Administering the API Gateway, 3scale 2.12

July 21, 2022

CHAPTER 12. INTEGRATING 3SCALE WITH AN OPENID CONNECT IDENTITY PROVIDER

To authenticate API requests, 3scale can integrate with an identity provider that complies with the OpenID Connect specification. For full compatibility with 3scale, the identity provider can be Red Hat Single Sign-On (RH-SSO) or a third-party identity provider that implements default Keycloak client registration. For compatibility with the 3scale API gateway (APIcast), any identity provider that implements OpenID Connect can be used.

The foundation for OpenID Connect is the OAuth 2.0 Authorization Framework (RFC 6749). OpenID Connect uses a JSON Web Token (JWT) (RFC 7519) in an API request to authenticate that request. When you integrate 3scale with an OpenID Connect identity provider, the process has two main parts:

- APIcast parses and verifies the JWT in the request. If successful, APIcast authenticates the identity of the API consumer client application.
- The 3scale Zync component synchronizes 3scale application details with the OpenID Connect identity provider.

3scale supports both of these integration points when RH-SSO is the OpenID Connect identity provider. See the supported version of RH-SSO on the Supported Configurations page. However, RH-SSO is not a requirement. You can use any identity provider that supports the OpenID Connect specification and the default Keycloak client registration. APIcast integration is tested with RH-SSO and ForgeRock.

The following sections provide information and instructions for configuring 3scale to use an OpenID Connect identity provider:

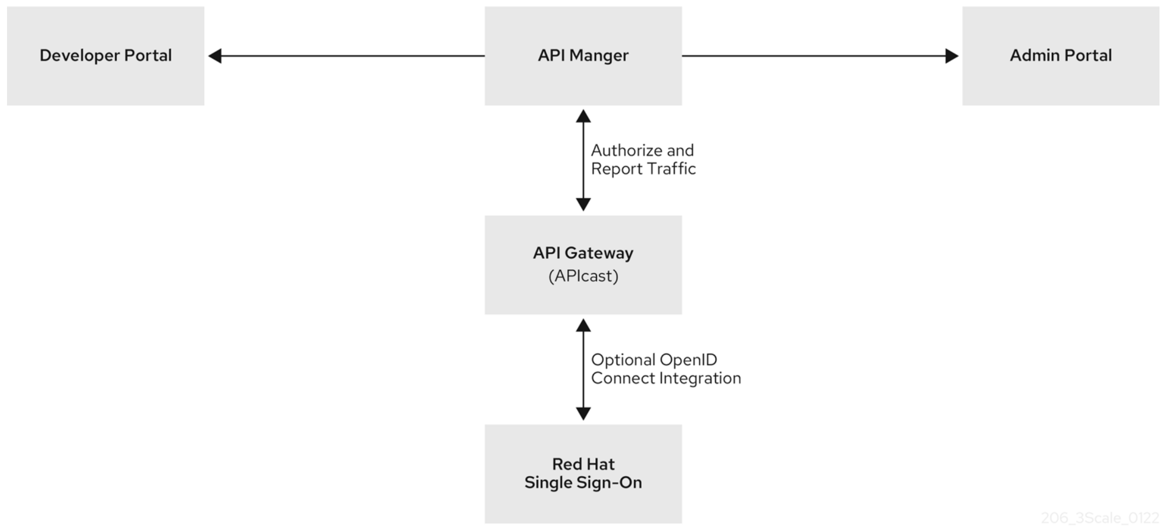
- Overview of integrating 3scale and an OpenID Connect identity provider
- How APIcast processes JSON Web Tokens
- How 3scale Zync synchronizes application details with OpenID Connect identity providers
- Integrating 3scale with Red Hat Single Sign-On as the OpenID Connect identity provider
- Integrating 3scale with third-party OpenID Connect identity providers
- Testing 3scale with OpenID Connect identity providers
- Example of a 3scale integration with and OpenID Connect identity provider

12.1. OVERVIEW OF INTEGRATING 3SCALE AND AN OPENID CONNECT IDENTITY PROVIDER

Each main 3scale component participates in authentication as follows:

- APIcast verifies the authenticity of the authentication tokens presented by API consumer applications. In a default 3scale deployment, APIcast can do this because it implements auto discovery of an API product's OpenID Connect configuration.
- API providers use the Admin Portal to set up authentication flows.
- If a 3scale-managed API does not authenticate requests with standard API keys or with application identifier and key pairs, then API providers must integrate 3scale with an OpenID Connect identity provider. In the figure below, the OpenID Connect identity provider is Red Hat Single Sign-On (RH-SSO).
- With authentication configured and a live Developer Portal, API consumers use your Developer Portal to subscribe to an application plan that provides access to a particular 3scale API product.
- When OpenID Connect is integrated with 3scale, subscription triggers the configured flow for API consumer applications to obtain JSON Web Tokens (JWTs) from the OpenID Connect identity provider. An API provider specifies this flow when configuring the API product to use OpenID Connect.

