IBM Business Automation Workflow User Management Services on containers

Contents

User Management Services on Containers

This document provides information for configuring and using User Management Services (UMS) in Business Automation Workflow on containers.

Before you begin

The information in this document is complementary to the information contained in .The details in this document are applicable to 21.0.3 and later versions of Business Automation Workflow

Preparing to install User Management Services

UMS includes two options: UMS SSO and UMS Teams. If you want your app users to benefit from single sign-on and shared teams, you must install UMS before you install any apps that rely on it. Before you deploy User Management Service, you must prepare your environment. These procedures include setting up the database and creating secrets.

Before you begin

Make sure that you performed LDAP Configuration. In addition, if you want all communications between UMS and your LDAP server to be encrypted, perform Preparing for SSL-enabled LDAP.

Procedure

To prepare to install User Management Services, perform the following actions:

1. Perform “Preparing the database for the User Management Services” on page 1.
2. Perform “Configuring the User Management Services dedicated pods option” on page 5.
3. Perform “Creating the UMS database admin secret ” on page 5.
4. Perform “Securing communications with the UMS database” on page 6.
5. Perform “Securing communications routes with User Management Services (UMS)” on page 8.

Preparing the database for the User Management Services

Create the database for UMS single sign-on and teams options.

About this task

The default Derby database is not suitable for a production system. When selected, the Derby database is created automatically.

If you are not using a Derby database, perform the preparati0on task for your database:

Preparing a Db2 database for the User Management Services

Create dB2 databases for UMS single sign-on and teams options, and optional failover servers.

Procedure

1. Create the UMS SSO option database.

   For example, you can create a database named UMSDB by running the following command:

   ```
   db2 create database UMSDB automatic storage yes using codeset UTF-8 territory US pagesize
   32768
   ```

2. Optional: Create a separate database for the UMS Teams option if you do not want it to share the UMS SSO oauth database. For example, named UMSTSDB.
3. Optional: Create one or more failover servers for the UMS database(s).

   To cover the possibility that the primary server is unavailable during the initial connection attempt, you can configure a list of failover servers, as described in Configuring client reroute for applications that use DB2® databases.
4. Make a note of the datasource information that you will need later to add to the `datasource_configuration` section of the custom resource:

```
datasource_configuration:
  dc_ums_datasource: # credentials are read from ums_configuration.admin_secret_name
    # oauth database config
    dc_ums_oauth_type: db2
    dc_ums_oauth_host: host_name
    dc_ums_oauth_port: 50000
    dc_ums_oauth_name: UMSDB
    dc_ums_oauth_schema: OAuth_DB_Schema
    dc_ums_oauth_driverfiles: db2jcc4.jar, db2jcc_license_cu.jar
    dc_ums_oauth_alternate_hosts: "server1.db2.example.com, server2.db2.example.com"
    dc_ums_oauth_alternate_ports: "50443, 51443"
    dc_ums_oauth_ssl: true
    # teamserver database config
    dc_ums_teamserver_type: db2
    dc_ums_teamserver_host: host_name
    dc_ums_teamserver_port: 50000
    dc_ums_teamserver_name: UMSTSDB
    dc_ums_teamserver_driverfiles: db2jcc4.jar, db2jcc_license_cu.jar
    dc_ums_teamserver_alternate_hosts: "server1.db2.example.com, server2.db2.example.com"
    dc_ums_teamserver_alternate_ports: "50443, 51443"
    dc_ums_teamserver_ssl: true
```

Important:

If you do not have a separate teams database, UMSTSDB, specify identical values for the `dc_ums_teamserver_` parameters as for the `dc_ums_oauth_` ones.

If you have multiple failover servers, for example,

- server1.db2.company.com on port 50443
- server2.db2.company.com on port 51443

You will specify them as comma-separated lists, for example:

```
    dc_ums_oauth_alternate_hosts: "server1.db2.example.com, server2.db2.example.com"
    dc_ums_oauth_alternate_ports: "50443, 51443"
    ...
    dc_ums_teamserver_alternate_hosts: "server1.db2.example.com, server2.db2.example.com"
    dc_ums_teamserver_alternate_ports: "50443, 51443"
```

Preparing an Oracle database for the User Management Services datasource

Create Oracle databases for User Management Services (UMS) single sign-on and teams options, and optional failover servers.

Procedure

1. Create a user, for example, C##UMS, to represent the schema and grant the user privileges to create resources:

```
sqlplus sys AS sysdba;

CREATE USER C##UMS IDENTIFIED BY secret_password;

GRANT CREATE TABLE TO C##UMS;
GRANT CREATE SESSION TO C##UMS;
GRANT CREATE SEQUENCE TO C##UMS;
GRANT UNLIMITED TABLESPACE TO C##UMS;
```

Where *secret_password* is the password that you must specify with the user in the `ibm-baw-ums-secret` secret.

2. Make a note of the following datasource information:
   - *host_name* is the name of your database host
   - *SID* is the SID of your database, for example `orcl`.

     Tip: To determine the SID of your database, on the database host perform:

     ```
     SELECT sys_context('userenv','instance_name') FROM dual;
     ```

     The response provides the SID, for example:

     ```
     SYS_CONTEXT('USERENV','INSTANCE_NAME')
     --------------------------------------------------------------------------------
     orcl
     ```

   - *DB_user_ID* is your database user ID, for example C##UMS
   - If you connect to an Oracle Real Application Clusters (RAC) environment using Single Client Access Name (SCAN), note *DB_service_name*, which is the database service name for the datasource, for example ORCL.

     Tip: To determine the service name, on the SCAN host perform:

     ```
     select name from v$database;
     ```

     The response provides the service name, which is case-sensitive, for example:

     ```
     NAME
     ---------
     ORCL
     ```

   Later, you will need update the `datasource_configuration` section of the custom resource:

   ```
   datasource_configuration:
     dc_ums_datasource: # credentials are read from ums_configuration.admin_secret_name
     # oauth database config
       dc_ums_oauth_type: oracle
       dc_ums_oauth_host: host_name
       dc_ums_oauth_port: 1521
       dc_ums_oauth_name: SID
       dc_ums_oauth_schema: DB_user_ID
       dc_ums_oauth_oracle_service_name: DB_service_name
       dc_ums_oauth_ssl: false
       dc_ums_oauth_driverfiles: ojdbc8.jar, orai18n.jar
     # teamserver database config
       dc_ums_teamserver_type: oracle
       dc_ums_teamserver_host: host_name
       dc_ums_teamserver_port: 1521
       dc_ums_teamserver_name: SID
       dc_ums_teamserver_oracle_service_name: DB_service_name
       dc_ums_teamserver_ssl: false
       dc_ums_teamserver_driverfiles: ojdbc8.jar, orai18n.jar
   ```

   Important: For Oracle RAC, specify the host name of the SCAN listener as the value of the `dc_ums_oauth_host` and `dc_ums_teamserver_host` parameters.

Preparing an MS SQL database for the User Management Services datasource

Configure a MS SQL database for User Management Services (UMS) single sign-on and teams options.

Procedure

1. Create your MS SQL database, for example by issuing the command:

```
create database umsdb
```

For more information, refer to your database documentation.

2. Make a note of your values for the following configuration parameters that you will need later to add to the `datasource_configuration` section of the custom resource:

```
datasource_configuration:
    dc_ums_datasource: # credentials are read from ums_configuration.admin_secret_name
    # oauth database config
      dc_ums_oauth_type: sqlserver
      dc_ums_oauth_host: host_name
      dc_ums_oauth_port: 1433
      dc_ums_oauth_name: UMSDB
      dc_ums_oauth_driverfiles: mssql-jdbc-8.2.2.jre8.jar
      dc_ums_oauth_ssl: true
    # teamserver database config
      dc_ums_teamserver_type: sqlserver
      dc_ums_teamserver_host: host_name
      dc_ums_teamserver_port: 1433
      dc_ums_teamserver_name: UMSDB
      dc_ums_teamserver_driverfiles: mssql-jdbc-8.2.2.jre8.jar
      dc_ums_teamserver_ssl: true
```

Where *host_name* is the host name of the database server, *1433* is the default port, and *UMSDB* is the name of the database.

Preparing a PostgreSQL database for the User Management Services datasource

Configure a PostgreSQL database for User Management Services (UMS) single sign-on and teams options.

Procedure

1. Using **psql** create a database and grant your database user privileges to create resources:

```
CREATE DATABASE umsdb OWNER db_user ENCODING UTF8;
GRANT ALL PRIVILEGES ON DATABASE umsdb to db_user;
```

Where *umsdb* is the name of the database and *db_user* is the database user.

2. Make a note of your values for the following configuration parameters that you will need later to add to the `datasource_configuration` section of the custom resource:

```
  datasource_configuration:
    dc_ums_datasource: # credentials are read from ums_configuration.admin_secret_name
      # oauth database config
      dc_ums_oauth_type: postgresql
      dc_ums_oauth_host: host_name
      dc_ums_oauth_port: 5432
      dc_ums_oauth_name: umsdb
      dc_ums_oauth_driverfiles: postgresql-42.2.14.jar
      # teamserver database config
      dc_ums_teamserver_type: postgresql
      dc_ums_teamserver_host: host_name
      dc_ums_teamserver_port: 5432
      dc_ums_teamserver_name: umsdb
      dc_ums_teamserver_driverfiles: postgresql-42.2.14.jar
```

Important: If you configured PostgreSQL for high availability following the Patroni architecture with an HAproxy for load balancing, make sure that you specify the host name and port number of the HAproxy as the values for `dc_ums_oauth_host and dc_ums_teamserver_host` and `dc_ums_oauth_port and dc_ums_teamserver_port`.

Configuring the User Management Services dedicated pods option

User Management Services provides different capabilities. The default is to run each capability in a dedicated pod so that each capability can scale horizontally as required. It is also possible to have a single pod that contains all UMS capabilities.

About this task

User Management Services (UMS) provides the following capabilities:

- UMS SSO
- UMS Users and Groups (based on SCIM)
- UMS Teams

By default each capability runs in a dedicated pod so that each capability can scale horizontally as required, independently of the other capabilities. You can configure these pods individually, based on the load and resource demand that you expect for each of the capabilities. Alternatively, you can configure all four capabilities to run a single pod, that can be scaled, but with every pod containing all UMS capabilities. This option might be appropriate for evaluation or demo purposes.

This is option is selected by the UMS configuration parameter `dedicated_pods`, which defaults to the value `true`, which is better suited for production environments.

Procedure

Later, when you are customizing your custom resource file in the task "Configuring User Management Services" on page 9:

- To have dedicated pods for each UMS capability:

  You will use the default `dedicated_pods: true` option and can adjust any configuration parameters to suit your requirements for each pod type. For the `replica_count`, `resources`, `autoscaling` and `logs` parameters, default values are provided, so it is optional to set their values explicitly.
- To have all UMS capabilities in one pod:

  You must specify `dedicated_pods: false` and specify the configuration parameters for the all-in-one pod type. For the `replica_count`, `resources`, `autoscaling` and `logs` parameters, default values are provided, so it is optional to set their values explicitly.

Creating the UMS database admin secret

To avoid passing sensitive information in configuration files, you must create a secret manually before you deploy User Management Services (UMS). The secret contains a UMS database admin user, database users, and passwords.

Procedure

For your database (Db2, Oracle, PostgreSQL or MSSQL), generate the UMS database admin secret `ibm-baw-ums-secret`.

1. Copy the following as `ums-dba-secret.yaml`, then edit it to specify the required user identifiers and passwords.

```
apiVersion: v1
kind: Secret
metadata:
  name: ibm-baw-ums-secret
type: Opaque
stringData:
  adminUser: "umsadmin"
  adminPassword: "password"
  oauthDBUser: "oauthDBUser"
  oauthDBPassword: "oauthDBPassword"
```

```
tsDBUser: "tsDBUser"
tsDBPassword: "tsDBPassword"
```

Where

adminUser / adminPassword
Specify the user and password values for an internal UMS admin user that will be created in a local user registry. This user must be unique and not exist in the LDAP user registry. If these parameters are left out, the values will be generated.

oauthDBUser / oauthDBPassword
Specify the user and password for the OAuth (SSO) database.

tsDBUser / tsDBPassword
Specify the user and password for the UMS Teams database. If you do not have a separate UMS Teams database, specify the same database user ID and password that you specify for `oauthDBUser` and `oauthDBPassword`.

2. Create the secret by running the following command:

```
oc create -f ums-dba-secret.yaml
```

3. Make a note of the secret name `ibm-baw-ums-secret`, that you will need later to specify for the `ums_configuration.admin_secret_name` in the custom resource.

Securing communications with the UMS database

To protect communications with the database, you must create a secret that contains the certificate for the User Management Services (UMS) database.

Procedure

Perform the step for the database type that you use for UMS:

1. If you are using Db2®, create a secret to ensure that all communications between UMS and Db2 are encrypted:
   a) Import the database CA Certificate to UMS and create a secret to store the certificate:

   ```
   oc create secret generic ibm-baw-ums-db2-cacert --from-
   file=cacert.crt=path_to_certificate_PEM_file
   ```

   Where *path_to_certificate_PEM_file* is the path to the PEM format certificate file. Do not change the part `--from-file=cacert.crt=`.
   b) Make a note of the datasource SSL information that you will need later to add to the `datasource_configuration` section of the custom resource file:

   ```
   datasource_configuration:
     dc_ums_datasource:
       ...
       dc_ums_oauth_ssl_secret_name: ibm-baw-ums-db2-cacert
       dc_ums_oauth_ssl: true
       ...
       dc_ums_teamserver_ssl_secret_name: ibm-baw-ums-db2-cacert
       dc_ums_teamserver_ssl: true
   ```

2. If you are using Oracle, ensure that all communications between UMS and Oracle are encrypted:
   a) Import the database CA Certificate to UMS and create a secret to store the certificate by running the following command:

   ```
   oc create secret generic ibm-baw-ums-oracle-cacert --from-file=cacert.crt=path-to-oracle-
   certificate-file
   ```

   Where `path-to-oracle-certificate-file` is the path to the local copy of the Oracle certificate file, which must be in PEM format. Do not change the part `--from-file=cacert.crt=`

b) Make a note of the name of the secret, for example `ibm-baw-ums-oracle-cacert` that you will need later to add to the `dc_ums_datasource` section of the custom resource file:

```
datasource_configuration:
  dc_ums_datasource:
    ...
    dc_ums_oauth_ssl_secret_name: ibm-baw-ums-oracle-cacert
    dc_ums_oauth_ssl: true
    ...
    dc_ums_teamserver_ssl_secret_name: ibm-baw-ums-oracle-cacert
    dc_ums_teamserver_ssl: true
```

3. If you are using MS SQL, ensure that all communications between UMS and MS SQL are encrypted:
   a) Obtain the base-64 encoded X.509 signer certificate of your MS SQL server.
   b) Create a configuration file for the secret, for example, named `mssql.yaml` and add the signer certificate of the MS SQL server as the value of the property `tls.crt` for the secret named `ibm-baw-ums-mssql-cert`, based on the following example:

```
apiVersion: v1
kind: Secret
metadata:
  name: ibm-baw-ums-mssql-cert
type: Opaque
stringData:
  tls.crt: |+
    -----BEGIN CERTIFICATE-----
    <Include the MS SQL certificate here>
    -----END CERTIFICATE-----
```

   c) Create a secret that contains the MS SQL certificate by using the following command:

```
oc create -f mssql.yaml
```

   d) Add the name of the MS SQL certificate to the list of trusted certificates in the custom resource:

```
shared_configuration:
  trusted_certificate_list:
    - ibm-baw-ums-mssql-cert
```

   During the UMS deployment, the operator adds the MS SQL signer certificate to the truststore of UMS.

   e) In the datasource configuration enable SSL and specify the hostname that is used in certificate.
4. If you are using PostgreSQL, ensure that all communications between UMS and PostgreSQL are encrypted:
   a) Import the database CA Certificate to UMS and create a secret to store the certificate by running the following command:

```
oc create secret generic ibm-baw-ums-postgresql-cacert --from-file=cacert.crt=path-to-
postgresql-certificate-file
```

   Where `path-to-postgresql-certificate-file` is the path to the local copy of the PostgreSQL certificate file, which must be in PEM format. Do not change the part `--from-file=cacert.crt=`.

   b) Make a note of the name of the secret, for example `ibm-baw-ums-postgresql-cacert` that you will need later to add to the `dc_ums_datasource` section of the custom resource file:

```
datasource_configuration:
  dc_ums_datasource:
    ...
    dc_ums_oauth_ssl_secret_name: ibm-baw-ums-postgresql-cacert
    dc_ums_oauth_ssl: true
    ...
    dc_ums_teamserver_ssl_secret_name: ibm-baw-ums-postgresql-cacert
    dc_ums_teamserver_ssl: true
```

Securing communications routes with User Management Services (UMS)

To protect external routes that client communications use with UMS in a production environment, you can use an external TLS certificate that is trusted by your clients. For a test environment without an external TLS certificate, you can let UMS generate certificates using `shared_configuration.root_ca_secret`.

