ホワイトペーパー：Azure API Management と OSS(Keycloak) を活用したセキュアな API 公開

著：日本マイクロソフト株式会社 パートナー事業本部 パートナー技術統括本部

株式会社 日立製作所 OSS ソリューションセンタ

目次

1. はじめに

デジタルトランスフォーメーションにおいて、事業者は、自組織内もしくは他組織に向けて API を公開する必要に迫られている中、API を効率的に公開するための共通機能を提供する API 管理基盤が注目を集めている。また、昨今 API の不正利用が相次いでおり、セキュリティとしては認可認証の規格である OAuth2.0 の適切な利用も必須である。

本書では、Azure が提供する API 管理基盤である「Azure API Management」と昨今存在感を高めている認可認証 OSS である「Keycloak」を組み合わせて、セキュアな API 管理基盤を実現できることを示す。2 章および 3 章では概要を説明し、4 章以降で実際の構築手順の例を示す。

2. Azure API Management の概要

まず、Azure API Management(以降、API Management)の概要を紹介する。

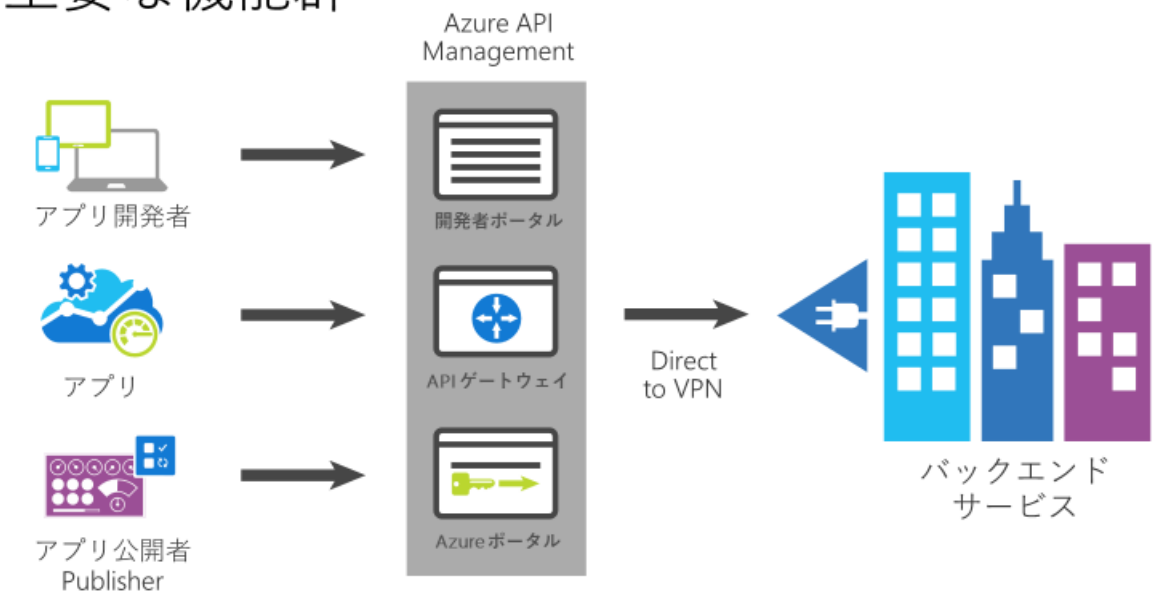
2.1. 関連するステークホルダーと主要な機能群

API Management は次の 3 つのコンポーネントで構成される。

- API ゲートウェイ
- Azure ポータル
- 開発者ポータル

それぞれのコンポーネントの利用者と機能例を次に示す。

主要な機能群



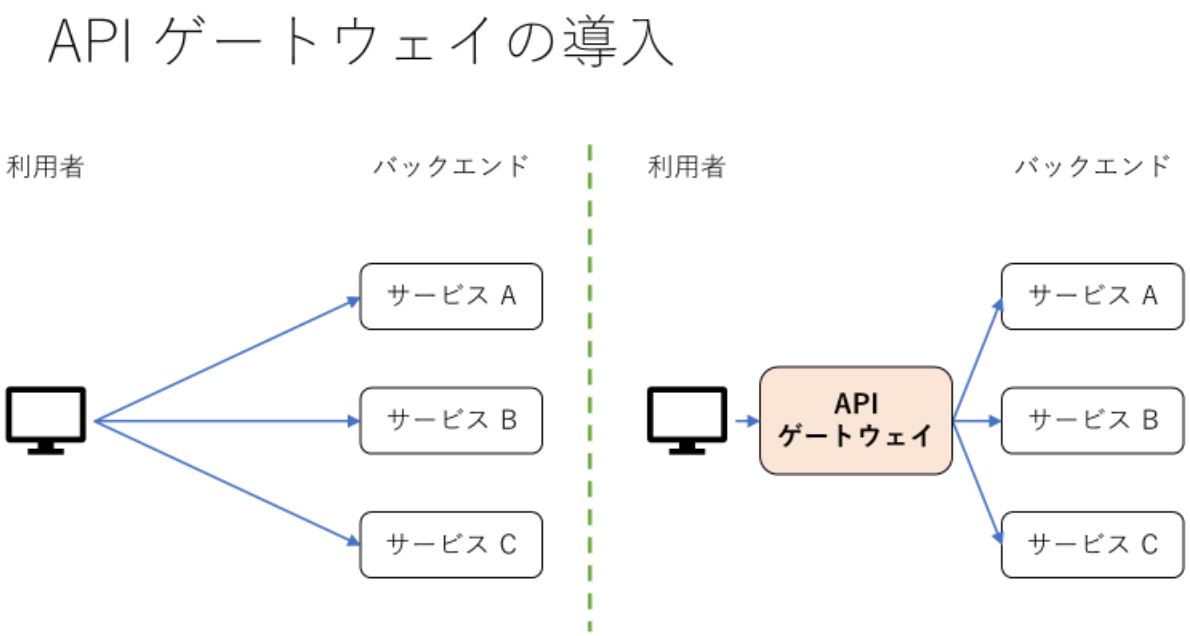
コンポーネント	利用者	機能例
API ゲートウェイ	アプリケーション	- API 呼び出しを受け入れ、バックエンドにルーティングする。 - API キー、JWT トークン、証明書、その他の資格情報を検証する。 - 流量の制限（クォータとレートの 2 種類可能） - コードを変更せずにその場で API を変換する。 - セットアップ時にバックエンドの応答をキャッシュする。 - 分析目的で呼び出しメタデータを記録する。
Azure ポータル	API Management の管理者	- API スキーマを定義またはインポートする。 - API を製品にパッケージする。 - API のクォータや変換などのポリシーを設定する。 - API の利用状況などを分析し、集計する。 - ユーザーを管理する。
開発者ポータル	アプリケーションの開発者	- API のドキュメントを閲覧する。 - 対話型コンソールを使用して API を試す。 - アカウントを作成して API キーを取得する。 - 自身の API 利用を分析する。

ここでいう開発者は、API を利用したアプリケーションの開発者を意図しており、API Management サービスインスタンス内のユーザーアカウントとして管理される。API Management の管理者が開発者を作成したり参加を呼びかけたりすることができるほか、開発者自身が開発者ポータルからサインアップすることもできる。

2.2. API ゲートウェイによるメリット

API Management で中核となる要素が API ゲートウェイである。本節では API ゲートウェイの概要とメリットを紹介する。

昨今のシステム構成では、複数のバックエンドのサービスを連携させることで、一つの顧客向けサービスを提供することが主流になってきている。すでに運用を継続しているシステム資産が存在していることが多く、一つのサービスで全ての機能を提供することは事実上まれであるともいえる。



このような場合、サービスの利用者(アプリ開発者)が直接各サービスを API コールするのではなく、上の図のように、間に API ゲートウェイを配置することが一般的である。

一般的に、API ゲートウェイに求められる機能要件を以下に示す。

- 再利用性：既存の API をそれ自体の変更なくサポートする
- 容易性：サービスの利用者が簡単に API 利用できるようにする
- セキュリティ：悪意のあるアクセスや過剰アクセスからバックエンドを保護する
- 分析：どの API が誰からどれほど利用されているか測定する

