Co-financed by the Connecting Europe Facility of the European Union

increasinG tRust with eId for Developing buSiness

D3.1 Business Attribute Aggregation (BAA)

Document Identification			
Status	Final	**Due Date**	31/05/2021
Version	1.0	**Submission Date**	01/06/2021

Related Activity	Act 3	**Document Reference**	D3.1
Related Deliverable(s)	D2.1, D3.2, D3.3	**Dissemination Level (*)**	PU
Lead Participant	ATOS	**Lead Author**	Ross Little
Contributors	UAEGEAN KOMPANY ADACOM INFOCERT	**Reviewers**	Peter Bainbridge-Clayton (KOMPANY) Pasquale Minervini, (INFOCER)

Keywords

Identity, Assurance, Trust, Framework, eIDAS, KYC, KYB, Claims, Business Registers, Banks, Relying Party, Data Consumer, Data Provider, OpenID Connect Provider, Legal Identity

(*) Dissemination level: **PU**: Public, fully open, e.g. web; **CO:** Confidential, restricted under conditions set out in Model Grant Agreement; **CI:** Classified EU RESTRICTED, EU CONFIDENTIAL, **Int =** Internal Working Document, information as referred to in Commission Decision 2001/844/EC.

Document Information

List of Contributors	
Name	Partner
Nikos Triantafyllou	UAEGEAN
Peter Bainbridge-Clayton	KOMPANY
Pasquale Minervini	INFOCERT
Stamoulis Zamanis	ADACOM
Miryam Villegas Jimenez	ATOS
Raquel Cortés Carreras	ATOS
Juan Carlos Perez Baun	ATOS

Document History			
Version	Date	Change editors	Changes
0.1	13/05/2021	Ross Little (ATOS)	Initial version
0.2	19/05/2021	Ross Little (ATOS)	Updated after internal review and ready for review by partners.
0.3	27/05/2021	Ross Little (ATOS)	Updated after peer review.
1.0	31/05/2021	Juan Alonso, Juan Carlos Perez (ATOS)	Review of final version before submission

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Ross Little (ATOS)	19/05/2021
Peer reviewers	Stamoulis Zamanis (ADACOM) Peter Bainbridge-Clayton (KOMPANY)	27/05/2021
Quality Manager	Juan Alonso (Atos)	31/05/2021

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AML	Anti Money Laundering
CI	Continuous Integration
CD	Continuous Development
EC	European Commission
D3.1	Deliverable number 1 belonging to Activity 3
DC	Data Consumer
DoA	Description of the Action
DP	Data Provider
DS	Data Subject
CDD	Customer due diligence
eIDAS	Regulation (EU) No. 910/2014 on electronic identification and trust services for electronic transactions in the internal market
FIs	Financial Institutions
IDA	Identity Assurance
IdP	Identity Provider
JSON	JavaScript Object Notation
JWKS	JSON Web Key Set
KYB	Know Your Business
KYC	Know Your Customer
LEI	Legal Entity Identifier
LOU	Local Unit Operator
OIDC	OpenID Connect
OIDC IDA	IDA OpenID Connect for IDentity Assurance
OP	OpenID Connect Provider
OS	Open Source
RP	Relying Party
SAML	Security Assertion Markup Language
VM	Virtual Machine
WG	Working Group

Executive Summary

The deliverable is a report covering the implementation of the Business Attribute Aggregator (BAA) module so that it satisfies the high-level design as carried out in Task 2.2. This report provides a recap of the high-level design before fully specifying the BAAs low-level design and interwork of each module is fully specified for the microservices developed and deployed for the GRIDS Project.

The BAA implementation follows the OIDC IDA 1.0 specification [7], and as a next release for the specification is still being actively worked on by the OIDC IDA WG, members of GRIDS have attended the WG teleconferences and contributed to new updates to the specification based on our needs and experiences.

The CI/CD process executed for the project is also elaborated on which facilitated partner code to be uploaded to the Atos GitLab and generate docker containers and upload on Nexus automatically.