Figure 12.1. shows the main 3scale components with an OpenID Connect identity provider



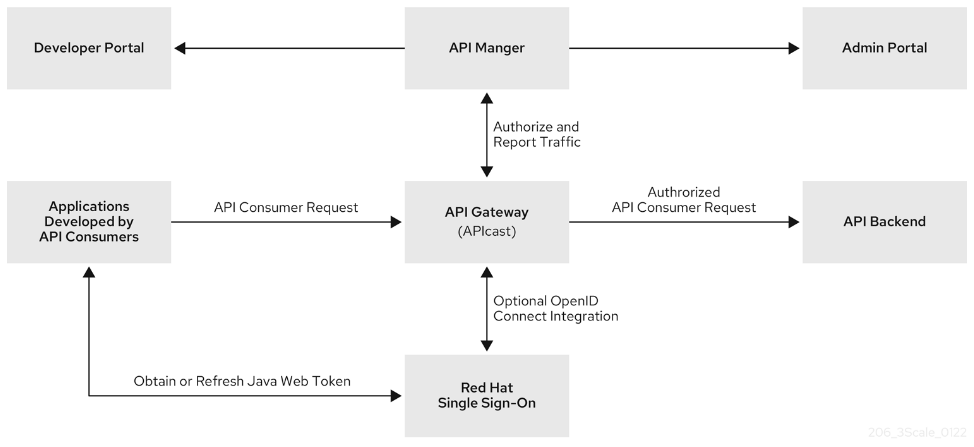
After subscribing to an application plan, an API consumer receives authentication credentials from the integrated OpenID Connect identity provider. These credentials enable authentication of requests that an API consumer application sends to an upstream API, which is the API that is provided by the 3scale API product that the API consumer has access to.

Credentials include a client ID and a client secret. An application created by an API consumer uses these credentials to obtain a JSON Web Token (JWT) from the OpenID Connect identity provider. When you configure your 3scale integration with OpenID Connect, you select the flow for how an API consumer application obtains a JWT. In an API consumer application that uses the default **Authorization Code** flow with RH-SSO, the application must do the following:

1. Initiate an OAuth authorization flow with the OpenID Connect identity provider before the first request to the upstream API backend. The authorization code flow redirects the end-user to RH-SSO. The end-user logs in to obtain an authorization code.
2. Exchange the authorization code for a JWT.
3. Receive a JWT from RH-SSO upon authentication.
4. Send an API request that contains the JWT to the upstream API backend.
5. Send subsequent API requests with the same JWT until it expires.
6. Refresh the JWT or send a new request to the OpenID Connect identity provider to obtain a new JWT. Which action is required depends on the OpenID Connect identity provider.

APIcast receives requests from API consumers and checks the JWT in the request. If APIcast verifies the JWT, APIcast sends the request, including the JWT, to the upstream API backend.

Figure 12.2. shows that the OpenID Connect identity provider is RH-SSO but configuration with other OpenID Connect identity providers is possible.


Developer Portal
API Manger
Admin Portal
Authorize and Report Traffic
Applications Developed by API Consumers
API Consumer Request
API Gateway (APIcast)
Authrorized API Consumer Request
API Backend
Optional OpenID Connect Integration
Obtain or Refresh Java Web Token
Red Hat Single Sign-On
206_3Scale_0122

12.2. HOW APICAST PROCESSES JSON WEB TOKENS

APIcast processes each request by checking the JSON Web Token (JWT) that the OpenID Connect identity provider returned when it authenticated an API consumer application. The request contains the JWT in the format that was issued by the integrated OpenID Connect identity provider. The JWT must be in the **`Authorization`** header and it must use the **`Bearer`** schema. For example, the header should look like this:

```
Authorization: Bearer
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL2lkcC5leGFtcGxlL
mNvbSIsInN1YiI6ImFiYzEyMyIsIm5iZiI6MTUzNzg5MjQ5NCwiZXhwIjoxNTM3ODk2MDk0LCJ
pYXQiOjE1Mzc4OTI0OTQsImp0aSI6ImlkMTIzNDU2IiwidHlwIjoiQmVhcmVyIn0.LM2PSmQ0k
8mR7eDS_Z8iRdGta-Ea-pJRrf4C6bAiKz-Nzhxpm7fF7oV3BOipFmimwkQ_-mw3kN--oOc3vU1
RE4FTCQGbzO1SAWHOZqG5ZUx5ugaASY-hUHIohy6PC7dQl0e2NlAeqqg4MuZtEwrpESJW-VnGd
ljrAS0HsXzd6nENM0Z_ofo4ZdTKvIKsk2KrdyVBOcjgVjYongtppR0cw30FwnpqfeCkuATeINN
5OKHXOibRA24pQyIF1s81nnmxLnjnVbu24SFE34aMGRXYzs4icMI8sK65eKxbvwV3PIG3mM0C4
ilZPO26doP0YrLfVwFcqEirmENUAcHXz7NuvA
```

There are a number of open source tools for securely decoding a JWT. Be careful that you do not decode a JWT in a public web tool. In a decoded JWT, you can see that the token has three parts:

- The header provides information about how the token was formed and what algorithm was used to sign the token.
- The payload identifies the API consumer that sent the request. The details can include the read and write actions that this API consumer can perform, an email address for the API consumer, and other information about the API consumer.
- The signature is a cryptographic signature that indicates that the token has not been altered.

