やメタデータを参照する場合
    - 活用例：API呼び出し時に使われたアクセス・トークンから、リソース・オーナー名やメタデータを取得してバックエンドAPIに渡す
    - API設定の「アセンブル」タブ上で、以下のコンテキスト変数で参照可能
      - » リソース・オーナー名の参照：apim.getvariable('oauth.resource-owner')
      - » OAuthメタデータの参照：apim.getvariable('oauth.miscinfo')
    - リソース・オーナー名やメタデータ以外にもアクセス・トークンに含まれている情報は同様にコンテキスト変数で参照可能(後続ページを参照)
  - クライアントからアクセス・トークンの情報を参照させる場合
    - 「OAuth2.0プロバイダーAPI」設定の「トークン・イントロスペクションの有効化」を有効にして、アクセス・トークンの内容を照会するAPIを利用可能にする
    - トークン・イントロスペクションの呼び出しには、アクセストークン、クライアントID、クライアント・シークレットが必要

<参考>
IBM API Connect Knowledge Center：OAuth メタデータ
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/con_metadata.html

1-Tips. OAuth認証とログのカスタマイズ

❑ OAuth利用時のコンテキスト変数の照会

- OAuthの標準仕様では、アクセス・トークンの内容についての規定はなく、各許可サーバーの実装にまかされている。API Connectでは、その情報は公開されていない。

- API Connectでは、APIの処理の中で、OAuthアクセス・トークンに関連する情報を参照して処理を行うことが可能
- API設定の「アセンブル」タブ上で、以下のコンテキスト変数で参照可能

項目	説明
oauth.access-token	要求が OAuth で認証される場合、この変数にはアクセス・トークン・ストリングが含まれます
oauth.resource-owner	要求が OAuth で認証される場合、この変数にはリソース所有者の名前が含まれます
oauth.scope	要求が OAuth で認証される場合、この変数には、このアクセス・トークンのスコープが含まれます
oauth.miscinfo	要求が OAuth で認証される場合、認証URLまたはメタデータURLのヘッダーで指定された情報が格納されます
oauth.not-before	要求が OAuth で認証される場合、この変数にはトークンの発行日付が含まれます
oauth.not-after	要求が OAuth で認証される場合、この変数にはトークンの有効期限が切れる日付が含まれます

1-Tips. OAuth認証とログのカスタマイズ

❑ APIのアクセス情報のログ出力方法1：分析ログの利用

- APIの呼び出しが行われると分析ログとして記録される（API Connect標準機能）
- APIの設定の「アセンブル」タブ上で、Activity-logポリシーを利用して、分析ログの出力内容を制御可能
  - 正常時とエラー時でそれぞれ、none, activity, header, payload から選択可能（デフォルトの設定は、正常時はactivity,　エラー時はpayload）

- APIマネージャーのカタログの「分析」画面のダッシュボード上で表示、データのダウンロード(csvファイル)が可能
- REST API呼び出しによるコマンドでの取得が可能
- 分析ログで出力可能な項目は下部リンクのKnowledge Centerを参照

- 注意点
  - 分析ログには、リソース・オーナーの情報やOAuthアクセス・トークンに含まれる情報は出力されない

<参考>
IBM API Connect Knowledge Center：API イベント・レコードのフィールド
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.apionprem.doc/rapim_analytics_apieventrecordfields.html

1-Tips. OAuth認証とログのカスタマイズ

❑ APIのアクセス情報のログ出力方法2：独自のログを出力

- APIの設定の「アセンブル」タブ上で、ログ出力を独自実装することが可能
  - XSLTポリシーでログ出力を独自実装する例
    - API Connectのコンテキスト変数で、データ項目の抽出
    - DataPowerのXSLT拡張でログ出力
  - リソース・オーナーの情報やOAuthアクセス・トークンに含まれる情報を出力可能

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:dp="http://www.datapower.com/extensions"
 xmlns:func="http://exslt.org/functions"
 xmlns:apim="http://www.ibm.com/apimanagement" extension-element-prefixes="dp func apim">