このような機能要件を満たす API ゲートウェイを導入することで、サービス利用者およびサービス側にとって次のようなメリットがある。

まず、ゲートウェイは既存のシステム資産や複数サービスとの組み合わせのような複雑さをサービスの利用者から隠蔽し、利用者に対して一貫性のある API を提供する。利用者に対して一貫性のある API を提供することで、サービス利用者にとっての利便性が向上する。

また、バックエンドのそれぞれのサービスには、認証・ロギング・過負荷からの保護など、共通要件が多く存在する。API ゲートウェイはこれらの共通要件をカバーし、サービス側にとってのメリットも提供する。昨今のモバイルファーストの流れや、利用するアプリケーションが複数存在する状況では、これらを一か所で集中管理することのメリットは大きい。

API Management ではこのような API ゲートウェイ機能の他に、ゲートウェイ機能を各ステークホルダーへ提供するための統合機能もあわせてサポートする。さらには、API Management には、「ポリシー」と呼ばれる機能があり、API の動作をカスタマイズすることができる(付録「ポリシーによるカスタマイズ」を参照)。またこのような豊富な機能をスモールスタートから本格的な本番利用を想定した様々な課金体系(付録の「価格レベル」を参照)で利用することができる。

以上で API Management の概要を紹介したが、さらなる詳細については公式ドキュメント[1]を参照頂きたい。

3. OAuth2.0 による API セキュリティと Keycloak

API のセキュリティとして最も基本的なものは、呼び出し元のユーザおよびアプリケーションを制限するための認可認証であり、認可認証の仕組みとしては「OAuth2.0」がデファクトである。OAuth2.0 は、「アクセストークン」を認可の結果として API 呼び出し元に発行するための枠組みであり、アクセストークンを払い出し管理するための「認可サーバ」が必要になってくる。

3.1. 認可サーバとして存在感が高まる Keycloak

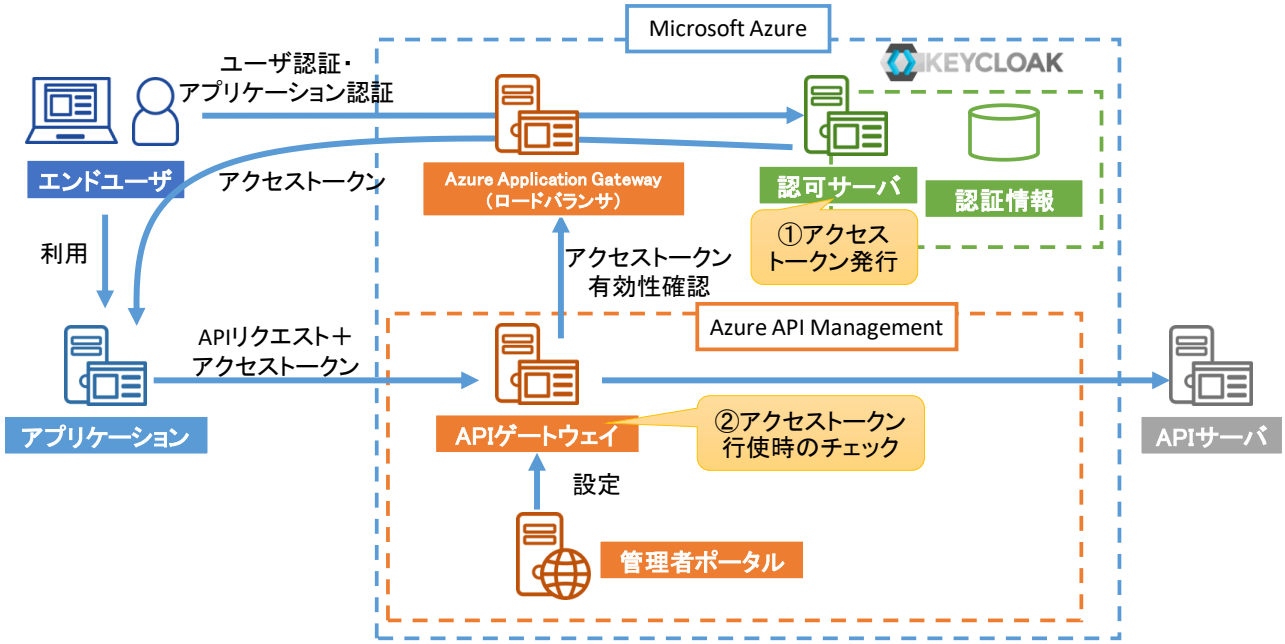
認可サーバは専門的なソフトウェアであり、また認証情報との繋ぎ込み等のカスタマイズが必要とされるため、多くの API 管理製品では認可サーバは別途用意する必要がある。

[1] https://docs.microsoft.com/ja-jp/azure/api-management/

認可サーバの実装としては、OSS である「Keycloak[2]」が急速に存在感を増している。Keycloak は、元々はシングルサインオンサーバのソフトウェアであるが、近年認可サーバに最適化された開発が OSS コミュニティで進んでいる。例えば OAuth2.0 をより厳密にした OpenID Connect に対応しているほか、最先端の仕様である Financial-Grade API(FAPI)への対応[3]も日々進んでいる 。

3.2. Azure API Management と Keycloak の構成

API management において OAuth2.0 の認可サーバとして Keycloak を使う場合の構成を図に示す。OAuth2.0 による API 保護は大きく「①アクセストークンの発行」と「②アクセストークン行使時のチェック」の 2 つのフェーズに分類できる。以下 2 つのフェーズにおける各構成要素の役割を示す。



① アクセストークンの発行

OAuth2.0 で定められたフローに従い、認可サーバ(Keycloak)が、エンドユーザやアプリケーションを認証し、アクセストークンをアプリケーションに発行する。Keycloak の場合、

[2] http://www.keycloak.org/

[3] https://qiita.com/yuichi-nakamura/items/53b35a9cbaf663fab441

アクセストークンは JWT(Json Web Token)形式になっており、どのような権限を持つのか、誰が認証されたのかという情報を封入することができる。

下図に Keycloak のアクセストークンの例を示す。例えば「scope」から始まる行は、認可された権限名を示している。


```
{
 "jti": "b9796e02-df59-4c60-a589-a34d7b356296",
 "exp": 1564103902,
 "nbf": 0,
 "iat": 1564103602,
 "iss": "https://okd-lb-router.oss.example.co.jp:8445/auth/realms/test_realm",
 "aud": "account",
 "sub": "d33286d6-4091-4b1a-83e1-ee09cf39c60e",
＜略＞
 “scope”: “openid email profile offline_access”,＃認可された権限名
 “email_verified”: false,
 “preferred_username”: “test_user” ＃ユーザ名
}
```


なお、トークン発行段階では、Keycloak は API management とは独立して動作することに注意する。また、Keycloak の冗長化構成を取るために、ロードバランサ(Azure の場合は Azure Application Gateway)を Keycloak の前に配置することが一般的である。

② アクセストークンの行使時のチェック

アプリケーションは、Keycloak から発行されたアクセストークンを HTTP ヘッダにセットし、API management 配下にある API をコールする。ここで、API リクエストは API management の API ゲートウェイを通過する。ポリシーを用いることで、アクセストークンのチェック処理を行うことができる。