APIcast checks the JWT for the following characteristics:

- **Integrity**: Is the JWT being altered by a malicious user? Is the signature valid?
  The JWT contains a signature that the token's receiver can verify to ensure that a known issuer signed the token. This verification also guarantees that its content remains as created. 3scale supports RSA signatures based on public/private key pairs. The issuer signs the JWT by using a private key. APIcast verifies the token by using a public key. APIcast uses OpenID Connect Discovery for getting the JSON Web Keys (JWK) that can be used to verify the JWT signature.
- **Timing**: Is the current time later than the time when the token becomes acceptable for processing? Has the JWT expired? In other words, APIcast checks the JWT **`nbf`** (not before time) and **`exp`** (expiration time) claims.
- **Issuer**: Was the JWT issued by an OpenID Connect identity provider that is known to APIcast? In other words, APIcast verifies that the issuer specified in the JWT is the same issuer that an API provider

configured in the **OpenID Connect Issuer** field. Specification of the issuer is part of the procedure for integrating 3scale and an OpenID Connect identity provider. This is the JWT `iss` claim.

- **Client ID**: Does the token contain a 3scale client application ID that is known to APIcast? This client ID must match a **ClientID Token Claim** that the API provider specified in the procedure for integrating 3scale with the OpenID Connect identity provider. This is the JWT `azp` (authorized party that the JWT was issued to) claim and the `aud` (audience) claim.

If any JWT validation or authorization checks fail, APIcast returns an `Authentication failed` error. Otherwise, APIcast sends the request to the 3scale upstream API backend. The `Authorization` header remains in the request, so the API backend can also use the JWT to check the user and client identity.

12.3. HOW 3SCALE ZYNC SYNCHRONIZES APPLICATION DETAILS WITH OPENID CONNECT IDENTITY PROVIDERS

Zync is a 3scale component that reliably pushes data about 3scale applications to an OpenID Connect identity provider. In this interaction, a 3scale application corresponds to an OpenID Connect identity provider client. In other words, Zync communicates with the OpenID Connect identity provider to create, update, and delete OpenID Connect clients.

Zync implements Keycloak default client registration. The use of this API means that the client representation is specific to Keycloak and RH-SSO. The identity provider returns a client ID and a client secret, which are the authentication credentials for the 3scale application.

Each time 3scale creates, updates, or deletes an application, Zync communicates with the OpenID Connect identity provider to update the corresponding client accordingly. Successful synchronization requires the following settings for a given 3scale product:

- The authentication mechanism is OpenID Connect.
- The **OpenID Connect Issuer Type** is either:
  - **RH-SSO** when Red Hat Single Sign-On is the OpenID Connect identity provider. With this issuer type, Zync sends client registration requests to the Keycloak/RH-SSO default client registration API.
  - **REST API** for other OpenID Connect identity providers. With this issuer type, Zync sends client registration requests as shown in the Zync REST API example.
- A URL such as the following is the **OpenID Connect Issuer**: `http://id:secret@example.com/api_endpoint`.

When deployed to an OpenShift cluster, there are two Zync processes:

- **`zync`** is a REST API that receives notifications from **`system-sidekiq`** and enqueues background jobs to **`zync-que`**. There are notifications for new, updated, and deleted 3scale applications.
- **`zync-que`** processes these background jobs, which communicate with **`system-app`** and with the OpenID Connect identity provider. For example, when RH-SSO is the configured OpenID Connect identity provider, Zync creates, updates and deletes clients in the RH-SSO realm.

12.4. INTEGRATING 3SCALE WITH RED HAT SINGLE SIGN-ON AS THE OPENID CONNECT IDENTITY PROVIDER

As an API provider, you can integrate 3scale with Red Hat Single Sign-On (RH-SSO) as the OpenID Connect identity provider. The procedures documented here are for a 3scale API product that requires OpenID Connect for authenticating API requests.

Part of this procedure is to establish an SSL connection between 3scale Zync and RH-SSO, because Zync communicates with RH-SSO to exchange tokens. If you do not configure the SSL connection between Zync and RH-SSO, the tokens would be open for anyone listening.

3scale 2.2 and later versions support custom CA certificates for RH-SSO with the SSL_CERT_FILE environment variable. This variable points to the local path of the certificates bundle. Integrating 3scale with RH-SSO as the OpenID Connect identity provider consists of configuring the following elements in the following order:

- If RH-SSO does *not* use a certificate issued by a trusted Certificate Authority (CA), you must configure 3scale Zync to use custom CA certificates. This is not required if RH-SSO uses a certificate issued by a trusted CA.
- Configure RH-SSO to have a 3scale client.
- Configure 3scale to work with RH-SSO.

