.DataPlatform v22.1

This manual describes how to install and set up eccenca DataPlatform.It is intended for system administrators, who are responsible for installing, configuring, maintaining and supporting the deployment of DataPlatform.

To use this manual, system administrators should have knowledge about Linux (Ubuntu) and the installation environment on which DataPlatform is deployed.

DataPlatform provides a basic default configuration. The default configuration can be changed by defining properties for the intended use. The properties defined overwrite the respective properties from the default configuration.

In this manual, YAML syntax is used to define those properties as the default configuration file format. Key/value pairs stored in a property file (.properties) are also a valid option.

The following example shows a YAML-based configuration:

```
foo:
  bar:
  - id: id1
    label: label1
  - id: id2
    label: label2
```

In the following example, the same configuration is represented as key/value pairs in a properties file:

```
foo.bar[0].id = "id1"
foo.bar[0].label = "label1"
foo.bar[1].id = "id2"
foo.bar[1].label = "label2"
```

Note: You cannot combine both formats, so either provide a YAML (.yml) or property file.

Note: YAML maps `OFF` to `FALSE` so make sure to add quotes if you want to disable logging:

```
logging:
  level:
    root: "OFF"
```

License

By default, DataPlatform is subject to the eccenca free Personal, Evaluation and Development License Agreement (PEDAL), a license intended for non-commercial usage. When your delivery includes a dedicated license file, you have to configure DataPlatform to enable your license. To change the default configuration, you have several options that are described in License configuration.

License configuration

- license
  - key
  - file

Property `license`

Use this root property and its sub-properties for the license configuration. If this root property is not configured, the default license included (PEDAL) is used.

Property `license.key`

Default	none
Required	no

Conflicts with	`license.file`
Valid values	PGP Key (Message)

Use this property to specify the license key as a YAML multiline string value of the `license.key` property:

```
license:
  key: |
    -----BEGIN PGP MESSAGE-----
    ...
    ...
    ...
    -----END PGP MESSAGE-----
```

Property `license.file`

Default	none
Required	no
Conflicts with	`license.key`
Valid values	location of the license file

Use this property to specify the location of the license file:

```
license:
  file: PATH_TO_LICENSE_FILE
```

License evaluation order

In case a dedicated license file is used, different configuration options can overwrite each other. The license is read in the following sequence:

1. `license.key` property
2. `license.file` property
3. `license.asc` file in the same folder, where the application is started from (in Standalone Mode)
4. Fallback to eccenca free Personal, Evaluation and Development License Agreement (PEDAL)

SPARQL endpoints

SPARQL endpoints declare how DataPlatform connects to a SPARQL-capable store or service. This includes stores that are capable of reading and writing RDF such as Virtuoso as well as read-only services like remote SPARQL HTTP endpoints (e.g. DBpedia).

With the default configuration, DataPlatform uses an in-memory database. This means, that no persistent storage is available, unless a store supporting data persistence is configured.

This table shows the supported SPARQL endpoint types together with the supported features:

Feature	GraphDB	AWS Neptune	Stardog	Virtuoso	HTTP	In-memory
Persistent storage	Yes	Yes	Yes	Yes	Yes	-
Read access	Yes	Yes	Yes	Yes	Yes	Yes
Write access	Yes	Yes	Yes	Yes	Yes	Yes
Rewrite authorization	Yes	Yes	Yes	Yes	Yes	Yes
Provisioned authorization	-	-	Yes	Yes	-	-

Note: The support to configure and connect to multiple SPARQL endpoints with a single DataPlatform instance will be removed in a future release.

Authorization strategies

DataPlatform supports the following authorization strategies:

- `NONE`: Do not use authorization for this endpoint (default).
- `REWRITE_FROM`: SPARQL FROM-clause rewriting used as authorization strategy for this endpoint.

- `PROVISIONED`: Authorization provisioned by the triple store. For productive setups, set a `userPasswordSalt`. Treat this parameter as secretly as the triple store administration password.

> Be aware that when authorization for an endpoint is NONE, the user gets administrator permissions which means for example the user can grant permissions and is allowed to do any action.
> Therefore the NONE strategy should only be used for demo or evaluation environments!

GraphDB

Use the following set of properties to connect to a GraphDB server using HTTP.

The GraphDB setup section provides more information on prerequisites for Stardog endpoint configuration.

GraphDB configuration

- sparqlEndpoints:
  - graphdb:
    - id
    - authorization
    - host
    - port
    - database
    - username
    - password
    - userPasswordSalt
    - sslEnabled
    - updateTimeoutInMilliseconds
    - connectionPool
      - maxConnectionsPerUser
      - maxIdleConnectionsPerUser
      - maxIdleMilliseconds
      - maxWaitMilliseconds

Property `sparqlEndpoints.graphdb[0].id`

Default	none
Required	yes
Conflicts with	other IDs, must be unique
Valid values	string

Use this property to specify the ID of the in-memory endpoint. This ID can be user-defined and must be unique.

Property `sparqlEndpoints.graphdb[0].authorization`

Default	`NONE`
Required	no
Conflicts with	none
Valid values	`NONE, REWRITE_FROM`

Use this property to specify the authorization strategy as explained in section Authorization strategies.

When setting the strategy to `PROVISIONED`, each user context will get its own dedicated GraphDB user; when setting the strategy to any other value, all user requests will use the same GraphDB user. This has an impact on connection pooling configuration (maxConnectionsPerUser, maxIdleConnectionsPerUser). If `PROVISIONED` is set, these values should fit the expected number of parallel requests per unique user. Otherwise they should fit the expected number of overall parallel requests.

Property `sparqlEndpoints.graphdb[0].host`

Default	none
Required	yes
Conflicts with	none
Valid values	string

Use this property to configure the hostname of the GraphDB server.

Property `sparqlEndpoints.graphdb[0].port`

Default	none
Required	yes
Conflicts with	none
Valid values	integer

Use this property to configure the port of the GraphDB server.

Property `sparqlEndpoints.graphdb[0].database`

Default	none
Required	yes
Conflicts with	none
Valid values	string

Use this property to configure the database to connect to.

Property `sparqlEndpoints.graphdb[0].username`

Default	none
Required	yes
Conflicts with	none
Valid values	string

Use this property to configure the username with which the connection to the server is established.

Property `sparqlEndpoints.graphdb[0].password`

Default	none
Required	yes
Conflicts with	none
Valid values	string

Use this property to configure the password with which the connection to the server is established.

Property `sparqlEndpoints.graphdb[0].userPasswordSalt`

Default	none
Required	no
Conflicts with	none
Valid values	string

Use this property to configure the salt value used for internal user password generation. This property is only relevant if the `authorization` is set to `PROVISIONED`.

If not provided, a default internal value is used, which is not recommended in a production setup. Configure and treat this parameter with the same secrecy as `password`.

Property `sparqlEndpoints.graphdb[0].sslEnabled`

Default	`false`
Required	no
Conflicts with	none
Valid values	Boolean

Use this property to enable encrypted data transfer from and to the GraphDB server. If enabled, the configured `port` must be the GraphDB SSL port and the server must be properly configured for this purpose. Refer to section GraphDB setup.

Property `sparqlEndpoints.graphdb[0].updateTimeoutInMilliseconds`

Default	`0`
Required	no
Conflicts with	none
Valid values	long

Use this property to set the upper bound for update operation execution time. If an update request consists of multiple update operations, the timeout applies to each update operation individually. To support this, the GraphDB server must be properly configured. Refer to section GraphDB setup.

Property `sparqlEndpoints.graphdb[0].connectionPool.maxConnectionsPerUser`

Default	`1000`
Required	no
Conflicts with	none
Valid values	integer

Use this property to configure the maximum amount of parallel open connections for a user provided by the pool. A negative value will remove the connection pool maximum size limitation. Notice that in `REWRITE_FROM` and `NONE` authorization strategies there is only one user for connection pooling purposes. See the authorization property description for more information.