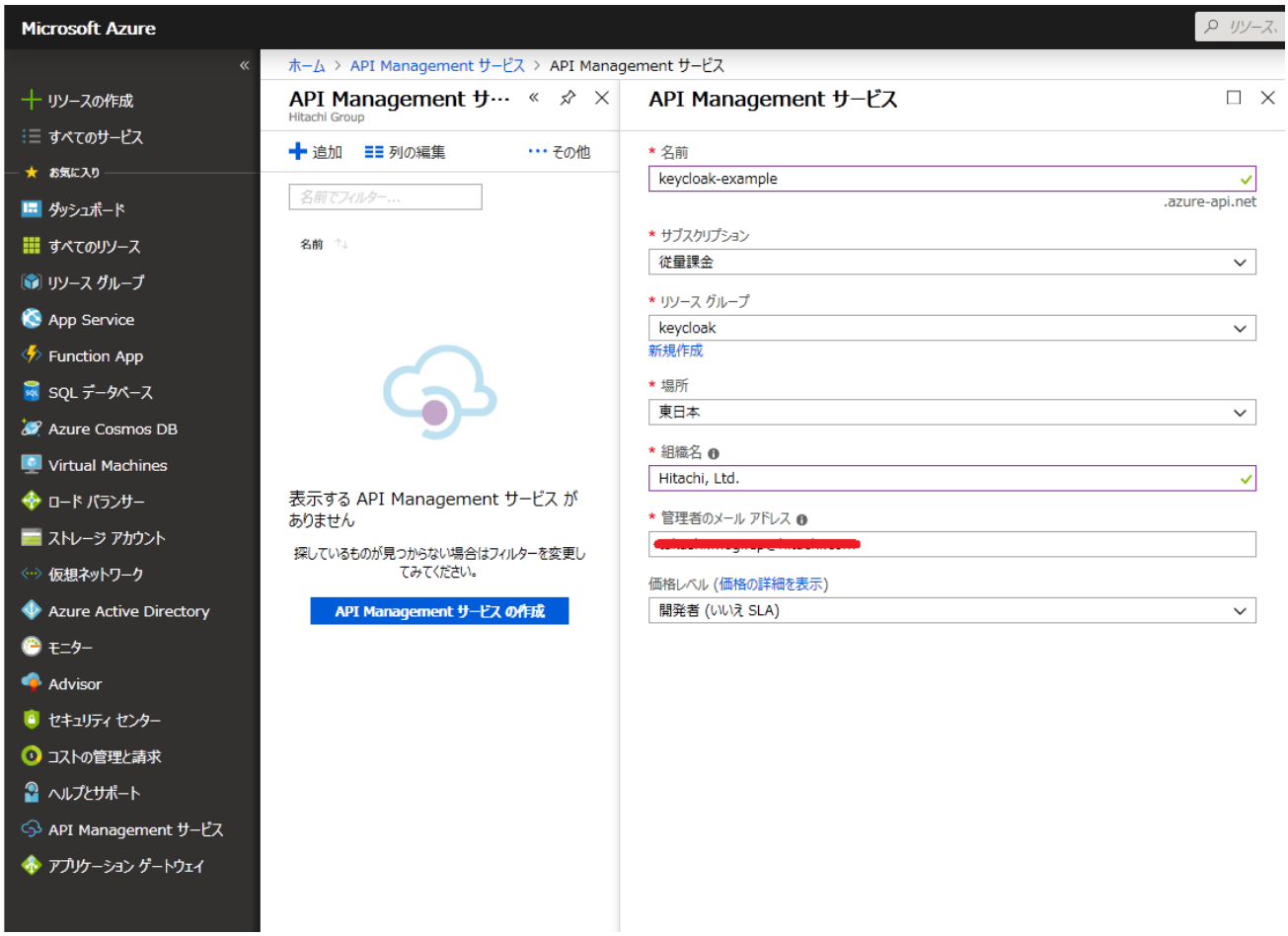
チェック処理として最も一般的なのは、アクセストークンの有効性チェックである。つまり、アクセストークンが改ざんされていないか、また失効していないかの確認である。改ざんチェックについては、JWT の署名を検証することで行うことができるが、失効確認については、アクセストークンを管理している Keycloak に問い合わせる必要がある。Keycloak は失効確認のための標準的なインターフェース(RFC7662: OAuth 2.0 Token Introspection、通称 Token Introspection)をサポートしているため、API ゲートウェイからは RFC7662 のインターフェースをコールすることになる。API Management においては、ポリシーの簡単なカスタマイズにより RFC7662 のインターフェースのコールが可能である。

さらには、JWT 形式であるアクセストークンの各要素に格納された値に基づいて API のアクセスを拒否したり許可したりするようなチェックもポリシーで可能である。

4 章以降では、実際に Azure API management と Keycloak を組み合わせて OAuth2.0 で API を保護する方法を紹介する。Azure API Management と Keycloak をそれぞれ立上げたうえで、ポリシーを用いてアクセストークン行使時のチェックをできるように設定していく。

4. Azure API Management サービスの作成

まずは Azure API Management サービスを作成する。Microsoft Azure Portal の**すべてのサービス**から、**API Management サービス**を選択し、API Management サービスを作成する。



ここで入力した名前をもとに、API ゲートウェイの URL(~.azure-api.net)および開発者ポータルの URL(~.portal.azure-api.net)が割り当てられる。必要に応じて独自のドメインを利用することもできるが、今回はデフォルトのドメインを利用した。

API Management サービスの作成が完了すると、API の作成や設定変更が可能となる。またデフォルトで Echo API という API が追加されているが、これは Microsoft が提供しているサンプル用の API である。今回はこの API を利用して設定を行っていく。

5. Keycloak サーバの構築・設定

まず、Keycloak そのもののインストールについては、公式ドキュメント[4]を参照されたい。Azure API Management との連携を行うにあたり、API ゲートウェイ・Keycloak 間のトークン検証のための通信を必ず HTTPS で行う必要がある。Keycloak では、デフォルトの設定で HTTPS が有効とならないため追加の設定が必要となる。今回は Keycloak 設定ではなく、Microsoft Azure のロードバランサー機能を利用して HTTPS 化したので、その設定方法を下記に示す。

5.1. Azure Application Gateway による SSL 終端

Keycloak サーバで HTTPS を有効化するためには Keycloak 側の設定で行うことも可能だが、今回は Azure Application Gateway を利用して SSL 終端を行った。Azure Application Gateway は Web アプリケーション向けのロードバランサーであり、Web アプリケーションの負荷分散や、SSL 終端処理といった機能を持っている。また、Azure Application Gateway を利用すると、外部から接続可能な URL が作成できる。アプリケーションゲートウェイの設定方法に関しては、Microsoft のドキュメント[5]を参照。

また、このドキュメントでは自己証明書を利用しているため、Azure API Management 側に CA 証明書を追加する必要がある。CA 証明書の追加に関してはこちら[6]を参照。

5.2. Keycloak の Proxy 設定

HTTPS の設定は、Azure のアプリケーションゲートウェイによってプロキシされているため、Keycloak 側で設定する必要はない。しかし、Keycloak をプロキシの背後で動作させるためには、下記に示すように http-listener に proxy-address-forwarding の設定を追加する必要がある。

---

[4] https://www.keycloak.org/docs/latest/server_installation/index.html

[5] https://docs.microsoft.com/ja-jp/azure/application-gateway/tutorial-ssl-cli

[6] https://docs.microsoft.com/ja-jp/azure/api-management/api-management-howto-ca-certificates

```
<subsystem xmlns="urn:jboss:domain:undertow:6.0">
   <buffer-cache name="default"/>
   <server name="default-server">
      <ajp-listener name="ajp" socket-binding="ajp"/>
      <http-listener name="default" socket-binding="http" redirect-socket="https
"
          proxy-address-forwarding="true"/>
      ...
   </server>
   ...
</subsystem>
```

上記の設定を行い Keycloak を起動させると、Azure Application Gateway を経由して Keycloak にアクセスが可能になる。

5.3. Keycloak の設定

Keycloak のアクセスが可能になったら、必要な情報を設定していく。

- Realm 作成

  今回は azure という名前で Realm を作成した。

- クライアント追加

  設定値は下記の通り。なお、他の部分は省略してデフォルト値のままとなっている。

#	項目名	設定値
1	Client ID	apim
2	Client Protocol	openid-connect
3	Access Type	confidential
4	Standard Flow Enabled	ON
5	Valid Redirect URIs	* (後で設定するため、適当な値を入れておく)