Prerequisites

- The RH-SSO server must be available over **`HTTPS`** and it must be reachable by **`zync-que`**. To test this, you can run **`curl https://rhsso-fqdn`** from within the **`zync-que`** pod, for example:

```
oc rsh -n $THREESCALE_PROJECT $(oc get pods -n $THREESCALE_PROJECT
--field-selector=status.phase==Running -o name | grep zync-que)
/bin/bash -c "curl -v https://<rhsso-fqdn>/auth/realms/master"
```

- OpenShift cluster administrator permissions.
- A 3scale API product for which you want to configure OpenID Connect integration with RH-SSO.

See the following sections for details:

- Configuring 3scale Zync to use custom Certificate Authority certificates
- Configuring RH-SSO to have a 3scale client
- Configuring 3scale to work with RH-SSO

12.4.1. Configuring 3scale Zync to use custom Certificate Authority certificates

This is not required when RH-SSO uses a certificate issued by a trusted CA. However, if RH-SSO does *not* use a certificate issued by a trusted CA you must configure 3scale Zync before you configure RH-SSO to have a 3scale client and before you configure 3scale to work with RH-SSO.

Procedure

1. Replace the placeholders and run the following command to get a certificate chain:

   ```
   echo -n | openssl s_client -connect <rhsso_fqdn>:<rhsso_port>
   -servername <rhsso_fqdn> --showcerts | sed -ne '/-BEGIN
   CERTIFICATE-/,/-END CERTIFICATE-/p' > customCA.pem
   ```

   Some versions of OpenSSL accept **`-showcerts`** instead of **`--showcerts`**. If necessary, modify the command according to the version you are using.

Replace the **`<rhsso_fqdn>`** placeholder with the fully qualified domain name (**`fqdn`**) in human-readable format, for example, **`host.example.com`**.

2. Validate the new certificate with the **`cURL`** command below. The expected response is a JSON configuration of the RH-SSO realm. If validation fails your certificate might not be correct.

```
curl -v https://<secure-sso-host>/auth/realms/master --cacert
customCA.pem
```

3. Gather the existing content of the **`/etc/pki/tls/cert.pem`** file on the **`zync-que`** pod:

```
oc exec <zync-que-pod-id> -- cat /etc/pki/tls/cert.pem > zync.pem
```

4. Append the contents of the custom CA certificate file to **`zync.pem`**:
```
cat customCA.pem >> zync.pem
```

5. Attach the new file to the **`zync-que`** pod as a configmap object:
```
oc create configmap zync-ca-bundle --from-file=./zync.pem
oc set volume dc/zync-que --add --name=zync-ca-bundle --mount-path
/etc/pki/tls/zync/zync.pem --sub-path zync.pem
--source='{"configMap":{"name":"zync-ca-bundle","items":[{"key":"zync
.pem","path":"zync.pem"}]}}'
```

This completes the addition of the certificate bundle to the **`zync-que`** pod.

6. Verify that the certificate is attached and the content is correct:

```
oc exec <zync-que-pod-id> -- cat /etc/pki/tls/zync/zync.pem
```

7. Configure the **`SSL_CERT_FILE`** environment variable on Zync to point to the new CA certificate bundle:

```
oc set env dc/zync-que SSL_CERT_FILE=/etc/pki/tls/zync/zync.pem
```

12.4.2. Configuring RH-SSO to have a 3scale client

In your OpenShift RH-SSO dashboard, configure RH-SSO to have a 3scale client. This is a specialized, administrative client. Each time an API consumer subscribes to an API in your Developer Portal, 3scale uses the RH-SSO administrative client that you create in this procedure to create a client for the API consumer application.