Property `sparqlEndpoints.graphdb[0].connectionPool.maxIdleConnectionsPerUser`

Default	`1`
Required	no
Conflicts with	cannot be larger than `maxConnectionsPerUser`
Valid values	integer

Use this property to configure the maximum amount of idle connections (connections returned to the pool and kept open for faster connection providing) for a user. Notice that in `REWRITE_FROM` and `NONE` authorization strategies there is only one user for connection pooling purposes. See the authorization property description for more information.

Property `sparqlEndpoints.graphdb[0].connectionPool.maxIdleMilliseconds`

Default	`60000`
Required	no
Conflicts with	none
Valid values	integer

Use this property to configure the maximum amount of time that a connection is allowed to idle until it is closed.

Property `sparqlEndpoints.graphdb[0].connectionPool.maxWaitMilliseconds`

Default	`10000`
Required	no
Conflicts with	none
Valid values	integer

Use this property to configure the maximum amount of time that a request is blocked waiting for a connection to be provided by the pool when its maximum capacity is reached. On timeout, the user gets an error response.

Configuration example

```
sparqlEndpoints:
  graphdb:
    - id: "default"
      authorization: REWRITE_FROM
      host: "http://store:7200"
      repository: "cmem"
      username: "admin"
      password: "changeme"
```

AWS Neptune

Use the following set of properties to connect to a AWS Neptune triple store.

AWS Neptune configuration

- sparqlEndpoints:
    - neptune:
        - id
        - authorization
        - host
        - port
        - aws:
            - region
            - authEnabled
        - s3:
            - bucketNameOrAPAlias
            - iamRoleArn
            - bulkLoadThresholdInMb
            - bulkLoadParallelism

Property `sparqlEndpoints.neptune[0].id`

Default	none
Required	yes
Conflicts with	other IDs, must be unique
Valid values	string

Use this property to specify the ID of the in-memory endpoint. This ID can be user-defined and must be unique.

Property `sparqlEndpoints.neptune[0].authorization`

Default	`NONE`
Required	no
Conflicts with	none
Valid values	`NONE, REWRITE_FROM`

Use this property to specify the authorization strategy as explained in section Authorization strategies.

Property `sparqlEndpoints.neptune[0].host`

Default	none
Required	yes
Conflicts with	none
Valid values	string

Use this property to configure the hostname of the server.

Property `sparqlEndpoints.neptune[0].port`

Default	none
Required	yes

Conflicts with	none
Valid values	integer

Use this property to configure the port of the server.

Property `sparqlEndpoints.neptune[0].aws.region`

Default	none
Required	yes
Conflicts with	none
Valid values	string

Use this property to configure the AWS Region where Neptune cluster is located i.e. "eu-central-1".

Property `sparqlEndpoints.neptune[0].aws.authEnabled`

Default	none
Required	yes
Conflicts with	none
Valid values	Boolean

Boolean value on whether IAM DB authentication is enabled on Neptune cluster. DataPlatform uses AWS default mechanism on supplying AWS IAM credentials (https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/credentials.html#credentials-chain).

Property `sparqlEndpoints.neptune[0].s3.bucketNameOrAPAlias`

Default	none
Required	no
Conflicts with	none
Valid values	string

Use this property to configure the name of the S3 bucket or access point alias for bucket (bucket needs to be in configured region).

Property `sparqlEndpoints.neptune[0].s3.iamRoleArn`

Default	`none`
Required	no
Conflicts with	none
Valid values	string

Use this property to configure the role which has read access to the bucket (neptune cluster must assume this role to read from S3 for bulk loading).

Property `sparqlEndpoints.neptune[0].s3.bulkLoadThresholdInMb`

Default	`150`
Required	no
Conflicts with	none
Valid values	integer

Use this property to set the limit on how much data of a GSP request (i.e. graph) is uploaded to neptune via HTTP (default 150). If limit is exceeded S3 bulk loading will be applied, the maximum working value for limit is 150 MB as this is Neptune limit for HTTP uploads (s. https://docs.aws.amazon.com/neptune/latest/userguide/limits.html).

Property `sparqlEndpoints.neptune[0].s3.bulkLoadParallelism`

Default	`LOW`

Required	no
Conflicts with	none
Valid values	`LOW, MEDIUM, HIGH, OVERSUBSCRIBE`

Use this property to configure the level of parallelism for bulk loading to Neptune, affects CPU usage. One of `LOW, MEDIUM, HIGH, OVERSUBSCRIBE`.

Configuration example

```
sparqlEndpoints:
  neptune:
  - id: neptune
    authorization: REWRITE_FROM
    host: <your-neptune-sparql-endpoint-uri>
    port: 8182
    aws:
      region: eu-central-1
      authEnabled: true
    s3:
      bucketNameOrAPAlias: <your-s3-bucket-name>
      iamRoleArn: <your-s3-iam-role>
      bulkLoadThresholdInMb: 150
      bulkLoadParallelism: HIGH
```

Stardog

Use the following set of properties to connect to a Stardog server using HTTP.

The Stardog setup section provides more information on prerequisites for Stardog endpoint configuration.

Stardog configuration

- sparqlEndpoints:
  - stardog:
    - id
    - authorization
    - host
    - port
    - database
    - username
    - password
    - userPasswordSalt
    - sslEnabled
    - updateTimeoutInMilliseconds
    - connectionPool
      - maxConnectionsPerUser
      - maxIdleConnectionsPerUser
      - maxIdleMilliseconds
      - maxWaitMilliseconds

Property `sparqlEndpoints.stardog[0].id`

Default	none
Required	yes
Conflicts with	other IDs, must be unique
Valid values	string

Use this property to specify the ID of the in-memory endpoint. This ID can be user-defined and must be unique.

Property `sparqlEndpoints.stardog[0].authorization`

Default	`NONE`
Required	no
Conflicts with	none
Valid values	`NONE, PROVISIONED, REWRITE_FROM`

Use this property to specify the authorization strategy as explained in section Authorization strategies.

When setting the strategy to `PROVISIONED`, each user context will get its own dedicated Stardog user; when setting the strategy to any other value, all user requests will use the same Stardog user. This has an impact on connection pooling configuration (maxConnectionsPerUser, maxIdleConnectionsPerUser). If `PROVISIONED` is set, these values should fit the expected number of parallel requests per unique user. Otherwise they should fit the expected number of overall parallel requests.

Property `sparqlEndpoints.stardog[0].host`

Default	none
Required	yes
Conflicts with	none
Valid values	string

Use this property to configure the hostname of the Stardog server.

Property `sparqlEndpoints.stardog[0].port`

Default	none
Required	yes
Conflicts with	none
Valid values	integer

Use this property to configure the port of the Stardog server.

Property `sparqlEndpoints.stardog[0].database`

Default	none
Required	yes
Conflicts with	none
Valid values	string

Use this property to configure the database to connect to.

Property `sparqlEndpoints.stardog[0].username`

Default	none
Required	yes
Conflicts with	none
Valid values	string

Use this property to configure the username with which the connection to the server is established.

Property `sparqlEndpoints.stardog[0].password`

Default	none
Required	yes
Conflicts with	none
Valid values	string

Use this property to configure the password with which the connection to the server is established.

Property `sparqlEndpoints.stardog[0].userPasswordSalt`

Default	none
Required	no
Conflicts with	none

Valid values	string

Use this property to configure the salt value used for internal user password generation. This property is only relevant if the `authorization` is set to `PROVISIONED`.

If not provided, a default internal value is used, which is not recommended in a production setup. Configure and treat this parameter with the same secrecy as `password`.

Property `sparqlEndpoints.stardog[0].sslEnabled`

Default	`false`
Required	no
Conflicts with	none
Valid values	Boolean

Use this property to enable encrypted data transfer from and to the Stardog server. If enabled, the configured `port` must be the Stardog SSL port and the server must be properly configured for this purpose. Refer to section Stardog setup.

Property `sparqlEndpoints.stardog[0].updateTimeoutInMilliseconds`

Default	`0`
Required	no
Conflicts with	none
Valid values	long

Use this property to set the upper bound for update operation execution time. If an update request consists of multiple update operations, the timeout applies to each update operation individually. To support this, the Stardog server must be properly configured. Refer to section Stardog setup.