Before you begin

Ensure that you have created the shared external TLS certificate secret as described in Providing certificates for external routes.

Procedure

1. If you are creating a test environment and do not want to deal with certificates, you do not need to generate any `ums_configuration.external_tls_*` secrets. Later, by removing them from the custom resource file, the `root_ca_secret` will be used to generate an internal TLS secret for all UMS services and an external TLS secret for each of the routes:

   `ums-route`, `ums-sso-route`, `ums-scim-route`, and `ums-teams-route`.
2. If you are creating a production environment, and you want to use the shared secret `shared_configuration.external_tls_certificate_secret` that contains a single HTTPS wildcard certificate that can secure all routes, you should not define any of the UMS secrets with names starting with the string `ibm-baw-ums-external-tls`. When they are not defined, the shared secret is used, if it is defined.
3. If you are creating a production environment and you do not want to use a shared secret, you must create one or more secrets that contain TLS certificates that represent the host names or a common hostname suffix of the routes that your clients connect to.

   a. Obtain or generate a TLS certificate that represents the host names or a common hostname suffix of the routes that your clients will use to connect to UMS:

   - `ums-route`
   - `ums-teams-route`
   - `ums-scim-route`
   - `ums-sso-route`

   Perform one of the following:

   - Obtain a certificate that is signed by an external certificate authority (CA) that is trusted by your clients.
   - Generate a certificate by using OpenSSL:

     i) Create a TLS certificate signing request by executing OpenSSL. Note that the final certificate should have a `Subject Alternative Names` (SAN) value that matches the hostname. Many certificate authorities allow you to specify SANs during the ordering process, otherwise you must provide the SAN directly in the certificate signing request (CSR).

     ```
     openssl req -new -newkey rsa:2048 -subj "/CN=UMS" -extensions SAN  -days
     365 -nodes -out ums.csr -config <(cat /etc/ssl/openssl.cnf <(printf "[SAN]
     \nsubjectAltName=DNS:ums.mycluster.com"))
     ```

     ii) Two files are generated: a private key (`privkey.pem`) and a certificate signing request that can be sent to your certificate authority for signing.

   b. Use the private key and your certificate authority's response to generate the following secrets

   - `ibm-baw-ums-external-tls-secret` for the `ums-route` route
   - `ibm-baw-ums-external-tls-teams-secret` for the `ums-teams-route` route

- `ibm-baw-ums-external-tls-scim-secret` for the `ums-scim-route` route
- `ibm-baw-ums-external-tls-sso-secret` for the `ums-sso-route` route

If the response from your certificate authority does not include all certificates from its signing chain, you can provide them in `ibm-baw-ums-external-tls-ca-secret`.

Create each TLS secret by running the following command:

```
oc create secret tls ibm-baw-ums-external-tls-secret --cert tls.crt --key tls.key
```

The result in a YAML structure like the following:

```
apiVersion: v1
kind: Secret
metadata:
  ...
type: kubernetes.io/tls
data:
  tls.crt: [very long base64 string of certificate]
  tls.key: [very long base64 string of private key]
```

For each secret, run your YAML file, for example:

```
oc create -f ums-external-tls-secret.yaml
```

c. Make a note of the secret names that you will need later to specify in the custom resource for the `ums_configuration.external_tls_*` parameters:

```
ums_configuration:
    hostname:
    ## optional: create routes for backwards compatibility
    backwards_compatibility_routes:
    ## optional for secure communication with UMS
    external_tls_secret_name:
    ## optional for secure communication with UMS
    external_tls_ca_secret_name:
    ## optional for secure communication with UMS
    external_tls_teams_secret_name:
    ## optional for secure communication with UMS
    external_tls_scim_secret_name:
    ## optional for secure communication with UMS
    external_tls_sso_secret_name:
```

Configuring User Management Services

User Management Services configuration settings are stored in the shared custom resource (CR) file for operator deployment.

About this task

Provide details for configuration settings that you want, including things that you have already created, like the names of your persistent volume claims, secrets, and data sources that you created previously during “Preparing to install User Management Services” on page 1. For more information about the settings, see “User Management Services configuration parameters” on page 49.

Procedure

1. Edit the `ums_configuration` section of the custom resource file.
2. Specify the name of the UMS database admin secret that you created for the `ums_configuration.admin_secret_name` during, for example `ibm-baw-ums-secret`.
3. Specify the UMS datasource settings for your database in the `datasource_configuration` section of the custom resource.

For example:

- For Db2:

```
datasource_configuration:
  dc_ums_datasource: # credentials are read from ums_configuration.admin_secret_name
    # oauth database config
    dc_ums_oauth_type: db2
    dc_ums_oauth_host: host_name
    dc_ums_oauth_port: 50000
    dc_ums_oauth_name: UMSDB
    dc_ums_oauth_schema: OAuth_DB_Schema
    dc_ums_oauth_driverfiles: db2jcc4.jar, db2jcc_license_cu.jar
    dc_ums_oauth_alternate_hosts: "server1.db2.example.com, server2.db2.example.com"
    dc_ums_oauth_alternate_ports: "50443, 51443"
    dc_ums_oauth_ssl: true
    # teamserver database config
    dc_ums_teamserver_type: db2
    dc_ums_teamserver_host: host_name
    dc_ums_teamserver_port: 50000
    dc_ums_teamserver_name: UMSTSDB
    dc_ums_teamserver_driverfiles: db2jcc4.jar, db2jcc_license_cu.jar
    dc_ums_teamserver_alternate_hosts: "server1.db2.example.com, server2.db2.example.com"
    dc_ums_teamserver_alternate_ports: "50443, 51443"
    dc_ums_teamserver_ssl: true
```

If you created a secret for the Db2 certificate to secure communications between UMS and Db2, specify the name of the secret, and enable SSL, for example:

```
datasource_configuration:
  dc_ums_datasource:
    ...
    dc_ums_oauth_ssl_secret_name: ibm-baw-ums-db2-cacert
    dc_ums_oauth_ssl: true
    ...
    dc_ums_teamserver_ssl_secret_name: ibm-baw-ums-db2-cacert
    dc_ums_teamserver_ssl: true
```

- For Oracle:

```
  datasource_configuration:
    dc_ums_datasource: # credentials are read from ums_configuration.admin_secret_name
    # oauth database config
      dc_ums_oauth_type: oracle
      dc_ums_oauth_host: host_name
      dc_ums_oauth_port: 1521
      dc_ums_oauth_name: SID
      dc_ums_oauth_schema: DB_user_ID
      dc_ums_oauth_oracle_service_name: DB_service_name
      dc_ums_oauth_ssl: false
      dc_ums_oauth_driverfiles: ojdbc8.jar, orai18n.jar
    # teamserver database config
      dc_ums_teamserver_type: oracle
      dc_ums_teamserver_host: host_name
      dc_ums_teamserver_port: 1521
      dc_ums_teamserver_name: SID
      dc_ums_teamserver_oracle_service_name: DB_service_name
      dc_ums_teamserver_ssl: false
      dc_ums_teamserver_driverfiles: ojdbc8.jar, orai18n.jar
```

Where *host_name* is the name of your database host, *SID* is the SID of your database, for example UMSDB, *DB_user_ID* is your database user ID, for example C##UMS.

Important: For Oracle RAC, specify the host name of the SCAN listener as the value of the `dc_ums_oauth_host` and `dc_ums_teamserver_host` parameters.

If you created a secret for the Oracle certificate to secure communications between UMS and Oracle, specify the name of the secret, and enable SSL, for example:

```
datasource_configuration:
  dc_ums_datasource:
    ...
    dc_ums_oauth_ssl_secret_name: ibm-baw-ums-oracle-cacert
    dc_ums_oauth_ssl: true
    ...
```

```
    dc_ums_teamserver_ssl_secret_name: ibm-baw-ums-oracle-cacert
    dc_ums_teamserver_ssl: true
```

- For MS SQL:

```
datasource_configuration:
    dc_ums_datasource: # credentials are read from ums_configuration.admin_secret_name
    # oauth database config
      dc_ums_oauth_type: sqlserver
      dc_ums_oauth_host: host_name
      dc_ums_oauth_port: 1433
      dc_ums_oauth_name: UMSDB
      dc_ums_oauth_driverfiles: mssql-jdbc-8.2.2.jre8.jar
      dc_ums_oauth_ssl: true
    # teamserver database config
      dc_ums_teamserver_type: sqlserver
      dc_ums_teamserver_host: host_name
      dc_ums_teamserver_port: 1433
      dc_ums_teamserver_name: UMSDB
      dc_ums_teamserver_driverfiles: mssql-jdbc-8.2.2.jre8.jar
      dc_ums_teamserver_ssl: true
```

Where *host_name* is the name of your database host, *1433* is the port number, *UMSDB* is the name of your database.

If you created a secret for the MS SQL certificate to secure communications between UMS and MS SQL, specify the name of the secret, and enable SSL, for example, `ibm-baw-ums-mssql-cert`:

```
datasource_configuration:
  dc_ums_datasource:
    ...
    dc_ums_oauth_ssl_secret_name: ibm-baw-ums-mssql-cert
    dc_ums_oauth_ssl: true
    ...
    dc_ums_teamserver_ssl_secret_name: ibm-baw-ums-mssql-cert
    dc_ums_teamserver_ssl: true
```

- For PostgreSQL:

```
  datasource_configuration:
    dc_ums_datasource: # credentials are read from ums_configuration.admin_secret_name
      # oauth database config
      dc_ums_oauth_type: postgresql
      dc_ums_oauth_host: host_name
      dc_ums_oauth_port: 5432
      dc_ums_oauth_name: umsdb
      dc_ums_oauth_driverfiles: postgresql-42.2.14.jar
      # teamserver database config
      dc_ums_teamserver_type: postgresql
      dc_ums_teamserver_host: host_name
      dc_ums_teamserver_port: 5432
      dc_ums_teamserver_name: umsdb
      dc_ums_teamserver_driverfiles: postgresql-42.2.14.jar
```

Where *host_name* is the name of your database host and `umsdb` is the name of your database.

If you created a secret for the PostgreSQL certificate to secure communications between UMS and PostgreSQL, specify the name of the secret, and enable SSL, for example, `ibm-baw-ums-postgresql-cacert`:

```
datasource_configuration:
  dc_ums_datasource:
    ...
    dc_ums_oauth_ssl_secret_name: ibm-baw-ums-postgresql-cacert
    dc_ums_oauth_ssl: true
    ...
    dc_ums_teamserver_ssl_secret_name: ibm-baw-ums-postgresql-cacert
    dc_ums_teamserver_ssl: true
```

Important: If you configured PostgreSQL for high availability following the Patroni architecture with an HAproxy for load balancing, make sure that you specify the host name and port number of the HAproxy as the values for `dc_ums_oauth_host and dc_ums_teamserver_host` and `dc_ums_oauth_port and dc_ums_teamserver_port`.

Important: If you do not have a separate teams database, UMSTSDB, specify identical values for the `dc_ums_teamserver_` parameters as for the `dc_ums_oauth_` ones.

Trouble: If you plan to use UMS integration with other capabilities, you might encounter registration failure errors during deployment. This can happen if the UMS deployment is not ready by the time the other containers come up. The situation resolves in the next operator loop, so the errors can be ignored.

4. Specify the certificates and routing for secure communications with UMS.
   a. If you are creating a test environment and you do not want to deal with certificates, you do not need the following secrets, and should remove them from the custom resource:
      - `external_tls_secret_name`
      - `external_tls_ca_secret_name`
      - `external_tls_teams_secret`
      - `external_tls_sso_secret`
      - `external_tls_scim_secret`

      This causes the `root_ca_secret` to be used to generate an internal TLS secret for all services and an external TLS secret for each of the routes `ums-route`, `ums-sso-route`, `ums-scim-route`, and `ums-teams-route`.

      If you do not specify a root signing CA in the `shared_configuration` section of the custom resource, `root_ca_secret` is generated by the operator with a self-signed root CA.
   b. If you are creating a production environment, you can perform one of the following options:
      - Specify the secrets that contain a TLS certificate that represents the host names of the routes that your clients connect to.
      - Use the shared secret `shared_configuration.external_tls_certificate_secret` that contains a single HTTPS wildcard certificate that can be used to secure all routes.
   c. If you upgraded from a version earlier than 21.0.2 and you have existing secrets that contain host names, they will not conform to the new host name convention, if you do not want to regenerate the secrets, you can set `backwards_compatibility_routes` to `true` to use the old host naming pattern.

```
ums_configuration:
  hostname: <ums-host>
  admin_secret_name: ibm-baw-ums-secret
  # optional: create routes for backwards compatibility
  backwards_compatibility_routes: false
  # optional for secure communication with UMS
  external_tls_secret_name: ibm-baw-ums-external-tls-secret
  # optional for secure communication with UMS
  external_tls_ca_secret_name: ibm-baw-ums-external-tls-ca-secret
  # optional for secure communication with UMS
  external_tls_teams_secret_name: ibm-baw-ums-external-tls-teams-secret
  # optional for secure communication with UMS
  external_tls_scim_secret_name: ibm-baw-ums-external-tls-scim-secret
  # optional for secure communication with UMS
  external_tls_sso_secret_name: ibm-baw-ums-external-tls-sso-secret
```

5. Decide whether you want each UMS service (UMS SSO, UMS Teams, UMS SCIM-based Users and Groups) to run in its own dedicated pod so that they can scale individually, which is the default. Or whether you want all UMS services to run in a single pod.

   Perform one of the following:

   a) For dedicated pods:

      i) In the section `ums_configuration` set

```
dedicated_pods: true
```

ii) For the UMS SSO service, default values are specified for the `replica_count`, `resource`, `autoscaling` and `logs` parameters. If the default values do not meet your requirements, you can change the values .

```
# Configuration for sso pods
sso:
  replica_count: 2
  resources:
    limits:
      cpu: 500m
      memory: 512Mi
    requests:
      cpu: 200m
      memory: 256Mi
    autoscaling:
      enabled: true
      minReplicas: 2
      maxReplicas: 5
      targetAverageUtilization: 98
  custom_xml:
  logs:
    traceSpecification: "*=info"
```

iii) For the UMS SCIM-based Users and Groups service, default values are specified for the `replica_count`, `resource`, `autoscaling` and `logs` parameters. You can change the values if the default values do not meet your requirements.

```
# configuration for scim pods
scim:
  replica_count: 2
  resources:
  limits:
    cpu: 500m
    memory: 512Mi
  requests:
    cpu: 200m
    memory: 256Mi
  autoscaling:
    enabled: true
    minReplicas: 2
    maxReplicas: 5
    targetAverageUtilization: 98
  custom_xml:
  logs:
    traceSpecification: "*=info"
```

iv) For the UMS Teams service, default values are specified for the `replica_count`, `resource`, `autoscaling` and `logs` parameters. You can change the values if the default values do not meet your requirements.

```
# configuration for teamserver pods
teamserver:
  replica_count: 2
  resources:
  limits:
    cpu: 500m
    memory: 512Mi
  requests:
    cpu: 200m
    memory: 256Mi
  autoscaling:
    enabled: true
    minReplicas: 2
    maxReplicas: 5
    targetAverageUtilization: 98
  custom_xml:
  logs:
    traceSpecification: "*=info"
```

b) To have all UMS services run in the same pod, set the value `dedicated_pods: false`.