- ユーザ追加

  ログインするための適当なユーザを追加しておく。

ここまでで、Keycloak の設定は完了となる。

6. API ゲートウェイでのアクセストークンのチェック

次に API ゲートウェイでアクセストークンをチェックするために API のポリシーを変更していく。Azure API Management のポリシーは、初期段階ではすべての通信に応答する設定となっているため、Keycloak から発行された有効なアクセストークン付のリクエストにのみ応答するように設定を行う。アクセストークンの有効性を調べる方法として、Microsoft が提供している validate-jwt というポリシーを利用する方法と、RFC 7662 Token Introspection を利用する方法がある。

デフォルトで追加された Echo API に対して validate-jwt ポリシーおよび Token Introspection を実行するポリシーの追加方法を下記に示す。

6.1. validate-jwt を利用したアクセストークンの検証

Keycloak から取得できるアクセストークンは、署名付き JWT が利用されている。Azure API Managemnt では validate-jwt というポリシーを利用することで、JWT の署名、有効期限および Claim の値を検証することができるため、有効なアクセストークンが付与されたリクエストのみ API へアクセス可能とすることができる。本ポリシーを用いた場合は、Keycloak への通信は発生しないが、トークンが Keycloak 側で失効した場合は検知することができないことに注意する。

ポリシーを有効にするためには対象の API でポリシーエディタを開き、下記に示す validate-jwt 要素を追加する。なお、<Azure Application Gateway のホスト名>と表記されている部分は、作成した Azure Application Gateway のホスト名に変更すること。

```
<policies>
  <inbound>
    <base />
    <validate-jwt header-name="Authorization"
                  require-scheme="Bearer"
                  require-signed-tokens="true"
                  require-expiration-time="true"
                  failed-validation-httpcode="401"
                  failed-validation-error-message="Unauthorized">
      <openid-config url="https://<Azure Application Gateway のホスト名>/auth/azure/.well-known/openid-configuration" />
      <issuers>
        <issuer>https://<Azure Application Gateway のホスト名>/auth/azure</issuer
```

```
>
      </issuers>
    </validate-jwt>
  </inbound>
...
</policies>
```

openid-config で指定している url の値には、OpenID Connect の設定を取得するための URL であるメタデータエンドポイントの URL を指定する。Keycloak の場合は /auth/azure/.well-known/openid-configuration となる。また、今回は Azure Application Gateway を利用して、Keycloak へのアクセスをプロキシしているため、メタデータエンドポイント URL のホスト名には Azure Application Gateway のホスト名を指定する必要がある。

上記の設定により、アクセストークンに指定された JWT の値が検証され、アクセス可否が判断されるようになる。なお、より詳細なアクセス制御が必要な場合は Azure API Management のドキュメント(JWT を検証する[7])を参照。

6.2. RFC 7662(Token Introspection)を利用したアクセストークンの検証

JWT の検証を行うと、IdP にアクセスせずにトークンの有効性をある程度確認できるが、より厳密な検証を行う場合は RFC 7662 で定義されている Token Introspection を利用することができる。Azure API Management では、Token Introspection ポリシーが用意されているわけではないが、外部へのリクエスト送信などを組み合わせて Token Introspection によるアクセストークンの検証を実現することができる。

Token Introspection を実行するポリシーは、Azure API Management のドキュメント(Azure API Management サービスからの外部サービスの使用[8])にサンプルが載っているので、それを参考に設定を行う。

---

[7] https://docs.microsoft.com/ja-jp/azure/api-management/api-management-access-restriction-policies#ValidateJWT

[8] https://docs.microsoft.com/ja-jp/azure/api-management/api-management-sample-send-request

```
<inbound>
  <set-variable name="token" value="@(context.Request.Headers.GetValueOrDefault
("Authorization","scheme param").Split(' ').Last())" />

  <send-request mode="new" response-variable-name="tokenstate" timeout="20" igno
re-error="true">
    <set-url>https://<Azure Application Gateway のホスト名>/auth/azure/protocol/o
penid-connect/token/introspect</set-url>
    <set-method>POST</set-method>
    <set-header name="Authorization" exists-action="override">
      <value>basic Base64(クライアント ID:クライアントシークレット)</value>
    </set-header>
    <set-header name="Content-Type" exists-action="override">
      <value>application/x-www-form-urlencoded</value>
    </set-header>
    <set-body>@($"token={(string)context.Variables["token"]}")</set-body>
  </send-request>

  <choose>
    <when condition="@((bool)((IResponse)context.Variables["tokenstate"]).Body.A
s<JObject>()["active"] == false)">
        <return-response response-variable-name="existing response variable">
            <set-status code="401" reason="Unauthorized" />
            <set-header name="WWW-Authenticate" exists-action="override">
                <value>Bearer error="invalid_token"</value>
            </set-header>
        </return-response>
    </when>
  </choose>
  <base />
</inbound>
```

Keycloak での Token Introspection のパスは/auth/azure/protocol/openid-connect/token/introspect となる。また、Token Introspection の実行にはクライアントの認証情報が必要となるため、クライアント ID、クライアントシークレットを利用した Basic 認証ヘッダも必要となる。

以上の設定により、API を OAuth2.0 で保護できるようになっている。

7. 動作確認

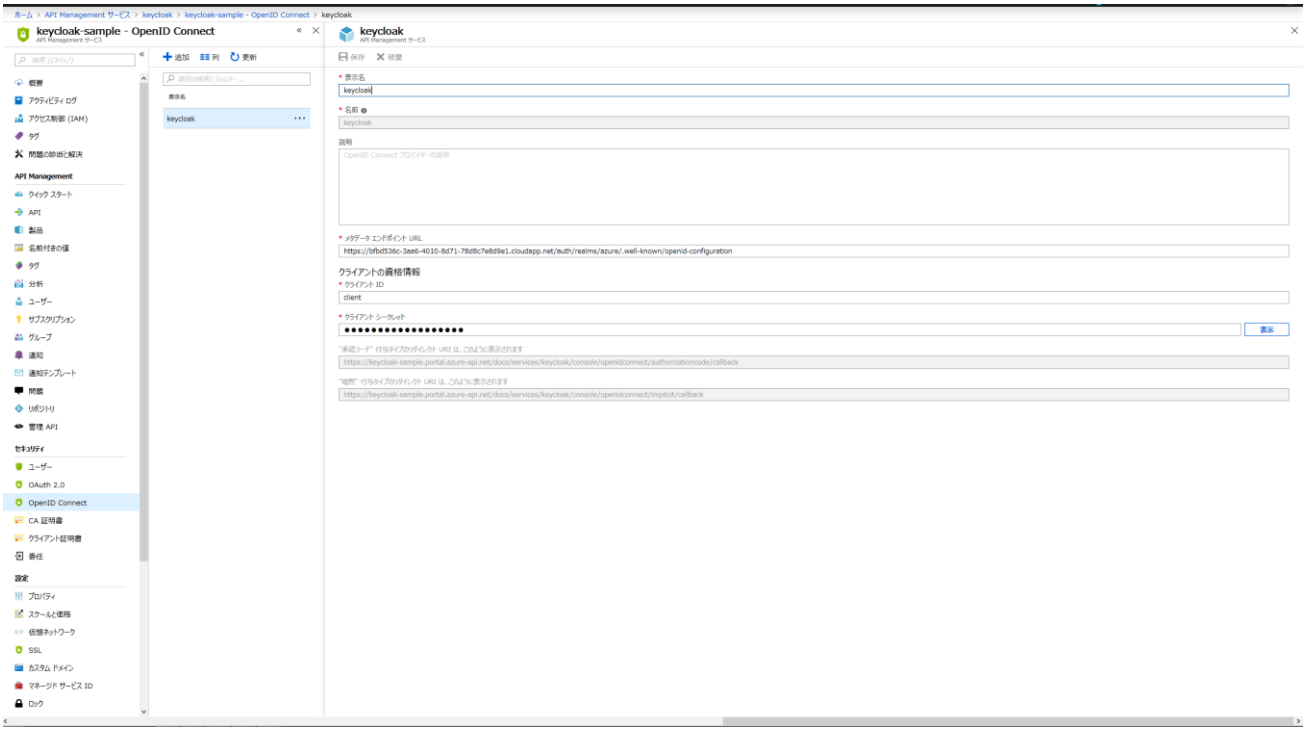
実際に動作確認をしていく。今回は、API Management の開発者ポータルから API をコールすることで動作確認を行う。