The result is a report that accurately reflects the developed BAA software modules and the design process.

1 Introduction

1.1 Purpose of the document

The purpose of this document is to detail the GRIDS Business Attribute Aggregator (BAA) implementation design and development process. The BAA enables the authentication of natural and legal persons over eIDAS and uses this assured identity data in the further collection of Know Your Customer (KYC) & Know Your Business (KYB) claims from Data Providers, that are connected over the GRIDS BAA in a Distributed Claims Approach as specified by the OIDC IDA 1.0 specification [7].

1.2 Relation to other project work

This deliverable covers the low-level design and implementation of the GRIDS BAA as per the Task 3.1 of the DOA and as such follows the high-level design specified in Task 2.2.

It supports the OIDC Distributed Claims Approach for querying Data Providers for KYC/KYB claims as per the interface realised in Task 3.2.

It interfaces with the eIDAS network via the SPHUB as is implemented in Task 3.3.

1.3 Structure of the document

This document is structured in 3 major chapters

- **Chapter 2** gives an overview of the high-level design elaborated in Task 2.2
- **Chapter 3** specifies the low-level design of the BAA
- **Chapter 4** describes the CI/CD process for GRIDS development

2 High Level Identity Assurance Overview

The implementation of the BAA is based on OpenID Connect for Identity Assurance 1.0 (aka "OIDC IDA 1.0"), Ref[3], where it is acting as an OpenID Connect Provider (OP) . It supports authentication of both natural and legal identities over eIDAS and the querying of Data Providers for KYC/KYB claims with the authenticated identity of the Data Subject. The querying of the Data Provider for KYC/KYB claims is supported in a distributed claims approach, as specified in the OIDC Core specification [6].

The following Figure 1 referenced from the high-level design [14] demonstrates the high level interwork following the OIDC authorisation and distributed claims flow.