Property `sparqlEndpoints.stardog[0].connectionPool.maxConnectionsPerUser`

Default	`1000`
Required	no
Conflicts with	none
Valid values	integer

Use this property to configure the maximum amount of parallel open connections for a user provided by the pool. A negative value will remove the connection pool maximum size limitation. Notice that in `REWRITE_FROM` and `NONE` authorization strategies there is only one user for connection pooling purposes. See the authorization property description for more information.

Property `sparqlEndpoints.stardog[0].connectionPool.maxIdleConnectionsPerUser`

Default	`1`
Required	no
Conflicts with	cannot be larger than `maxConnectionsPerUser`
Valid values	integer

Use this property to configure the maximum amount of idle connections (connections returned to the pool and kept open for faster connection providing) for a user. Notice that in `REWRITE_FROM` and `NONE` authorization strategies there is only one user for connection pooling purposes. See the authorization property description for more information.

Property `sparqlEndpoints.stardog[0].connectionPool.maxIdleMilliseconds`

Default	`60000`
Required	no
Conflicts with	none
Valid values	integer

Use this property to configure the maximum amount of time that a connection is allowed to idle until it is closed.

Property `sparqlEndpoints.stardog[0].connectionPool.maxWaitMilliseconds`

Default	`10000`
Required	no
Conflicts with	none
Valid values	integer

Use this property to configure the maximum amount of time that a request is blocked waiting for a connection to be provided by the pool when its maximum capacity is reached. On timeout, the user gets an error response.

Configuration example

```
sparqlEndpoints:
  stardog:
  - id: stardog
    authorization: PROVISIONED
    host: localhost
    port: 5820
    database: database
    username: username
    password: password
    userPasswordSalt: $0M3Th1ng
```

Virtuoso

You can connect DataPlatform to an OpenLink Virtuoso Universal Server using a JDBC connection.

The section Virtuoso Setup provides further information on prerequisites for Virtuoso endpoint configuration.

Virtuoso configuration

- sparqlEndpoints
  - virtuoso
    - id
    - authorization
    - host
    - port
    - username
    - password
    - userPasswordSalt
    - sslEnabled

Property `sparqlEndpoints.virtuoso[0].id`

Default	none
Required	yes
Conflicts with	other IDs, must be unique
Valid values	string

Use this property to specify the ID of the in-memory endpoint. This ID can be user-defined and must be unique.

Property `sparqlEndpoints.virtuoso[0].authorization`

Default	`NONE`
Required	no
Conflicts with	none
Valid values	`NONE, PROVISIONED, REWRITE_FROM`

Use this property to specify the authorization strategy as explained in section Authorization strategies.

Property `sparqlEndpoints.virtuoso[0].host`

Default	none
Required	yes

Conflicts with	none
Valid values	string

Use this property to set the hostname of the Virtuoso server.

Property `sparqlEndpoints.virtuoso[0].port`

Default	none
Required	no
Conflicts with	none
Valid values	integer

Use this property to set the port of the Virtuoso server.

Property `sparqlEndpoints.virtuoso[0].username`

Default	none
Required	yes
Conflicts with	none
Valid values	string

Use this property to configure the username with which the connection to the server is established.

Property `sparqlEndpoints.virtuoso[0].password`

Default	none
Required	yes
Conflicts with	none
Valid values	string

Use this property to configure the password with which the connection to the server is established.

Property `sparqlEndpoints.virtuoso[0].userPasswordSalt`

Default	none
Required	no
Conflicts with	none
Valid values	string

Salt value used for internal user password generation. This property is only relevant if the property `authorization` is set to `PROVISIONED`.

If this property is not provided, a default internal value is used, which is not recommended in a production setup. Configure and treat this parameter with the same secrecy as `password`.

Property `sparqlEndpoints.virtuoso[0].sslEnabled`

Default	`false`
Required	no
Conflicts with	none
Valid values	Boolean

Use this property to enable encrypted data transfer from and to the Virtuoso server. If enabled, the configured `port` must be the Virtuoso SSL port and the server must be properly configured for this purpose. Refer to section Virtuoso setup.

Configuration example

```
sparqlEndpoints:
  virtuoso:
  - id: virtuoso
    authorization: PROVISIONED
    host: localhost
    port: 1111
    username: dba
    password: dba
    userPasswordSalt: $0M3Th1ng
```

Remote / HTTP

Use the following set of properties to connect to arbitrary HTTP SPARQL services.

Remote / HTTP configuration

- sparqlEndpoints
  - http
    - id
    - authorization
    - queryEndpointUrl
    - updateEndpointUrl
    - graphStoreEndpointUrl
    - username
    - password

Property `sparqlEndpoints.http[0].id`

Default	none
Required	yes
Conflicts with	other IDs, must be unique
Valid values	string

Use this property to specify the ID of the HTTP endpoint.

Property `sparqlEndpoints.http[0].authorization`

Default	`NONE`
Required	no
Conflicts with	`sparqlEndpoints.http[0]. graphStoreEndpointUrl`
Valid values	`NONE, REWRITE_FROM`

Use this property to specify the authorization strategy as explained in section Authorization strategies.

Note: The authorization strategy `REWRITE_FROM` is not supported for SPARQL 1.1 Graph Store requests while using a remote HTTP endpoint. Do not configure a `graphStoreEndpointUrl` or use the authorization strategy `NONE`.

Property `sparqlEndpoints.http[0].queryEndpointUrl`

Default	none
Required	no
Conflicts with	none
Valid values	string

Use this property to configure the endpoint to which SPARQL 1.1 queries are sent. At least one of `queryEndpointUrl`, `updateEndpointUrl` and `graphStoreEndpointUrl` must be provided.

Property `sparqlEndpoints.http[0].updateEndpointUrl`

Default	none

Required	no
Conflicts with	none
Valid values	string

Use this property to configure the endpoint to which SPARQL 1.1 update queries are sent. At least one of `queryEndpointUrl`, `updateEndpointUrl` and `graphStoreEndpointUrl` must be provided.

Property `sparqlEndpoints.http[0].graphStoreEndpointUrl`

Default	none
Required	no
Conflicts with	`sparqlEndpoints.http[0]. authorization`
Valid values	string

Use this property to configure the endpoint to which SPARQL 1.1 Graph Store Protocol requests are sent. Provide at least one of the properties `queryEndpointUrl`, `updateEndpointUrl` or `graphStoreEndpointUrl`.

Property `sparqlEndpoints.http[0].username`

Default	none
Required	only if `password` is provided
Conflicts with	`password` (see "Required")
Valid values	string

Basic authentication is used if this parameter is provided.

Property `sparqlEndpoints.http[0].password`

Default	none
Required	only if `username` is provided
Conflicts with	`username` (see "Required")
Valid values	string

Basic authentication is used if this parameter is provided.

Configuration example

```
sparqlEndpoints:
  http:
  - id: remote
    authorization: NONE
    queryEndpointUrl: http://remote.server/query
    updateEndpointUrl: http://remote.server/update
    graphStoreEndpointUrl: http://remote.server/gsp
```

In-memory

You can configure one ore more in-memory SPARQL endpoints. Based on Jena Models, in-memory endpoints do not provide persistent storage. Hence, shutting down a DataPlatform configured with an in-memory endpoint deletes your data and therefore you should use it only for testing purposes.

In-memory configuration

- sparqlEndpoints
  - inMemory
    - id
    - authorization
    - files

Property `sparqlEndpoints.inMemory[0].id`

Default	none
Required	yes
Conflicts with	other IDs, must be unique
Valid values	string

Use this property to specify the ID of the in-memory endpoint. This ID can be user-defined and must be unique.

Property `sparqlEndpoints.inMemory[0].authorization`

Default	`NONE`
Required	no
Conflicts with	none
Valid values	`NONE, REWRITE_FROM`

Use this property to specify the authorization strategy as explained in section Authorization strategies.

