meʀʌtive™

Cúram 8.1.2

Integrating Cúram™ with IBM Watson® Assistant Guide

Note

Before using this information and the product it supports, read the information in

Edition

This edition applies to Cúram 8.1, 8.1.1, and 8.1.2.

© Merative US L.P. 2012, 2024

Merative and the Merative Logo are trademarks of Merative US L.P. in the United States and other countries.

Contents

1 Configuring Watson Assistant settings

Complete the following steps to configure IBM® Watson™ Assistant settings in Cúram.

About this task

For more information about Watson™ Assistant web chat configuration, see [Configuration](#).

For more information about the issuer for Watson™ Assistant, see [Securing the web chat](#).

Procedure

1. Log in to Cúram as a system administrator.
2. Select **System Configurations** > **Shortcuts** > **Application Data**.
3. Select the **Watson Assistant Configuration** category and click **Search**.
4. To update any of the following properties, select **...** > **Edit Value** to update the value and click **Save**.

Property	Value
curam.watson.assistant.accepted.application.codes	The application codes for the user workspaces where you want the web chat to be displayed. You can specify multiple workspaces with a comma-separated list. For example, you can specify `DefaultApp,CustomApp` to show the web chat to users of the `DefaultApp` and `CustomApp` applications. Application codes are not case-sensitive.
curam.watson.assistant.enabled	Set to `true` to enable the Watson Assistant web chat widget to be displayed in the specified workspaces.
curam.watson.assistant.integration.id	The integration ID of your web chat integration, which is a UUID, such as `250058a9-11f5-4deb-a91e-7ac24878c6a3`.
curam.watson.assistant.issuer	The issuer of the JSON Web Tokens (JWT).
curam.watson.assistant.region	The data center where your web integration was created. For example, `us-south`.
curam.watson.assistant.service.instance.id	The service instance ID of the assistant that hosts your web chat integration, which is a UUID such as `fba14cb7-95da-45e3-bf4a-e61d80e46598`. You can find it in the Assistant URL.

5. Click **Publish** to publish your changes.

2 Generating the JWT keystore and certificates

A JSON Web Token (JWT) is used to secure requests between Cúram and IBM® Watson™ Assistant. The JWT is digitally signed by the Cúram application by using a public/private key pair, which is stored in a keystore.

The private key in the keystore must be accessible by the Cúram runtime code so the JWT can be generated and signed when an outbound request to Watson™ Assistant is made.

You must add the public key in two locations:

- Watson™ Assistant needs the public key so that requests from Cúram to Watson™ Assistant are trusted.
- The Cúram application server's truststore, so if Watson™ Assistant needs to make an API call back into Cúram to retrieve data in order to answer questions, the JWT that is passed along with the request is accepted. For more information about configuring the application server to allow authentication with JWT, see the *Configuring application servers for JSON Web Tokens* related link.

Self-signed certificates for development and test

For development and test environments, you can generate a keystore with a self-signed cert with Cúram Ant build scripts if needed.

For production environments, a keystore with a certificate that is signed by a Trusted Certificate Authority is recommended.

To generate and use a self-signed cert, use the `configtest` or `configure` Ant build scripts. These scripts generate a keystore and self-signed cert by using the properties that are specified in the `/EJBServer/project/properties/JWTKeystore.properties` file. Both the generated keystore and properties file are added to a `JWTCryptoConfig.jar` file, which is placed in the `JAVA_HOME/jre/lib/ext` directory. The public certificate is also exported to this directory, so it can be retrieved later and added to Watson Assistant. The `configure` build script also imports the public certificate into the application servers truststore.