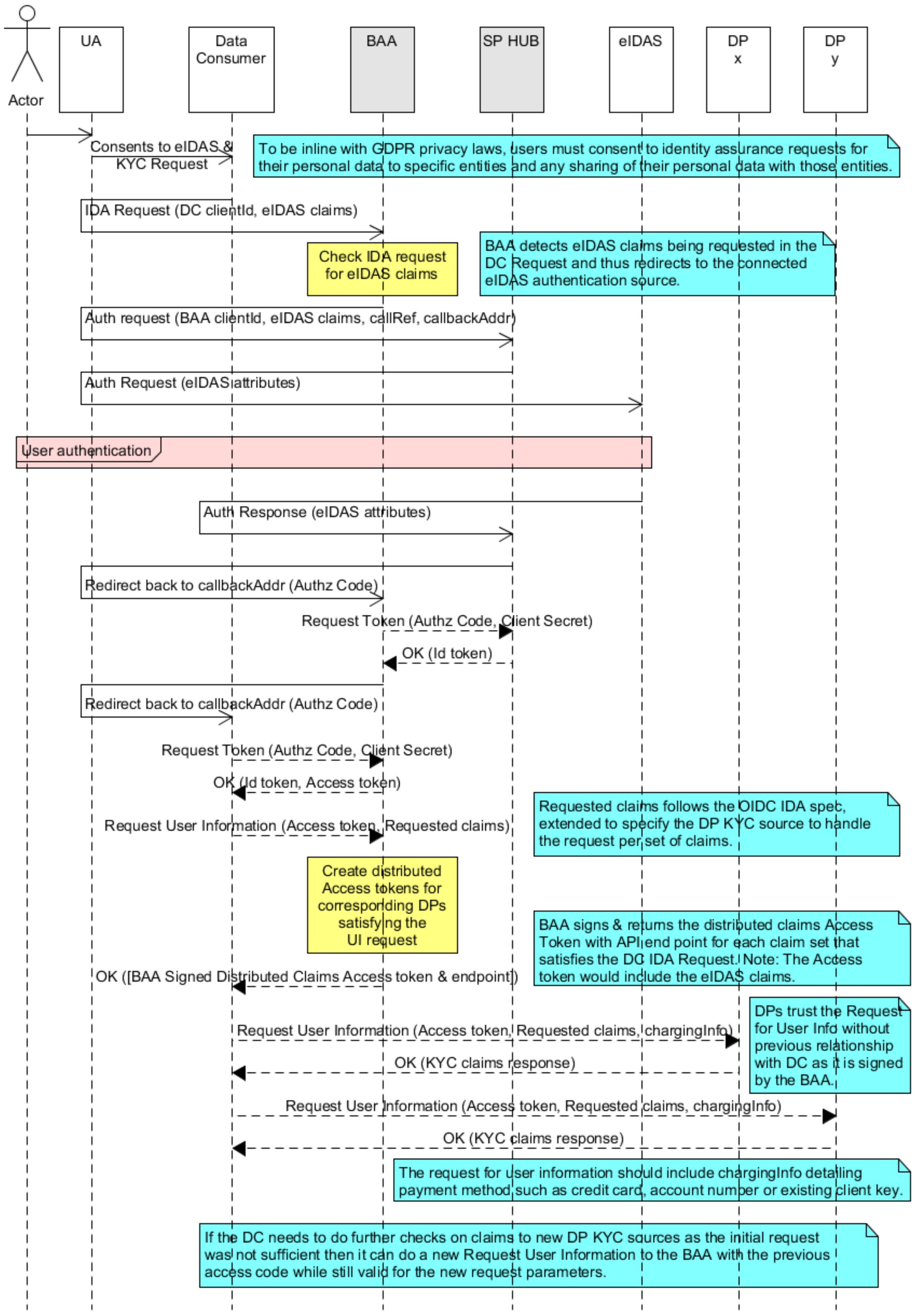

Actor
UA
Data Consumer
BAA
SP HUB
eIDAS
DP x
DP y
Consents to eIDAS & KYC Request
To be inline with GDPR privacy laws, users must consent to identity assurance requests for their personal data to specific entities and any sharing of their personal data with those entities.
IDA Request (DC clientId, eIDAS claims)
Check IDA request for eIDAS claims
BAA detects eIDAS claims being requested in the DC Request and thus redirects to the connected eIDAS authentication source.
Auth request (BAA clientId, eIDAS claims, callRef, callbackAddr)
Auth Request (eIDAS attributes)
User authentication
Auth Response (eIDAS attributes)
Redirect back to callbackAddr (Authz Code)
Request Token (Authz Code, Client Secret)
OK (Id token)
Redirect back to callbackAddr (Authz Code)
Request Token (Authz Code, Client Secret)
OK (Id token, Access token)
Request User Information (Access token, Requested claims)
Requested claims follows the OIDC IDA spec, extended to specify the DP KYC source to handle the request per set of claims.
Create distributed Access tokens for corresponding DPs satisfying the UI request
BAA signs & returns the distributed claims Access Token with API end point for each claim set that satisfies the DC IDA Request. Note: The Access token would include the eIDAS claims.
OK ([BAA Signed Distributed Claims Access token & endpoint])
Request User Information (Access token, Requested claims, chargingInfo)
DPs trust the Request for User Info without previous relationship with DC as it is signed by the BAA.
OK (KYC claims response)
Request User Information (Access token, Requested claims, chargingInfo)
OK (KYC claims response)
The request for user information should include chargingInfo detailing payment method such as credit card, account number or existing client key.
If the DC needs to do further checks on claims to new DP KYC sources as the initial request was not sufficient then it can do a new Request User Information to the BAA with the previous access code while still valid for the new request parameters.

Figure 1 Distributed Claims approach

2.1 BAA External Interfaces

The BAA supports the following external interfaces:

2.1.1 OpenID Configuration Endpoint

The configuration of any GRIDS Endpoint: BAA or Data Provider can be retrieved via OpenID Configuration published by the respective Issuer. OpenID Providers supporting Discovery MUST make a JSON document available at the path formed by concatenating the string /well-known/openid-configuration to the Issuer [4].