 <!-- Contains the APIM functions -->
 <xsl:import href="local:///isp/policy/apim.custom.xsl" />
 <xsl:template match="/">
  <xsl:variable name="request-uri" select="apim:getContext('request.uri')" />
  <xsl:variable name="client-id" select="apim:getContext('client.app.id')" />
  <xsl:variable name="resource-owner" select="apim:getContext('oauth.resource-owner')" />
  <xsl:variable name="oauth-scope" select="apim:getContext('oauth.scope')" />
  <xsl:message dp:priority="info" dp:type="HTTPAccess">
   <xsl:value-of select="$request-uri" /><xsl:text>,</xsl:text>
   <xsl:value-of select="$client-id" /><xsl:text>,</xsl:text>
   <xsl:value-of select="$resource-owner" /><xsl:text>,</xsl:text>
   <xsl:value-of select="$oauth-scope" /><xsl:text>,</xsl:text>
  </xsl:message>
 </xsl:template>
</xsl:stylesheet>
```

コンテキスト変数による値の取得

DataPowerのXSLT拡張によるログ出力

構成例2. トークン失効管理

2-1. DataPower Gatewayを利用したトークン失効管理

2-2. 外部のサーバーを利用したトークン失効管理

- この構成を実現する動機と実現方法
- 実装概要図
- トランザクション・フロー
- API Connect 設定のポイント
- トークン失効管理の方法

実現方式ごとの失効管理機能の整理

2-1. DataPower Gatewayを利用したトークン失効管理

この構成を実現する動機

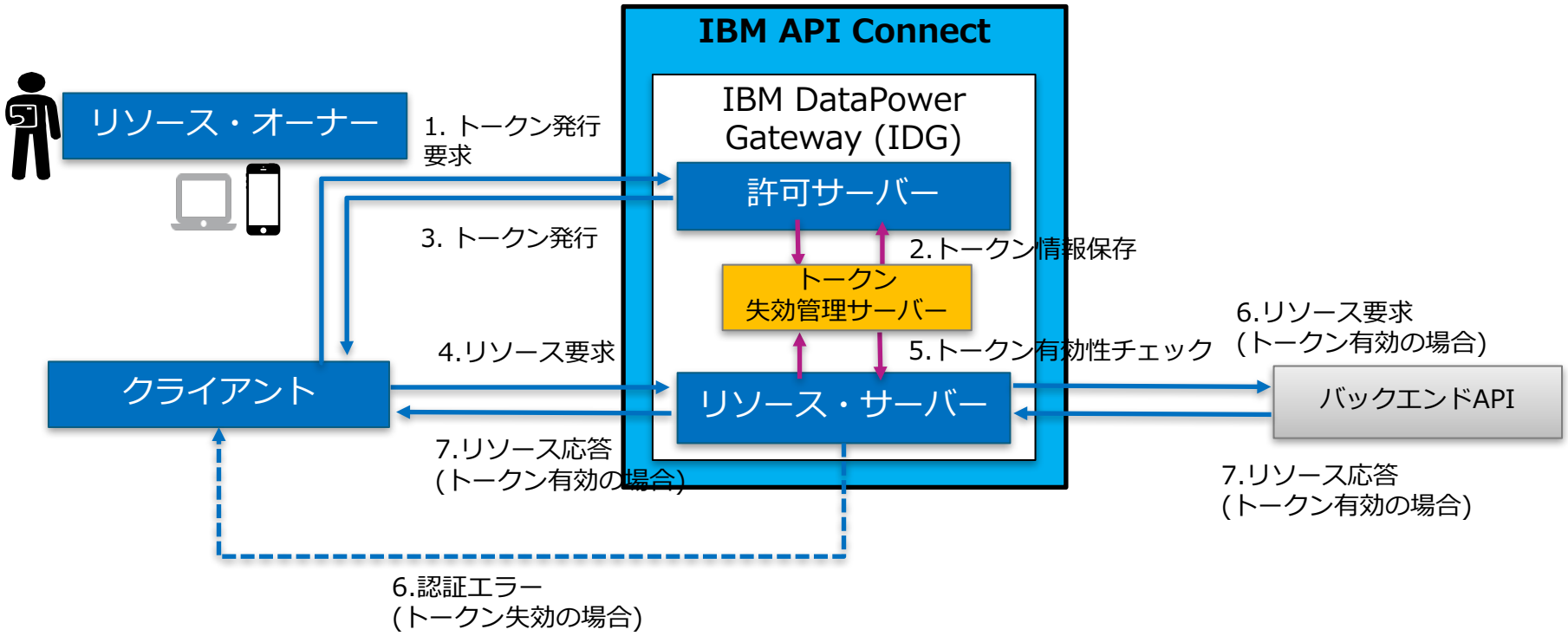
- トークンの有効期限を迎える前に任意のタイミングでトークンを失効したい
  - トークンが外部に漏洩した場合にトークンを失効したい
  - エンドユーザーの依頼にもとづきトークンを失効したい
- IBM DataPower Gateway上でトークン失効を管理したい

IBM API Connect での実現方法

- トークン失効の設定はAPI Connectの「OAuth 2.0プロバイダーAPI」で設定可能
  - IBM DataPower Gateway上でのトークン失効管理は、Bluemix版のAPI Connectではサポートされません。Bluemix版のAPI Connectでトークン失効管理したい場合は、後述の「失効URL」を使用する
- DataPower上でトークン失効管理するにあたり、Quota Enforcement Serverの設定を有効にする
  - 許可コードの再利用を防ぐには、クラスターを構成しているDataPower間で許可コードを共有する必要があるので、DataPower側の設定でQuota Enforcement Serverの設定を有効にする必要がある

2-1. DataPower Gatewayを利用したトークン失効管理