7.1. Azure API Management サービスへの IdP 情報の登録

自開発者ポータルにある API ドキュメントで IdP(OAuth/OpenID Connect のサーバ、今回の場合は Keycloak)を利用できるようにするための情報を登録する。この情報を登録しておくと、開発者ポータルの API ドキュメントからアクセストークンの発行、アクセストークン付リクエストの送信が可能になる。設定方法を下記に示す。

7.1.1. IdP 情報の登録

まずは、Azure API Management のセキュリティ -> OpenID Connect を選択して、新規に IdP の情報を追加する。



設定する情報は下記の通り。

- 表示名

  登録する IdP の表示名。この表示名は開発者ポータルの API ドキュメントに表示される。

- メタデータエンドポイント URL

  OpenID Connect の設定を取得するための URL である、メタデータエンドポイントの URL を指定する。Keycloak の場合は/auth/azure/.well-known/openid-configuration となる。また、今回は Azure Application Gateway を利用して、Keycloak へのアクセスをプロキシしているため、メタデータエンドポイント URL のホスト名には Azure Application Gateway のホスト名を指定する必要がある。

- クライアント ID、クライアントシークレット

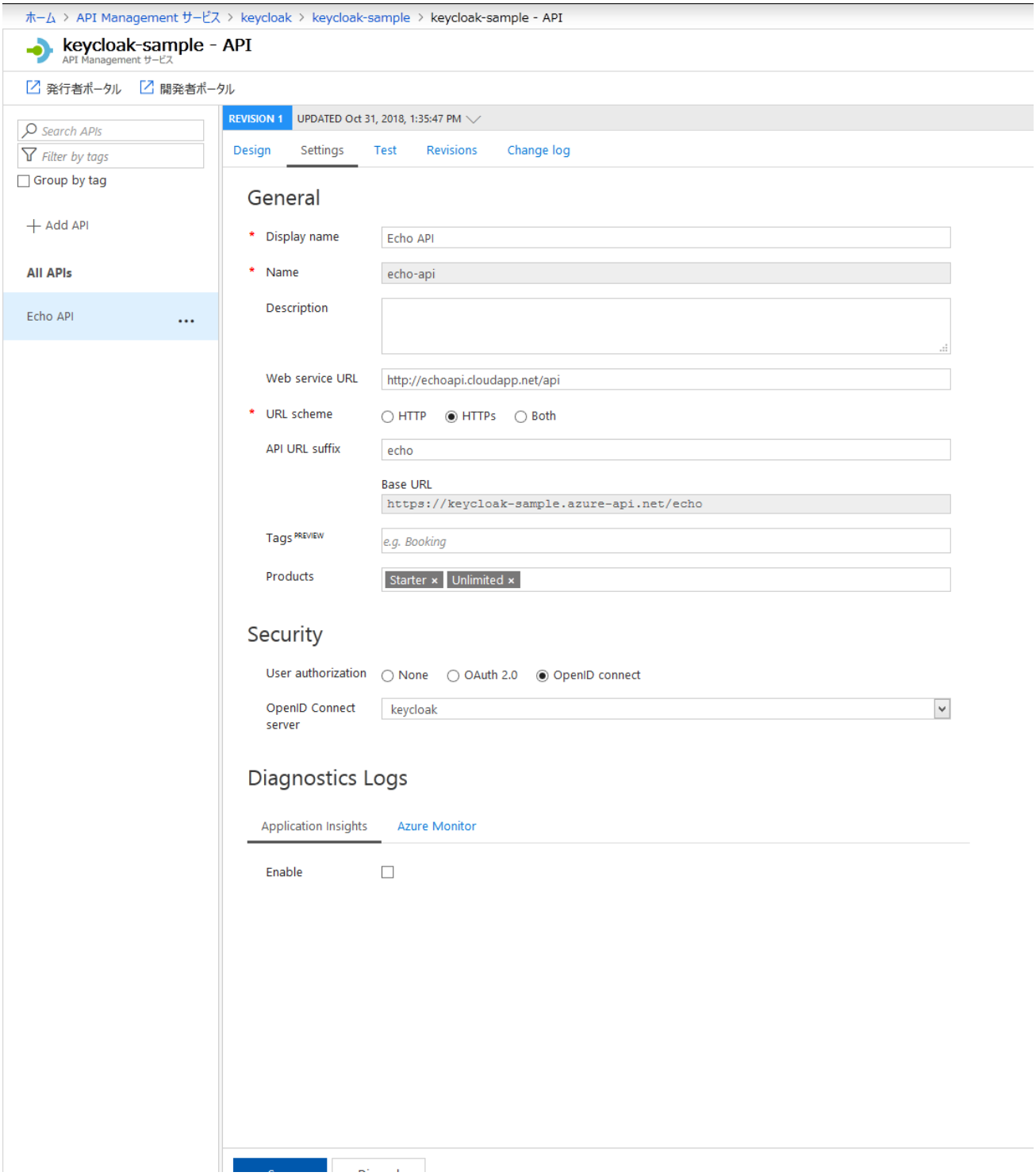
  Keycloak に登録したクライアントの ID(apim)および生成されたシークレットを入力する。

また、IdP 情報追加時に「“認証コード”付与タイプのリダイレクト URL」「“暗黙”付与タイプのリダイレクト URL」という二つの URL が表示される。これは、API ドキュメントで認可フローを実行した際のリダイレクト URL となる。これらの値を、Keycloak に事前に登録しておいたクライアントの Valid Redirect URIs に設定する。

以上で、Azure API Management への IdP 情報の登録が完了する。

7.1.2. API の設定

次に上記で設定した IdP 情報を API で利用できるようにする。OpenID Connect によるアクセスコントロールを適用する API(今回はデフォルトで追加されている Echo API)の Settings を開く。そして、Security の Open ID Connect を選択すると、先ほど登録した IdP の情報がプルダウンで選択できるようになる。