Procedure

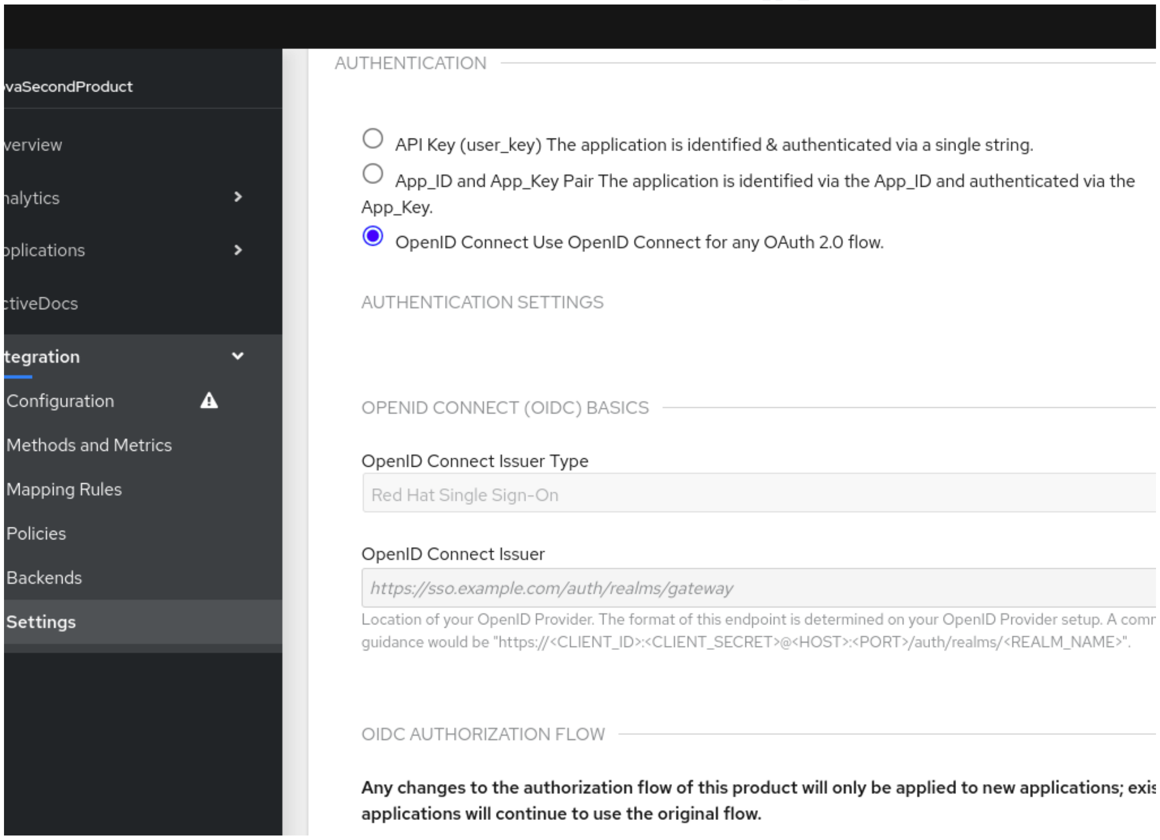
1. In the RH-SSO console, create a realm for a 3scale client or select an existing realm to contain your 3scale client.
2. In the new or selected realm, click **Clients** in the left navigation panel.
3. Click **Create** to create a new client.
4. In the **Client ID** field, specify a name that helps you identify this client as the 3scale client, for example `oidc-issuer-for-3scale`.
5. Set the **Client Protocol** field to `openid-connect`.
6. Save the new client.
7. In the settings for the new client, set and save the following:
    1. **Access Type** to `confidential`.
    2. **Standard Flow Enabled** to `OFF`.
    3. **Direct Access Grants Enabled** to `OFF`.
    4. **Service Accounts Enabled** to `ON`. This setting enables this client to issue service accounts.
8. Set the service account roles for the client:
    1. Navigate to the **Service Account Roles** tab of the client.
    2. In the **Client Roles** drop down list, click **realm-management**.
    3. In the **Available Roles** pane, select **manage-clients** and assign the role by clicking **Add selected >>**.
9. Note the client credentials:
    1. Make a note of the client ID (`<client_id>`).
    2. Navigate to the **Credentials** tab of the client and make a note of the **Secret** field (`<client_secret>`).
10. To facilitate testing the authorization flow, add a user to the realm:
    1. On the left side of the window, expand **Users**.
    2. Click **Add user**.
    3. Enter a username, set **Email Verified** to `ON`, and click **Save**.
    4. On the **Credentials** tab, set the password. Enter the password in both fields, set the **Temporary** switch to `OFF` to avoid the password reset at the next login, and click **Set Password**.
    5. When the pop-up window displays, click **Change password**.

12.4.3. Configuring 3scale to work with RH-SSO

Configure 3scale to work with RH-SSO by specifying integration settings in the 3scale Admin Portal.

Procedure

1. In the 3scale Admin Portal, in the top level selector, click **Products** and select the 3scale API product for which you are enabling OpenID Connect authentication.
2. Navigate to **[Your_product_name] > Integration > Settings**.
3. Under **Authentication** , select **OpenID Connect Use OpenID Connect for any OAuth 2.0 flow**.
   Figure 12.3. displays the OPENID CONNECT (OIDC) BASICS section.



AUTHENTICATION

API Key (user_key) The application is identified & authenticated via a single string.

App_ID and App_Key Pair The application is identified via the App_ID and authenticated via the App_Key.

[✓ OpenID Connect Use OpenID Connect for any OAuth 2.0 flow.]

AUTHENTICATION SETTINGS

OPENID CONNECT (OIDC) BASICS

OpenID Connect Issuer Type: Red Hat Single Sign-On

OpenID Connect Issuer: https://sso.example.com/auth/realms/gateway

Location of your OpenID Provider. The format of this endpoint is determined on your OpenID Provider setup. A comr guidance would be "https://<CLIENT_ID>:<CLIENT_SECRET>@<HOST>:<PORT>/auth/realms/<REALM_NAME>".