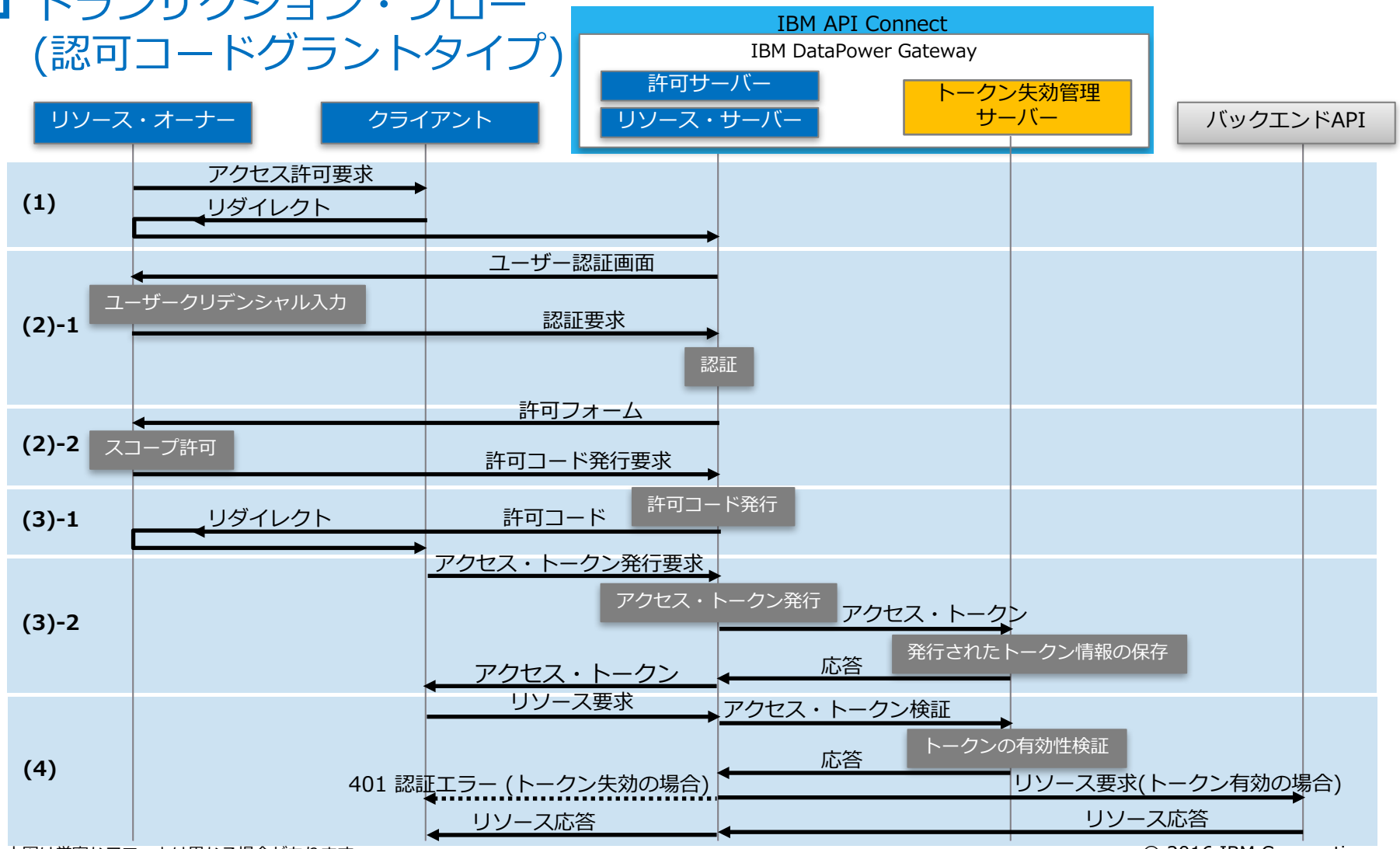
❑ 実装概要図



P.67 トランザクション・フローの(3)-2, (4)の部分の処理概要
トークンが失効している場合は、HTTP 401 Unauthorized(認証エラー)がクライアントに戻されます

2-1. DataPower Gatewayを利用したトークン失効管理

❑ トランザクション・フロー (認可コードグラントタイプ)



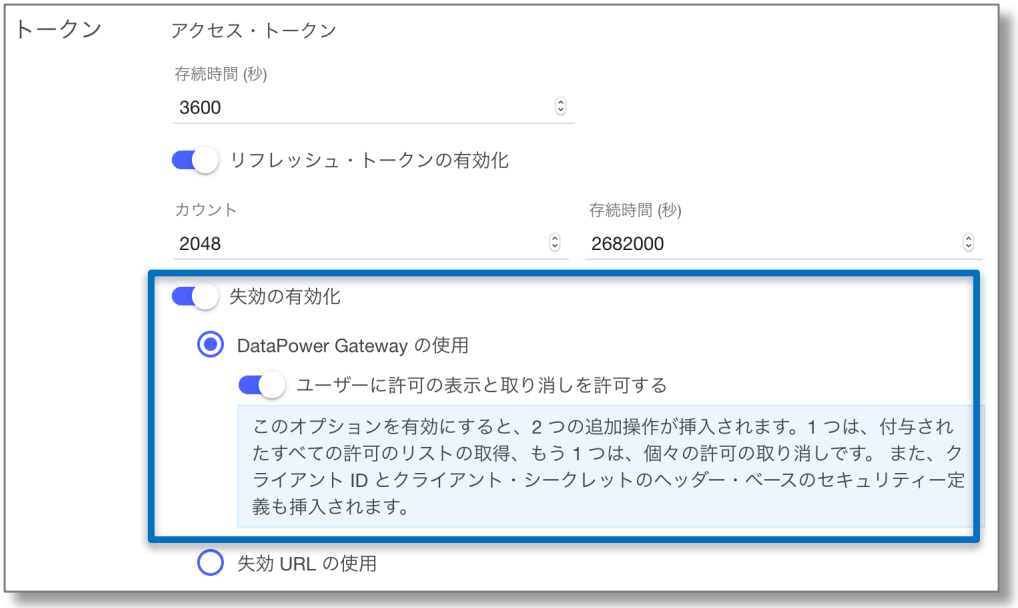
本図は厳密なフローとは異なる場合があります。

2-1. DataPower Gatewayを利用したトークン失効管理

❑ API Connect 設定のポイント(1)

❍ API Connectの「OAuth 2.0 プロバイダーAPI」に以下を設定

項目	設定値	備考
トークン	「失効の有効化」を有効化し、「DataPower Gatewayの使用」を選択して、「ユーザーに許可の表示と取り消しを許可する」を有効にする	- トークンの発行状況の表示と失効(削除)のインターフェースが提供される - 以下の2つのパスが有効となる GET /oauth2/issued DELETE /oauth2/issued



2-1. DataPower Gatewayを利用したトークン失効管理

❑ API Connect 設定のポイント(2)

- DataPower上でQuota Enforcement Serverの設定を有効にする
- DataPowerのdefaultドメインで、「割り当て量制約サーバー の構成(Config Quota Enforcement Server)」の、「管理状態」を「有効(enabled)」にする。
  - 物理・仮想版ともにQuota Enforcement Serverの「管理状態」はデフォルトで「有効(enabled)」

- 「データ・ストレージのロケーション」を設定することで、DataPower上でのトークン失効情報の保管先が決まる。DataPowerのメモリー上に保管する場合は「(なし)」に設定する。RAIDボリューム上に保管する場合は「raid0」 (事前にraidのセットアップが必要)に設定する。
- 複数台のDataPowerでクラスターを構成している場合は、「ピア·グループ·モード(Peer group mode)」を有効にすることで、OAuthデータがクラスター内で同期される
  - クラスター構成の場合は、全てのDataPowerで上記の設定を行います

2-1. DataPower Gatewayを利用したトークン失効管理

トークン失効管理方法

- 「OAuth 2.0 プロバイダーAPI」の「ユーザーに許可の表示と取り消しを許可する」オプションを有効にすることで、DataPower上の「トークン発行状況の表示」と「トークンの失効(削除)」を行うAPIが提供される

- トークン発行状況の表示
  - 下記のAPIにGET要求を行うことで、トークンの発行状況(スコープ、リソースオーナ名、トークンの有効期限など)を参照可能
    - /oauth2/issued

- トークン失効(削除)
  - 下記のAPIにDELETE要求を行うことで、トークン失効が可能
    - /oauth2/issued
  - リソースオーナー単位(エンドユーザー単位)でのトークン失効が可能
    - トークン単位(アクセス・トークン、リフレッシュ・トークン)での失効はできない
  - API呼び出し時に、クライアントID、クライアントシークレット、リソースオーナーのユーザー&パスワードの入力が必要

2-1. DataPower Gatewayを利用したトークン失効管理

トークン発行状況の表示方法

- cURLによるリクエスト例