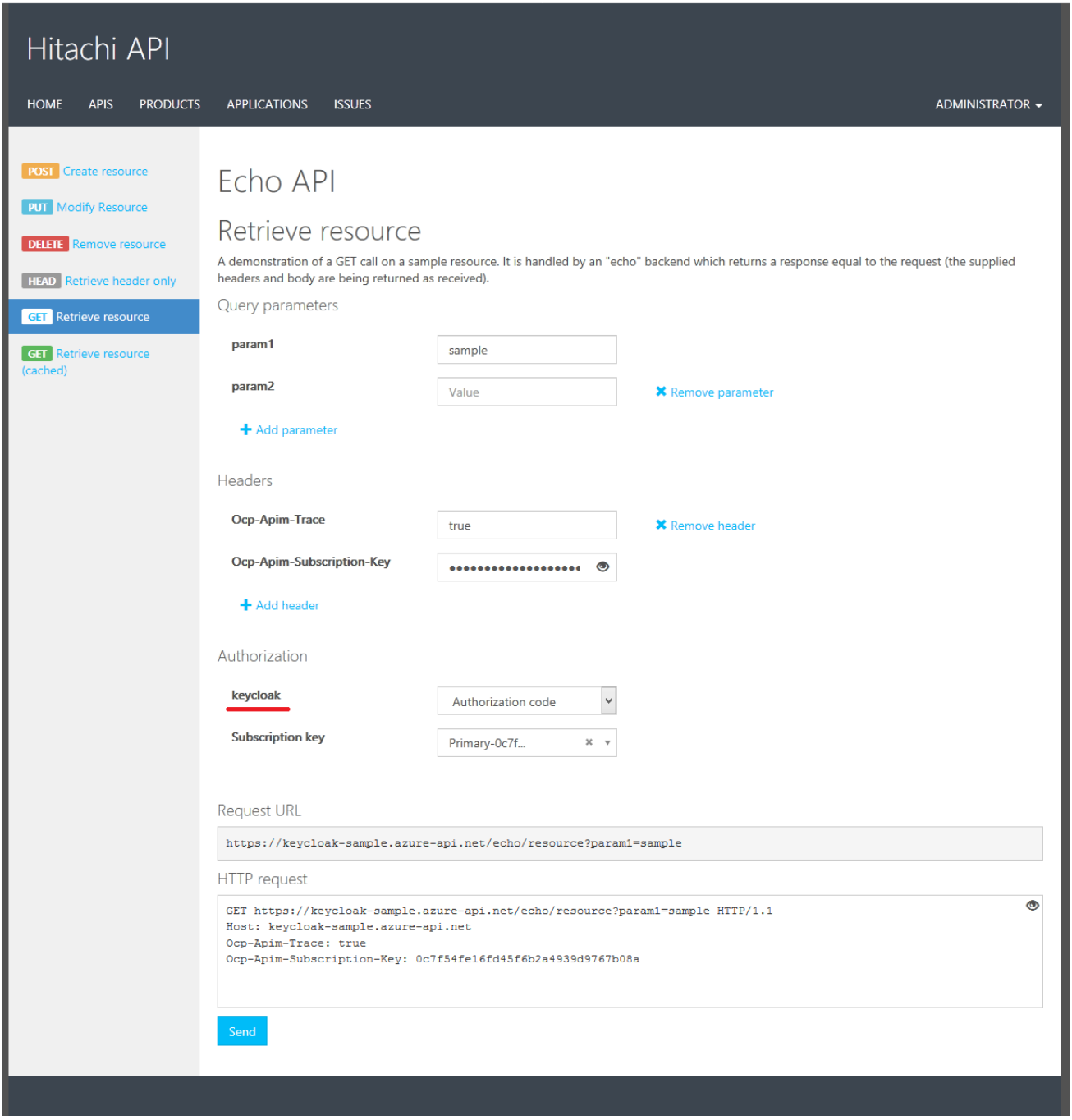
ここで、IdP として登録した Keycloak を選択して設定を完了する。これで、開発者ポータルの API ドキュメントから、Keycloak を選択してアクセストークンの発行、アクセストークン付きリクエストの実行が可能になる。

以上で、Azure API Management と Keycloak の連携設定は完了し、OpenID Connect によるアクセス制御が可能となる。

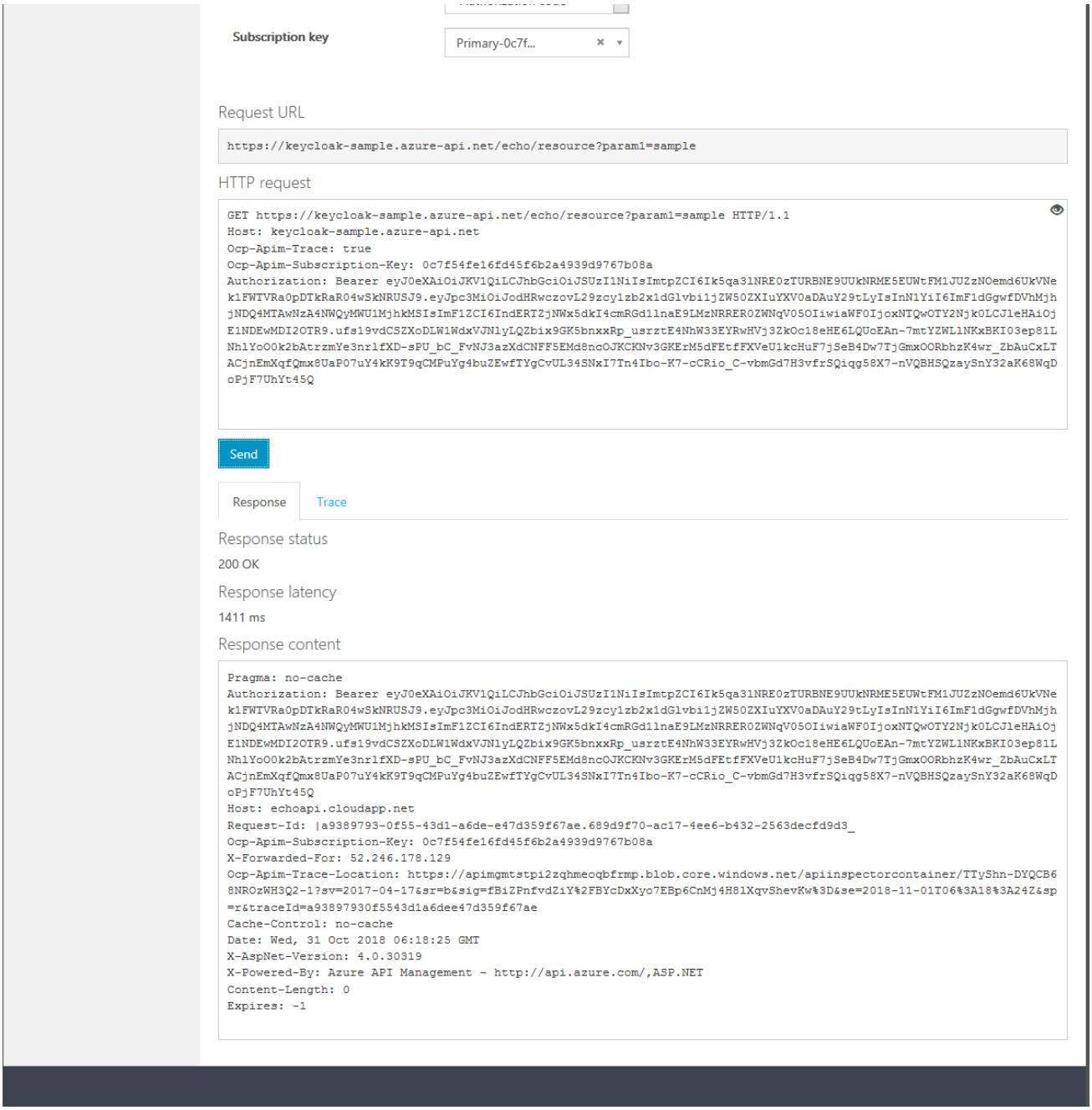
7.2. API リクエストの実行

ここからは実際の API アクセスを開発者ポータルより実行し、設定が正しく行われているかを確認していく。

7.2.1. API ドキュメントからの API 実行



ここで認証方法として Keycloak での Authorization Code を選択して API を実行する。API を実行しようとすると、Keycloak の認証画面が表示され、ログインに成功するとアクセストークンが発行される。この状態で API を実行すればアクセストークン付きのリクエストが実行され、Keycloak との連携設定が完了していれば 200 OK のレスポンスが帰ってくる。



7.2.2. コンソールからの確認

次にコマンドなどを利用して、有効なアクセストークンが付与されていないリクエストが失敗することを確認する。curl で実行する場合は下記の通り。

```
# curl -v -H 'Ocp-Apim-Subscription-Key: <リクエスト時に使用したサブスクリプションキー>' \
 https://<Azure API Management のサービス名>.azure-api.net/echo/resouce
```

リクエストを実行する際に、サブスクリプションキーというものが必要となる。このサブスクリプションキーは、API をサブスクリプション(つまり、API プロダクトを購入した人)を判別するためのもので、開発者ポータルから登録すると発行される。API ドキュメントから API を実行すると、Ocp-Apim-Subscription-Key という HTTP ヘッダに自動で付与されるので、この値をコピーしてくる。

上記のリクエストを実行すると、有効なアクセストークンが付与されていないため 401 Unauthorized のレスポンスが帰ってくることが確認できる。

以上で、Azure API Management と Keycloak の連携が確認できた。

8. おわりに

以上のように、Azure 上に OAuth に対応した API 管理を容易に実現できる。本書では触れていないが、さらに高度な設定も可能である。例えば validate-jwt ポリシーのほうでは、アクセストークン中の属性(scope 等)に基づき API アクセスの拒否許可を設定することもできる。詳細については、公式ドキュメント[9]を参照頂きたい。