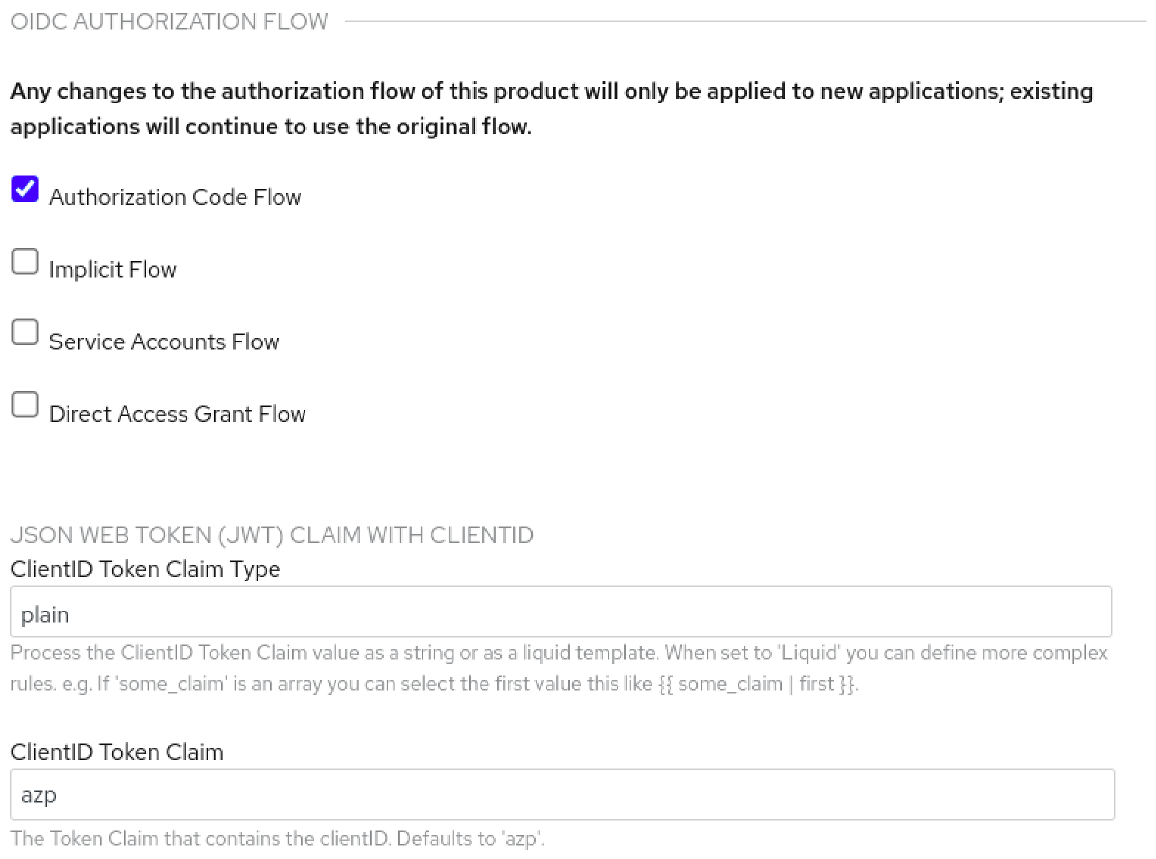
OIDC AUTHORIZATION FLOW

Any changes to the authorization flow of this product will only be applied to new applications; exis applications will continue to use the original flow.

4. In the **OpenID Connect Issuer Type** field, ensure that the setting is **Red Hat Single Sign-On**.
5. In the **OpenID Connect Issuer** field, enter the URL for the configured OpenID Connect identity provider. The format for this URL looks like this:
   `https://<client_id>:<client_secret>@<rhsso_host>:<rhsso_port>/auth/realms/<realm_name>`

6. Replace the placeholders with the previously noted RH-SSO client credentials, the host and port for your RH-SSO server, and the name of the realm that contains the RH-SSO client.
7. Under **OIDC AUTHORIZATION FLOW**, select one or more of the following:
    - **Authorization Code Flow** - In RH-SSO this is the Standard Flow.
    - **Implicit Flow**
    - **Service Accounts Flow** - Also referred to as Client Credentials Flow.
    - **Direct Access Grant Flow** - Also referred to as Resource Owner Password Credentials.

    Figure 12.4. shows where you select the authorization flow:

    OIDC AUTHORIZATION FLOW

    Any changes to the authorization flow of this product will only be applied to new applications; existing applications will continue to use the original flow.

    [✓ Authorization Code Flow]
    Implicit Flow
    Service Accounts Flow
    Direct Access Grant Flow

    JSON WEB TOKEN (JWT) CLAIM WITH CLIENTID

    ClientID Token Claim Type: plain

    Process the ClientID Token Claim value as a string or as a liquid template. When set to 'Liquid' you can define more complex rules. e.g. If 'some_claim' is an array you can select the first value this like {{ some_claim | first }}.