2.1.2 Client Registration Endpoint

The Client Registration Endpoint is an OAuth 2.0 [15] Protected Resource through which a new client registration can be requested. The OP MAY require an initial access token that is provisioned out-of-band (in a manner that is out of scope for this specification) to restrict registration requests to only authorized Clients or developers [5].

2.1.3 Authorization Endpoint

The Authorisation Endpoint supports an OAuth 2.0 Authorization Request that requests that the End-User be authenticated by the Authorization Server [6].

2.1.4 Token Endpoint

To obtain an Access token, an ID Token, and optionally a Refresh Token, the RP (Client) sends a Token Request to the Token Endpoint to obtain a Token Response, when using the Authorization Code Flow [6].

2.1.5 Userinfo Endpoint

The UserInfo Endpoint is an OAuth 2.0 Protected Resource that returns Claims about the authenticated End-User. To obtain the requested Claims about the End-User, the Client makes a request to the UserInfo Endpoint using an Access token obtained through OpenID Connect Authentication [6].

In this implementation the Userinfo Endpoint will return the userinfo response with distributed claim requests towards remote Userinfo Endpoints supported by GRIDS Data Providers to request Identity Assured claims as specified in the OIDC IDA 1.0 specification [7].

2.1.6 Client Introspection Endpoint

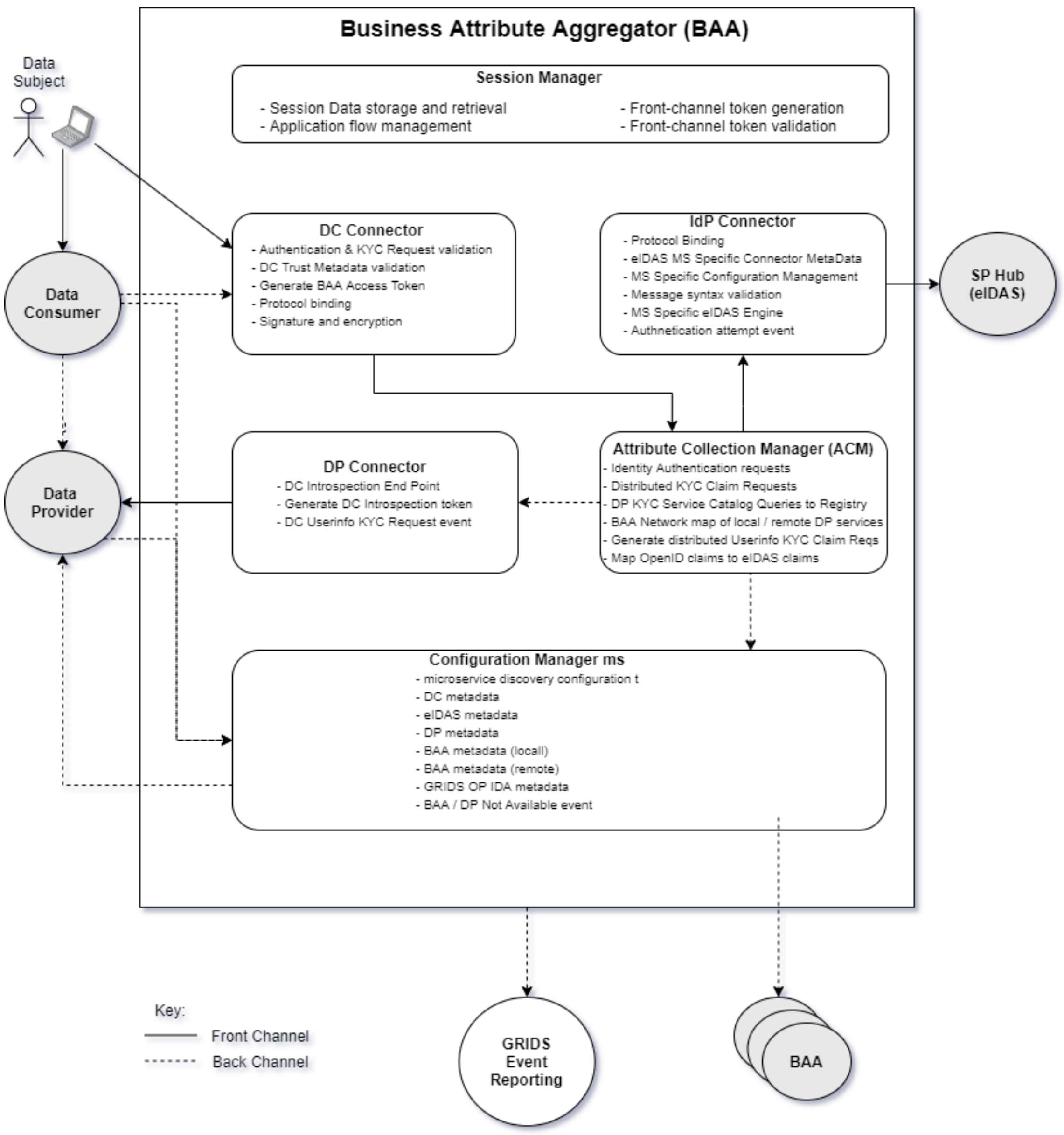
The client introspection endpoint is an OAuth 2.0 endpoint, used by the Data Providers, that queries a client introspection token and returns a JSON document representing the metadata for the Data Consumer client [8].