また、エンタープライズ用途には、Keycloak の商用版である Red Hat Single Sign-On の利用を推奨する。日立では Keycloak の開発に積極的に関わるなどコミュニティへの貢献を行っており、そこで得られた知見を元に Red Hat Single Sign-On の商用サポートを提供している。Azure 上で OAuth に対応した API 管理を実現する際には、ぜひご相談頂きたい。

[9] https://docs.microsoft.com/ja-jp/azure/api-management/api-management-access-restriction-policies#ValidateJWT

9. 付録

9.1. 価格レベル

9.1.1. 各プランと機能の差分について

API Management には価格レベルがそれぞれ設定されており、利用できる機能群も異なる。以下に機能対応表を示す。

Developer レベルについて、非運用環境のユースケースと評価のためのものであることに注意が必要である。 SLA が提供されないため、開発環境やテスト環境の用途で用いることを前提としている。

	Consumption	Developer	Basic	Standard	Premium
Azure AD 統合	いいえ	はい	いいえ	可能	はい
Virtual Network (VNet) のサポート	いいえ	はい	いいえ	いいえ	はい
複数リージョンのデプロイ	いいえ	いいえ	いいえ	いいえ	はい
複数のカスタム ドメイン名	いいえ	いいえ	いいえ	いいえ	はい
開発者ポータル	いいえ	可能	はい	はい	はい
ビルトイン キャッシュ	いいえ	可能	はい	はい	はい
ビルトイン分析	いいえ	可能	はい	はい	はい
セルフホステッド ゲートウェイ	いいえ	はい	いいえ	いいえ	はい
SSL 設定	はい	はい	はい	はい	はい
外部キャッシュ	はい	はい	はい	はい	はい
クライアント証明書認証	はい	はい	はい	はい	はい
バックアップと復元	いいえ	可能	はい	はい	はい
Git による管理	いいえ	可能	はい	はい	はい
ダイレクト管理 API	いいえ	可能	はい	はい	はい
Azure Monitor のログとメトリック	いいえ	可能	はい	はい	はい
静的 IP	いいえ	可能	はい	はい	はい

9.1.2. Consumption プラン

従来まで、ゲートウェイとしてのリソースを確保して提供する固定額の価格プランのみ提供していたが、使用料に基づいた課金体系である Consumption プランのサポートを 2019 年 5 月より一般提供を開始した(2018 年 12 月にプレビュー提供開始)。現在は利用できる Azure リージョンが限定されている状況であるが、API Management を使用できるすべてのリージョンで順次使用可能になる予定だ。

この Consumption プランにより、それまで要望の多かった以下のようなシナリオに対応できるようになった。

- Azure Functions や Logic Apps など、サーバーレス テクノロジを使用して実装されているマイクロサービス向けの API ゲートウェイ。
- Service Bus キューおよびトピック、Azure Storage など、サーバーレスの Azure リソースに対して、簡素で安全なファサードを提供する API ゲートウェイ。
- 通常はトラフィックが少ないが急増することもあるバックエンド向けの API ゲートウェイ。
- テスト環境における API ゲートウェイ。

利用に際して以下の点に注意する必要がある。

- 既出の機能一覧のとおり、利用できる機能が制限されているため、想定するシナリオで要件を満たしているか確認する必要がある。
- バッファされるペイロードのサイズや、リクエスト URL のサイズ、後述するポリシーのドキュメントサイズに制限がある。

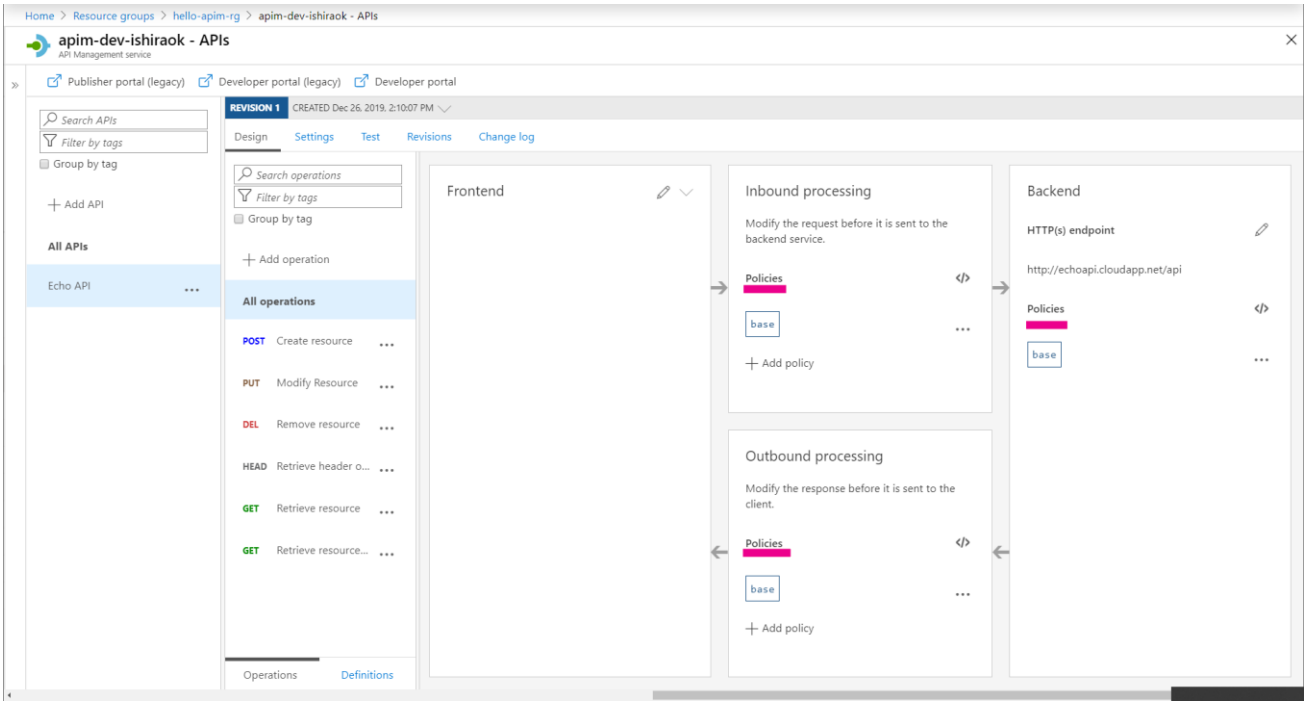
以上で価格レベルの概要を解説したが、詳細については、公式ドキュメント（https://azure.microsoft.com/ja-jp/pricing/details/api-management/）を参照されたい。

9.2. ポリシーによるカスタマイズ

9.2.1. ポリシーの概要

ポリシー機能は、Azure API Management のカスタマイズのための強力な機能である。ポリシーは、クライアントからの要求、それに対する応答に対して順に実行される一連のセットであり、API の動作を変更することができる。下記は、Azure ポータルにおけるポリ