Default values are specified for the `replica_count`, `resource`, `autoscaling` and `logs` parameters. You can change the values if the default values do not meet your requirements.

```
#### If dedicated_pods is set to false, the UMS capabilities sso, scim, teamserver
#### run in the same pods with this configuration.
```

```
    replica_count: 2
    resources:
      limits:
        cpu: 500m
        memory: 512Mi
      requests:
        cpu: 200m
        memory: 256Mi
    autoscaling:
      enabled: true
      min_replicas: 2
      max_replicas: 5
      target_average_utilization: 98
    custom_xml:
    logs:
      traceSpecification: "*=info"
```

6. If you want to change the database connection pool sizes, health parameters, or certificate checking options, you can modify the settings that are described in “UMS advanced parameters” on page 65 by using the `custom_xml` section.
7. From 23.0.1, JDBC drivers are provided for all databases.
   a. For 21.0.3 and previous versions:

      If you are using an Oracle, MS SQL, or PostgreSQL database, specify as shown in the `ums_configuration`:

      ```
      use_custom_jdbc_drivers: true
      existing_claim_name: operator-shared-pvc
      ```

      Where `operator-shared-pvc` is the persistent volume claim that you prepared for the operator.
   b. For 23.0.1 and later versions:

      If you want to use custom JDBC drivers, instead of the provided default JDBC drivers, specify as shown in the `ums_configuration`

      ```
      use_custom_jdbc_drivers: true
      existing_claim_name: operator-shared-pvc
      ```

      Where `operator-shared-pvc` is the persistent volume claim that you prepared for the operator.
8. Customize any other UMS configuration settings as necessary to suit your requirements. For example, in the sections `oauth`, `resources`, `autoscaling`, or `logs`:

```
  oauth:
    # optional: full DN of an LDAP group that is authorized to manage OIDC clients, in
addition to primary admin from admin secret
    client_manager_group:
    # optional: full DN of an LDAP group that is authorized to manage app_tokens, in
addition to primary admin from admin secret
    token_manager_group:
    # optional: lifetime of OAuth access_tokens. default is 7200s
    access_token_lifetime:
    # optional: lifetime of app-tokens. default is 366d
    app_token_lifetime:
    # optional: lifetime of app-passwords. default is 366d
    app_password_lifetime:
    # optional: maximimum number of app-tokens or app-passwords per client. default is 100
    app_token_or_password_limit:
    # optional: encoding / encryption when sotring client secrets in OAuth database.
Default is xor for compatibility. Recommended value is PBKDF2WithHmacSHA512
    client_secret_encoding:
    use_custom_binaries: false
    service_type: Route
    routes_ingress_annotations:

  resources:
    limits:
      cpu: 500m
      memory: 512Mi
    requests:
      cpu: 200m
      memory: 256Mi
  ## Horizontal Pod Autoscaler
```

```
  autoscaling:
    enabled: true
    min_replicas: 2
    max_replicas: 5
    target_average_utilization: 98
  use_custom_jdbc_drivers: false
  use_custom_binaries: false
  custom_secret_name:
  custom_xml:
  logs:
    console_format: json
    console_log_level: INFO
    console_source: message,trace,accessLog,ffdc,audit
    trace_format: ENHANCED
    trace_specification: "*=info"
```

9. Specify the following:

```
shared_configuration
  sc_deploy_zen_with_iaf: false
```

10. Specify an LDAP group that is authorized to administer UMS Teams by adapting the following snippet, where *UMSTeamsAdmins* is the name of the UMS Teams administration group:

```
ums_configuration:
  teamserver:
    admingroup: 'cn=UMSTeamsAdmins,dc=example,dc=org'
```

Remember: After deployment, you can administer teams for your business needs by logging on to the UMS Teams Management UI at `https://ums_host/teamserver/ui` using a user ID that is in the UMS Teams administration group.

Completing post-deployment tasks for User Management Service

Containers:
20.x: You can perform optional configuration tasks for User Management Service (UMS) to configure Business Automation Workflow or IBM® Process Federation Server to use UMS, or delegating authentication to an OIDC or SAML identity provider..

Procedure

1. Optional: If you want to configure Business Automation Workflow or IBM Process Federation Server to use UMS single sign-on, perform the following task: .
2. Optional: .
3. Optional: .

Delegating authentication to a Security Assertion Markup Language (SAML) identity provider

Optional: User Management Services (UMS) can delegate authentication to SAML identity provider (IdP). For more information, see Delegating authentication to a third-party identity provider.

Before you begin

Your OIDC Identity Provider (IdP) must be accessible and you need the following information for your IdP:

Procedure

To configure UMS to delegate authentication to a SAML IdP, perform the following steps:

1. Enable the samlWeb-2.0 feature in UMS.
   a) Edit your custom resource `ums_configuration.custom_xml` file to include the following snippet:

```
custom_xml: |
  <server>
    <featureManager>
      <feature>samlWeb-2.0</feature>
    </featureManager>
    <samlWebSso20 id="defaultSP" httpsRequired="true">
      <authFilter>
        <userAgent id="disable" matchType="equals" agent="samldefault" />
      </authFilter>
    </samlWebSso20>
     <samlWebSso20 id="umsSP" httpsRequired="true" idpMetadata="/opt/ibm/wlp/usr/shared/
resources/custom-binaries/idpMetadata.xml" >
      <authFilter>
        <requestUrl id="authorizeEndpoint" urlPattern="/oidc/endpoint/ums/authorize"
matchType="contains"/>
        <requestHeader id="allowBasicAuth" matchType="notcontain" name="Authorization"
value="Basic" />
      </authFilter>
    </samlWebSso20>
  </server>
```

   Important: Because the SAML configuration is not yet complete, the `authFilter` configuration is used to prevent UMS from delegating authentication at this point. The `defaultSP` configuration is included to prevent the SAML feature from creating an incomplete (but active) default configuration.

   b) Apply the UMS configuration, by running the following command:

```
oc apply -f ums-saml.yaml
```

   Where `ums-saml.yaml` is the full custom resource file that includes the previous snippet.

2. Configure UMS to delegate authentication to the IdP.
   a) Create a persistent volume (PV) and a persistent volume claim (PVC) to make the `idpMetadata.xml` file available to UMS.
      i) Create a `ums-saml-pvc.yaml` file based on the following sample:

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: data-pv
  labels:
    type: icp4a-pv
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: inf-node
  mountOptions:
    - nolock
  nfs:
    path: /data
    server: NFS_server_hostname_or_IP_address
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: data-pvc
spec:
  accessModes:
    - ReadWriteMany
  volumeMode: Filesystem
  storageClassName: inf-node
  resources:
    requests:
      storage: 1Gi
  selector:
```

```
    matchLabels:
      type: icp4a-pv
  volumeName: data-pv
```

ii) Create the PV and PVC, by running the following command:

```
oc apply -f ums-saml-pvc.yaml
```

b) Copy the `idpMetadata.xml` metadata file for your IdP to the `custom-binaries`:

```
/data
   +-- custom-binaries
      +-- idpMetadata.xml
```

During deployment, the operator mounts `idpMetadata.xml` from this location into the `/opt/ibm/wlp/usr/shared/resources/custom-binaries/` directory on the file system where UMS is installed.

3. Edit the custom resource file.

a) In the section `ums_configuration`, configure the parameter `existing_claim_name` to point to the persistent claim that you defined in the previous step and set the parameter `use_custom_binaries` to `true`:

```
ums_configuration:
    …
    existing_claim_name: data-pvc
    …
    use_custom_binaries: true
    …
```

b) Modify the `ums_configuration.custom_xml` property to point to the location of the `idpMetadata.xml` file and configure the `authFilter` to delegate authentication to the IdP for all URLs that contain the string `/oidc/endpoint/ums/authorize`.

- If you use dedicated pods for UMS capabilities, add the following configuration information to the `custom_xml` parameter in the configuration of the `sso` pod in the `ums_configuration` section.
- If you deploy all UMS capabilities in one pod, add the following configuration information to the `custom_xml` parameter in the `ums_configuration` section.

```
custom_xml: |
  <server>
    <featureManager>
      <feature>samlWeb-2.0</feature>
    </featureManager>
    <samlWebSso20 id="defaultSP" httpsRequired="true">
      <authFilter>
        <userAgent id="disable" matchType="equals" agent="samldefault" />
      </authFilter>
    </samlWebSso20>
     <samlWebSso20 id="umsSP" httpsRequired="true" idpMetadata="/opt/ibm/wlp/usr/shared/
resources/custom-binaries/idpMetadata.xml" >
      <authFilter>
        <requestUrl id="authorizeEndpoint" urlPattern="/oidc/endpoint/ums/authorize"
matchType="contains"/>
        <requestHeader id="allowBasicAuth" matchType="notcontain" name="Authorization"
value="Basic" />
      </authFilter>
    </samlWebSso20>
  </server>
```

c) Apply the updated configuration by running the following command:

```
oc apply -f ums-saml.yaml
```

d) After the resources are updated, verify that UMS delegates authentication to the identity provider by navigating your browse to one of the Business Automation Workflow resources that use UMS SSO, for example UMS Teams:

```
https:/ums-teams-route/teamserver/ui/
```

Where *ums-teams-route* is the host name of the UMS Teams server.

Verify that you are redirected to the SAML endpoint of your identity provider, and that after entering correct credentials, you are successfully logged into the UMS Teams UI.

e) If UMS does not delegate authentication to the identity provider, verify that the `idpMetadata.xml` file is available in the `custom-binaries` directory of the UMS pod by running the following command:

```
oc exec <ums-pod> -- ls /opt/ibm/wlp/usr/shared/resources/custom-binaries/
```

If necessary, modify the custom resource to point to that directory, and apply the changes:

```
<samlWebSso20
   id="umsSP"
   httpsRequired="true"
   idpMetadata="/opt/ibm/wlp/usr/shared/resources/custom-binaries/idpMetadata.xml" />
```

Results

This configuration ensures a seamless single sign-on experience for browser users.

What to do next

If you want to create a client application that will invoke the REST APIs of an Business Automation Workflow component that uses UMS single sign-on (SSO), the client app must present an access token. To learn how to obtain an access token from a command line, browser, mobile, or liberty-based application, see “Invoking OAuth 2.0 protected APIs” on page 22.

Delegating authentication to an OIDC Identity Provider

Optional: User Management Services (UMS) can delegate authentication to an OIDC Identity Provider (IdP). For more information, see Delegating authentication to a third-party identity provider.

Before you begin

Your OIDC Identity Provider (IdP) must be accessible and you need the following information for your IdP:

- `authorizationEndpointUrl`
- `tokenEndpointUrl`
- `issuerIdentifier`
- `jwkEndpointUrl`
- `signatureAlgorithm`

Procedure

To configure UMS to delegate authentication to an OIDC IdP, perform the following steps:

1. Register UMS as OIDC client of the OIDC IdP by following the instructions of your identity provider to register UMS as an OIDC client.
2. Obtain the signer certificate of your OIDC IdP.
3. Create an OIDC IdP secret.
   a) Create a configuration file for the secret, for example, named `idp-secret.yaml`, and add the signer certificate that you obtained in the previous step as the value of the property `tls.crt`:

```
apiVersion: v1
kind: Secret
metadata:
  name: idp-tls
type: Opaque
stringData:
  tls.crt: |+
    -----BEGIN CERTIFICATE-----
    <include the IdP certificate>
    -----END CERTIFICATE-----
```

Where *idp-tls* is the name of the secret.

4. Create the secret.

   In the namespace where UMS will be deployed, run the command:

   ```
   oc apply -f idp-secret.yaml
   ```

5. To configure UMS to delegate authentication to the IdP, make the following edits to the Custom Resource (CR):

   a) Add the name of the IdP certificate secret to the `shared_configuration.trusted_certificate_list` section of the CR.

   For example:

   ```
       trusted_certificate_list:
         - idp-tls
   ```

   During deployment, the operator adds the IdP signer certificate to the truststore of UMS.

   b) In the `ums_configuration` section, specify the OpenID Client configuration in the `custom_xml` parameter, including the information about the IdP and specify the `authFilter` value to redirect only URLs that point to `/oidc/endpoint/ums/authorize`.

   - If you use dedicated pods for UMS capabilities, add the following configuration information to the `custom_xml` parameter in the configuration of the `sso` pod in the `ums_configuration` section.
   - If you deploy all UMS capabilities in one pod, add the following configuration information to the `custom_xml` parameter in the `ums_configuration` section.

   For example:

   ```
       custom_xml: |
         <server>
           <openidConnectClient id="client_id"
             clientId="client_id"
             clientSecret="client_secret"
             authorizationEndpointUrl="authorizationEndpointUrl"
             tokenEndpointUrl="tokenEndpointUrl"
             issuerIdentifier="issuerIdentifier"
             jwkEndpointUrl="jwkEndpointUrl"
             signatureAlgorithm="signatureAlgorithm">
             <authFilter>
               <requestUrl matchType="contains" urlPattern="/oidc/endpoint/ums/authorize"></
   requestUrl>
               <requestHeader id="allowBasicAuth" matchType="notcontain"
   name="Authorization" value="Basic" />
             </authFilter>
           </openidConnectClient>
         </server>
   ```

   Tip: For more information about the parameters in the OIDC client configuration see Configuring an OpenID Connect Client in Liberty.

   During deployment, the operator adds the OIDC client configuration to the server configuration of UMS.

   c) If you use dedicated pods for UMS capabilities. also include the following `featureManager` section in `custom_xml` to add the required `openidConnectClient-1.0` feature.

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

   For example:

   ```
       # Configuration for sso pods
       sso:
         ...
         custom_xml: |
           <server>
             <featureManager>
               <feature>openidConnectClient-1.0</feature>
   ```

```
            </featureManager>
            <openidConnectClient id="client_id"
              clientId="client_id"
              clientSecret="client_secret"
              authorizationEndpointUrl="authorizationEndpointUrl"
              tokenEndpointUrl="tokenEndpointUrl"
              issuerIdentifier="issuerIdentifier"
              jwkEndpointUrl="jwkEndpointUrl"
              signatureAlgorithm="signatureAlgorithm">
              <authFilter>
                <requestUrl matchType="contains" urlPattern="/oidc/endpoint/ums/
authorize"></requestUrl>
                <requestHeader id="allowBasicAuth" matchType="notcontain"
name="Authorization" value="Basic" />
              </authFilter>
            </openidConnectClient>
          </server>
```

Configuring UMS routes for load balancing

If you are not using Red Hat OpenShift Kubernetes Service (ROKS) or OpenShift Container Platform (OCP) and you have a load balancer in front of your User Management Service services, you must define `serviceType: NodePort` endpoints.

Procedure

1. If the `ums_configuration.hostname` parameter is defined, define a load balancer route for each of the following endpoints:
   - `ums-sso-`*`hostname`*`:`*`port`*
   - `ums-teams-`*`hostname`*`:`*`port`*
   - `ums-scim-`*`hostname`*`:`*`port`*
   - `ums-`*`hostname`*`:`*`port`*

   Where *hostname* is the value of the `ums_configuration.hostname` parameter, and *port* is the port number (typically 443).
2. If the `ums_configuration.hostname` parameter is not defined, define a load balancer route for the deprecated catch all endpoint:
   - `ums.`*`hostname`*`:`*`port`*

   Where *hostname* is the value of the `shared_configuration.sc_deployment_hostname_suffix` parameter, and *port* is the port number (typically 443).

Using the User Management Services

User Management Services (UMS) provide single sign-on (SSO) and Teams options, which provide users of multiple applications with a single sign-on experience and centrally administered teams.

User Management Service single sign-on

You can use the User Management Service (UMS) single sign-on (SSO) option to provide users of multiple applications with a single sign-on experience.

You can use UMS SSO to provide a common login page for Business Automation Workflow apps that are enabled to delegate authentication to the service. If you have multiple deployments, users can have a single sign-on experience when they interact with more than one of them.

Because Business Automation Workflow combines several technologies and runtime servers in your virtual cloud-based environments, UMS helps you manage this complexity by consolidating aspects of user management in a single place.

UMS SSO brings the following advantages:

- Reuses existing customizations of Trust Association Interceptors for single sign-on.
- Provides an authentication scheme that is based on the open standards OpenID Connect and OAuth 2.0.
- Familiarity for many administrators from a configuration and operations perspective.

If an unauthenticated user requests access to a protected resource that is owned by Business Automation Workflow, then the user is redirected to UMS to sign on. After the authentication completes successfully, the user is redirected back to the web application, which then checks the user's authorization and, if successful, returns the protected resource. The OpenID Connect protocol requires that the Offering Party and Relying Party are made known to each other as part of the configuration. This happens as part of the automated setup process when Business Automation Workflow is installed using the operator.

The following sections describe what happens between login and logout for containers that delegate authentication to UMS.

OpenID Connect (OIDC) login

When an unauthenticated user requests a protected URL from an application, for example IBM Business Automation Studio, the browser is redirected to UMS for authentication. Upon authentication in UMS, a session with UMS is established that uses cookies, and the user is redirected back to Business Automation Studio to complete the login sequence. Business Automation Studio also establishes a session with the browser by using cookies. Two independent sessions with two servers are open from the same browser.

Single Sign-On

When the same user from the same browser attempts to access a different application or another instance of the same application, the user is redirected to UMS for authentication. Because the browser already has an established cookie-based session with UMS, the user is not prompted for credentials. The user is redirected to the second application, which completes the login sequence and another cookie-based session is established.

Identity propagation

As part of the login procedure, an app, for example Business Automation Studio, obtains a set of tokens that can be forwarded to other connected systems. This enables Business Automation Studio to invoke APIs on behalf of the current end user.

Tip: You can register additional clients with UMS so that your custom web apps or mobile apps can use UMS for authentication and invoke APIs on behalf of the end user. For more information about how to register clients, see "Invoking OAuth 2.0 protected APIs" on page 22.

Logout

Users often interact with one or more applications from their browser. When a user clicks **Logout** in an application, a request is sent from the browser to the server and the server-side session is invalidated and cookies are removed.

Each application is configured to redirect a browser to the UMS logout endpoint `/oidc/endpoint/ums/logout`. UMS invalidates the session and clears the related cookies.

Important: Other applications are not notified about the logout and session termination in UMS. The user sessions with other applications remain open. To close all open sessions, a user must close the browser or **Logout** from each application. Because App Engine does not expose a logout page for the user to perform a logout, the user must close the browser.

Invoking OAuth 2.0 protected APIs

For your client application to invoke a REST API that is provided by a component that has been configured to delegate authentication to User Management Service (UMS) single sign-on (SSO), the client app must present an access token.

The following topics describe how different types of applications can use appropriate OAuth 2.0 authorization flows to obtain an access token.

How a custom command line application obtains an access token from UMS SSO

Because a custom command line application cannot redirect a user to a browser UI for authentication, such applications can use the Resource Owner Password Credentials flow to obtain an access token from User Management Service (UMS) single sign-on (SSO) that can be used to invoke an OAuth 2.0 protected REST API.

Understanding the Resource Owner Password Credential flow

In the Resource Owner Password Credentials flow, resource owner credentials, such as username and password, are used directly to obtain an `access_token`. The custom command line application therefore initially needs to obtain credentials from the resource owner (the end user). It can then invoke UMS SSO, authenticate as a registered client application and exchange the user's username and password combination for an `access_token`. This flow can also be used with client types other than command line, but this is the most typical usage scenario.

Design considerations for the custom command line application

1. The custom command line application must register with UMS SSO as an OIDC Relying Party, for example:

```
curl -v -k -s -X POST -H "Content-Type:application/json" -u "umsadmin:passw0rd" -d @-
"https://<ums-host>/oidc/endpoint/ums/registration" <<+++
{
  "scope": "openid",
  "preauthorized_scope": "openid",
  "introspect_tokens": true,
  "client_id": "customApp",
  "client_secret": "passw0rd",
  "client_name": "customApp",
  "grant_types": ["password"],
  "response_types": [ "token"]
}
+++
```

   Where

   - *passw0rd* is an example client_secret to authenticate as the custom command line application to UMS - make sure that you use a much stronger secret
   - *customApp* is a human-readable identifier for your custom command line application
   - `grant_types` must be set to “`password`”
   - `response_types` must be set to “`token`”

2. Then the app can obtain an access token. For example:

```
curl -k -X POST -u "customApp:passw0rd" -d
"grant_type=password&scope=openid&username=user_name&password=user_password" "https://ums-
host/oidc/endpoint/ums/token"
```

   Where

   - option `-u "customApp:passw0rd"` is used by the client to authenticate with UMS, it is the combination of the values for `client_id` and `client_secret` that you registered in the previous step.

- `grant_type` must be set to "`password`".
- *`user_name`* and *`user_password`* are the credentials of the resource owner user name for whom the access token is being requested.
- *`ums-host`* is the hostname of the UMS server.

The response contains the access token, `access_token`, for example:

```
{
  "access_token": "uEsdnucnBtjt8llTYQDqKHxcPF7a06YLX1IbzQH8",
  "token_type": "Bearer",
  "expires_in": 7199,
  "scope": "openid"
}
```

3. The custom command line application uses the `access_token` in the authorization header of the request to invoke the OAuth 2.0 protected REST API. For example:

```
curl -k -s -H "Authorization: Bearer $access_token" https://my.server:9443/rest/bpm/wle/v1/
user/current
```

How a browser or mobile application obtains an access token from UMS

A browser or mobile application can use the Implicit flow to obtain an access token from User Management Service (UMS) single-sign-on (SSO) that can be used to invoke an OAuth 2.0 protected REST API.

Understanding the Implicit flow

In the Implicit flow, the client is issued an access token directly. This contrasts with the Authorization flow, which issues the client with an authorization code.

When UMS issues an access token, the client identity is verified by using the redirection URI that is used to deliver the access token to the client. The URI must be registered when the client application is registered with UMS SSO. It is not possible to make use of a client secret because a browser cannot keep secrets.

Design Considerations for the custom app

1. The client app must register with UMS SSO as an OIDC Relying Party. For example:

```
curl -v -k -s -X POST -H "Content-Type:application/json" -u "umsadmin:passw0rd" -d @-
"https://<ums-host>/oidc/endpoint/ums/registration" <<+++
{
 "scope": "openid",
 "preauthorized_scope": "openid",
 "introspect_tokens": true,
 "client_id": "browser",
 "client_name": "browser",
 "response_types": ["token"],
 "grant_types": ["implicit"],
 "redirect_uris": [ "https://your-browser-app-IP/tokenReceiver.html " ]
}
+++
```

Where

- `grant_types` must be set to `implicit`.
- `response_types` must be set to `token`.
- `redirect_uris` acts as a whitelist, you can specify a list of URIs.

Tip: If you want to use wild-cards or regular expressions in the `redirect_uris`, you must include

```
"allow_regexp_redirects": true
```

For more information, see Configuring an OpenID Connect Provider to accept client registration requests.

2. The browser application requests an access token from the OAuth authorization server, for example, by making the user's browser visit the `/authorize` endpoint and including a link that redirects to UMS SSO for authentication. For example:

```
<body>
  <h1>OAuth 2.0 Implicit</h1>
  <p>Have a look at the URL bar. If there is no token, click
  <a href="https://<ums-host>/oidc/endpoint/ums/
authorize?response_type=token&client_id=browser&scope=openid&state=none&redirect_uri=http://
192.168.99.100:8080">here</a></p>
  </p>
...
</body>
```

   In this example, the browser application passes the following parameters:

   - `response_type=token` because the app is requesting an `access_token` (not an `id_token`)
   - `client_id=browser`: The app needs to identify itself so that the authorization server can verify that the `redirect_uri` is one of the preregistered URLs and so that it can also include the `client_id` in the token information.
   - `scope=openid` is pre-authorized.
   - `state=none` is a short-cut in this example. This parameter is meant to allow the client (browser app) to make sure that it sent the user to the authorization server and prevent any unsolicited invocations. It can also help re-establishing context if the browser app had saved several parameters, such as in HTML5 session storage.
   - `redirect_uri` is the URI to come back to after the user has authenticated and given their consent (depending on the scopes).

3. When the request is redirected back to the browser application, several parameters are passed in the URL, including the access token.

   In the following code snippet, `readTokenFromUrl()` demonstrates how to parse name value pairs out of the URL bar and store them in a complex variable named `params`.

```
var api_base_url = "https://sample.api.server/rest/objectstore" //construct the target API
URL for the service you want to invoke
var params = {};

function readTokenFromUrl() {
    var queryString = location.hash.substring(1);
    var regex = /([^&=]+)=([^&]*)/g;
    var m;
    while (m = regex.exec(queryString)) {
      params[decodeURIComponent(m[1])] = decodeURIComponent(m[2]);
    }

    if (params.access_token==undefined) {
    addTextNode("access_token", "not available yet");
    } else {
    addTextNode("access_token", params.access_token);
    addTextNode("scopes", params.scope);
    addTextNode("token_type", params.token_type);
    addTextNode("expires_in", params.expires_in);
  }
}

function getValue() {
  var key = document.getElementById("key").value;
    var xmlhttp = new XMLHttpRequest();
    try {
        xmlhttp.onreadystatechange=function() {
                  if (xmlhttp.readyState==4 && xmlhttp.status==200) {

document.getElementById("response").innerText=xmlhttp.responseText;
                }
      }
        xmlhttp.open("GET",api_base_url+"/" + key, true);
        xmlhttp.setRequestHeader("Authorization","Bearer " + params.access_token);
        xmlhttp.send();
```

```
    } catch (err) { Console.log(err.message); }
}

function setValue(key, value) {
  var key = document.getElementById("key").value;
  var value = document.getElementById("value").value;
    var xmlhttp = new XMLHttpRequest();
    try {
        xmlhttp.onreadystatechange=function() {
                  if (xmlhttp.readyState==4) {
              if (xmlhttp.status==200)
{ document.getElementById("response").innerText="success" }
          } else if (xmlhttp.satus==403)
{  document.getElementById("response").innerText="not authorized" }
        }
        xmlhttp.open("PUT",api_base_url+"/" + key, true);
        xmlhttp.setRequestHeader("Authorization","Bearer " + params.access_token);
        xmlhttp.setRequestHeader("Content-Type","application/json");
        xmlhttp.send(value);

    } catch (err) {return err.message; }
}

function addTextNode(elementId, text) {
  var textnode = document.createTextNode(text);
  var parent = document.getElementById(elementId);

  parent.appendChild(textnode);
}
```

4. The code in the custom application can now use the access token to make XML HTTP Requests (XHR) to invoke OAuth 2.0 protected REST APIs. It must pass the value of the access token with each API request in the request header `Authorization`.

How a Liberty-based application obtains an access token from UMS SSO

A Liberty-based application can use the Authorization Code flow to act as an OIDC Relying Party (RP) to delegate authentication to UMS single sign-on (SSO), which acts as an OIDC Offering Party (OP).

Understanding the Authorization code flow

1. The user's browser has attempted to access a protected resource on the RP and is redirected to the OP (UMS SSO), including values for `client_id` and `redirect_url` as registered.
2. The user authenticates.
3. The OP redirects back to the RP, including an authorization code.
4. The RP sends the authorization code and client credentials using a back-channel communication.
5. The OP responds with `id_token`, `refresh_token`, `access_token`.

Register your server with UMS as shown below and configure your server as OIDC client based on the UMS configuration available at: `https://<ums-host>/oidc/endpoint/ums/.well-known/openid-configuration`

Design Considerations for the custom app

1. Register the Liberty server as an OIDC client with UMS SSO. For example:

```
curl -k -s -X POST -H "Content-Type:application/json" -u "umsadmin:passw0rd" -d @- "https://
ums-host/oidc/endpoint/ums/registration" <<+++ | jq '.'
{
   "scope": "openid profile",
   "preauthorized_scope": "openid profile",
   "introspect_tokens": true,
   "client_id": "liberty",
   "client_secret": "liberty-secret",
   "client_name": "liberty",
   "redirect_uris": [
      "https://liberty-host:10001/oidcclient/redirect/umsClient"
   ]
```

```
}
+++
```

Where

- `-u "umsadmin:passw0rd"`: The user and password used by the client to authenticate with UMS SSO.
- Optional: `"client_id": "liberty"` corresponds to the configuration in the `server.xml` configuration file.
- Optional: `"client_secret": "liberty-secret"` corresponds to the configuration in the `server.xml` configuration file.
- `"redirect_uris"` are the URIs to which the user (and the authorization code) are redirected upon successful authentication. This must match one of the redirect URIs that were specified when the custom application registered with UMS SSO.
- `"scope": "openid profile"` are the scopes that are represented by the access token, `access_token`, that is included in the final response to the authentication request. The `openid` scope is required to complete the flow. The `profile` scope should allow calling the `UserInfo` endpoint. Additional scopes are possible and depend on the application(s) where the token will be used.

Note: If you omit the optional `client_id` and `client_secret` parameters, random values are generated and returned in the registration response.

2. Extend the Liberty server configuration in `wlp/usr/servers/`*`server_Name`*`/server.xml` to configure your UMS host name and port number. Choose a `clientId` and a `clientSecret` and add them to the configuration.

```
<variable name="ums_prefix" value="https://<ums-host>" />

<openidConnectClient id="umsClient"
  authorizationEndpointUrl="${ums_prefix}/oidc/endpoint/ums/authorize"
  clientId="liberty"
  clientSecret="liberty-secret"
  tokenEndpointUrl="${ums_prefix}/oidc/endpoint/ums/token"
  validationEndpointUrl="${ums_prefix}/oidc/endpoint/ums/introspect"
  validationEndpointUrl="${ums_prefix}/oidc/endpoint/ums/userinfo"
  userInfoEndpointEnabled="true"
  userInfoEndpointUrl="${ums_prefix}/oidc/endpoint/ums/userinfo"
  validationMethod="userinfo"
  inboundPropagation="supported"
  signatureAlgorithm="RS256"
  jwkEndpointUrl="${ums_prefix}/oidc/endpoint/ums/jwk">
</openidConnectClient>
```

3. When an end user attempts to access a protected resource on the RP without having an established authenticated session, the RP redirects the user's browser to the UMS authorization endpoint `https://ums-host/oidc/endpoint/ums/authorize?response_type=code&client_id=liberty&state=001558899960297TBjhvTZSn&redirect_uri=https%3A%2F%2Fliberty-host%3A10001%2Foidcclient%2Fredirect%2FumsClient&scope=openid+profile`.

   In the Authorization flow, the request to handle delegated authentication contains the following parameters:

   - `response_type=code` indicates the authorization code flow.
   - `client_id=liberty` corresponds to the configuration in the `server.xml`configuration file.
   - `state=`*`random_number`* a nonce that the RP can use later to validate that an invocation belongs to a flow that it triggered.
   - `redirect_uri` specifies the URI to which the user (and the authorization code) are redirected upon successful authentication. This must match one of the redirect URIs that were specified when the custom application registered with UMS.
   - `scope=openid+profile` specify the scopes that are represented by the access token, `access_token`, that is included in the final response to the authentication request. The `openid`

scope is required to complete the flow. The `profile` scope should allow calling the `UserInfo` endpoint. Additional scopes are possible and depend on the application(s) where the token will be used.

4. The request returns an authorization code. The Liberty server application can call the UMS SSO token endpoint to exchange the authorization code for `access_token`, `refresh_token`, and `id_token`.

   UMS SSO provides all three tokens if the token endpoint is invoked with a valid recent authorization code and a matching `client_id` and `client_secret` for the client that requested authentication.

5. The Liberty server application can now establish an authenticated session based on the information in the `id_token`. Several configurations are possible:

   - The default is to use all information from the `id_token` to create a security context.
   - Use just the user name from the `id_token` and create a security context with information that is found in the locally configured user registry for that user name.
   - Perform some sort of identity mapping. For example, the `id_token` might contain an email address that represents the user, but the RP creates a security context for the employee serial number, which might be contained in some other claim of the `id_token`.

Using long-lived access tokens

If it is inconvenient for your programmatic clients that the User Management Service (UMS) `access_token` tokens have a default validity of two hours, you can exchange an `access_token` for an `app_token` that is valid for 366 days.

Understanding app tokens

App tokens were introduced to support enforcing multi-factor authentication (MFA) for command line clients. Because Liberty does not support MFA natively, it is expected that authentication is delegated further to an existing identity provider that supports MFA. Therefore, the client should use the browser-based `implicit flow` to obtain an initial `access_token`, which is then exchanged for an `app_token` that has a much longer validity.

Design considerations for the custom app

1. The custom application must register with UMS SSO, specifying the `implicit` `grant_type` and `token` `response_type`. For clarity and convenience, in the following examples, the UMS admin credentials are stored in the environment variables `umsuser` and `umspassword` and `localhost:9443` are used for the UMS host and port:

   a. Create a new file named `json.txt` with content similar to the following, substituting your values where necessary:

   ```
   {
     "scope": "openid",
     "preauthorized_scope": "openid",
     "introspect_tokens": true,
     "response_types": ["token"],
     "grant_types": ["implicit"],
     "redirect_uris": [ "http://192.168.99.100:8080", "myapp://token" ],
     "appTokenAllowed": true
   }
   ```

   b. Run the following commands:

   ```
   umsuser=umsadmin
   umspassword=passw0rd
   curl -k -s -X POST -H "Content-Type:application/json" -u "$umsuser:$umspassword" -d
   @json.txt "https://localhost:9443/oidc/endpoint/ums/registration"
   ```

   Example response:

   ```
   {
       "client_id_issued_at": 1572602365,
   ```