The values that are used for the keystore and certificate are in the file `/EJBServer/project/properties/JWTKeystore.properties` as shown in this example.

```
KEYSTORE PROPERTIES
jwt.keystore.filename=jwtkeystore.p12
jwt.keystore.location= (Leave empty when you use the generated self-signed certificate)
jwt.keystore.password=password
jwt.keystore.type=PKCS12

jwt.key.alias=jwtkeyalias (The alias name for the private key used for signing the JWT)
jwt.cert.alias=jwtcertalias (The alias name of the public cert in the truststore)
jwt.public.cert.filename=jwtcert.cer (The name of the file that will be created which
 contains the cert's public key)

CERTIFICATE PROPERTIES
(These are used to create a self-signed cert, if using the default generated keystore,
 at build time.
These properties are not required if using an existing keystore with existing
 certificate).
jwt.cert.common.name=JWT
jwt.cert.organisation.unit=
jwt.cert.organisation=
jwt.cert.locality=
jwt.cert.state=
jwt.cert.country=
jwt.cert.key.validity=1825
jwt.key.alg=RSA
jwt.key.size=2048
```

Use existing keystore and certificate

You can use an existing keystore and certificate instead of the default keystore and self-signed certificate by updating the appropriate properties in the `/EJBServer/project/properties/JWTKeystore.properties` file.

1. If needed, customize the properties in the *`/EJBServer/project/properties/JWTKeystore.properties`* file.

```
jwt.keystore.filename= (The name of the keystore file)
jwt.keystore.location= (The absolute path to the keystore file)
jwt.keystore.password= (The password of the keystore)
jwt.keystore.type= (The type of the keystore)
jwt.key.alias= (The alias of the private key to use for signing the JWT)
```

2. From *EJBServer* directory, run the `configtest` build target to generate the *JWTCrypto.jar* file and the public key *jwtcert.cer* in the *$JAVA_HOME/jre/lib/ext/* directory.

Troubleshooting

- **Rerunning `configtest` build target and reimporting the JWT public certificate into the truststore**

  If you rerun the `configtest` build target after you followed the steps for importing the JWT certificate into the truststore, you must reimport the certificate again because the original certificate is no longer valid. Reimporting the certificate again applies only to applications deployed on WebSphere® or on WebSphere® Liberty.

  For more information, see the *Configuring application servers for JSON Web Tokens* related link.

- **Starting the Watson™ Assistant chatbot**
  If you can't start Watson™ Assistant chatbot, search for all files that start with 'jwtcert'. Where there are multiple entries, delete the old files. Then, rerun the build `configtest` target.

Related concepts

[Configuring application servers for JSON Web Tokens on page 15](#)
You must configure the application server in your development or test environment to accept
JSON Web Tokens for the integration with IBM® Watson™ Assistant.

2.1 Adding the public key to Watson™ Assistant

Complete the following steps to add your public key to IBM® Watson™ Assistant.

About this task

For more information about security for Watson™ Assistant, see [Securing the web chat](#).

Procedure

1. Open the *$JAVA_HOME/jre/lib/ext/jwtcert.cer* file with a text editor to copy the public key details.
2. Log in to IBM® Cloud® and open your Watson Assistant service.
3. Click the **Launch Watson Assistant** button and select **Integrations** for your assistant.
4. Under **Integrations**, select **Web chat**.
5. Select the **Security** tab, paste the public key in the **Your public key** section, and save.

Cúram 8.1.2 14

3 Configuring application servers for JSON Web Tokens

You must configure the application server in your development or test environment to accept JSON Web Tokens for the integration with IBM® Watson™ Assistant.

3.1 Configuring WebSphere® Application Server for JWT

Complete the following steps to configure IBM® WebSphere® Application Server to accept JSON Web Tokens (JWT).

Complete the following steps in the WebSphere® Application Server documentation, see Configuring authentication with JSON Web Tokens (JWT).

When you configure the new interceptor, add the following relying party custom properties for Cúram.

- The following properties are required for JWT authentication.
  - `provider_<id>.useJwtFromRequest`: (must be set to required)
  - `provider_<id>.identifier`
  - `provider_<id>.issuerIdentifier` This value must match the `iss` attribute value in the JWT token.
- Configure the following property to obtain the JWT signer certificate.
  - `provider_<id>.signVerifyAlias` This value must match the alias name that is used to import the signer certificate into the truststore.
- Configure the following property to enable the OIDC TAI to intercept requests.
  - `provider_<id>.filter Authorization%=Bearer`

    If the request includes a header that is called Authorization and that contains Bearer in the value, use the TAI interceptor.