シー設定画面である。フロントエンドからバックエンドへの要求、応答に対してそれぞれインバウンドプロセス、アウトバウンドプロセスを挟み込むことができる。



代表的な機能としては以下の機能が挙げられる。

- XML 形式から JSON 形式への変換
- 開発者からの呼び出しの回数を制限する、呼び出しレート制限
- OAuth のアクセストークンのチェック

挟み込む箇所としては、次の 4 か所でポリシーを構成できる。

- inbound
- backend
- outbound
- on-error

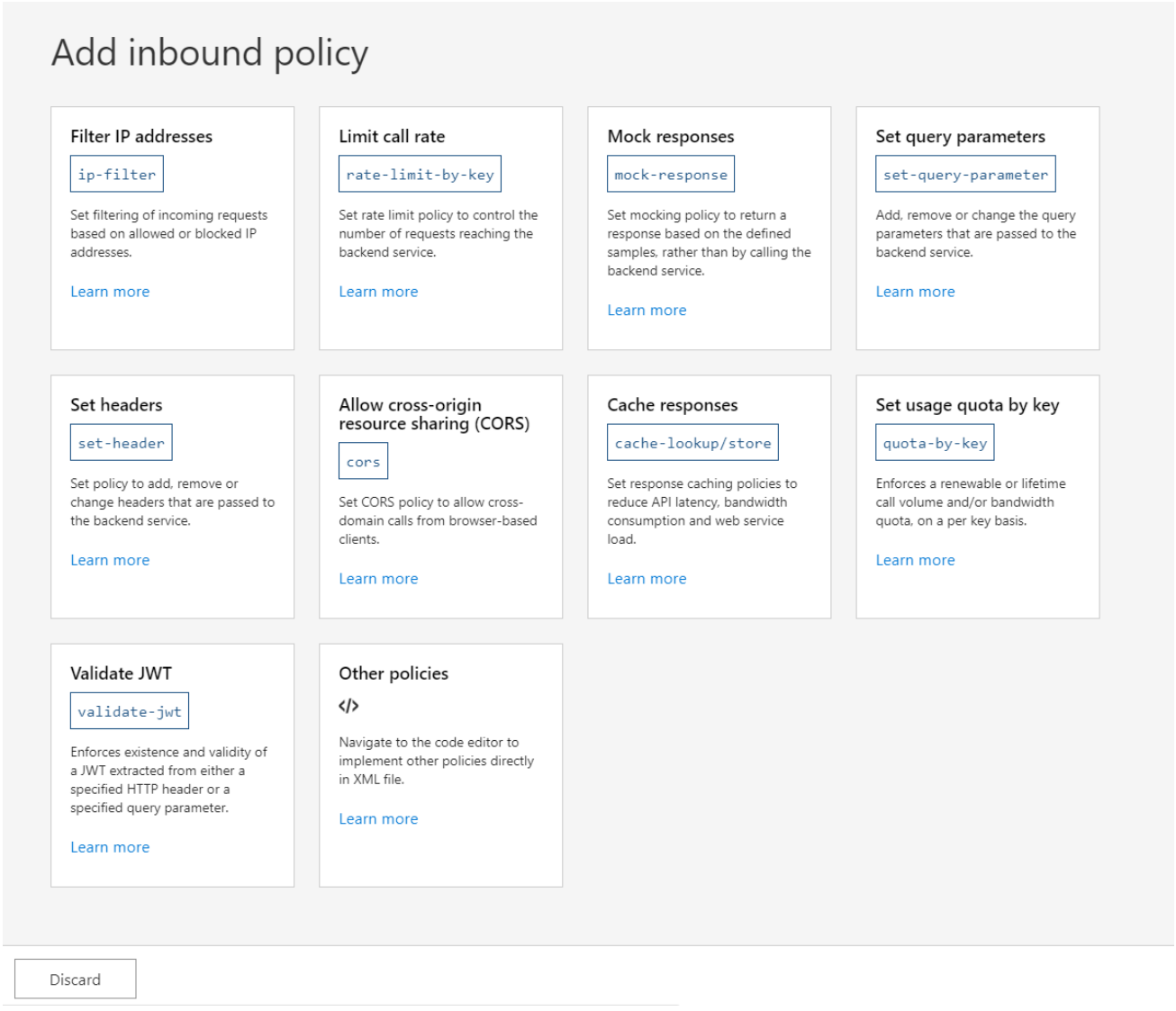
これらは以下のように XML によって定義されている。

```
<policies>
  <inbound>
    <!-- リクエストに適用されるステートメント -->
  </inbound>
  <backend>
    <!-- バックエンドサービスに転送される前に適用されるステートメント -->
  </backend>
  <outbound>
    <!-- レスポンスに適用されるステートメント -->
```

```
  </outbound>
  <on-error>
    <!-- エラー条件があった場合に適用されるステートメント -->
  </on-error>

</policies>
```

上記の画面では、この定義を GUI で構成することができる。例えば、Inbound Processing の“Add Policy”をクリックすると以下のように標準機能として提供されるポリシーを選択して追加することができる。これらのうち、代表的なポリシーについては、付録「Azure API Management の代表的なポリシー」に紹介しているので参照されたい。



9.2.2. Azure API Management の代表的なポリシー

本章では代表的なポリシーと定義リファレンスを示す。そのほか全ての標準ポリシーについては、API Management ポリシーリファレンス(https://docs.microsoft.com/ja-jp/azure/api-management/api-management-policies)を参照頂きたい。

アクセス制限ポリシー

ある API クライアントの乱用による他の API クライアントへの悪影響を与えないように、アクセスを通常利用の範囲に制限する手段を提供する。以下に主なポリシー機能を紹介する。

1. 呼び出しレート“rate-limit”
   呼び出し数を毎分最大 10 回に制限する、などで、アクセス急増を防ぐ。
2. クォータ“quota”
   1 か月あたりの呼び出しの合計を 1,000,000 回＆帯域幅を 10,000 KB に制限する、などで、呼び出しの総量に条件を与える。

- HTTP ヘッダーを確認する – HTTP ヘッダーの存在と値の両方、またはそのどちらかを適用する。
- 呼び出しレートをサブスクリプション別に制限する – 呼び出しレートをサブスクリプションに基づいて制限することで、API の使用量の急増を防ぐ。
- 呼び出しレートをキー別に制限する – 呼び出しレートをキーに基づいて制限することで、API の使用量の急増を防ぐ。
- 呼び出し元 IP を制限する – 特定の IP アドレスとアドレス範囲の両方、またはそのどちらかからの呼び出しをフィルター処理（許可/拒否）する。
- 使用量のクォータをサブスクリプション別に設定する – 更新可能な呼び出しまたは有効期間中の呼び出しのボリュームと帯域幅クォータの両方またはそのどちらかをサブスクリプションに基づいて適用する。
- 使用量のクォータをキー別に設定する – 更新可能な呼び出しまたは有効期間中の呼び出しのボリュームと帯域幅クォータの両方またはそのどちらかをキーに基づいて適用する。
- JWT を検証する – 指定された HTTP ヘッダーまたは指定されたクエリ パラメーターから抽出した JWT の存在と有効性を適用する。

認証ポリシー

- 基本認証 -基本認証を使用してバックエンド サービスで認証する。
- クライアント証明書による認証 -クライアント証明書を使用してバックエンド サービスで認証する。
- マネージド ID による認証 - マネージド ID を使用してバックエンド サービスで認証する。

キャッシュ ポリシー

- キャッシュから取得 - キャッシュを検索して、キャッシュに格納された有効な応答があればそれを返す。
- キャッシュに格納 - 指定されたキャッシュ制御の構成に従って応答をキャッシュに格納する。
- キャッシュから値を取得 - キャッシュされたキー別の項目を取得する。
- 値をキャッシュに格納 - 項目をキー別にキャッシュに格納する。
- キャッシュから値を削除 - キー別にキャッシュ内の項目を削除する。

変換ポリシー

- JSON から XML への変換 - 要求本文または応答本文を JSON から XML に変換する。
- XML から JSON への変換 - 要求本文または応答本文を XML から JSON に変換する。
- 本文内の文字列の検索および置換 - 要求または応答の部分文字列を検索して、別の部分文字列に置き換える。
- コンテンツ内の URL のマスク - ゲートウェイを経由して同じリンクをポイントするよう、応答本文のリンクを書き換える（マスクする）。
- バックエンド サービスの設定 -受信要求のバックエンド サービスを変更する。
- 本文の設定 - 着信要求と発信要求のメッセージ本文を設定する。
- HTTP ヘッダーの設定 - 既存の応答または要求ヘッダーまたは新しい応答または要求ヘッダーに値を割り当てる。
- クエリ文字列パラメーターの設定 - 要求クエリ文字列パラメーターの追加、値の置換、または削除を行う。
- URL の書き換え - 要求 URL をパブリックな形式から Web サービスで想定されている形式に変換する。
- XSLT を使用した XML の変換 - 要求本文または応答本文に含まれる XML に XSL 変換を適用する。

これらのポリシー機能を用いて、バックエンドのサービスに変更なく、クライアントアプリケーションに対して、適切なゲートウェイ機能を提供していくことができる。また、これらの変更はポリシー定義を行うだけで実装することができるため、迅速にかつ安全に挙動の変更を行うことができる。