```
    "registration_client_uri": "https://localhost:9443/oidc/endpoint/ums/registration/
1657e00324474d8b9fbe637b0883515d",
    "client_secret_expires_at": 0,
    "token_endpoint_auth_method": "client_secret_basic",
    "scope": "openid",
    "grant_types": ["implicit"],
    "response_types": ["token"],
    "application_type": "web",
    "preauthorized_scope": "openid",
    "introspect_tokens": true,
    "resource_ids": [],
    "proofKeyForCodeExchange": false,
    "publicClient": false,
    "appPasswordAllowed": false,
    "appTokenAllowed": true,
    "hash_itr": 0,
    "hash_len": 0,
    "client_id": "1657e00324474d8b9fbe637b0883515d",
    "client_secret": "5a8pw7rQWN3VeKGcfvsIs039zHHrmiliprDAVKlLvbK9yrnhHgOulvKpbLgN",
    "client_name": "1657e00324474d8b9fbe637b0883515d",
    "redirect_uris": ["http://192.168.99.100:8080", "myapp://token"],
    "allow_regexp_redirects": false
}
```

c. Note the values for `client_id` and `client_secret`. For demonstration purposes, you might store them in environment variables:

```
clientid=1657e00324474d8b9fbe637b0883515d
clientsecret=5a8pw7rQWN3VeKGcfvsIs039zHHrmiliprDAVKlLvbK9yrnhHgOulvKpbLgN
```

2. The app obtains a short-lived access token. If UMS is not configured to delegate authentication further, we can use basic authorization, for example:

```
curl -k -s -v -u "$umsuser:$umspassword" 'https://localhost:9443/oidc/endpoint/ums/authorize?
response_type=token&client_id=$clientid&scope=openid&state=none&redirect_uri=myapp://token'
```

Where

- `response_type=token` indicates that the app wants an access token (not an id token).
- `client_id=$clientid` identifies itself so that the authorization server can verify that the redirect_uri is in the preregistered whitelist and that it can include the client_id in the token information.
- `scope=openid` is required.
- `state=none` is a short-cut in this example. You can use this parameter to help re-establishing the context, for example, if the browser app had saved some parameters in HTML5 session storage or to make sure that the client actually sent the user to the authorization server.
- `redirect_uri` the URI to come back to after the user authenticated and gave consent (depending on scopes).

The 302 redirect response will contain an HTTP response header location that is pointing to the registered and requested location (myapp://token) with an appended fragment that includes among other things your `access_token`, for example:

```
location: myapp://
token#scope=openid&access_token=C87wJE2Mst7DEdBJMfV9Rw6I8RUlZ9bilITmZqhK&token_type=Bearer&ex
pires_in=7199&state=none
```

The short-lived `access_token` is `C87wJE2Mst7DEdBJMfV9Rw6I8RUlZ9bilITmZqhK` expires in 7199 seconds.

3. Exchange the `access_token` for long-lived `app_token` by using the access token to call the UMS app-tokens endpoint. For example:

```
curl -k -X POST -u "$clientid:$clientsecret" -d "app_name=myapp" -H "Accept: application/
json" -H "access_token: $accesstoken" "https://localhost:9443/oidc/endpoint/ums/app-tokens"
```

The response contains the app-token: GGONoU7vG9uJ6jJQOwKO7nBouq9sB08WHiOLzv0qBO

```
{
    "app_token": "GGONoU7vG9uJ6jJQOwKO7nBouq9sB08WHiOLzv0qBO",
    "app_id": "Lk088OmmPgwAmglhR4skWd4OzKmYT4nprrR4xR3e",
    "created_at": "1572603852658",
    "expires_at": "1580379852658"
}
```

For demonstration purposes, you might store the token in an environment variable:

```
apptoken=GGONoU7vG9uJ6jJQOwKO7nBouq9sB08WHiOLzv0qBO
```

Tip: You can set the following lifetimes and limits for app tokens and app passwords in the custom resource file:

ums_configuration.oauth.app_token_lifetime
The lifetime of app tokens. The default is 366d.

ums_configuration.oauth.app_password_lifetime
The lifetime of app passwords. The default is 366d.

ums_configuration.oauth.app_token_or_password_limit
The maximum number of app tokens or app passwords per client. The default is 100.

4. Use the app_token just like any other access_token to invoke an API. for example:

```
curl -k -s -H "Authorization: Bearer $apptoken" https://sample-host/rest/bpm/wle/v1/user/
current
```

User Management Service Teams

You can use the User Management Service (UMS) Teams option to administer global teams across different Business Automation Workflow components.

The UMS Teams option is a microservice that manages global teams for Business Automation Workflow apps and components.

What are UMS Teams?

A team is a collection of users, groups, and other teams that already exist. Unlike users and groups, teams are not stored in LDAP. The nesting depth of teams is unlimited, but cycles are not allowed. The UMS Teams option provides the team management capability, but does not perform authorization on behalf of the apps and components, which apply their own authorization policies. Teams can be associated with resources in the various Business Automation Workflow components, such as desktops in Navigator or apps in App Engine to control authorization to these resources.

This approach has several advantages over using your company-wide user registry, such as LDAP, groups directly:

- Teams can be short-lived or frequently changing, whereas user registry groups are often long-living.
- Global teams can be defined and changed without modifying your company user registry.
- A company-wide administrator is not required to manage these business-oriented teams.
- Avoids the performance problems that can be caused when a company-wide user registry has too many groups.

Identifying teams across different environments

Because each environment (development, test, production) is connected to a different instance of the UMS Teams server, when you move resources between environments the teams that are associated with the resources are not automatically moved with the resources, so you must create corresponding teams on the target environment, if they do not already exist. However, because creating a team in multiple environments with the same team name and for the same purpose will have different unique team IDs

(UUID), it might not be easy to identify corresponding teams just by their display name, which is not guaranteed to be unique, for example, many teams might be named "managers".

To avoid confusion, use team distinguished names (which are guaranteed to be unique in a given environment) consistently across all your environments to identify teams that serve the same purpose. This will make it easier to identify the correct team in a target environment and map it to the resources that you deploy.

Using multiple email addresses

When the user repository contains multiple email addresses for a user, by default, UMS Teams returns only the primary email address of the user, for example when using the REST API call `GET /teamserver/rest/users`. But you can search for users that match with an alternative email address.

You can configure the UMS federated user repository so that UMS Teams delivers all email addresses, but in this case, it marks the first email address as the primary email address, regardless of any other marker that might exist to indicate the primary email address. To enable UMS to return all email addresses, add the following code to the `federatedRepository` section of `server.xml` file, or in the `ums-configmap.yaml.j2` when used to generate the `server.xml` file:

```
<extendedProperty dataType="String" name="emails" entityType="PersonAccount"
multiValued="true" />
```

Administrating UMS Teams

If you are an administrator of the UMS Teams feature, you can create and manage teams by using either the **Team Management** UI or the UMS Teams REST APIs.

Managing teams

Administrators can use the **Team Management** page to view, create, modify, or delete teams. You can build a team out of users, groups, and other teams.

Using the Team Management page

1. In a browser, open the URL `https://`*`ums_host`*`/teamserver/ui`, where *ums_host* is the host name of your UMS server.
2. If you are prompted to log in to your account, enter a valid administrator user name and password.
3. The **Team List** view is displayed as a table of the first page of existing teams.
   - To filter the list of teams by Display Name, click the magnifier icon then enter the text to match. To search on a different field, you can change the search scope from **Display Name** to **Distinguished name**, **Description** or **Team ID**.
   - To sort the teams by a column field, click the column heading.
   - To edit a team, click on its display name.
   - To delete one or more teams, select the teams then click **Delete selected teams**.
   - To copy a team ID to the clipboard, select the team's menu icon then click **Copy team ID**.
   - To create a new team, click **Create a team**, then on the **Basic Team Data** tab, enter the **Display Name**, **Distinguished Name**, and **Description**.
     – To add users to the team, select the **Users** tab, then for each user, enter their distinguished user name and click **Add**.

       On this tab, you can also search for users and add them to the team: Click **Search**, to open the window, enter at least three characters of the string to enable the **Search** button then click it. In the search results you can select one or more users then click **Add selected** to add them to the current team.

  - To add groups to the team, select the **Groups** tab, then for each group, enter the distinguished group name and click **Add**.

    On this tab, you can also search for a group by its display name (as defined in the user registry) and add it to the team: Click **Search**, to open the window, enter at least three characters of the string to enable the **Search** button then click it. In the search results you can select one or more groups then click **Add selected** to add them to the current team.
  - To add an existing team to the new team, select the **Teams** tab, and perform one of the following:
    - If you copied the ID of an existing team (as described earlier), paste the team ID and click **Add**.
    - On this tab, you can also search for existing teams and add them to the team: Click **Search**, to open the window, enter at least three characters of the string to enable the **Search** button then click it. In the search results you can select one or more teams then click **Add selected** to add them to the current team.
  - When you are finished, click **Save** to save the updates and exit or click **Save and Back** to save the updates and remain in the editor so that you can make further changes or to validate that the users or groups in a modified team exist in the user registry.
  - After saving any changes, you can validate that the users or groups in a team exist in the user registry, by clicking **Validate** on the **Users** or **Groups** tab. The results are indicated by check mark icons in the list of users or groups next to those that were verified to exist in the user registry.
- To display all team members, including those that are in member groups and teams, select the **All Users** tab and click **Load all**. The members of the team are calculated and displayed in a table with the following details for each member:
  - Display name
  - Distinguished user name
  - User name
  - Email

User Management Service Teams access control

You can control who is authorized to create User Management Service (UMS) teams, change a team, or read the details of a team either by using J2EE roles when installing the UMS Teams service, or, by specific teams whose members have particular access rights on other teams.

The access permissions have two levels of granularity:

- "Global access permissions" on page 31
- "Team scoped access permissions" on page 32

Global access permissions

The following permissions provide access control for all teams.

Administrators with full access
Global administrators act like a superuser, and can call any UMS Teams service REST API. Global administrators are:

- All members of the J2EE role `teamserveradmin`.
- All members of the global administrators team, which has the uuid `10000000-0000-0000-0000-000000000000`.

Users that can create or update a team
All users of UMS Teams are members of the J2EE role `teamserveruser`. These users can create or write a team if they satisfy one or more of the following conditions:

- The user is member of the creators team, which has the uuid `20000000-0000-0000-0000-000000000000`.
- The user is a global administrator.

Reading details of a team

When creating or updating a team, the user can use the following REST APIs to find out which users exist in the user repository:

- GET /teamserver/rest/groups
- GET /teamserver/rest/users

To call these APIs, a user must satisfy one or more of the following conditions:

- The user is a global administrator.
- The user is member of the "creators" team, which has the uuid `20000000-0000-0000-0000-000000000000`.
- The user is member of the "repository readers" team, which has the uuid `30000000-0000-0000-0000-000000000000`.
- If the user passes the uuid of a team as context and has the permission to update that team. When a user can update a team, the user also has a reasonable need for obtaining the list of users or groups from the user repository.

Granting users authorization

To add a user to one of the predefined teams that has a uuid, *uuid*, a global administrator can call one of the following REST API calls:

- PUT /teamserver/rest/teams/*uuid*
- PATCH /teamserver/rest/teams/*uuid*

For example:

```
PUT /teamserver/rest/teams/30000000-0000-0000-0000-000000000000
```

For examples that use PUT and PATCH, see “Examples: Using the User Management Service Teams REST API” on page 34.

Team scoped access permissions

Each team has the following administrative fields:

`owner`
A user who is the main administrator of the team. The owner cannot be `null`.

`administratorTeam`
A team whose members are also administrators of the team. Members of this team can read and write the entire contents of the team, including the `admin` part. The `administratorTeam` field can be `null`.

`writerTeam`
A team whose members have the permission to change the team content. Members of this team can read and write the entire content of the team except the `admin` part. The `writerTeam` field can be `null`.

`readerTeam`
A team whose members have the permission to read the team content. Members of this team can read the entire contents of the team except the `admin` part. The `readerTeam` field can be `null`.

What the owner of a team can and can't do

The owner (or administrator) of team has the permission to choose which user should be the owner of the team, or which team should be used as reader team, writer team, or administrator team. This does however not imply any permissions on the chosen team.

In the following example, although John Doe is the owner of the exampleteam team and can specify which teams are the readerTeam, writerTeam, and administratorTeam, he can't choose the individual members of those teams unless he is also owner, administrator or writer of them.

```
{
  "uuid": "a1c81405-429a-4fc2-803e-b6e8cf03f7f1",
  "distinguishedName": "cn=exampleteam,ou=team,dc=example,dc=com",
  "displayName": "Example Team",
  "description": "This is a new team",
  "users": [
    "cn=John Doe,ou=User,dc=example,dc=com",
    "cn=Jane Doe,ou=User,dc=example,dc=com"
  ],
  "groups": [
    "cn=Example,ou=Group,dc=example,dc=com"
  ],
  "teams": [
    "a3c90404-419a-4fc5-804e-b7e9cf14f8f2"
  ],
  "admin": {
    "owner": "cn=John Doe,ou=User,dc=example,dc=com",
    "administratorTeam": "aabb0000-517a-44de-413e-a78ccef3f2f2",
    "writerTeam": "ccdd0101-517a-34de-112e-a7616273f343",
    "readerTeam": "eeff0202-732a-5a4f-ee12-b86211c79ff1",
  }
  "metadata": {
    "created": "2020-10-25T14:37:08.198Z",
    "lastModified": "2020-10-25T14:37:08.198Z"
  }
}
```

User Management Service Teams REST API

Your applications can access and administer User Management Service (UMS) teams by using REST calls.

Documentation

After you have deployed UMS Teams, you can access the OpenAPI documentation for the REST APIs that are available on the server by directing your browser to the URL `https://`*`ums_host`*`/ibm/api/explorer`

The Team Service API on the context path `/teamserver` manages global teams as a central service for enabled Business Automation Workflow apps and components.

Teams

The teams REST API supports the discovery, creation, retrieval, modification, and deletion of team resources.

The following summary lists the operations that are available on the `teams` context path:

/teamserver/rest/teams

- GET: Retrieve team definitions.
- POST: Create a Team definition.

/teamserver/rest/teams/{uuid}

- GET: Retrieve a team definition.
- PUT: Replace a team definition.
- DELETE: Delete a team definition.
- PATCH: Update a team definition incrementally.

/teamserver/rest/teams/{uuid}/contained_users

- GET: Retrieve the users that are contained in the specified team.

/teamserver/rest/teams/{uuid}/contained_groups

- GET: Retrieve the groups that are contained in the specified team.

Users

The users REST API supports retrieving information about the current user, checking their team memberships, and checking their permissions.

The following summary lists the operations that are available on the `users` context path:

/teamserver/rest/users/current_user

- GET: Retrieve information about the current user.

/teamserver/rest/users/current_user/teams

- GET: Retrieve information about the which teams the current user is a member of.

/teamserver/rest/users/current_user/member_of_any_team

- GET: Returns true if the current user is a member of any of a specified list of teams.

/teamserver/rest/users/current_user/permission

- GET: Returns a list of team actions, where the value `true` indicates that the current user is allowed to perform the action. For example:

```
{
  "canListMyTeams": true,
  "canListAllTeams": true,
  "canViewTeamDetails": true,
  "canCreateTeam": true,
  "canModifyTeam": true,
  "canReplaceTeam": true,
  "canDeleteTeam": true
}
```

/teamserver/rest/users

- GET: Retrieves information about the list of users. You can use it to search for users that match a filter.

  Important: If users can have have multiple email addresses, see "Using multiple email addresses" on page 30.

/teamserver/rest/users/{userDN}/teams

- GET: Returns a list of team definitions for the teams that the user is a member of.

/teamserver/rest/users/{userDN}/member_of_any_team

- GET: Returns true if the specified user is a member of any of a specified list of teams.

Groups

The groups REST API supports retrieving information about the groups.

The following summary lists the operations that are available on the `groups` context path:

/teamserver/rest/groups

- GET: Retrieves information about the list of groups from the user repository, for example LDAP, by using SCIM. It allows you to search for groups that match a filter.

Examples: Using the User Management Service Teams REST API

The following examples illustrate how to perform common actions on teams by using the REST API.

- "Creating a new team" on page 35

Creating a new team

To create a new UMS team you can use the following REST API call:

```
POST /teamserver/rest/teams
```

You must pass the team definition as input parameters in a JSON object in the request body of the call. A team definition consists of the team's distinguished and optional display name, optional description, and its members, which can consist of LDAP users, LDAP groups, and other teams. The distinguished name is required, and must be unique, which helps when you have to move a team from a test system to a production system. It is meaningful to follow the DN naming rules when creating distinguished names for teams. It is also advisable to provide a display name when creating a new team to make it easy to look-up when searching for a particular team.

To add users and groups as member of the new team, you must provide their distinguished names from the connected LDAP repository. To include another team, you must specify its internal ID (uuid).

For example, to create a new team that is build out of the users "John Doe", "Joe Bloggs" and the group "Department 4711" you can pass the following JSON object:

```
{
  "distinguishedName": "cn=Authors,ou=bpm,dc=example,dc=com",
  "displayName": "Authors",
  "description": "This team writes the technical documentation.",
  "users": [
    "cn=John Doe,ou=User,dc=example,dc=com",
    "cn=Joe Bloggs,ou=User,dc=example,dc=com"
  ],
  "groups": [
    "cn=Department 4711,ou=Group,dc=example,dc=com"
  ]
}
```

The response includes the internal ID (uuid) of the new team, which you will need for example, to modify the team or add this team as member of another team:

```
{
  "description": "This team writes the technical documentation.",
  "displayName": "Authors",
  "distinguishedName": "cn=authors,ou=bpm,dc=example,dc=com",
  "groups": [
    "cn=Department 4711,ou=Group,dc=example,dc=com"
  ],
  "metadata": {
    "created": "2020-02-18T14:28:33.040Z",
    "lastModified": "2020-02-18T14:28:33.040Z"
  },
  "teams": [],
  "users": [
    "cn=Joe Bloggs,ou=User,dc=example,dc=com",
    "cn=John Doe,ou=User,dc=example,dc=com"
  ],
  "uuid": "e60d02c1-7e55-4534-81f8-b3c079e83ee3"
}
```

Creating a new team that includes an existing team

To create another team that includes the "Authors" as a team member you would call POST `/teamserver/rest/teams` with the following JSON object:

```
{
  "distinguishedName": "cn=Reviewers,ou=bpm,dc=example,dc=com",
  "displayName": "Reviewers",
  "description": "This team is responsible for reviewing the documentation.",
  "teams": [
    "e60d02c1-7e55-4534-81f8-b3c079e83ee3"
  ]
}
```

Retrieving a team

To retrieve an UMS team you can use the following REST API call:

```
GET /teamserver/rest/teams/{uuid}
```

For example, to retrieve the "Authors" team (that has the uuid e60d02c1-7e55-4534-81f8-b3c079e83ee3) you can invoke the following REST API call:

```
GET /teamserver/rest/teams/e60d02c1-7e55-4534-81f8-b3c079e83ee3
```

The response object returns the information about the team. For example:

```
{
  "description": "This team writes the technical documentation.",
  "displayName": "Authors",
  "distinguishedName": "cn=authors,ou=bpm,dc=example,dc=com",
  "groups": [
    "cn=Department 4711,ou=Group,dc=example,dc=com"
  ],
  "metadata": {
    "created": "2020-02-18T14:28:33.040Z",
    "lastModified": "2020-02-18T14:28:33.040Z"
  },
  "teams": [],
  "users": [
    "cn=Joe Bloggs,ou=User,dc=example,dc=com",
    "cn=John Doe,ou=User,dc=example,dc=com"
  ],
  "uuid": "e60d02c1-7e55-4534-81f8-b3c079e83ee3"
}
```

Incrementally updating a team

To update an UMS team incrementally you can use the following REST API call:

```
PATCH /teamserver/rest/teams/{uuid}
```

You must pass the update operation in a JSON object in the request body of the call.

For example, to update the description of the "Authors" team you can invoke the following REST API call:

```
PATCH /teamserver/rest/teams/e60d02c1-7e55-4534-81f8-b3c079e83ee3
```

and pass the following JSON object:

```
{
  "operations": [
    {
      "op": "replace",
      "path": "description",
      "value": "This team is responsible for the product documentation."
    }
  ]
}
```

Replacing a team

To replace an UMS team you can use the following REST API call:

```
PUT /teamserver/rest/teams/{uuid}
```

You must pass the new team definition as an input parameter in a JSON object in the request body of the call.

For example, to remove the user "John Doe" you can invoke the following REST API call:

```
PUT /teamserver/rest/teams/e60d02c1-7e55-4534-81f8-b3c079e83ee3
```

and pass the following JSON object:

```
{
  "distinguishedName": "cn=Authors,ou=bpm,dc=example,dc=com",
  "displayName": "Authors",
  "description": "This team is responsible for the product documentation.",
  "users": [
    "cn=Joe Bloggs,ou=User,dc=example,dc=com"
  ],
  "groups": [
    "cn=Department 4711,ou=Group,dc=example,dc=com"
  ]
}
```

Retrieving a list of teams that you are a member of

To retrieve a list of all UMS teams that you are a member of you can use the following REST API call:

```
GET /teamserver/rest/teams?my_teams=true
```

Note that only UMS Team administrators are authorized to perform this call without the my_teams query parameter to retrieve all teams with arbitrary filters. UMS Team users are restricted to see only those teams that they are a member of. To limit the list to those teams whose display name starts with "Aut" you can add the filter query parameter. For more information about filters, see the OpenAPI documentation for the full list of filters. Also, notice that for better readability the required URL query parameters encoding is omitted:

```
GET /teamserver/rest/teams?my_teams=true&filter=displayName SW "Aut"
```

Here is the example response:

```
{
  "items": [
    {
      "description": "This team is responsible for the product documentation.",
      "displayName": "Authors",
      "distinguishedName": "cn=authors,ou=bpm,dc=example,dc=com",
      "groups": [
        "cn=Department 4711,ou=Group,dc=example,dc=com"
      ],
      "metadata": {
        "created": "2020-02-18T14:28:33.040Z",
        "lastModified": "2020-02-18T14:33:57.688Z"
      },
      "teams": [],
      "users": [
        "cn=Joe Bloggs,ou=User,dc=example,dc=com"
      ],
      "uuid": "e60d02c1-7e55-4534-81f8-b3c079e83ee3"
    }
  ],
  "metadata": {
    "startIndex": 1,
    "totalSize": 1
  }
}
```

Deleting a team

To delete an UMS team you can use the following REST API call:

```
DELETE /teamserver/rest/teams/{uuid}
```

The delete request removes the corresponding team from the database and also removes all references from other teams to that team.

For example, to delete the "Authors" team you can invoke the following REST API call:

```
DELETE /teamserver/rest/teams/e60d02c1-7e55-4534-81f8-b3c079e83ee3
```

User Management Service Teams GraphQL API

Your applications can access and administer User Management Service (UMS) Teams by using GraphQL calls.

GraphQL is a powerful query language for APIs standardized by the GraphQL Organization. While the normal REST API returns the response always in a fixed format, GraphQL allows you to specify the exact list of fields that you want to receive in a response to a REST API call. The UMS Teams server supports GraphQL with two REST endpoints:

/teamserver/rest/graphql

- GET: Retrieve team data according to a GraphQL query.
- POST: Retrieve or modify data according to a GraphQL query or mutation.

  Remember: In GraphQL terminology, a modification is know as a "mutation".

Important:

Whereas the UMS Teams REST API (such as GET `teamserver/rest/teams`) usually omits any fields with a null value in the response, GraphQL sets such requested fields explicitly to `null`.

UMS Teams does not support GraphQL subscriptions.

User Management Service Teams GraphQL schema

UMS Teams supports GraphQL queries with respect to a schema that is defined in the SDL language.

```
schema {
    query: Query
    mutation: Mutation
}

# Root type for all queries
type Query {

    # Retrieve a team definition
    team(

        # The unique identifier of the team definition.
        uuid: String!,

        # If "shallow", the team definition contains only the directly
        # contained users, groups and teams. If "deep", the team definition
        # contains the recursively contained users, groups and teams
        # (but LDAP groups are not expanded).
        membership: MembershipEnum = shallow

    ): Team

    # Retrieve a teams collection
    teams(
        # When true, only those team are returned that contain the current user.
        myteams: Boolean = false,

        # When specified, only those team definitions matching the filter
        # expression are returned. A filter is a logical expression consisting
        # of an attribute test (e.g., displayName EQ "ABC") or logical
        # combinations (AND, OR, NOT) of attribute tests. Operator precedence
```

```
        # can be overridden by braces ```()```.
        filter: String = null,

        # Specifies the attributes whose value is used to order the returned
        # items.
        sortBy: TeamSortByEnum = displayName,

        # Specifies whether the sort order is ascending or descending.
        sortOrder: SortOrderEnum = ascending,

        # Specifies the index of the first retrieved result within the list
        # of all elements that match the search filter. The index starts at 1.
        startIndex: Int = 1,

        # Non-negative integer that specified the desired maximum number of
        # retrieved elements per page. If this parameter is missing, all
        # elements are retrieved.
        maxCount: Int = -1,

        # If "shallow", the team definition contains only the directly
        # contained users, groups and teams. If "deep", the team definition
        # contains the recursively contained users, groups and teams (but LDAP
        # groups are not expanded).
        membership: MembershipEnum = shallow

    ): TeamCollection!

    # Check if the current user is a member of any of the provided teams
    userMemberOfAnyTeam(

        # A list of unique identifiers of team definitions.
        teamIds: [String!]

    ): UserMemberOfAnyTeam!

    # Retrieve the info of the current user
    userInfo: UserInfo!

    # Retrieve the permissions of the current user
    userPermission: UserPermission!
}

# Root type for all mutations
type Mutation {
    # Create a team definition
    createTeam(
        # The team definition when creating a new team
        team: TeamInput!
    ): Team

    # Replace a team definition
    replaceTeam(
        # The unique identifier of the team definition.
        uuid: String!

        # The new team definition for the team with the uuid
        team: TeamInput!
    ): Team

    # Delete a team definition
    deleteTeam(
        # The unique identifier of the team definition.
        uuid: String!
    ): String
}

# The new team definition
input TeamInput {
    # The team definition's distinguished name
    # Distinguished names are used to identify a teams in a readable and unique way.
    # The distinguished name must be unique among all teams.
    distinguishedName: String!

    # The team definition's display name
    displayName: String

    # The team definition's description
    description: String

    # The distinguished names of users that belong to the team
    users: [String!]
```

```
    # The distinguished names of groups that belong to the team
    groups: [String!]

    # The UUIDs of subteams that belong to the team
    teams: [String!]
}

# The team definition
type Team {
    # The team definition's unique identifier
    uuid: String!

    # The team definition's distinguished name.
    # Distinguished names are used to identify a teams in a readable and unique way.
    # The distinguished name must be unique among all teams.
    # While the uuid is immutable, the distinguished name can be modified by the user.
    distinguishedName: String!

    # The team definition's display name
    displayName: String

    # The team definition's description
    description: String

    # The distinguished names of users that belong to the team
    users: [String!]

    # The distinguished names of groups that belong to the team
    groups: [String!]

    # The subteams that belong to the team
    teams: [Team!]

    # Meta data about the creation and update timestamps
    metadata: MetaData
}

# Result of userMemberOfAnyTeam query
type UserMemberOfAnyTeam {
    # True, if the current user is a member of any of the provided teams
    memberOfAnyTeam: Boolean!
}

# The information for a user.
type UserInfo {
    # This is a short name uniquely identifying a user. This is expected to be stable
    # and typically matches what the user specified during login.
    userName: String

    # The user's distinguished name.
    distinguishedName: String

    # The distinguished names of groups the user belongs to.
    groups: [String!]
}

# The permissions of a user indicate what operations the user is allowed to perform.
type UserPermission {
    # True if the user is permitted to retrieve the list of his own teams.
    canListMyTeams: Boolean!

    # True if the user is permitted to retrieve the list of all teams.
    canListAllTeams: Boolean!

    # True if the user is permitted to view the details of each team.
    canViewTeamDetails: Boolean!

    # True if the user is permitted to create a new team.
    canCreateTeam: Boolean!

    # True if the user is permitted to modify each existing team.
    canModifyTeam: Boolean!

    # True if the user is permitted to replace each existing team.
    canReplaceTeam: Boolean!

    # True if the user is permitted to delete each existing team.
    canDeleteTeam: Boolean!
}

# Meta data about the creation and update timestamps
type MetaData {
```

```
    # The time stamp when the object was created
    created: String!

    # The time stamp when the object was modified
    lastModified: String!
}

# Collection of team definitions.
type TeamCollection {
    # An array of team definitions.
    items: [Team!]

    # Meta data about a paginated collection.
    metadata: PagedCollectionMetaData
}

# Meta data about a paginated collection.
type PagedCollectionMetaData {
    # The total number of elements that match the search filter.
    # A negative number is interpreted as unlimited total number.
    totalSize: Int!

    # The start index in the total list of elements if only a page of elements
    # is returned. The first element has index 1.
    startIndex: Int!

    # The total number of available pages if only a page of elements is returned.
    pageSize: Int

    # The page index if only a page of elements is returned. The first page has
    # page index 1.
    pageIndex: Int
}


# Enumeration for team membership deepness
enum MembershipEnum {
    # Team definitions contains only the directly contained users, groups and teams.
    shallow

    # Team definition contains the recursively contained users, groups and teams.
    deep
}

# Enumeration for team sorting
enum TeamSortByEnum {
    # Sort by distinguishedName
    distinguishedName
    # Sort by displayName
    displayName
    # Sort by description
    description
    # Sort by unique identifier
    uuid
    # Sort by creation timestamp
    created
    # Sort by modification timestamp
    lastModified
}

# Enumeration for sort order
enum SortOrderEnum {
    # Ascending order
    ascending
    # Descending order
    descending
}
```

User Management Service Teams GraphQL error handling

The GraphQL system does not use the normal REST API error handling.

Most of the time, the two REST endpoints on `teamserver/rest/graphql` will return the HTTP status OK (200), even when there is an error.

A response that includes an error might look like the following:

```
{
    "data": {
        "team": {
```

```
            "uuid": "ddc12371-b71a-4a9a-b819-6cb29713ff65",
            "displayName": "Team1"
        }
    },
    "errors": [
        {
            "locations": [
                {
                    "column": 33,
                    "line": 1
                }
            ],
            "message": "CWLUM1017E: Error in GraphQL query: ..."
        }
    ],
    "extensions": {}
}
```

- The `data` field represents the valid response data. It might or might not be present, depending on whether an error stopped the entire processing, whether there is no error, or an error is recoverable.
- The `errors` field contains the errors that occurred during the GraphQL processing, for example, when the input query was syntactically wrong or if the caller is not authorized to perform the request. When there are no errors, the `errors` field is missing.
- The `extensions` field is usually missing. The `extension` field might be filled with a GraphQL trace log if the server has logging and tracing enabled at the level `FINER`.

There are cases when the two REST endpoints on `teamserver/rest/graphql` will return an HTTP status code that is not the OK code (200). This happens if the error is outside the GraphQL processing, for example, when the JSON input of the POST request cannot be parsed. In this case, the response will not be in the GraphQL response format because the GraphQL processing was not even started.

User Management Service Teams GraphQL example queries

GraphQL queries can be sent through the GET API as well as through the POST API. The following sections provide examples of GraphQL queries on teams and users:

GET API

```
GET https://host:port/teamserver/rest/graphql?
query=someQuery&variables=someVariableMap&operation_name=opName
```

The query parameters are:

- **query** - The query string in GraphQL format.
- **variables** - A variable map in JSON format.
- **operation_name** - A name selecting a named query if the query string contains multiple named queries.

Note that query parameters require URL encoding.

POST API

```
POST https://{host}:{port}/teamserver/rest/graphql
```

The POST API is similar to the GET API, but instead of query parameters, a JSON payload must be specified.