The client introspection endpoint and client introspection token are made available to the Data Provider (DP) in the distributed claims Userinfo request.

3 BAA Low-level design specification

The Business Attribute Aggregator is based on the existing ESMO microservices platform and inherits its architecture. GRIDS will add new modules and functionalities on the core microservice architecture design.

3.1 Logical Architecture Overview To be updated



Business Attribute Aggregator (BAA)
Data Subject
Session Manager
- Session Data storage and retrieval
- Application flow management
- Front-channel token generation
- Front-channel token validation
DC Connector
- Authentication & KYC Request validation
- DC Trust Metadata validation
- Generate BAA Access Token
- Protocol binding
- Signature and encryption
IdP Connector
- Protocol Binding
- eIDAS MS Specific Connector MetaData
- MS Specific Configuration Management
- Message syntax validation
- MS Specific eIDAS Engine
- Authnetication attempt event
SP Hub (eIDAS)
Data Consumer
Data Provider
DP Connector
- DC Introspection End Point
- Generate DC Introspection token
- DC Userinfo KYC Request event
Attribute Collection Manager (ACM)
- Identity Authentication requests
- Distributed KYC Claim Requests
- DP KYC Service Catalog Queries to Registry
- BAA Network map of local / remote DP services
- Generate distributed Userinfo KYC Claim Reqs
- Map OpenID claims to eIDAS claims
Configuration Manager ms
- microservice discovery configuration t
- DC metadata
- eIDAS metadata
- DP metadata
- BAA metadata (locall)
- BAA metadata (remote)
- GRIDS OP IDA metadata
- BAA / DP Not Available event
Key:
Front Channel
Back Channel
GRIDS Event Reporting
BAA

Figure 2 BAA Logical modular microservices architecture

3.2 Modules Specification

This section specifies the GRIDS components and details their low-level implementation details and also includes implementation notes that will serve as guides for the development of the modules.

3.2.1 Data Consumer Connector (DCC)

The Data Consumer Connector microservice module implements the OIDC authorization interface to the Data Consumer (DC), which is also known as the Relying Party (RP) in claims-based applications.

3.2.1.1 OIDC Provider Metadata with IDA Trust framework

The DCC will support a configuration end point as described in OIDC Discovery 1.0 provider config information [16]. This will make available the OIDC end points and metadata needed for Data Consumer clients communicating with the BAA.

The BAA acts as an OP and will follow the OIDC IDA 1.0 specification to advertise the KYC/KYB Claims and trust framework it supports over distributed claim sources in the GRIDS network.

The DCC will offer a Configuration Endpoint, where its OP Metadata is published based on current compilation of supported claims and trust frameworks for the totality of the DPs connected over the GRIDS Network and the locally connected IdP(s).