Property `sparqlEndpoints.inMemory[0].files`

Default	none
Required	no
Conflicts with	none
Valid values	list of files in form of file URI scheme or absolute filename

Use this property to preload the endpoint with local or remote .rdf files. Specify the property with file URI scheme:

```
files:
- file://hostname/path/to/file1.ttl
- file://hostname/path/to/file2.ttl
```

Or specify it by using an absolute file path:

```
files:
- /home/user/path/to/file1.ttl
- /home/user/path/to/file2.ttl
```

Configuration example

```
sparqlEndpoints:
  inMemory:
  - id: inMemory
    authorization: NONE
    files:
    - example_data.trig
```

OWL imports resolution

DataPlatform supports the interpretation of `owl:imports` statements. That means if a graph imports a second graph, the data of the second graph is included when querying the first graph. This applies both to SPARQL 1.1 Queries as well as SPARQL 1.1 Update queries by extending the dataset definition of the query: for selecting queries, by extending the `FROM` and `FROM NAMED` dataset definition, for update queries by extending the `USING` and `USING NAMED` dataset definition.

An OWL imports statement’s subject URI must be equal to the graph URI containing it, otherwise it is ignored.

The example below shows a OWL imports statement in TriG syntax:

```
<urn:graphA> {
  <urn:graphA> <owl:imports> <urn:graphB> .
}
```

Property `sparqlEndpoints.owlImportsResolution`

Default	`true`
Required	no
Conflicts with	none
Valid values	Boolean

Use this property to enable OWL imports resolution.

```
sparqlEndpoints:
  owlImportsResolution: false
```

Making a SPARQL endpoint accessible via HTTP

If you want to use a configured SPARQL endpoint via the HTTP SPARQL 1.1 Protocol or SPARQL 1.1 Graph Store HTTP Protocol, add the appropriate ID to the list of SPARQL proxies.

HTTP SPARQL endpoint configuration

Property `proxy.endpointIds[0]`

Default	`default`
Required	no
Conflicts with	none
Valid values	list of strings

Use this property to specify which SPARQL endpoints should be proxied. Only valid SPARQL endpoint IDs previously configured as `sparqlEndpoints.ENDPOINT_TYPE[i].id` can be used.

Property `proxy.labelProperties[0]`

Default	`http://www.w3.org/2000/01/rdf-schema#label`
Required	no
Conflicts with	none
Valid values	list of strings

Use this property to specify which RDF properties should be used to provide label values when matching IRIs against a search term during rewriting `SELEC`
-queries.

Note: This configuration property affects

- modification of `SELECT`-queries for search triggered by the `search-string` query parameter (see section *POST - SPARQL query via HTTP POST* of the *Developers Manual*).
- Results of `SELECT`-queries when the resolveLabels property is set to `LABELS`

Configuration example

```
proxy:
  endpointIds:
  - my_virtuoso
  labelProperties:
  - "http://www.w3.org/2000/01/rdf-schema#label"
  - "http://www.w3.org/2004/02/skos/core#prefLabel"
  - "http://www.w3.org/2004/02/skos/core#altLabel"
  languagePreferences:
  - "en"
  - ""
```

Property `proxy.languagePreferences[0]`

Default	en, ``
Required	no
Conflicts with	none
Valid values	list of strings

Specifies base language preferences for this instance. **Note:** This configuration property affects results of `SELECT`-queries when the resolveLabels property is set to `LABELS`.

The following example showcases a setup in which for each Resource all `rdfs:label`, Literals with *lang*uage `es`, then `en` and in the end those without a language are evaluated. If nothing nothing matches here, `skos:prefLabel` is examined in the same way.

Configuration example

```
proxy:
  endpointIds:
  - my_stardog
  labelProperties:
  - "http://www.w3.org/2000/01/rdf-schema#label"
  - "http://www.w3.org/2004/02/skos/core#prefLabel"
  languagePreferences:
  - "es"
  - "en"
  - ""
```

Authentication

Access to DataPlatform resources is restricted using OAuth 2.0.

OAuth 2.0 resource server

In order to protect access to it's resources, DataPlatform acts as an OAuth 2.0 resource server accepting and responding to a protected resource request using a JSON Web Token (JWT).

The OAuth 2.0 specification as well as the JSON Web Token specification don't define any mandatory claims to be contained in a JWT access token. However, if the property `spring.security.oauth2.resourceserver.jwt.issuer-uri` is set, the `iss` (issuer) claim is required to be contained in the JWT. It's value must be equal to the configured issuer URI. Additionally, in order to identify the requesting principal, either the username claim or the clientId claim must be contained in the JWT.

Resource server configuration

- spring
  - security
    - oauth2
      - resourceserver
        - anonymous
        - jwt
          - issuerUri
          - jwkSetUri
          - claims
            - username
            - groups
            - clientId

Property `spring.security.oauth2.resourceserver.anonymous`

Default	`false`
Required	no
Conflicts with	none
Valid values	Boolean

Use this property to allow anonymous access to protected resources.

Property `spring.security.oauth2.resourceserver.jwt.issuerUri`

Default	none
Required	yes
Conflicts with	`spring.security.oauth2.resourceserver.jwt.jwkSetUri`
Valid values	string

Use this property to specify the URI that an OpenID Connect Provider asserts as its Issuer Identifier if it supports OpenID Connect discovery.

If this property is set, the `iss` (issuer) claim is required to be contained in the JWT. The value of the claim has to be the same value as the configured issuer URI.

Note: If the authorization server is down when DataPlatform queries it (given appropriate timeouts), then startup will fail. Also, if the authorization server doesn’t support the Provider Configuration endpoint, or if DataPlatform must be able to start up independently from the authorization server, use the property jwk-set-uri instead.

Property `spring.security.oauth2.resourceserver.jwt.jwkSetUri`

Default	none
Required	yes
Conflicts with	`spring.security.oauth2.resourceserver.jwt.issuerUri`
Valid values	string

Use this property to specify the JSON Web Key URI to use to verify the JWT token.

Property `spring.security.oauth2.resourceserver.jwt.claims`

Use the following configuration options to specify the claims conveyed by a JWT used to access protected resources of DataPlatform. If nothing is configured, a default configuration is provided with the following configuration:

```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          claims:
            username: preferred_username
            groups: groups
            clientId: clientId
```

Property `spring.security.oauth2.resourceserver.jwt.claims.username`

Default	`preferred_username`
Required	no
Conflicts with	none
Valid values	string

Use this property to specify the claim providing the account name of the user accessing a protected resource.

Property `spring.security.oauth2.resourceserver.jwt.claims.groups`

Default	`groups`
Required	no
Conflicts with	none
Valid values	string | list of strings

Use this property to specify the claim identifying the roles (authorities) of the user accessing a protected resource.

Property `spring.security.oauth2.resourceserver.jwt.claims.clientId`

Default	`clientId`
Required	no
Conflicts with	none
Valid values	string

Use this property to specify the claim providing the OAuth 2.0 client ID to which a token was issued.

Configuration example

```
spring:
  security:
    oauth2:
      resourceserver:
        anonymous: true
        jwt:
          issuerUri: http://keycloak/auth/realms/cmem
```

Authorization

DataPlatform supports authorization of RDF named graphs and actions. Authorization for clients and/or users is specified by the access conditions model which is described in section Access conditions. You can configure root access for a specific group of users who are given unrestricted access regardless of the defined access conditions. Refer to section Root Access for more information.

Authorization configuration

- authorization
  - rootAccess
  - abox
    - adminGroup
    - publicGroup
    - anonymousUser
    - prefix
  - accessConditions
    - url
    - endpointId
    - graph

Property `authorization.rootAccess`

Default	`true`
Required	no
Conflicts with	none
Valid values	Boolean

Use this property to enable or disable root access (see section Root access).

Property `authorization.abox`

Use the following configuration options to specify values used by DataPlatform when working with RDF data, such as default URIs and prefixes.

Property `authorization.abox.adminGroup`

Default	`elds-admins`
Required	no
Conflicts with	none
Valid values	string

Use this property to configure the group that gets root access if enabled (see section Root access).

Property `authorization.abox.publicGroup`

Default	`urn:elds-backend-public-group`

Required	no
Conflicts with	none
Valid values	string

Use this property to configure the URI of the public user group (see section Public access).