```
{
    "query" : "{ team(uuid:\"72d8e3b6-d4a3-4d0f-a5a7-2e63cea03460\") { displayName }}"
}
```

The POST payload corresponds to the GET query parameters and looks in general form like this:

```
{
    "query" : "....",
    "variables" : {
        "variable1" : "Test",
        "variable2" : 123
    },
    "operationName" : "TestOp"
}
```

Using GET to request the display name of a specific team

The query parameter:

```
{ team(uuid:"72d8e3b6-d4a3-4d0f-a5a7-2e63cea03460") { displayName }}
```

A curl call for `localhost` would look like this:

```
curl -i -k -H "Authorization: Bearer oauthtoken" -H "Accept: application/json" -H "Content-
Type: application/json" -X POST https://host:port/teamserver/rest/graphql -d '{ "query" :
"{ team(uuid:\"72d8e3b6-d4a3-4d0f-a5a7-2e63cea03460\") { displayName }}" }'
Reply
```

Here is an example response:

```
{
    "data": {
        "team": {
            "displayName": "Team3"
        }
    }
}
```

Using POST to request the display name of a specific team

```
{
    "query" : "{ team(uuid:\"72d8e3b6-d4a3-4d0f-a5a7-2e63cea03460\") { displayName }}"
}
```

Remember: Because the value of the `query` field is a string that contains the GraphQL query, all quotes inside the GraphQL query must be escaped.

A curl call for `localhost` would look like this:

```
curl -i -k -H "Authorization: Bearer oauthtoken" -H "Accept: application/json" -H "Content-
Type: application/json" -X POST https://localhost:9443/teamserver/rest/graphql -d '{ "query" :
"{ team(uuid:\"72d8e3b6-d4a3-4d0f-a5a7-2e63cea03460\") { displayName }}" }'
```

Requesting more details for a specific team

Here is a more interesting example for a query string that is formatted, without URL encoding or quote escapes:

```
{
    team(uuid:"66c0fffd-402f-4062-9280-9cbcc6df5da2", membership:deep) {
        uuid
        displayName
        description
        teams {
            uuid
            displayName
        }
    }
}
```

The corresponding example response:

```
{
    "data": {
        "team": {
            "uuid": "66c0fffd-402f-4062-9280-9cbcc6df5da2",
            "displayName": "Team3",
            "description": "This is a third test team",
            "teams": [
                {
                    "uuid": "4eadae15-85e2-434a-924a-81e2f840f485",
                    "displayName": "Team1"
                },
                {
                    "uuid": "9aeb2195-c96a-452a-859f-14f1e740d661",
                    "displayName": "Team2"
                }
            ]
        }
    }
}
```

Using variables in queries

GraphQL queries can contain variables. In this case, you must provide the variable map in the GET request by using the `variables` parameter, or in the POST request in the `variables` payload field. For example:

```
query TestOp($muuid: String!) {
    team(uuid:$muuid) {
        uuid
        displayName
        description
        teams {
            uuid
            displayName
        }
        metadata {
            created
            lastModified
        }
    }
}
```

A suitable variable map could be:

```
{
    "muuid" : "9aeb2195-c96a-452a-859f-14f1e740d661"
}
```

Operation names

Because the query in the previous example is named, you could pass the string `TestOp` as the `operation_name` parameter. Operation names are optional for unique queries and are only needed when the query string contains multiple named queries, for example, when an application always passes a query string with all possible queries and wants to select which query from the possible ones should be executed.

Requesting a list of teams

Here is a query for the list of teams:

```
{
    teams(maxCount:2, startIndex:3, sortOrder:descending) {
        items {
            uuid
            displayName
            teams {
                uuid
                displayName
            }
        }
        metadata {
            startIndex
            totalSize
            pageIndex
            pageSize
        }
    }
}
```

The example response might be:

```
{
    "data": {
        "teams": {
            "items": [
                {
                    "displayName": "Team2",
                    "teams": [
                        {
                            "displayName": "Team1",
                            "uuid": "6c009429-5240-4a29-b13a-36344e6a5504"
                        }
                    ],
                    "uuid": "21172f8a-c242-406d-9b64-e84911d2862e"
                },
                {
                    "displayName": "Team1",
                    "teams": [],
                    "uuid": "6c009429-5240-4a29-b13a-36344e6a5504"
                }
            ],
            "metadata": {
                "pageIndex": 2,
                "pageSize": 7,
                "startIndex": 3,
                "totalSize": 13
            }
        }
    }
}
```

Requesting a list of teams with a filter

Here is a query for a list of teams with a filter. Note the need to escape quotes inside the filter string:

```
{
    teams(filter: "displayName EQ \"Team1\"", membership:shallow) {
        items {
            displayName
        }
    }
}
```

Requesting information about the current user

Here is a query to return the information about the current user:

```
{
  userInfo {
      principalName
```

```
      distinguishedName
      realm
      groups
  }
}
```

The example response might be:

```
{
   "data": {
      "userInfo": {
         "principalName": "John.Doe@example.com",
         "distinguishedName": "cn=John Doe,ou=User,dc=example,dc=com",
         "realm": "PrimaryRealm",
         "groups": [ "cn=Example,ou=Group,dc=example,dc=com" ]
      }
   }
}
```

Checking if the current user is a member of any of the provided teams

Here is a query to check if the current user is a member of any of the provided teams:

```
{
    userMemberOfAnyTeam(teamIds:"47c48aea-54ba-4007-9886-ba9cd81c67c0") {
        memberOfAnyTeam
    }
}
```

The example response might be:

```
{
    "data": {
        "userMemberOfAnyTeam": {
            "memberOfAnyTeam": true
        }
    }
}
```

Checking the permissions of the current user:

Here is a query to check the permissions of the current user:

```
{
  userPermission {
      canCreateTeam
      canDeleteTeam
      canListAllTeams
      canListMyTeams
      canModifyTeam
      canReplaceTeam
      canViewTeamDetails
  }
}
```

The example response might be:

```
{
   "data": {
      "userPermission": {
         "canCreateTeam": false,
         "canDeleteTeam": false,
         "canListAllTeams": false,
         "canListMyTeams": true,
         "canModifyTeam": false,
         "canReplaceTeam": false,
         "canViewTeamDetails": false
      }
   }
}
```

User Management Service Teams GraphQL example mutations

You can use mutations to create, replace, or delete teams. Mutations should always use the POST API.

The following sections provide examples:

- "Creating a new team" on page 47
- "Replacing a team" on page 48
- "Deleting a team" on page 48

Creating a new team

Here is an unformatted example payload that creates a new team:

```
{
    "query" : "mutation { createTeam(team: { displayName: \"ABC\", distinguishedName:
\"cn=abc,ou=bpm,dc=example,dc=com\", description: ...   }) { uuid displayName distinguishedName
description ... } } "
}
```

Remember: Because the value of the `query` field is a string that contains the GraphQL query, all quotes inside the GraphQL query must be escaped.

Here is a query string (excluding the `"query":`) of another example, in clear formatted text:

```
mutation {
    createTeam(
        team: {
            displayName: "ABC",
            distinguishedName: "cn=abc,ou=bpm,dc=example,dc=com",
            description: "Desc",
            users: [
                "U1"
            ],
            teams: [
                "5183d3c6-d25e-4fe7-847e-8b2bd5a0e1d0"
            ]
        }
    ) {
        uuid
        displayName
        distinguishedName
        description
        users
        groups
        teams {
            uuid
        }
    }
}
```

Here is an example response:

```
{
    "data": {
        "createTeam": {
            "uuid": "9b62c908-b356-4de2-b784-527a782649dd",
            "displayName": "ABC",
            "distinguishedName": "cn=abc,ou=bpm,dc=example,dc=com",
            "description": "Desc",
            "groups": [],
            "teams": [
                {
                    "uuid": "5183d3c6-d25e-4fe7-847e-8b2bd5a0e1d0"
                }
            ],
            "users": [
                "U1"
            ]
        }
    }
}
```

Replacing a team

Here is an example query string to replace a team, in clear text:

```
mutation {
    replaceTeam(
        uuid: "9b62c908-b356-4de2-b784-527a782649dd",
        team: {
            displayName: "DEF",
            distinguishedName: "cn=def,ou=bpm,dc=example,dc=com",
            users: [
                "U2", "U3"
            ],
            teams: [
                "936783c6-d24e-41f1-217e-1a3cd4a1e201"
            ]
        }
    ) {
         uuid
        displayName
        distinguishedName
        description
        users
        groups
        teams {
            uuid
            displayName
            description
            teams {
                uuid
                teams {
                    uuid
                    teams {
                        uuid
                        teams {
                            uuid
                            teams {
                                uuid
                                teams {
                                    uuid
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Note: In the previous example, the nesting of the teams illustrates how it is possible to query the result arbitrarily deep in the relationship nesting

Deleting a team

Here is an example query string to delete a team, in clear text:

```
mutation {
    deleteTeam(
        uuid: "9b62c908-b356-4de2-b784-527a782649dd"
    )
}
```

Here is an example response:

```
{
    "data": {
        "deleteTeam": "9b62c908-b356-4de2-b784-527a782649dd"
    }
}
```

User Management Services configuration parameters

Containers:
Configuration parameters for the User Management Services (UMS) on Kubernetes.

UMS data source parameters

Containers:
Configuration parameters for the User Management Service (UMS) data source. These are specified in the section `dc_ums_datasource`.

Most UMS data source configuration parameters are optional, the following parameters are required:

- `datasource_configuration.dc_ums_datasource.dc_ums_oauth_type`

  If the OAuth database type is db2 or `oracle` then the following parameters are also required:

  - `datasource_configuration.dc_ums_datasource.dc_ums_oauth_host`
  - `datasource_configuration.dc_ums_datasource.dc_ums_oauth_port`
  - `datasource_configuration.dc_ums_datasource.dc_ums_oauth_name`
- `datasource_configuration.dc_ums_datasource.dc_ums_teamserver_type`

  If the teams database type is db2 then the following parameters are also required:

  - `datasource_configuration.dc_ums_datasource.dc_ums_teamserver_host`
  - `datasource_configuration.dc_ums_datasource.dc_ums_teamserver_port`
  - `datasource_configuration.dc_ums_datasource.dc_ums_teamserver_name`

Table 1. UMS data source configuration parameters for the `datasource_configuration.dc_ums_datasource` section

Parameter	Description	Default/Example values	Required

`dc_ums_oauth_type`	The type of OAuth database.
	Important: Derby can only be used for test scenarios. It will not work in scenarios with more than one UMS pod. All data is lost when the pod is restarted.
		`derby`
		`db2`
		`oracle`
		`sqlserver`
		`postgresql`
			Yes

`dc_ums_oauth_host`	The host name of the OAuth database. It must be an accessible address, such as an IP, hostname, or Kubernetes service name.		If the OAuth database is db2 or `oracle`.
`dc_ums_oauth_port`	The OAuth database port number.	`50000`	If the OAuth database is db2 or `oracle`.
`dc_ums_oauth_name`	The name of the OAuth database.	`UMSDB`	

Table 1. UMS data source configuration parameters for the `datasource_configuration.dc_ums_datasource` section (continued)

Parameter	Description	Default/Example values	Required
`dc_ums_oauth_schema`	For Oracle databases, the schema name must be the user name of the database.		Can be specified if a schema was created.
`dc_ums_oauth_oracle_service_name`	If you connect to an Oracle Real Application Clusters (RAC) environment using Single Client Access Name (SCAN), configure the database service name in addition to the name of the Oauth database.		If you connect to an Oracle Real Application Clusters (RAC) environment using Single Client Access Name (SCAN).
`dc_ums_oauth_ssl`	Specify `true` if SSL will be used to secure the OAuth database connection.	The default value is `false`	If SSL will be used to secure the OAuth database connection.
`dc_ums_oauth_ssl_secret_name`	The name of the SSL secret.	`ibm-dba-ums-db2-cacert`	If SSL will be used to secure the OAuth database connection.
`dc_ums_oauth_driverfiles`	If you are using a database of type other than Db2 or derby, copy the driver files to the connected persistent volume (PV). Use the property `spec.ums_configuration.existing_claim_name` to point to the PV claim. During the deployment Operator picks up the driver files and configures the connection to the database	`db2jcc4.jar` `db2jcc_license_cu.jar`. **Note:** Db2 driver files are loaded automatically, only provide Oracle driver files if you are using Oracle.	If you are using a database of type other than Db2 or derby.
`dc_ums_oauth_alternate_hosts`	Only specify alternate OAuth database hosts if the OAuth database type is set to db2HADR.		If the OAuth database type is set to db2HADR.
`dc_ums_oauth_alternate_ports`	Only specify alternate OAuth database ports if the OAuth database type is set to db2HADR.		If the OAuth database type is set to db2HADR.

Table 1. UMS data source configuration parameters for the `datasource_configuration.dc_ums_datasource` section (continued)

Parameter	Description	Default/Example values	Required

`dc_ums_teamserver_type`	The type of UMS Teams database.
	Important: Derby can only be used for test scenarios. It will not work in scenarios with more than one UMS pod. All data is lost when the pod is restarted.
		`derby`
		`db2`
		`oracle`
		`sqlserver`
		`postgresql`
			Yes

`dc_ums_teamserver_host`	The host name of the UMS Teams db2 database.		If the UMS Teams database is db2.
`dc_ums_teamserver_port`	The UMS Teams db2 database port.	`50000`	If the UMS Teams database is db2.
`dc_ums_teamserver_name`	The name of the UMS Teams database.	`UMSTEAMSDB`	If the UMS Teams database is db2.
`dc_ums_teamserver_schema`	Can be specified if a schema was created. For Oracle databases, the schema name must be the user name of the database.		Can be specified if a schema was created.
`dc_ums_teamserver_oracle_service_name`	If you connect to an Oracle Real Application Clusters (RAC) environment using Single Client Access Name (SCAN), configure the database service name in addition to the name of the UMS Teams database.		If you connect to an Oracle Real Application Clusters (RAC) environment using Single Client Access Name (SCAN).
`dc_ums_teamserver_ssl`	Specify `true` if SSL is be used to secure the UMS Teams database connection.	The default value is `false.`	If SSL is used to secure the UMS Teams database connection.
`dc_ums_teamserver_ssl_secret_name`	If SSL is used to secure the UMS Teams database connection, specify the name of the SSL secret.	`ibm-dba-ums-db2-cacert`	If SSL is used to secure the UMS Teams database connection.

Table 1. UMS data source configuration parameters for the `datasource_configuration.dc_ums_datasource` section (continued)			
Parameter	**Description**	**Default/Example values**	**Required**
`dc_ums_teamserver_driverfiles`	During the deployment Operator picks up the driver files and configures the connection to the UMS Teams database	`db2jcc4.jar db2jcc_license_cu.jar`.	No
`dc_ums_teamserver_alternate_hosts`	Only specify alternate UMS Teams database hosts if the UMS Teams database type is set to db2HADR.		If the UMS Teams database type is set to db2HADR
`dc_ums_teamserver_alternate_ports`	Only specify alternate UMS Teams database ports if the UMS Teams database type is set to db2HADR.		If the UMS Teams database type is set to db2HADR

UMS parameters

Containers:
Configuration parameters for User Management Services (UMS). These are specified in the section `ums_configuration`.

Most configuration parameters are optional, only two parameters are required:

- `ums_configuration.images.ums.repository`: The repository from where the UMS image is pulled.
- `ums_configuration.images.ums.tag`: The UMS image tag.