```
$ curl -i -k -u user01:password -H 'accept: application/json' -H 'x-ibm-client-id: 93be043e-de8e-4348-a2b1-b09a53353706' -H 'x-ibm-client-secret: H3lC1tO0kJ0xQ7aO2yK6vO7uC7bL2nR4rQ2yF2pI3kY4wI7vL7' -X GET 'https://<IDG_IPAddress>/org/sb/oauth-end/oauth2/issued'
```

- 応答電文例
  - トークン発行状況が返る

```
X-Backside-Transport: FAIL FAIL
Connection: Keep-Alive
Transfer-Encoding: chunked
Cache-Control: private, no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Security-Policy: default-src 'self'; style-src 'unsafe-inline'
Content-Type: application/json; charset=UTF-8


[
 {
  "clientId":"93be043e-de8e-4348-a2b1-b09a53353706",
  "clientName":"getDateApp",
  "owner":"user01",
  "scope":"getDate",
  "issuedAt":1469589686,
  "expiredAt":1472271686,
  "refreshTokenIssued":true
 }
]
```

2-1. DataPower Gatewayを利用したトークン失効管理

トークン失効(削除)方法

cURLによるリクエスト例

```
$ curl -i -k -u user01:password -H 'accept: application/json' -H 'x-ibm-client-id: 93be043e-de8e-4348-a2b1-b09a53353706' -H 'x-ibm-client-secret: H3lC1tO0kJ0xQ7aO2yK6vO7uC7bL2nR4rQ2yF2pI3kY4wI7vL7' -H 'content-type: application/x-www-form-urlencoded' -X DELETE 'https ://<IDG_IPAddress>/org/sb/oauth-end/oauth2/issued?client-id=93be043e-de8e-4348-a2b1-b09a53353706'
```

応答電文例

トークン失効に成功した場合

```
X-Backside-Transport: FAIL FAIL
Connection: Keep-Alive
Transfer-Encoding: chunked
Cache-Control: private, no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Security-Policy: default-src 'self'; style-src 'unsafe-inline'
Content-Type: application/json; charset=UTF-8

{ "status":"success" }
```

トークン失効に失敗した場合

```
X-Backside-Transport: FAIL FAIL
Connection: Keep-Alive
Transfer-Encoding: chunked
Cache-Control: private, no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Security-Policy: default-src 'self'; style-src 'unsafe-inline'
Content-Type: application/json; charset=UTF-8