    ClientID Token Claim: azp

    The Token Claim that contains the clientID. Defaults to 'azp'.

    This configures how API consumers obtain JSON Web Tokens (JWTs) from the OpenID Connect identity provider. When 3scale integrates RH-SSO as the OpenID Connect identity provider, Zync creates RH-SSO clients that have only the **Authorization Code Flow** enabled. This flow is recommended as the most secure and suitable for most cases. Be sure to select an OAuth 2.0 flow that is supported by your OpenID Connect identity provider.
8. Scroll down and click **Update Product** to save the configuration.
9. In the left navigation panel, click **Integration > Configuration**.
10. Scroll down to click **Promote v. *x* to APIcast Staging**.

Next steps

Test the integration with RH-SSO as the identity provider. When everything is working as it should, return to the **Integration > Configuration** page and scroll down to promote the APIcast staging version to be the production version.

12.5. INTEGRATING 3SCALE WITH THIRD-PARTY OPENID CONNECT IDENTITY PROVIDERS

As an API provider, you can configure an HTTP integration between 3scale and a third-party OpenID Connect identity provider. That is, you can configure an OpenID Connect identity provider other than Red Hat Single Sign-On. 3scale uses this integration to authenticate requests from API consumers and to update the third-party identity provider with the latest 3scale application details.

Most of the work required to integrate 3scale with a third-party OpenID Connect identity provider involves these tasks:

- Meeting the 3scale Zync-related prerequisites.
- Configuring your OpenID Connect identity provider to authorize requests from 3scale applications.

Prerequisites

- 3scale Zync is installed.
- Your chosen third-party OpenID Connect identity provider:
  - Adheres to Zync's OpenAPI specification as provided by 3scale.
  - Allows registration of a client with **`<client_id>`** and **`<client_secret>`** declared as a parameter in the request. 3scale is always the source of client identity management in the integration between 3scale and a third-party OpenID Connect identity provider.
  - Is configured for authorizing requests from 3scale applications.
- If your configuration cannot fulfill the previous prerequisite, you must implement a custom adapter that is based on the Zync abstract adapter. Zync uses this adapter to interact with your OpenID Connect identity provider. To create this adapter, you can modify rest_adapter.rb, which is part of the 3scale Zync REST API example.
  You can include the **`rest_adapter.rb`** module in the **`zync-que`** pod according to the method that best fits your requirements. For example, you could mount a **`configMap`** through a volume or you can build a new image for Zync.

Procedure

1. In the 3scale Admin Portal, in the top level selector, click **Products** and select the 3scale API product for which you are enabling OpenID Connect authentication.
2. Navigate to **[Your_product_name] > Integration > Settings**.
3. Under **Authentication**, select **OpenID Connect Use OpenID Connect for any OAuth 2.0 flow**. This displays the **OPENID CONNECT (OIDC) BASICS** section.
4. In the **OpenID Connect Issuer Type** field, ensure that the setting is **REST API**.
5. In the **OpenID Connect Issuer** field, enter the URL for your OpenID Connect identity provider. The format for this URL looks like this:
   `https://<client_id>:<client_secret>@<oidc_host>:<oidc_port>/<endpoint>`
   For example, in the Zync `rest_adapter.rb` example, the URL endpoint is hard-coded as `{endpoint}/clients`. Your endpoint might be `{endpoint}/register` or something else.
6. Under **OIDC AUTHORIZATION FLOW**, select one or more of the following:
   - **Authorization Code Flow**
   - **Implicit Flow**
   - **Service Accounts Flow**
   - **Direct Access Grant Flow**
7. This configures how API consumer applications receive JSON Web Tokens (JWTs) from the OpenID Connect identity provider. The **Authorization Code Flow** is recommended as the most secure and suitable for most cases. Be sure to select an OAuth 2.0 flow that is supported by your OpenID Connect identity provider.
8. Scroll down and click **Update Product** to save the configuration.
9. In the left navigation panel, click **Integration > Configuration**.
10. Scroll down to click **Promote v. *x* to APIcast Staging**.

Next steps

Test the integration with the third-party identity provider. When everything is working as it should, return to the **Integration > Configuration** page and scroll down to promote the APIcast staging version to be the production version.

12.6. TESTING 3SCALE INTEGRATIONS WITH OPENID CONNECT IDENTITY PROVIDERS

After integrating 3scale with an OpenID Connect identity provider, test the integration to confirm that:

- API consumers receive access credentials when they subscribe to a 3scale-managed API.
- APIcast can authenticate requests from API consumers.