Import the JWT signer's public certificate into the default truststore
Using the default self-signed certificate

The `configtest`, `configure`, `installapp`, and `configurewebserverplugin` ant build scripts all generate a self-signed certificate that is used to sign the JWT. The `configure` build script also imports the corresponding public certificate into the application server's default truststore. However, where the other build scripts are run after the configure build script then a new self-signed certificate is generated. The newer public certificate must be imported into the truststore.

The following list outlines how to use the default self-signed certificate:

1. In the administrative console, click **Security** > **SSL certificate and key management** > **Key stores and certificates** > **NodeDefaultTrustStore** > **Signer certificates**.
2. Click **jwtcertalias**, where an entry of this name exists, and click **Delete**.
3. Click **Add**.
4. Enter `jwtcertalias` for the **Alias** field. The name must also match the alias name that you specified for the `signVerifyAlias` TAI custom property.

5. Enter the absolute path to the public certificate for the **File name** field. When the self-signed certificate was created, a file that contains the public certificate was created in the directory `<JAVA_HOME>/jre/lib/ext` . For example: `<JAVA_HOME>/jre/lib/ext/jwtcert.cer`.

Using an existing keystore and signer certificate

If you are using an existing keystore and signer certificate instead of the generated self-signed certificate, use the preceding steps to import the public certificate into the application server's truststore. Ensure that you first run the `configtest` ant build script. The script exports the public key certificate to `<JAVA_HOME>/jre/lib/ext/jwtcert.cer`.

3.2 Configuring WebLogic Server for JWT

Complete the following steps to configure Oracle WebLogic Server to accept JSON Web Tokens (JWT).

Procedure

1. From the WebLogic Server **Admin** pane, select **Security Realms**.
2. Select **myrealm** in the **Realms** list.
3. Select the **Providers** tab.
4. Select the **Authentication** tab.
5. Click **New**.
6. Enter these values for the following fields:

    - **Name**: The name of your new provider.
    - **Type**: CuramJWTIdentityAsserter
7. Click **OK**.
8. Click **Save**.
9. Restart the server.

3.3 Configuring WebSphere® Liberty for JWT

Import the JWT signer's public certificate into the default truststore and then update the `server.xml` configuration file.

Importing the JWT signer's public certificate into the default truststore
Using the default self-signed certificate

The `configtest`, `configure`, `installapp`, and `configurewebserverplugin` ant build scripts all generate a self-signed certificate that is used to sign the JWT. The `configure` build script also imports the corresponding public certificate into the application server's default truststore. However, where the other build scripts are run after the configure build script then a new self-signed certificate is generated. The newer public certificate must be imported into the truststore.

First, delete any existing public certificate entries with the alias name `jwtcertalias`, or the value that is used in the `jwt.cert.alias` property in the `/EJBServer/project/ properties/JWTKeystore.properties` file where it was updated.

To do this, run the `keytool` command that is provided with the Java™ SE Development Kit, or equivalent. For example:

```
keytool -delete -alias <alias_name> -keystore <keystore_file> -storetype <store_type> -
storepass <store_password>
```

Then, import the new public certificate file into the application server's keystore. For example:

```
keytool -import -trustcacerts -alias <alias_name> -storetype <store_type> -storepass
 <store_password> -keystore <keystore_file> -file <certificate_file>
```

where:

- `<alias_name>` is the string identifier for the truststore entry. This is defined in the `jwt.cert.alias` property in the `/EJBServer/project/properties/ JWTKeystore.properties` file. The default name `isjwtcertalias`.
- `<keystore_file>` is the absolute path to a keystore or truststore file that is used by the application server. The simplest option is to use the default keystore, which is already defined in the `<keystore>` element in the `${server.config.dir}/adc_conf/ server_security.xml` configuration file. The default keystore is located in the directory `${server.config.dir}/resources/security`.
- `<store_type>` is the type of keystore, for example JKS or PKCS12.
- `<store_password>` is the password of the keystore or truststore.
- `<certificate_file>` is the absolute path to the file that contains the public certificate. When the self-signed cert was created, a file that contains the public certificate is created in the `<JAVA_HOME>/jre/lib/ext` directory. For example: `<JAVA_HOME>/jre/lib/ ext/jwtcert.cer`.