Note: If you change this property, you also need to change existing URI descriptions and existing access conditions.

Property `authorization.abox.anonymousUser`

Default	`urn:elds-backend-anonymous-user`
Required	no
Conflicts with	none
Valid values	string

Use this property to configure the URI of the public user (see section Public access).

Note: If you change this property, you also need to change existing URI descriptions and existing access conditions.

Property `authorization.abox.prefix`

Default	`https://ns.eccenca.com/`
Required	no
Conflicts with	none
Valid values	string

Use this property to set the namespace of URIs created by DataPlatform.

Note: If you change this property, you also need to change the corresponding shape definitions for access conditions (more precisely, the URI template), as well as existing URI descriptions and existing access conditions.

Property `authorization.accessConditions`

Note: The access conditions model is empty if you provide no configuration in this section.

Property `authorization.accessConditions.url`

Default	none
Required	no
Conflicts with	`endpointId`, `graph`
Valid values	string

Use this property to set the URL of the access conditions model file. This can be either a remote (`http://...`) or a local (`file:...`) .rdf file. Refer to section Access conditions for more information on the access conditions model.

Property `authorization.accessConditions.endpointId`

Default	none
Required	only if `graph` provided
Conflicts with	`url`
Valid values	string

Use this property to set the endpoint containing the access conditions model graph specified by `authorization.accessConditions.graph`. The endpoint ID must be one of the configured endpoints (see section SPARQL endpoints). Refer to section Access conditions for more information on the access conditions model.

Property `authorization.accessConditions.graph`

Default	`urn:elds-backend-access-conditions-graph`
Required	only if `endpointId` provided
Conflicts with	`url`
Valid values	string

Use this property to set the graph containing the access conditions model.

Note: If you change this property, you also need to change the corresponding shape definitions for access conditions (more precisely, the UI SPARQL queries).

Refer to section Access conditions for more information on the access conditions model.

Configuration example

```
authorization:
  rootAccess: true
  abox:
    prefix: https://ns.eccenca.com/
    anonymousUser: urn:elds-backend-anonymous-user
    publicGroup: urn:elds-backend-public-group
    adminGroup: elds-admins
  accessConditions:
    endpointId: default
    graph: http://example.org/accessConditions
```

Root access

DataPlatform allows root access for a specific administrator group (see property authorization.abox.adminGroup). You can toggle root access using the property `authorization.rootAccess`. Regardless of the access conditions declared in the access conditions model (see Access conditions), all members of the administrator group are permitted to read and write all graphs of all endpoints and are allowed to perform all actions.

For example, the following configuration grants root access to any user in the group `admins`:

```
authorization:
  rootAccess: true
  abox:
    adminGroup: admins
```

For a detailed explanation of the configuration options used, refer to the sections Authorization configuration and In-memory provider.

Access conditions

Access conditions are defined in the access conditions model, an RDF named graph containing instances of the OWL class `eccauth:AccessCondition` defined by the eLDS Auth schema Ontology.

An access condition consists of the following elements:

- `type`: Only RDF resources of type `eccauth:AccessCondition` are valid access conditions.
- `requirements`: Conditions to be fulfilled by the logged-in user such as group membership. All conditions must be fulfilled, otherwise the access condition is not fulfilled.
- `grants`: Actions or data the user is allowed to execute and/or access such as accessing a specific API or reading a graph.

For simplicity, the Turtle serialization syntax is used in the following examples, but you can define the access conditions model in any RDF serialization.

Predefined URIs

DataPlatform recognizes a set of specific URIs with a precise meaning, as listed below:

- `urn:elds-backend-all-graphs`: Represents all RDF named graphs. You can use it as the object of either the `eccauth:readGraph` or the `eccauth:writeGraph` property to enable read or write access for all graphs.
- `urn:elds-backend-all-actions`: Represents all actions. You can use it as object of the `eccauth:allowedAction` property to allow performing all actions.
- `urn:elds-backend-public-group`: Represents the group which every user is member of. You can use it as object of the `eccauth:requiresGroup` property to define access conditions for all users including anonymous users. Refer to section Public Access for more information about public/anonymous access.
- `urn:elds-backend-anonymous-user`: Represents the anonymous user account. Refer to section Public Access for more information about public/anonymous access.
- `urn:elds-backend-actions-auth-access-control`: Represents the action needed to use the Authorization management API (see the Developer Manual). You can use it as object of the `eccauth:allowedAction` property to grant access to the Authorization management API if the user fulfills the access condition.

Access condition URIs

You have to define access condition as named individuals. Blank nodes are not allowed.

This example shows how an access condition URI and type definition should look like:

```
@prefix : <https://ns.eccenca.com/> .
@prefix eccauth: <https://vocab.eccenca.com/auth/> .

:SampleAccessCondition a eccauth:AccessCondition .
```

Access condition requirements

You can define requirements for an access condition. The requesting user must fulfill the complete set of requirements in order to get granted the specified rights. If you define no requirements for an access condition it is automatically fulfilled by every user. Refer to section Properties of chapter eLDS Auth schema ontology for a complete list of available requirements that you can specify.

Group specific access conditions

To restrict an access condition to a specific group of users, use the property `eccauth:requiresGroup`. The following example shows how to restrict an access condition to a specific group of users:

```
@prefix : <https://ns.eccenca.com/> .
@prefix eccauth: <https://vocab.eccenca.com/auth/> .

:SampleAccessCondition a eccauth:AccessCondition ;
  eccauth:requiresGroup :ExampleUserGroup ;
  eccauth:readGraph :ExampleGraph ;
.
```

The example defines that only users which are member of the group `:ExampleUserGroup` can read the graph `:ExampleGraph`.

Note: LDAP or in-memory group and user names are URL-encoded and concatenated with the prefix of the property `authorization.abox.prefix` for matching against groups referenced in the access conditions model. For example, the URI corresponding to the group with ID 'Test:Group' is `:Test%3AGroup`.

Endpoint specific access conditions

By default, every defined access condition applies to all configured endpoints. If you need to define access conditions for a specific endpoint, use the property `eccauth:requiresEndpoint`.

```
@prefix : <https://ns.eccenca.com/> .
@prefix eccauth: <https://vocab.eccenca.com/auth/> .

:SampleAccessCondition a eccauth:AccessCondition ;
  eccauth:requiresEndpointId "exampleEndpoint" ;
  eccauth:requiresGroup :ExampleUserGroup ;
  eccauth:readGraph :ExampleGraph ;
.
```

The access condition as shown in the example above applies only to the endpoint with the ID `exampleEndpoint`. Additionally, all access conditions without an `eccauth:requiresEndpointId` property will also match.

Note: This only affects interactions with the configured endpoint (see SPARQL Endpoints). The access condition is ignored in any other case.

Access condition grants

You can define permissions granted to users which fulfill an access condition. Refer to section Properties of chapter eLDS Auth schema ontology for a complete list of available grants that you can specify. Continuing the previous example, this one demonstrates how to specify multiple grants:

```
@prefix : <https://ns.eccenca.com/> .
@prefix eccauth: <https://vocab.eccenca.com/auth/> .

:SampleAccessCondition a eccauth:AccessCondition ;
  eccauth:requiresEndpointId "exampleEndpoint" ;
  eccauth:requiresGroup :ExampleUserGroup ;
  eccauth:writeGraph :ExampleGraph ;
  eccauth:readGraph <urn:elds-backend-all-graphs> ;
.
```

This access condition defines two grants for the endpoint `exampleEndpoint` to all users of the group `:ExampleUserGroup`:

- Users can modify the `:ExampleGraph` graph.
- Users can read from all graphs.

Adding new access conditions during runtime

The Access Conditions Management endpoint (see section *Authorization and authentication management API* of the Developer Manual) allows adding new access conditions if the requesting user is allowed to. This feature is only enabled if an endpoint is used as the store for the access conditions model (see properties `authorization.accessConditions.endpointId` and `authorization.accessConditions.graph`). A configuration using a local access condition model file is read-only.