{ “status”:“failure" }
```

構成例2. トークン失効管理

2-1. DataPower Gatewayを利用したトークン失効管理

2-2. 外部のサーバーを利用したトークン失効管理

- この構成を実現する動機と実現方法
- 実装概要図
- トランザクション・フロー
- API Connect 設定のポイント
- トークン管理サーバーの実装概要

実現方式ごとの失効管理機能の整理

2-2. 外部のサーバーを利用したトークン失効管理

❑ この構成を実現する動機

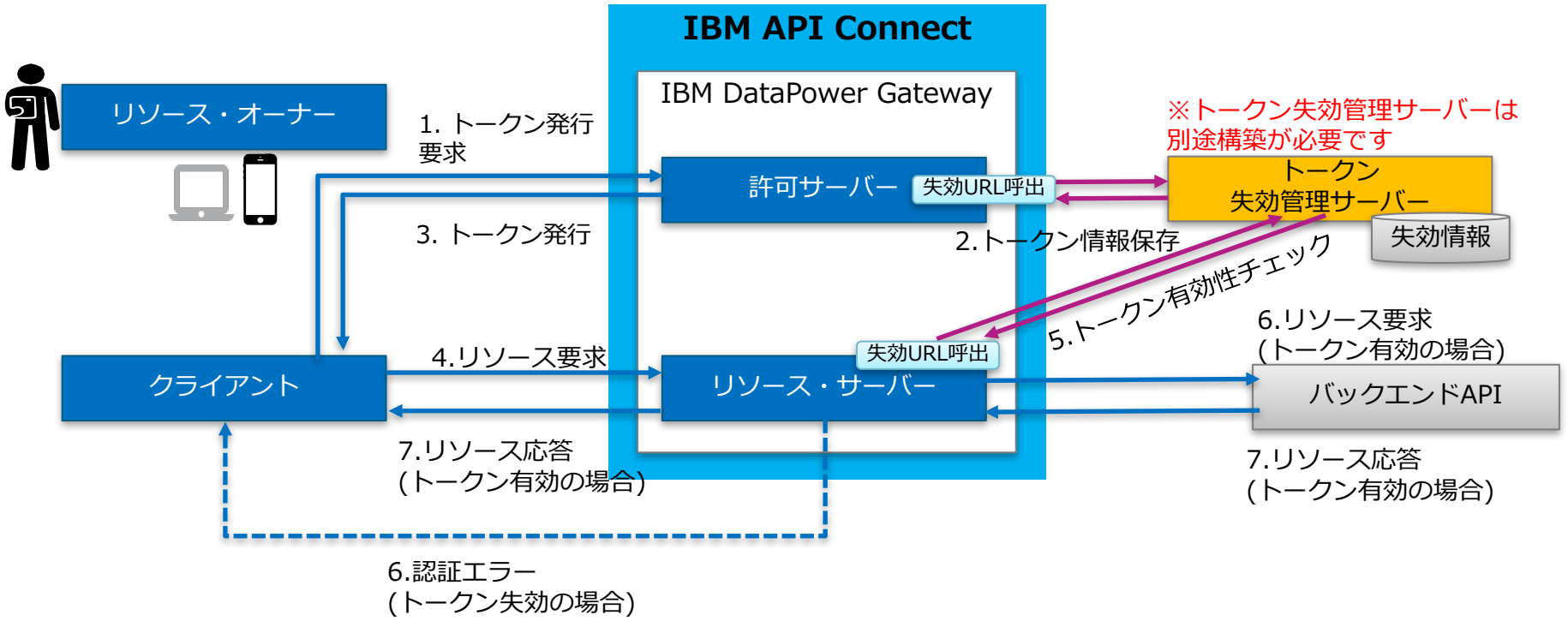
- トークンの有効期限を迎える前に任意のタイミングでトークンを失効したい
  - トークンが外部に漏洩した場合にトークンを失効したい
  - エンドユーザーの依頼にもとづきトークンを失効したい
- トークン失効のロジックをカスタマイズしたい
  - トークン単位(アクセス・トークン、リフレッシュ・トークン)で失効させたい
- Bluemix版のAPI Connectでトークン失効を管理したい

❑ IBM API Connect での実現方法

- トークン失効の設定はAPI Connectの「OAuth 2.0プロバイダーAPI」で設定可能
  - トークン管理を行う外部サーバーのURLを許可サーバーの「失効URL」に指定
- トークン管理を行う外部サーバーとして、アプリケーションやデータベースを別途構築・開発する必要あり
  - 以下の機能を実現するアプリケーションを開発
    - DataPowerが発行するトークン情報を外部サーバーのDBに保存
    - DBに保存されたトークン情報を元に、リソース要求時に付与されるトークンの有効性を検証
    - 不要になったトークン情報を適宜削除するメンテナンス・アプリケーション
      - » API Connectの機能としてトークン情報のメンテナンスが提供されないため、メンテナンス・アプリケーションがないと保管されたトークン情報が外部サーバー上に溜まり続けることになります

2-2. 外部のサーバーを利用したトークン失効管理

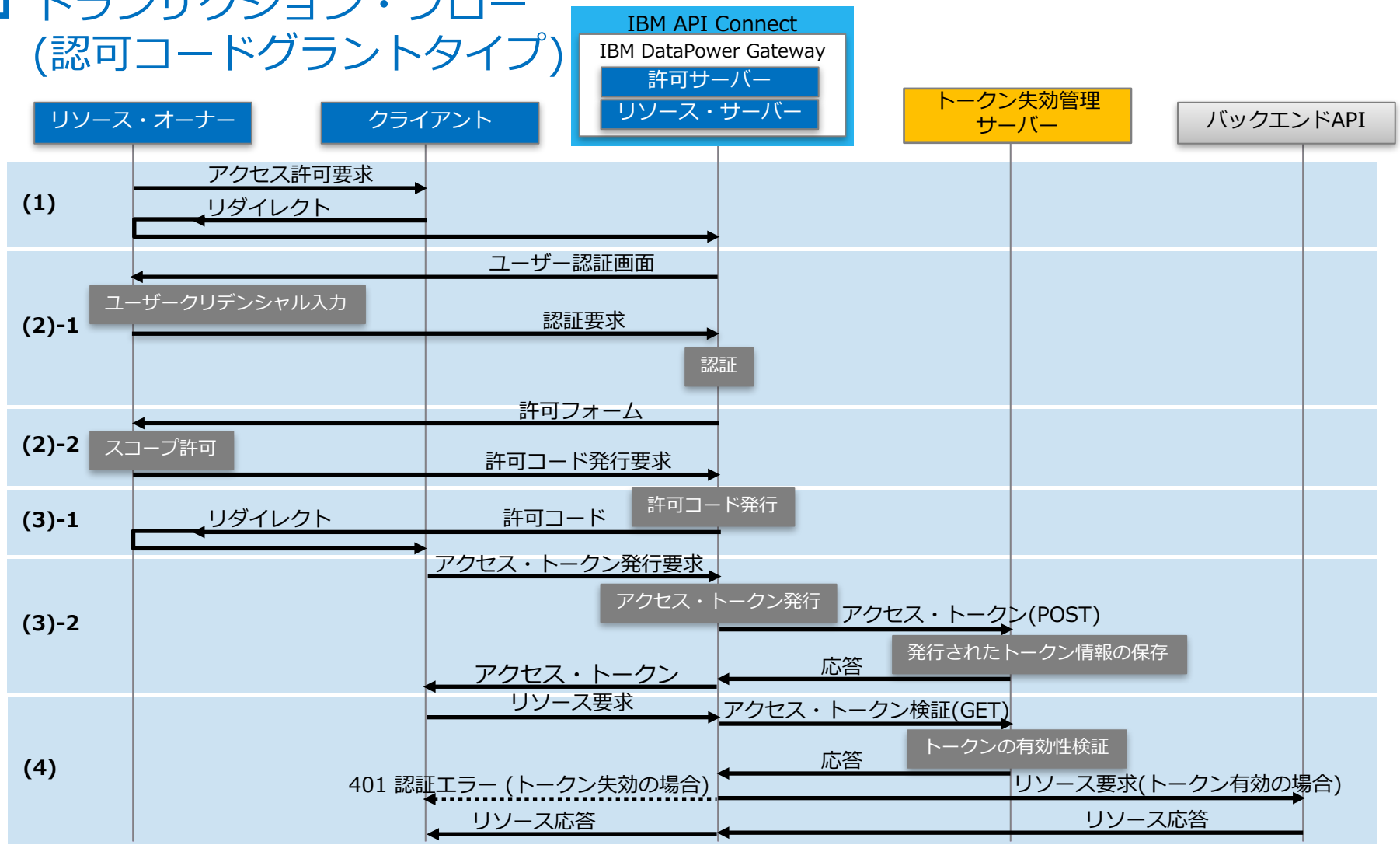
❑ 実装概要図



P.76 トランザクション・フローの(3)-2, (4)の部分の処理概要
トークンが失効している場合は、HTTP 401 Unauthorized(認証エラー)がクライアントに戻されます

2-2. 外部のサーバーを利用したトークン失効管理

❑ トランザクション・フロー (認可コードグラントタイプ)



本図は厳密なフローとは異なる場合があります。

2-2. 外部のサーバーを利用したトークン失効管理

❑ API Connect 設定のポイント

- API Connectの「OAuth 2.0 プロバイダーAPI」に以下を設定

項目	設定値	備考
トークン	「失効の有効化」を有効化し、「失効URLの使用」を選択して、「失効URL」を入力	- 「失効URL」には、トークン管理を行う外部サーバーのURLを入力

2-2. 外部のサーバーを利用したトークン失効管理

トークン管理サーバーの実装概要

POST要求

- アクセストークン/リフレッシュトークン発行時に、トークン失効管理サーバーに対して、発行したトークン情報がPOST(HTTPのbodyメッセージ)で連携される
- POST要求例(API Connectの機能によりトークン失効管理サーバーに送信される)
  - トークン失効管理サーバーのアプリケーションで送信されるトークン情報をDBに保存
  - トークン情報のどの項目を保存するかを、失効する単位に応じて取捨選択する
    - » トークン単位で失効管理するのであれば、アクセス・トークン、リフレッシュ・トークンの情報を保管
    - » リソースオーナー単位(エンドユーザー単位)で失効管理するのであれば、リソースオーナの情報を保管

```
POST <revocationURL> HTTP/1.1
User-Agent: IBM-APIManagement/4.0
Content-Type: application/xml