Prerequisites

- Integration between 3scale and your OpenID Connect identity provider is in place for a particular 3scale API product. This integration uses the **Authorization Code Flow**.
- An application plan is available for API consumers to subscribe to in your Developer Portal. This application plan provides access to a 3scale-managed API, that is, a 3scale product, for which you configured OpenID Connect authentication.
- An application that sends requests to the upstream API. The upstream API is a backend of the 3scale product that the API consumer application can access as a result of the subscription. Alternatively, you can use Postman to send requests.

Procedure

1. In the Developer Portal, subscribe to an application plan.
   This creates an application in the Developer Portal. The OpenID Connect identity provider should return a client ID and a client secret that you can see in your application's page in the Developer Portal.
2. Note the client ID and the client secret for the application.
3. Verify that your OpenID Connect identity provider now has a client with the same client ID and client secret. For example, when Red Hat Single Sign-On (RH-SSO) is the OpenID Connect identity provider, you will see a new client in the configured RH-SSO realm.
4. In the application page in the Developer Portal, in the **REDIRECT URL** field, enter the URL for the application that sends API requests to the upstream API.
5. Verify that your OpenID Connect identity provider has the correct redirect URL.
6. Discover the URL that receives authentication requests for your OpenID Connect identity provider by using this endpoint:
   `.well-known/openid-configuration`
   For example:
   `https://<rhsso_host>:<rhsso_port>/auth/realms/<realm_name>/.well-known/openid-configuration`

7. For the base URL, use the value that an API provider configured in the OpenID Connect Issuer field.
8. An API consumer who is writing an application that consumes the upstream API, does the following:
    1. Initiates an authorization flow with your OpenConnect identity provider. This request must contain the 3scale application's client ID and client secret. In some cases, the end-user identity is also required.
    2. Receives the identity provider's response, which contains an authorization code.
9. The API consumer's application does the following:
    1. Exchanges the authorization code for a JWT.
    2. Receives a JWT from RH-SSO upon authentication.
    3. Sends an API request that contains the JWT to the upstream API backend.

If APIcast accepts the JWT in the request, your application will receive a response from the API backend.

Alternatively, in place of an API consumer application, use Postman to test that the token flow is correctly implemented.

12.7. EXAMPLE OF A 3SCALE INTEGRATION WITH AN OPENID CONNECT IDENTITY PROVIDER

This example shows the flow when you integrate 3scale with Red Hat Single Sign-On (RH-SSO) as the OpenID Connect identity provider. This example has the following characteristics:

- In the Admin Portal, an API provider defined a 3scale API product and configured that product to use RH-SSO as the OpenID Connect identity provider.
- This product's OpenID Connect configuration includes:
    - Public base URL: `https://api.example.com`
    - Private base URL: `https://internal-api.example.com`
    - OpenID Connect Issuer: `https://zync:41dbb98b-e4e9-4a89-84a3-91d1d19c4207@idp.example.com/auth/realms/myrealm`
    - **Authorization Code Flow**, which is the standard flow, is selected.
- In the 3scale Developer Portal, there is an application with the following characteristics. This application is the result of an API consumer subscribing for access to a 3scale API product provided by a particular application plan in the Developer Portal.
    - **Client ID**: `myclientid`
    - **Client Secret**: `myclientsecret`

    - **Redirect URL**: **`https://myapp.example.com`**
- In RH-SSO, in the **`myrealm`** realm, there is a client with these same characteristics:
    - **Client ID**: **`myclientid`**
    - **Client Secret**: **`myclientsecret`**
    - **Redirect URL**: **`https://myapp.example.com`**
- The **`myrealm`** realm has this user:
    - Username: **`myuser`**
    - Password: **`mypassword`**
- 3scale Zync client in **`myrealm`** has the correct **Service Account** roles

The flow is as follows:

1. An end-user (API consumer) sends an Authorization request to the authentication server, RH-SSO in this example, at the following endpoint:
   `https://idp.example.com/auth/realms/myrealm/protocol/openid-connect/auth`
   In the request, the application provides these parameters:
    - Client ID: **`myclientid`**
    - Redirect URL: **`https://myapp.example.com`**
2. The application redirects the end-user to the RH-SSO log-in window.
3. The end-user logs in to RH-SSO with these credentials:
    - Username: **`myuser`**
    - Password: **`mypassword`**
4. Depending on the configuration, and whether this is the first time that the end-user is authenticating in this specific application, the consent window might display.
5. RH-SSO issues an authorization code to the end-user.
6. The API consumer application uses the following endpoint to send a request to exchange the authorization code for a JWT:
   `https://idp.example.com/auth/realms/myrealm/protocol/openid-connect/token`
   The request contains the authorization code and these parameters:
    - Client ID: **`myclientid`**
    - Client secret: **`myclientsecret`**
    - Redirect URL: **`https://myapp.example.com`**.
7. RH-SSO returns a JSON Web Token (JWT) with an access_token field such as **`eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lk…xBArNhqF-A`**.
8. The API consumer application sends an API request to **`https://api.example.com`** with a header such as:
   `Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lk…xBArNhqF-A.`
9. The application should receive a successful response from **`https://internal-api.example.com`**.