For example, consider the following existing access condition:

```
@prefix : <https://ns.eccenca.com/> .
@prefix eccauth: <https://vocab.eccenca.com/auth/> .

:AllowExampleGraphsGrant a eccauth:AccessCondition ;
  eccauth:requiresGroup :ExampleUserGroup ;
  eccauth:grantWriteGraphPattern "http://example.org/*" ;
.
```

This access condition allows a user of the group `:ExampleUserGroup` to add a new access condition granting write access to any graph with a URI starting with `http://example.org/`, as for example:

```
@prefix : <https://ns.eccenca.com/> .
@prefix eccauth: <https://vocab.eccenca.com/auth/> .

:NewAccessCondition a eccauth:AccessCondition ;
  eccauth:writeGraph <http://example.org/ExampleGraph> ;
.
```

Refer to section *Authorization and authentication management API* of the Developer Manual for more information on the functionality and restrictions of this feature.

Public access

DataPlatform allows public (unauthenticated) access.

Therefore the application uses a specific user group - the public group (`urn:elds-backend-public-group`) - which any user belongs to including unauthenticated/anonymous user. To configure the URI of the public group use the property `authorization.abox.publicGroup`.

To define publicly available graphs and/or actions you need to declare an access condition which requires the public group. Furthermore, you can define additional requirements for the public group like the restriction to a specific endpoint.

```
@prefix : <https://ns.eccenca.com/> .
@prefix eccauth: <https://vocab.eccenca.com/auth/> .

:PublicAccessCondition a eccauth:AccessCondition ;
  eccauth:requiresGroup <urn:elds-backend-public-group> ;
  eccauth:readGraph :PublicGraph ;
```

```
.

:PublicAccessConditionExampleEndpoint a eccauth:AccessCondition ;
  eccauth:requiresEndpointId "exampleEndpoint" ;
  eccauth:requiresGroup <urn:elds-backend-public-group> ;
  eccauth:readGraph :ExampleGraph ;
.
```

In the example shown above, any user can read the graph `:PublicGraph` in every configured endpoint without further limitations (as stated by `:PublicAccessCondition`). Additionally, any user can read the graph `:ExampleGraph` of the endpoint `exampleEndpoint`.

eLDS Auth schema ontology

The eLDS Auth schema ontology is a vocabulary for context-sensitive authorization of RDF named graphs and arbitrary actions. You can use it to create access conditions defining requirements which must be fulfilled to get certain grants. The delegation of grants is based on a whitelist. That means a user is granted if the conjunction of requirements of an access condition is fulfilled. In the case of DataPlatform the vocabulary is utilized to grant read and write access to RDF named graphs and to allow the execution of actions.

Namespace	`https://vocab.eccenca.com/auth/`
Preferred prefix	`eccauth`

Example usage

The following example shows how to use the vocabulary to grant access to an RDF named graph:

```
@prefix : <https://ns.eccenca.com/> .
@prefix eccauth: <https://vocab.eccenca.com/auth/> .

:AccessConditionX a eccauth:AccessCondition ;
  eccauth:requiresGroup :GroupUsers ;
  eccauth:readGraph :GraphUserData .

:AccessConditionY a eccauth:AccessCondition ;
  eccauth:requiresGroup :GroupAdmins ;
  eccauth:writeGraph :GraphUserData .
```

The example describes two access conditions: * `:AccessConditionX` grants read access to the graph `http://example.org/GraphUserData` for all users of the group `:GroupUsers`. * `:AccessConditionY` grants write access to the same graph but only for users of the group `:GroupAdmins`.

Classes

- **eccauth:Principal**
  A principal can be any entity, such as an individual, a group, etc.
- **eccauth:Account**
  The class of concrete principals which can be authenticated.
- **eccauth:Group**
  A group represents a collection of accounts.
- **eccauth:AccessCondition**
  An access condition defines a set of requirements which must be fulfilled by a session to get specified grants. The set of requirements is considered a conjunction.
- **eccauth:Session**
  A session is a period of time where an account is successfully authenticated. It provides information about who is authenticated, how and when authentication was established.
- **eccauth:Graph**
  A set of triples according to the RDF 1.1 specification.
- **eccauth:Action**
  An action defines an activity or function.

Properties

- **eccauth:memberOf**
  Indicates the membership of an account to a group.
- **eccauth:openedBy**
  The account which is the originator of a session.
- **eccauth:hasAttribute**
  Super property of all properties used to define session characteristics. Do not use this property directly, use appropriate sub-properties instead.
- **eccauth:hasEndpointId**
  The ID of the endpoint the session applies for.
- **eccauth:requiresAttribute**
  Super property of all properties used to define requirements of an access condition. Do not use this property directly, use appropriate sub-

properties instead. The set of object values bound by sub-properties of this property must be met as conjunction in order to fulfill the requirements of an access condition.

- **eccauth:requiresGroup**
  The group the account must be member of to meet the access condition.
- **eccauth:requiresAccount**
  A specific account required by the access condition.
- **eccauth:requiresEndpointId**
  The ID of the endpoint needed to meet the access condition.
- **eccauth:isAllowed**
  Super property of all properties used to define grants if the requirements of an access condition are met. Do not use this property directly, use appropriate sub-properties instead.
- **eccauth:readGraph**
  Grants read access to a graph.
- **eccauth:writeGraph**
  Grants read/write access to a graph.
- **eccauth:allowedAction**
  Grants permission to execute an action.
- **eccauth:grantReadGraphPattern**
  Grants management of conditions granting read access on graphs matching the defined pattern. A pattern consists of a constant string and a wildcard (`*`) at the end of the pattern or the wildcard alone. Examples: `http://example.org/&#42;` matches any string starting with `http://example.org/`, `urn:r` matches the exact string `urn:r` and `*` matches any string.
- **eccauth:grantWriteGraphPattern**
  Grants management of conditions granting write access on graphs matching the defined pattern. Notice that write grant allowance implies read grant allowance. A pattern consists of a constant string and a wildcard (`*`) at the end of the pattern or the wildcard alone. Examples: `http://example.org/&#42;` matches any string starting with `http://example.org/`, `urn:r` matches the exact string `urn:r` and `*` matches any string.
- **eccauth:grantAllowedActionPattern**
  Grants management of conditions granting action allowance for actions matching the defined pattern. A pattern consists of a constant string and a wildcard (`*`) at the end of the pattern or the wildcard alone. Examples: `http://example.org/&#42;` matches any string starting with `http://example.org/`, `urn:r` matches the exact string `urn:r` and `*` matches any string.

Application logging

By default, DataPlatform only logs to the console. You can change the log level or configure logging into a file.

There are multiple levels of logging you can choose from that are explained in the table below.

Level	Description
TRACE	The TRACE level designates informational events of very low importance.
DEBUG	The DEBUG level designates informational events of lower importance.
INFO	The INFO level designates informational messages highlighting overall progress of the application.
WARN	The WARN level designates potentially harmful situations.
ERROR	The ERROR level designates error events which may or not be fatal to the application.

The levels can also be configured on runtime via the `loggers` HTTP endpoint as described in section *Application loggers* of the Developer Manual.

Application logging configuration

Property `logging.level.root`

Default	`WARN`
Required	no
Conflicts with	none
Valid values	`TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `OFF`

Use this property to change the root log level to one of the available levels.

Property `logging.level.[PACKAGE]`

Default	level from `logging.level.root`
Required	no
Conflicts with	none
Valid values	`TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `OFF`

Use this property to specify log levels from individual packages. Packages that are not configured individually inherit the logging level of the `logging.level.root` property.

Property `logging.file`

Default	none
Required	no
Conflicts with	none
Valid values	string (file path)

Use this property to specify where you want to store your logging file. Specifying a file leads to both, logging to standard output and the file.

File output creates an auto-rotating file with 10 MB file size each.

Configuration example

```
logging:
  level:
    root: WARN
    com.eccenca.elds.backend: DEBUG
    org.springframework: INFO
  file: /var/logs/dataplatform.log
```

Application logging with Logback

Logging for DataPlatform can also be configured with Logback, which, for example, allows a more granular control on file rolling strategies. For further information on configuration options, refer to the Logback's Configuration manual section and the Spring Boot's Configure Logback for Logging manual section.