Using an existing keystore and signer certificate

If you are using an existing keystore and signer certificate instead of the generated self-signed certificate, use the preceding steps to import the public certificate into the application server's truststore. Ensure that you first run the `configtest` ant build script. The script exports the public key certificate to `<JAVA_HOME>/jre/lib/ext/jwtcert.cer`.

Updating the `server.xml` configuration file

The following list outlines the changes that you must make to the `server.xml` configuration file:

1. Add these features to the `<featureManager>` element:

```
<feature>openidConnectClient-1.0</feature>
<feature>transportSecurity-1.0</feature>
```

2. Add a new `<authFilter>` element:

```
<authFilter id="jwtAuthFilter">
    <requestHeader id="authHeader" name="Authorization" value="Bearer"
 matchType="contains"/>
</authFilter>
```

This defines a filter for any incoming requests that contain the Authorization request header, by using the Bearer scheme. This is used so that OpenIDConnect functionality is applied only to incoming requests that include this header.

3. Add a new `<openidConnectClient>` element:

```
<openidConnectClient
 id="<unique_id>"
 inboundPropagation="required"
 issuerIdentifier="<issuer_identifier>"
 audiences="ALL_AUDIENCES"
 signatureAlgorithm="RS256"
 trustStoreRef="<keystore_ref>"
 trustAliasName="<alias_name>"
 authFilterRef="jwtAuthFilter"
 />
```

where:

- `<unique_id>` can be any string ID for the element.
- `<keystore_ref>` is the reference ID of the keystore that contains the public certificate. For default configurations, this is defaultKeyStore as defined in the `/adc_conf/server_security.xml` configuration file.
- `<alias_name>` is the alias of the entry in the keystore or truststore for the public certificate and it must match the alias name that is given when the public certificate was imported. This is `jwtcertalias` for default configurations that use the generated self-signed certificate.

4. `<issuer_identifier>` must exactly match the value of the `iss` field of the JWT. This value is configured by using the `curam.watson.assistant.issuer` system property in Cúram. When the token is generated, the value is added to the JWT.

5. Restart the server.

4 Getting started with your virtual assistant

To help you get started, see the open-source GitHub repository that provides sample code and guidance for a caseworker assistant called Case Assistant. Case Assistant enables caseworkers to get information more quickly, reducing the amount of navigation to complete their tasks. Caseworkers can ask about verifications on a case or client, and about caseworker policies and procedures.

Use Case Assistant and Watson™ Assistant to build your own live virtual assistant into Cúram. Refer to the sample dialog skills and cloud functions, and see how to query Cúram data with REST APIs. For more information, see the *Cúram Virtual Assistant Cookbook* related link.

Related information

[Cúram Virtual Assistant Cookbook](#)

Notices

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the Merative website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of Merative

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of Merative.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

Merative reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by Merative, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

MERATIVE MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Merative or its licensors may have patents or pending patent applications covering subject matter described in this document. The furnishing of this documentation does not grant you any license to these patents.

Information concerning non-Merative products was obtained from the suppliers of those products, their published announcements or other publicly available sources. Merative has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-Merative products. Questions on the capabilities of non-Merative products should be addressed to the suppliers of those products.

Any references in this information to non-Merative websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this Merative product and use of those websites is at your own risk.

Privacy policy

The Merative privacy policy is available at https://www.merative.com/privacy.

Trademarks