<?xmlversion="1.0" encoding="UTF-8"?>
<token>
<token_type>bearer</token_type>
<access_token>AAENYy1hbGwtcmVmcmVzaOfNeQKX8ZeojsBY9v0FI7/OerQvzKHq...</access_token>
<expires_in>3600</expires_in>
<scope>3600</scope>
<resource-owner>alice</resource-owner>
<client_id>83d9cdcd-ba72-4d00-abae-005da8da5fb1</client_id>
<refresh_token>AAGnYvhbGwtR8JshWETNeOKX8ZeojsBYr3JUnRCE4/OkirOreLGe...</refresh_token>
</token>
```

失効管理する単位に応じて、保管する情報の取捨選択が必要

- トークン失効管理サーバーからの応答例
  - トークン情報保存後に、POST要求を受け取った応答としてアプリケーションからDataPowerに対して、以下のステータス・コードを返すよう実装する

```
HTTP/1.1 200 OK
```

<参考>
IBM API Connect Knowledge Center : OAuth 失効 URL
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/ref_oauth_sampleurl.html

2-2. 外部のサーバーを利用したトークン失効管理

トークン管理サーバーの実装概要

- GET要求
  - 以下のリクエストが実行されると、DataPowerからトークン失効管理サーバーに対してGET(HTTPのheader)でトークンの有効性チェックが行われる
    - アクセストークンを使ったサービス
    - APIの呼び出しリフレッシュトークンを使ったアクセストークンの再発行
  - GET要求例(API Connectの機能によりトークン失効管理サーバーに送信される)

```
GET  <revocationURL>HTTP/1.1
access-token: AAETb2F1dGgtcmV2b2tlLWN1c3RvbfZaRlVbnPSc1
client-id: 760d75a2-44b1-4485-8c6f-0d264fcf7398
resource-owner: alice
```

  - トークン失効管理サーバーからの応答例
    - DataPowerからのGET要求に対して、トークン失効管理サーバーから以下の応答（失効と判定されたトークンまたはリソースオーナの情報など）を返すようアプリケーションを実装する必要あり

```
HTTP/1.1 200 OK
Content-Type: application/xml
Cache-Control: public, max-age=120
Date: Fri, 08 May 2015 21:49:03 GMT


<?xml version="1.0" encoding="UTF-8"?>
<oauth-revocation>
 <resource-owner before="2015-05-01T09:30:10Z">alice</resource-owner>
<oauth-revocation>
```

Cache-Controlを利用することで、トークン失効管理サーバーからの応答結果をDataPower上に最大2分間(120秒)キャッシュさせ、トークン失効管理サーバーへの照会頻度を軽減することが可能

後続ページで、応答例のパターンを紹介

<参考>
IBM API Connect Knowledge Center : OAuth 失効 URL
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/ref_oauth_sampleurl.html

2-2. 外部のサーバーを利用したトークン失効管理

トークン管理サーバーの実装概要

トークン失効時のトークン失効管理サーバーからの応答例(1)

- 特定の日付(2016年10月1日9時30分)より前にリソースオーナーuser01に対して発行されたトークンを失効したい場合は、以下の応答を返却する

```
HTTP/1.1 200 OK
Content-Type: application/xml
Cache-Control: public, max-age=120
Date: Sat, 08 Oct 2016 21:49:03 GMT

<?xml version="1.0" encoding="UTF-8"?>
<oauth-revocation>
<resource-owner before="2016-10-01T09:30:10Z">user01</resource-owner>
<oauth-revocation>
```

失効する「特定の日付」と「リソースオーナー」を返却

- 特定の日付(2016年10月1日9時30分)より前に発行された全てのトークンを失効したい場合は、以下の応答を返却する

```
HTTP/1.1 200 OK
Content-Type: application/xml
Cache-Control: public, max-age=120
Date: Sat, 08 Oct 2016 21:49:03 GMT


<?xml version="1.0" encoding="UTF-8"?>
<oauth-revocation>
<everytoken before="2016-10-01T09:30:10Z" />
<oauth-revocation>
```

失効する「特定の日付」を返却

<参考>
IBM API Connect Knowledge Center：OAuth 失効 URL
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/ref_oauth_sampleurl.html

2-2. 外部のサーバーを利用したトークン失効管理

❑ トークン管理サーバーの実装概要

❍ トークン失効時のトークン失効管理サーバーからの応答例(2)

- トークン単位(特定のアクセス・トークン、リフレッシュ・トークン)で失効したい場合は、以下の応答を返却する

```
HTTP/1.1 200 OK
Content-Type: application/xml
Cache-Control: public, max-age=120
Date: Sat, 08 Oct 2016 21:49:03 GMT