Property `logging.configuration`

Default	none
Required	no
Conflicts with	none
Valid values	string (file path)

Use this property to specify where the Logback configuration is located.

Configuration example

```
logging:
  configuration: ${ELDS_HOME}/etc/dataplatform/logback.xml
```

The following example `logback.xml` file defines a rolling file strategy where files are rotated on a time base (1 day) with a limit of 7 files, which means that the logging files contain a log history of a maximum of 1 week.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="TIME_BASED_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>/opt/elds/var/log/dataplatform.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <!-- daily rollover, history for 1 week -->
      <fileNamePattern>/opt/elds/var/log/dataplatform.%d{yyyy-MM-dd}.log</fileNamePattern>
      <maxHistory>7</maxHistory>
    </rollingPolicy>
    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>
  <logger name="com.eccenca" level="INFO">
    <appender-ref ref="TIME_BASED_FILE" />
  </logger>
</configuration>
```

Audit trail logging

DataPlatform is able to log the access of each user to named graphs in form of an audit trail log under the logger name `audit`.

Audit trail logging configuration

- auditTrail
    - enabled
    - auditedGraphs

Property `auditTrail.enabled`

Default	false
Required	no
Conflicts with	none
Valid values	Boolean

Use this property to enable logging of read and write access to every graph access. If `auditTrail.auditedGraphs` is specified, only those graphs are logged.

Note: If audit trail logging is enabled, RDF upload over the Graph Store Protocol interface is limited to triple formats. Any attempt to upload a quad format results in an HTTP 415 error.

Property `auditTrail.auditedGraphs`

Default	none
Required	no
Conflicts with	none
Valid values	Boolean

Use this property to specify graphs whose read and write access you want to be logged. Omit this value to log access to all graphs.

Configuration example

```
auditTrail:
  enabled: true
  auditedGraphs:
  - "example.org/data"
  - "aksw.org"
```

Audit log output structure

```
1999-03-14 13:37:08.015  INFO 80085 --- [nio-9090-exec-2] audit : Graph access: user=urn:user, action=read,
graph=urn:graph
```

As shown in the log output example above, every log contains the users' URI, the action (read or write) and the affected graphs.

Every access attempt to an audited graph is logged regardless of access control. Additionally, each unauthorized graph access is logged, leading to two log entries: one for the graph access itself and one for the reason why it was unauthorized:

```
1999-03-14 13:37:08.015  INFO 80085 --- [nio-9090-exec-2] audit : Graph access: user=urn:user, action=read,
graph=urn:graph
1999-03-14 13:37:08.015  INFO 80085 --- [nio-9090-exec-2] audit : Unauthorized graph access: user=urn:user,
action=read, graph=urn:graph
```

In the case that a user is allowed to read all graphs and the SPARQL query does not define an RDF dataset, the `ALL` keyword is logged, since all graphs in the store are queried:

```
1999-03-14 13:37:08.015  INFO 80085 --- [nio-9090-exec-2] audit : Graph access: user=urn:user, action=read,
graph=ALL
```

For SPARQL update operations that do not define an RDF dataset and the user is allowed to write all graphs, the `DEFAULT` keyword is logged, since the default graph of the store is presumed:

```
1999-03-14 13:37:08.015  INFO 80085 --- [nio-9090-exec-2] audit : Graph access: user=urn:user, action=read,
graph=DEFAULT
```

Embedded Tomcat

The URL under which DataPlatform is accessible has the following form: `PROTOCOL://HOST\:PORT/CONTEXT_PATH`

where:

- `PROTOCOL`: `http` or `https` depending on SSL configuration (see section SSL support)
- `HOST`: The hostname pointing to the server where DataPlatform is installed
- `PORT`: The TCP port where the embedded server is available (see the property server.port)
- `CONTEXT_PATH`: The context path under which DataPlatform is available (see the property server.servlet.contextPath)

Embedded Tomcat configuration

- server
    - port
    - servlet
        - contextPath

Property `server.port`

Default	9090
Required	no
Conflicts with	none
Valid values	integer

Use this property to set the TCP port where the embedded server is available.

Property `server.servlet.contextPath`

Default	''
Required	no
Conflicts with	none
Valid values	string

Use this property to define the context path under which DataPlatform is available. If this property is provided, use a leading slash.

Configuration example

```
server:
  port: 9090
  servlet:
    contextPath: /dataplatform
```

Caching

DataPlatform provides caching support which is enabled by default with an in-memory Caffeine cache.

Caching configuration

- spring
    - cache
        - type
    - redis
        - host
        - port

Property `spring.cache.type`

Default	`CAFFEINE`
Required	no
Conflicts with	none
Valid values	`CAFFEINE, REDIS, NONE`

Use this property to define the type of cache to use. The default type (`CAFFEINE`) provides an in-memory cache suitable for simple standalone installations or test deployments.

Set `REDIS` to use a Redis cache. Use this cache type in scenarios with higher scalability demands or clustered setups. If this cache type is used, you must set the spring.cache.redis.host and spring.cache.redis.port properties, too.

To disable caching, set the type to `NONE` (not recommended).

Property `spring.cache.redis.host`

Default	none
Required	only if `spring.cache.type=REDIS` provided
Conflicts with	none
Valid values	string

Use this property to set the hostname where the Redis cache is available.

Property `spring.cache.redis.port`

Default	none
Required	only if `spring.cache.type=REDIS` provided
Conflicts with	none
Valid values	integer

Use this property to set the TCP port where the Redis cache is available.

Configuration example

```
spring:
  cache:
    type: REDIS
  redis:
    host: localhost
    port: 6379
```

HTTPS support

Standalone mode

If DataPlatform is executed in standalone mode (see Standalone), the embedded servlet container can be configured to support one-way (server certification) or two-way (server and client certification) SSL. A KeyStore is required for one-way SSL and both a KeyStore as well as a TrustStore are required for two-way SSL.

Refer to the Oracle documentation to see how to create KeyStore and TrustStore files.

SSL support configuration

- server
    - ssl

- key-store
- key-store-password
- client-auth

Property `server.ssl.key-store`

Default	none
Required	no
Conflicts with	none
Valid values	string

Use this property to define the path to the KeyStore used for one-way or two-way SSL authentication.

In case of two-way authentication, a TrustStore must also be configured. This configuration must be provided as Java system properties either directly in the execution command or as part of the `JAVA_TOOL_OPTIONS` environment variable, e.g.:

```
JAVA_TOOL_OPTIONS=${JAVA_TOOL_OPTIONS} -Djavax.net.ssl.trustStore=path_to_trust_store.jks -Djavax.net.ssl.
trustStorePassword=trust_store_password
```

Property `server.ssl.key-store-password`

Default	none
Required	only if `key-store` is provided
Conflicts with	none
Valid values	string

Use this property to set the password to unlock the KeyStore used for one-way or two-way SSL authentication.

Property `server.ssl.client-auth`

Default	none
Required	no
Conflicts with	none
Valid values	none, `NEED`, `WANT`

Use this property to define the client identification policy.

If `WANT` is set, client identification is optional. If `NEED` is set, client identification is mandatory, so unauthenticated clients are refused.

Configuration example

```
server:
  ssl:
    key-store: ./key-store.jks
    key-store-password: jks-password
    client-auth: NEED
```

Proxy deployment

If DataPlatform is running behind a proxy server (e.g. Apache) then you must use all of the following properties to enforce HTTPS:

- server
  - tomcat
    - remoteIpHeader
    - protocolHeader
- security
  - requireSsl

property `server.tomcat.remoteIpHeader`

default	none

required	no
conflicts with	none
valid values	string

Use this property to set the request header which is required to identify the originating IP address of the client connecting to DataPlatform through an HTTP proxy.

property `server.tomcat.protocolHeader`

default	none
required	no
conflicts with	none
valid values	string

Use this property to set the request header which is required to identify the originating protocol of an HTTP request through an HTTP proxy.

property `security.requireSsl`

default	`false`
required	no
conflicts with	none
valid values	Boolean

Use this property to enable secure channels for all requests.

Configuration recommendation