Table 2. UMS configuration parameters for the `ums_configuration` section			
Parameter	**Description**	**Default/ Example values**	**Required**
`existing_claim_name`	The name of the Persistent Volume Claim for JDBC drivers and custom binaries.		No
`existing_claim_name_logstore`	The existing PVC for UMS logs, FFDC and access logs.		No
`use_custom_jdbc_drivers`	If the JDBC driver offered over `shared_configuration.sc_drivers_url` or the default JDBC drivers from ICP4BA should not be used, set this to true, so that the JDBC driver is read from the PV set as `existing_claim_name`. For more information on the `shared_configuration.sc_drivers_URL`, see Preparing customized versions of JDBC drivers	The default value is `false`.	No

Table 2. UMS configuration parameters for the ums_configuration section (continued)			
Parameter	**Description**	**Default/ Example values**	**Required**
dedicated_pods	Specifies whether the UMS capabilities each run in dedicated pods. To run the UMS capabilities sso, scim, and teamserver in separate pods, use the value true. To run all UMS capabilities in one pod, use the value false.	In an enterprise deployment the default value is true. In a demo deployment, the default value is false.	No
pod_disruption_budget.min_available • If you are not using dedicated pods locate the parameter in the ums_configuration section. • If you are using dedicated pods you must specify the parameter for each UMS capability's pod separately within the ums_configuration, for example: – ums_configuration.sso.pod_disruption_budget.min_available – ums_configuration.scim.pod_disruption_budget.min_available – ums_configuration.teamserver.pod_disruption_budget.min_available	Specifies the minimum number of pods that are available for the pod disruption budget.	The default value is 1	
replica_count	The number of pod replicas running by default.	The default value is 2.	No

Table 2. UMS configuration parameters for the `ums_configuration` section (continued)

Parameter	Description	Default/ Example values	Required

`backwards_compatibility_routes`	From 21.0.2, UMS uses the following pattern for host names:
	`ums-<suffix>`
	`ums-sso-<suffix>`
	`ums-teams-<suffix>`
	`ums-profiles-<suffix>`
	If you are upgrading and want routes to be created for backwards compatibility using the previously defined host names and certificates, set this to `true`. The old hostname pattern was:
	`ums.<suffix>`
	`ums-sso.<suffix>`
	`ums-teams.<suffix>`
	`ums-profiles.<suffix>`
		The default value is `false`.	No

`service_type`	The type to expose the service as, for example, `Route` for external access or `NodePort` for internal tests.	The default value is `Route`.	No
`iam.delegation_enabled`	Specifies whether authentication is delegated to the Common Services Identity Access Management (IAM).	On OCP and ROKS, the default value is `true`. Otherwise, the default is `false`.	No
`iam.namespace`	The namespace where IAM is installed.	The default value is `ibm-common-services`.	No

Table 2. UMS configuration parameters for the `ums_configuration` section (continued)			
Parameter	**Description**	**Default/ Example values**	**Required**
`hostname`	The name of the host where the User Management Service will run.	If not specified, `hostname` is generated from `shared_configuration.sc_deployment_hostname_suffix`.	No
`port`	The port that will be used to access the User Management Service, for example, 443 when using SSL.	The default value is 443.	No
`images.ums.repository`	The repository from where the UMS image is pulled.	• If the repository `sc_image_repository` is available, it is used as the default. • Otherwise, `cp.icr.io/cp/cp4a/ums/ums` is used as the default value.	Yes

Table 2. UMS configuration parameters for the ums_configuration section (continued)

Parameter	Description	Default/ Example values	Required
images.ums.tag	The UMS image tag.	• If the repository sc_image_repository is available, it is used as the default. • Otherwise, if the current repository in not cp.icr.io, the current version is used as the default, for example 21.0.2. • Otherwise, if the repository cp.icr.io is used, the image digest is used as the default value.	No
admin_secret_name	The name of the secret that was generated for the UMS secret and database secret.	If not specified, the secret ibm-dba-ums-secret must be created.	No

Table 2. UMS configuration parameters for the ums_configuration section (continued)			
Parameter	**Description**	**Default/ Example values**	**Required**
external_tls_secret_name	Enables SSL with an existing certificate for the ums-route route. If this is set this is used rather than using shared_configuration.external_tls_certificate_secret.	If this is not set, the default is to use shared_configuration.external_tls_certificate_secret, but if that is also not set, then no external TLS certificate is used.	No
external_tls_ca_secret_name	Certificate Authority (CA) used to sign the external TLS secret. If you don't want to provide a CA to sign the external TLS certificate, leave this empty, then .	The default is not to use a CA to sign the external TLS certificate.	No
external_tls_teams_secret_name	A secret that specifies the TLS certificate that represents the hostname or a common hostname suffix of the ums-teams-route route that your clients will use to connect to UMS. If this is set this is used rather than using shared_configuration.external_tls_certificate_secret.	If this is not set, the default is to use shared_configuration.external_tls_certificate_secret, but if that is also not set, then no external TLS certificate is used.	No

Table 2. UMS configuration parameters for the `ums_configuration` section (continued)

Parameter	Description	Default/ Example values	Required
`external_tls_scim_secret_name`	A secret that specifies the TLS certificate that represents the hostname or a common hostname suffix of the `ums-scim-route` route that your clients will use to connect to UMS. If this is set this is used rather than using `shared_configuration.external_tls_certificate_secret`.	If this is not set, the default is to use `shared_configuration.external_tls_certificate_secret`, but if that is also not set, then no external TLS certificate is used.	No
`external_tls_sso_secret_name`	A secret that specifies the TLS certificate that represents the hostname or a common hostname suffix of the `ums-sso-route` route that your clients will use to connect to UMS. If this is set this is used rather than using `shared_configuration.external_tls_certificate_secret`.	If this is not set, the default is to use `shared_configuration.external_tls_certificate_secret`, but if that is also not set, then no external TLS certificate is used.	No
`oauth.client_manager_group`	The full DN of an LDAP group that is authorized to manage OIDC clients, in addition to the primary admin from the admin secret.		No
`oauth.token_manager_group`	The full DN of an LDAP group that is authorized to manage tokens, in addition to the primary admin from the admin secret.		No

Table 2. UMS configuration parameters for the `ums_configuration` section (continued)			
Parameter	**Description**	**Default/ Example values**	**Required**
`oauth.access_token_lifetime`	The lifetime of OAuth access_tokens.	The default value is `7200s`.	No
`oauth.app_token_lifetime`	The lifetime of app-tokens.	The default value is `366d`.	No
`oauth.app_password_lifetime`	The lifetime of app-passwords.	The default value is `366d`.	No
`oauth.app_token_or_password_limit`	The maximum number of app-tokens or app-passwords per client.	The default value is `100`.	No
`oauth.client_secret_encoding`	The encoding / encryption when storing client secrets in the OAuth database.	The default value is `xor` for compatibility. Recommended value is `PBKDF2WithHmacSHA512`.	No
`custom_secret_name`	The name of the existing secret for sensitive Liberty configuration, specified in XML format.		No

Table 2. UMS configuration parameters for the `ums_configuration` section (continued)			
Parameter	**Description**	**Default/ Example values**	**Required**
For UMS `resources`, `autoscaling`, `custom_xml`, and `logs.trace_specification`: • If you are not using dedicated pods locate them in the `ums_configuration` section. • If you are using dedicated pods you must specify each UMS capability's pod separately within the `ums_configuration`, for example: – `ums_configuration.sso` – `ums_configuration.scim` – `ums_configuration.teamserver`	Kubernetes controls resources such as CPU and memory using requests and limits mechanisms. Requests are what the container is guaranteed to get. Limits make sure a container never goes above a certain value. A limit value cannot be lower than the corresponding request value. If you are not using dedicated pods for UMS capabilities (`ums_configuration.dedicated_pods = false`) you can specify `resources`, `autoscaling`, `custom_xml`, and `logs.trace_specification` for `ums_configuration`. If you are using dedicated pods for UMS capabilities (`ums_configuration.dedicated_pods = true`), you can specify `resources`, `autoscaling`, `custom_xml`, and `logs.trace_specification` for each UMS capability: `sso`, `scim`, and `teamserver`.	The default values are listed in the following rows.	No
• If you are not using dedicated pods: `resources.limits.cpu` • If you are using dedicated pods: – `sso.resources.limits.cpu` – `scim.resources.limits.cpu` – `teamserver.resources.limits.cpu`	The maximum CPU limit.	The default value is `500m`.	No
• If you are not using dedicated pods: `resources.limits.memory` • If you are using dedicated pods: – `sso.resources.limits.memory` – `scim.resources.limits.memory` – `teamserver.resources.limits.memory`	The maximum memory limit.	The default value is `512Mi`.	No

Table 2. UMS configuration parameters for the `ums_configuration` section (continued)			
Parameter	**Description**	**Default/ Example values**	**Required**
• If you are not using dedicated pods: `resources.limits.ephemeral_storage` • If you are using dedicated pods: – `sso.resources.limits.ephemeral_storage` – `scim.resources.limits.ephemeral_storage` – `teamserver.resources.ephemeral_storage`	The maximum ephemeral storage limit.	The default value is `500Mi`.	No
• If you are not using dedicated pods: `resources.requests.cpu` • If you are using dedicated pods: – `sso.resources.requests.cpu` – `scim.resources.requests.cpu` – `teamserver.resources.requests.cpu`	The minimum CPU.	The default value is `200m`.	No
• If you are not using dedicated pods: `resources.requests.memory` • If you are using dedicated pods: – `sso.resources.requests.memory` – `scim.resources.requests.memory` – `teamserver.resources.requests.memory`	The minimum memory.	The default value is `256Mi`.	No

Table 2. UMS configuration parameters for the `ums_configuration` section (continued)			
Parameter	**Description**	**Default/ Example values**	**Required**
• If you are not using dedicated pods: `resources.requests.ephemeral_storage` • If you are using dedicated pods: – `sso.resources.requests.ephemeral_storage` – `scim.resources.requests.ephemeral_storage` – `teamserver.requests.ephemeral_storage`	The minimum ephemeral storage limit.	The default value is `500Mi`.	No
• If you are not using dedicated pods: `autoscaling.enabled` • If you are using dedicated pods: – `sso.autoscaling.enabled` – `scim.autoscaling.enabled` – `teamserver.autoscaling.enabled`	If `true`, pods are automatically scaled within the specified range.	The default value is `true`.	No
• If you are not using dedicated pods: `autoscaling.min_replicas` • If you are using dedicated pods: – `sso.autoscaling.min_replicas` – `scim.autoscaling.min_replicas` – `teamserver.autoscaling.min_replicas`	The minimum number of replicas for autoscaling.	The default value is 2.	No

Table 2. UMS configuration parameters for the `ums_configuration` section (continued)			
Parameter	**Description**	**Default/ Example values**	**Required**
• If you are not using dedicated pods: `autoscaling.max_replicas` • If you are using dedicated pods: – `sso.autoscaling.max_replicas` – `scim.autoscaling.max_replicas` – `teamserver.autoscaling.max_replicas`	The maximum number of replicas for autoscaling.	The default value is 5.	No
• If you are not using dedicated pods: `autoscaling.target_average_utilization` • If you are using dedicated pods: – `sso.autoscaling.target_average_utilization` – `scim.autoscaling.target_average_utilization` – `teamserver.autoscaling.target_average_utilization`	The average CPU utilization for autoscaling. When the average utilization exceeds this target, then new pods are created.	The default value is 98.	No
`use_custom_binaries`	Specify if any custom binaries are used.	The default value is `false`.	No
`custom_secret_name`	The name of the existing secret for sensitive Liberty configuration, specified in XML format.		No
• If you are not using dedicated pods: `custom_xml` • If you are using dedicated pods: – `sso.custom_xml` – `scim.custom_xml` – `teamserver.custom_xml`	Custom configuration settings (optional, multi-line value). For LDAP configuration use `spec.ldap_configuration` parameters.		No

Table 2. UMS configuration parameters for the `ums_configuration` section (continued)			
Parameter	**Description**	**Default/ Example values**	**Required**
`logs.console_format`	The format of the UMS logs console.	The default value is `json`.	No
`logs.console_log_level`	The log level for the UMS logs console.	The default value is `INFO`.	No
`logs.console_source`	UMS logs console source.	The default value is `message,trace,accessLog,ffdc,audit`.	No
`logs.trace_format`	The format of the UMS logs trace.	The default value is `ENHANCED`.	No
`logs.max_files`	The maximum number of log files to use.	The default value is 2.	No
`logs.max_file_size`	The maximum size of the log files in MB.	The default value is `20`.	No
• If you are not using dedicated pods: `logs.trace_specification` • If you are using dedicated pods: – `sso.logs.trace_specification` – `scim.logs.trace_specification` – `teamserver.logs.trace_specification`	The UMS logs trace specification.	The default value is `*=info`.	No

Table 2. UMS configuration parameters for the `ums_configuration` section (continued)			
Parameter	**Description**	**Default/ Example values**	**Required**
`teamserver.admingroup`	The full DN of an LDAP group that is authorized to administer UMS Teams.	• For the IBM Automation Document Processing demo pattern, the default is `cn=ADPEnvironmentOwners,dc=example,dc=org`. • For all other demo patterns, the default is `cn=TeamsAdmins,dc=example,dc=org`. • For all enterprise (non-demo) patterns, there is no default.	No

UMS advanced parameters

Containers:
Configuration parameters for User Management Services (UMS).

Using these optional advanced UMS configuration parameters is described in the following sections:

- “Updating parameters if you do not have dedicated pods for UMS services enabled” on page 66
- “Updating parameters if you have dedicated pods for UMS services enabled” on page 66
- “UMS database JDBC connect pool sizes” on page 66

- “UMS Health host/port, logging, and certificate checking” on page 67
- “UMS Teams certificate checking” on page 68

Updating parameters if you do not have dedicated pods for UMS services enabled

Because all UMS services run together in shared pods, you must use the `ums_configuration.custom_xml` property in the Custom Resource file to overwrite the default values of any of the advanced parameters. For example:

```
ums_configuration:
  custom_xml: |
    <server>
      <variable name="Parameter_Name" value="Value"/>
    </server>
```

Updating parameters if you have dedicated pods for UMS services enabled

Because each UMS service runs in its own pod, to overwrite the default values of any of these advanced parameters you must specify the `custom_xml` property for the appropriate UMS service pods separately in the Custom Resource file. For example:

```
ums_configuration:

  sso:
    custom_xml: |
      <server>
        <variable name="Parameter_Name" value="Value" />
      </server>

  scim:
    custom_xml: |
      <server>
        <variable name="Parameter_Name" value="Value" />
      </server>

  teamserver:
    custom_xml: |
      <server>
        <variable name="Parameter_Name" value="Value" />
      </server>
```

Important: Not all parameters apply to all pods. If you have dedicated pods, refer to the "Valid for pods" columns in the following tables to see which pods each parameter can be specified for.

UMS database JDBC connect pool sizes

You can configure the following database parameters:

Table 3. Optional UMS database advanced configuration parameters

Parameter Name	Description	Valid for pods	Default value
`ums.oauthdb.maxPoolSize`	The maximum size of the pool of UMS JDBC connections can be tuned to better utilize the CPU of the UMS SSO pod.	`sso`	`100`
`ums.oauthdb.minPoolSize`	The minimum size of the pool of UMS JDBC connections can be tuned to better utilize the CPU of the UMS SSO server pod.	`sso`	`2`
`ums.tsdb.maxPoolSize`	The maximum size of the pool of UMS JDBC connections can be tuned to better utilize the CPU of the UMS Teams server pod.	`teamserver`	`100`

Table 3. Optional UMS database advanced configuration parameters (continued)			
Parameter Name	**Description**	**Valid for pods**	**Default value**
`ums.tsdb.minPoolSize`	The minimum size of the pool of UMS JDBC connections can be tuned to better utilize the CPU of the UMS Teams server pod.	`teamserver`	2

UMS Health host/port, logging, and certificate checking

To configure UMS Health, you can use the following advanced parameters for all pods:

Table 4. Optional UMS Health advanced configuration parameters			
Parameter Name	**Description**	**Valid for pods**	**Default value**
`ums.health.useLocalHostAndPort`	Specifies whether local host and local port are used instead of server host and server port if the health modules are automatically detected or the URLs of modules do not specify host and port explicitly. This setting can be needed if a reverse proxy or load balancer is used. By default, server host and server port are used in this case, that is, the load balancer or reverse poxy address, or in general, the same host and port the original request was sent to. This setting only has an effect if the fallback host and port is not specified.	All pods	`false`
`ums.health.fallbackHostAndPort`	The fallback host and port are used when the health modules are automatically detected or the URLs of modules do not specify host and port explicitly. If the fallback host and port is not specified, either the server host and server port or the local host and local port are used in the case, depending on the `useLocalHostAndPort` setting.	All pods	`https://127.0.0.1:9443`
`ums.health.logHealthFailuresOnStartup`	Specifies whether on server startup, all failing results of health calls are logged as warnings. This logging stops when the first health call returns success. This feature can help to analyze situations when the server fails to start.	All pods	`true`
`ums.health.disableCNCheck`	Configures whether the common name verification of server SSL certificates is disabled. This allows UMS to connect to an OpenID Connect provider with an SSL certificate that does not match its host name.	All pods	`false`
`ums.health.disableCertificateCheck`	Configures whether the certificate verification is disabled. This allows connection to an OpenID Connect provider whose certificate is not in the truststore.	All pods	`false`

UMS Teams certificate checking

You can configure the following advanced parameters:

Table 5. Optional UMS Teams advanced configuration parameters

Parameter Name	Description	Valid for pods	Default Value
`ums.teams.registration.disableCNCheck`	Configures whether the common name verification of server SSL certificates is disabled. This allows UMS Teams to connect to an OpenID Connect provider with an SSL certificate that does not match its host name.	`sso` and `teamserver`	`false`
`ums.teams.registration.disableCertificateCheck`	Configures whether the certificate verification is disabled. This allows UMS Teams to connect to an OpenID Connect provider whose certificate is not in the truststore.	`sso` and `teamserver`	`false`
`ums.teams.scim.disableCNCheck`	Configures whether the common name verification of server SSL certificates is disabled. This allows UMS Teams to connect to a SCIM server with an SSL certificate that does not match its host name.	`teamserver`	`false`
`ums.teams.scim.disableCertificateCheck`	Configures whether the certificate verification is disabled. This allows UMS Teams to connect to a SCIM server whose certificate is not in the truststore.	`teamserver`	`false`