<?xml version="1.0" encoding="UTF-8"?>
<oauth-revocation>
<token type="access">AAEkNWUwOGU1MTYtNzRhNy00M2M2LTllNWMtMDk2ZTkzZDg2NjFh5Ip0rQNhhb1_
.....sOLojLHTQUPmOZHyeDous</token>
<token type="refresh">fZaRlVbnPSc1UGTjCRdq4mPbOosD2+aZlKbJ6bTeWfZaRlVbnPSc1UGTjCRdq4mPbOo6
.....aOLRejtLWiORWEsdbJerhsh</token>
<oauth-revocation>
```

失効するアクセス/リフレッシュトークンの情報を返却

<参考>
IBM API Connect Knowledge Center：OAuth 失効 URL
https://www.ibm.com/support/knowledgecenter/ja/SSMNED_5.0.0/com.ibm.apic.toolkit.doc/ref_oauth_sampleurl.html

構成例2. トークン失効管理

2-1. DataPower Gatewayを利用したトークン失効管理
2-2. 外部のサーバーを利用したトークン失効管理

- この構成を実現する動機と実現方法
- 実装概要図
- トランザクション・フロー
- API Connect 設定のポイント
- トークン管理サーバーの実装概要

実現方式ごとの失効管理機能の整理

実現方式ごとの失効管理機能の整理

- API Connectで実装可能な２種類のトークン失効管理機能には、それぞれ以下の特徴がある

項目	DataPower Gatewayを利用したトークン失効	外部のサーバーを利用したトークン失効
概要	・API Connectの機能を利用して、DataPower Gateway内部で失効管理する	・外部のサーバー上で失効管理する
失効権限	・リソース・オーナー権限でのみ失効可能	・柔軟に実装可能
失効単位	・リソース・オーナーが特定クライアントに許可したトークンの一時停止・再開・失効が可能 ・クライアント単位やトークン単位での失効は不可	・柔軟に実装可能 ・クライアント単位やトークン単位での失効も可能
制約・前提等	・DataPowerを冗長構成とる場合は3ノード以上必要 ・API Connectでは、照会と失効のAPIが提供されるのみであるため、ユーザー(リソース・オーナー)に失効の操作をさせるための画面は別途開発する必要あり ・リソース・オーナー権限でしか失効できないため、システム運用として、トークンの失効管理はできない ・製品機能が利用できるため比較的構築・運用の負荷が低い	・外部のサーバーを構築・開発が必要 ・運用のインターフェースを構築開発が必要 ・すべて構築・運用が必要となるため、構築・運用の負荷が高い

構成例3. Gatewayでのフロント処理の追加

- この構成を実現する動機と実現方法
- 実装概要図
- 設定のポイント
- 構成手順

3. Gatewayでのフロント処理の追加

この構成を実現する動機

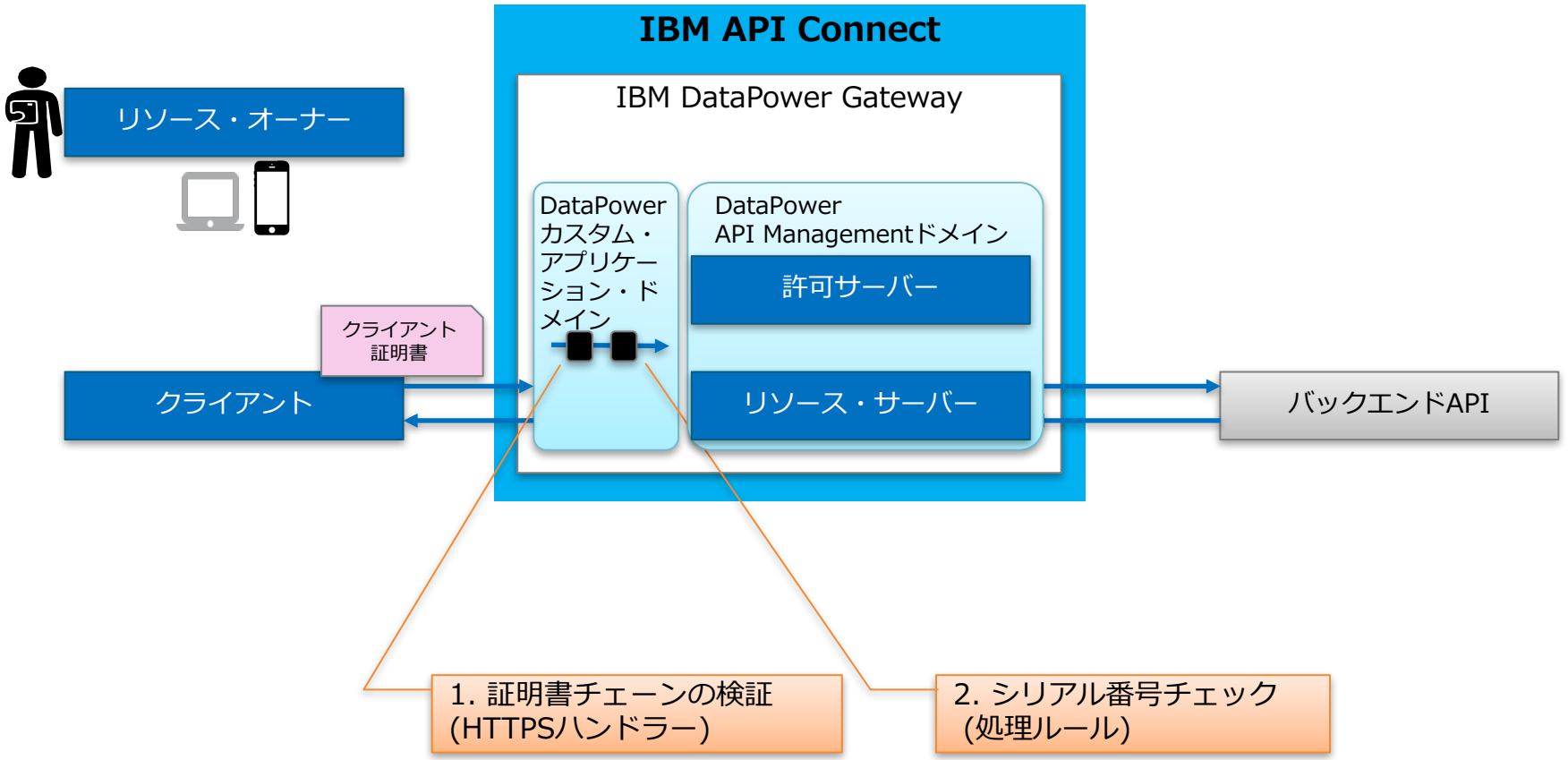
- API Connectの標準機能だけでは実現しにくい処理を、DataPower Gatewayのフロント処理として実装
  - API呼び出しに対する全流量制御、アクセスログ出力、クライアント証明書のチェック、HTTP->HTTPS変換、などを実装
- 外部アクセスに対するセキュリティ強化のために、クライアント証明書を使用したクライアント認証の実施
  - クライアント証明書の証明書チェーン検証、有効期限チェック
  - クライアント証明書のシリアルチェック

実現方法(クライアント証明書による認証実施例)

- DataPower Gatewayでのカスタム・アプリケーション・ドメインの作成
  - API Connectを構成するとDataPower Gateway上に作成されるAPI Managementドメインの前段に、プロキシーとして稼動するアプリケーション・ドメインを作成して配置
  - クライアント認証が必要なリクエストは、このプロキシー経由で、後段のAPI ManagementドメインのAPIの呼び出す
  - クライアント証明書の検証
    - HTTPSフロントサイドハンドラーのSSLサーバー・プロファイル設定で、クライアント証明書の証明書チェーン、有効期間等の検証