```
server:
  tomcat:
    remoteIpHeader: x-forwarded-for
    protocolHeader: x-forwarded-proto

security:
  requireSsl: true
```

Note: This configuration recommendation provides settings for headers most commonly used by proxies. Make sure to add **all three properties** in order to enforce HTTPS.

Cross-origin resource sharing (CORS)

DataPlatform supports Cross-origin resource sharing (CORS).

CORS configuration

- http
    - cors
        - allowedOrigins
        - allowedMethods
        - allowedHeaders
        - exposedHeaders
        - allowCredentials
        - maxAge

Property `http.cors.allowedOrigins`

Default	*
Required	no
Conflicts with	none
Valid values	list of strings

Use this property to define the list of allowed origins. The values must be either specific origins, e.g. `http://example.org`, or `*` for all origins.

Property `http.cors.allowedMethods`

Default	`OPTIONS, HEAD, GET, POST, PUT, DELETE, PATCH`
Required	no
Conflicts with	none
Valid values	list of strings

Use this property to define the list of allowed HTTP methods. The special value `*` allows all methods.

Property `http.cors.allowedHeaders`

Default	`Authorization, X-Requested-With, Content-Type, Content-Length, ETag`
Required	no
Conflicts with	none
Valid values	list of strings

Use this property to define the list of allowed HTTP headers. The special value `*` may be used to allow all headers.

Property `http.cors.exposedHeaders`

Default	`WWW-Authenticate, Link, ETag`
Required	no
Conflicts with	none
Valid values	list of strings

Use this property to define the list of headers that an actual response might have and can be exposed.

Property `http.cors.allowCredentials`

Default	`false`
Required	no
Conflicts with	none
Valid values	Boolean

Use this property to define whether the browser should send credentials, such as cookies along with cross domain requests.

Property `http.cors.maxAge`

Default	`3600`
Required	no
Conflicts with	none
Valid values	non-negative integer

Use this property to define how long in seconds the response from a pre-flight request can be cached by clients.

Configuration example

```
http:
  cors:
    allowedOrigins:
    - http://example.org
    - https://example.com
```

File upload limits

You can increase DataPlatform's maximal file upload size and request size.

File upload limits configuration

- spring
    - http
        - multipart:
            - maxFileSize
            - maxRequestSize

Property `spring.http.multipart.maxFileSize`

Default	`1024MB`
Required	no
Conflicts with	none
Valid values	string

Use this property to define the maximum size of an uploaded file in number of bytes. Values can use the suffixed "MB" or "KB" (e.g. '1024MB').

Note: If DataPlatform is deployed in a Servlet container, make sure to also configure support for large file sizes.

Property `spring.http.multipart.maxRequestSize`

Default	`1024MB`
Required	no
Conflicts with	none
Valid values	string

Use this property to define the maximum size of HTTP request in number of bytes. Values can use the suffixed "MB" or "KB" (e.g. '1024MB').

Configuration example

```
spring:
  http:
    multipart:
      maxFileSize: 1024MB
      maxRequestSize: 1024MB
```

Network timeouts for proxied requests

If DataPlatform is accessed behind a HTTP server, you should increase the network timeout for proxied requests if large files uploads are expected. Apache HTTP Server provides a default timeout of 300 seconds which will most likely not be big enough for large file uploads.

OpenAPI Specification and Swagger UI

You can activate endpoints to expose an OpenAPI compliant specification of the available DataPlatform APIs. Developers can make use of this information to understand the API and to bootstrap client integration code.

OpenAPI Documentation

- springdoc
    - api-docs
        - enabled

Property `springdoc.api-docs.enabled`

Default	`false`
Required	no
Conflicts with	none

Valid values	true, false

Use this property to enable and expose endpoint that provide the OpenAPI compliant specification of the DataPlatform APIs. The following endpoints will become available when this option is set to `true`:

- <DATA_PLATFORM_URI>/v3/api-docs
- <DATA_PLATFORM_URI>/v3/api-docs.yaml
- <DATA_PLATFORM_URI>/v3/api-docs/swagger-config

Swagger UI

- springdoc
  - swagger-ui
    - enabled

Property `springdoc.swagger-ui.enabled`

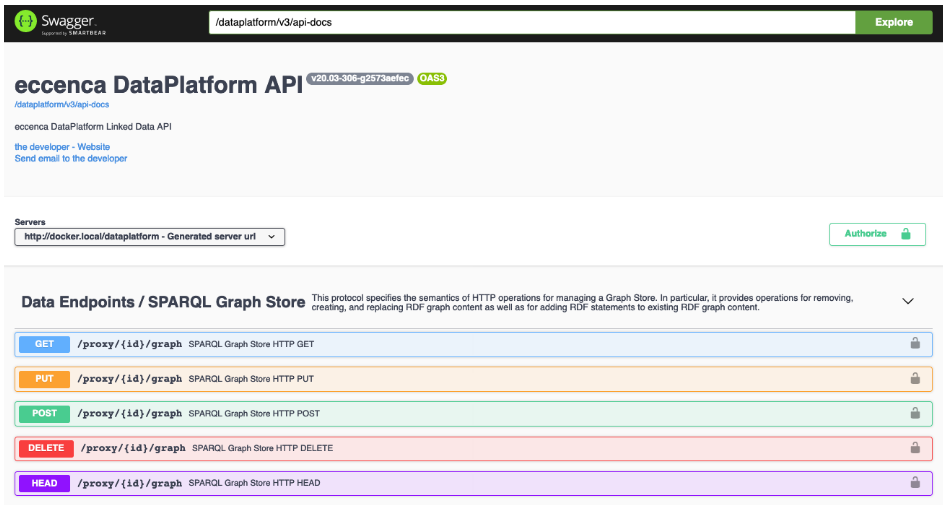
Default	`false`
Required	no
Conflicts with	none
Valid values	true, false

Note: Swagger UI requires the API-Documentation to be enabled as the ui is making use of the exposed specification.

Use this property to enable and expose a Swagger UI browser interface that can be used to explore and interact with the APIs. The following endpoints will become available when this option is set to `true`:

- <DATA_PLATFORM_URI>/swagger-ui

Use your web browser to explore <DATA_PLATFORM_URI>/swagger-ui :



eccenca DataPlatform API v20.03-306-g2573aefec OAS3

/dataplatform/v3/api-docs

eccenca DataPlatform Linked Data API

the developer - Website
Send email to the developer

Servers
http://docker.local/dataplatform - Generated server url

Data Endpoints / SPARQL Graph Store — This protocol specifies the semantics of HTTP operations for managing a Graph Store. In particular, it provides operations for removing, creating, and replacing RDF graph content as well as for adding RDF statements to existing RDF graph content.

GET	`/proxy/{id}/graph`	SPARQL Graph Store HTTP GET
PUT	`/proxy/{id}/graph`	SPARQL Graph Store HTTP PUT
POST	`/proxy/{id}/graph`	SPARQL Graph Store HTTP POST
DELETE	`/proxy/{id}/graph`	SPARQL Graph Store HTTP DELETE
HEAD	`/proxy/{id}/graph`	SPARQL Graph Store HTTP HEAD

The servers URLs can be customized by setting the environment variable OPENAPI_SERVER_URLS on the machine or in the docker container that runs DataManager:

```
export OPENAPI_SERVER_URLS="https://my-custom.domain.com:443/dataplatform"
```

Configuration example

In order to activate OpenAPI Documentation and Swagger UI provide the following in your DataPlatform `application.yml`:

```
## SpringDoc Endpoints
springdoc:
  swagger-ui:
    enabled: true
  api-docs:
    enabled: true
```

Activation with Environment Variables

Other than through the `application.yml` the endpoints can also be activated by setting the environment variables `SPRINGDOC_SWAGGER_UI_ENABLED` and `SPRINGDOC_API_DOCS_ENABLED` to `true` on the machine or in the docker container that runs DataManager:

```
export SPRINGDOC_SWAGGER_UI_ENABLED=true
export SPRINGDOC_API_DOCS_ENABLED=true
```