    - ドメイン内の処理ルールでの証明書内容(証明書のシリアル番号等)の検証

3. Gatewayでのフロント処理の追加

❑ 実装概要図



3. Gatewayでのフロント処理の追加

❑ 設定のポイント

- DataPower Gateway上にユーザー独自のカスタム・アプリケーション・ドメインを作成し、HTTPSフロントサイドハンドラーを構成するため、既存のAPI ManagementドメインのHTTPSフロントサイドハンドラーとのポートのバッティングに留意する
  - 同じネットワーク・インターフェースを使用する場合は、デフォルト(443)以外のポートを指定する
  - デフォルトのポート(443)を使用したいのであれば、この独自アプリケーションでリクエストを受け付ける別のネットワーク・インタフェース(IPアドレス)を割り振る必要がある
    - 外部からのリクエスト時のIPアドレスとポートの組み合わせが、API ConnectのIPアドレスとポートの組み合わせとバッティングしないことに留意する。APIのエンドポイントのパスは変更する必要はない

- シリアル番号の突合せする情報は、DataPower上のファイルシステムにプロパティ・ファイルとして保持することが想定されるため、そのプロパティ･ファイルを更新、管理するための運用も検討が必要

3. Gatewayでのフロント処理の追加

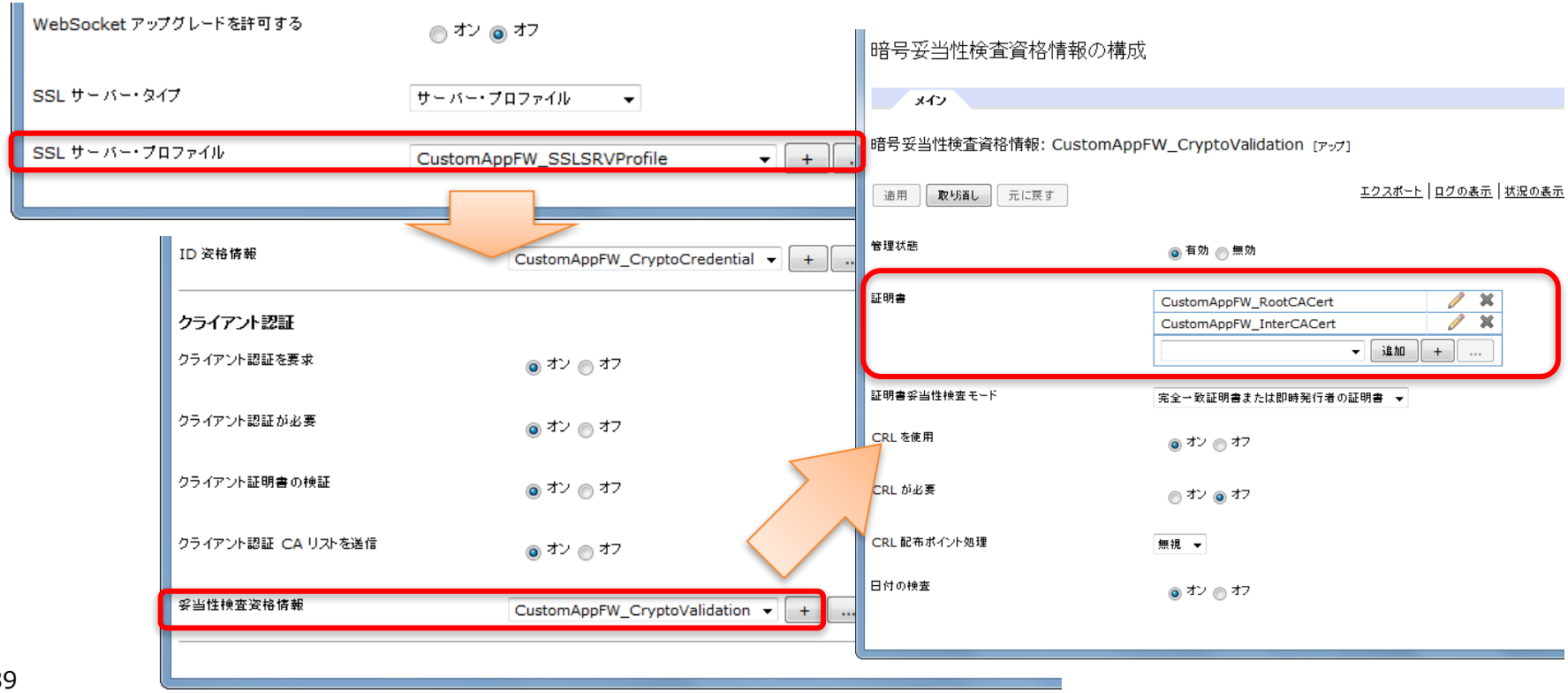
構成手順(1)

- DataPower上で、認証処理を行うアプリケーション・ドメインを作成
- 処理を実現するサービス(マルチプロトコル・ゲートウェイ)の作成
  - HTTPSフロントサイド・ハンドラーの作成
  - 処理ポリシー、処理ルールの作成

  - API ConnectのHTTPSフロントサイド・ハンドラーのIPアドレスとポートの組み合わせと、作成するアプリケーション・ドメインのHTTPSフロントサイド・ハンドラーのIPアドレスとポートの組み合わせに留意する

3. Gatewayでのフロント処理の追加

- 構成手順(2)
  - マルチプロトコル・ゲートウェイのフロントサイド・プロトコルのHTTPSハンドラーでのSSLサーバープロファイル設定、妥当性検査資格情報で、ルート証明書、中間証明書を設定



3. Gatewayでのフロント処理の追加

❑ 構成手順(3)

- DataPower上でのシリアル番号チェック
  - DataPowerのXSLTアクションで、クライアント証明書のシリアル番号の照会が可能(dp:auth-infoおよびdp:get-cert-serial)
  - クライアント証明書のシリアル番号をチェックすることで、パスワード認証とあわせた2要素認証を実現

```
<!-- 証明書情報の取得 -->
      <xsl:variable name="base64Cert" select="dp:auth-info('ssl-client-cert')"/>
      <!-- 証明書のCNとシリアルを取得 -->
      <xsl:variable name="certSubject" select="dp:get-cert-subject(concat('cert:',$base64Cert))"/>
      <xsl:variable name="cn" select="substring-before(substring-after($certSubject,'CN='),',')" />
      <xsl:variable name="certSerialNum" select="dp:get-cert-serial(concat('cert:',$base64Cert))"/>

<!-- クライアント証明書ありの場合 -->
  <!--以下、シリアル番号が記述されたプロパティ・ファイルを読み込み、番号のマッチングを行いシリアル番号チェックを実行 -->
・・・・
```

- 参考
  - DataPower Cryptographic extension functions
    - get-cert-serial()
    - http://www.ibm.com/support/knowledgecenter/ja/SS9H2Y_7.5.0/com.ibm.dp.doc/get-cert-serial_cryptographicfunction.html