Implementation note:
Accessing the OP Metadata is described here.
OP Metadata to support is described here.
OIDC IDA v1.0 specification OP Metadata is described here.
The latest specification of OIDC IDA being worked on is available here.

The following non-normative example of OP well-known configuration metadata from the OIDC IDA spec [7] shows how the BAA will advertise all the verified claims with trust framework supported over the GRIDS BAA.

```
{
...
   "verified_claims_supported":true,
   "trust_frameworks_supported":[
     "nist_800_63A_ial_2",
     "nist_800_63A_ial_3"
   ],
   "evidence_supported":[
      "id_document",
      "utility_bill",
      "qes"
   ],
   "id_documents_supported":[
       "idcard",
       "passport",
       "driving_permit"
   ],
   "id_documents_verification_methods_supported":[
       "pipp",
       "sripp",
       "eid"
   ],
   "claims_in_verified_claims_supported":[
      "given_name",
      "family_name",
      "birthdate",
        "place_of_birth",
        "nationalities",
        "address"
    ],
...
}
```

The basic well-known configuration data for the BAA is created on the DCC module, as per the example in section 7.8. Additionally, this will be complemented with:

- the OP IDA verified claims metadata: This will be supported by the OP Metadata being available on the CM (see section 3.2.5.1) and includes:
    - all verified KYC/KYB claims supported by the DPs over the GRIDS BAA.
    - all verified identity claims supported by IdPs on the BAA.
- the list of trusted BAA authorization servers in the whole GRIDS network that can generate distributed claims towards the DPs and that DPs need to validate them as trusted issuers and verify their signed requests. It is included in OP Metadata claim as: "verified_claims_trusted_issuers" and is obtained from the CM query on the GRIDS IDA metadata. This will support a future scenario where there can be multiple BAAs in a GRIDS network.

Note: The DCC will obtain all the above GRIDS IDA metadata by calling the Configuration Manager module for the GRIDS using: `/metadata/externalEntities/{collectionId}` where collectionId is "GRIDS". See section 3.2.5.5 for more information.

The well-known configuration OP Metadata is periodically generated (configurable) so to publish the DP KYC/KYB verified claims that are available over the BAA.

An example of the well-known configuration OP Metadata publicly available on the BAA DCC module is given in section 7.8.

3.2.1.2 JSON Web Key Set (JWKS)

The DCC will create a JSON Web Key Set (JWKS) specific for the BAA that it resides on.

3.2.1.3 Data Consumer Registration

The DC Connector will handle all Data Consumer registrations to the BAA node. It will provide this on a managed basis, so that the DC client application first passes an approval process before being issued with an Access token to get authorized access to the GRIDS BAA registration endpoint, and be dynamically issued with a Client ID and Client Secret.

Implementation note: The BAA OpenID Provider Metadata will be obtained at the discovery endpoint for the BAA and will provide the client registration endpoint, userinfo endpoint introspection endpoint, claims supported etc. The link found here is a useful reference. The IDA Specification also will add the IDA metadata to be supported by the BAA. The following links here and here are useful references.

In order for a client to use the BAA they need to register. The DCC module of the BAA will implement OpenID Connect Dynamic Client Registration, which extends OAuth 2.0 Dynamic Client Registration Protocol and OAuth 2.0 Dynamic Client Registration Management Protocol[1].

Specifically, GRIDS BAA aims to support two modes of registration:

- admin registration (console or REST endpoints) and
- client self-registration through the Client Registration Service of the BAA OIDC module

The Client Registration Service provides built-in support for Client Representations, OpenID Connect Client Meta Data and SAML Entity Descriptors. To invoke the Client Registration Services you usually need a token. The token can be a bearer token, an initial access token or a registration access token.

Bearer-Token:

The bearer token can be issued on behalf of a user or a Service Account [19].

Initial Access token:

The recommended approach to registering new clients is by using initial access tokens. An initial access token can only be used to create clients and has a configurable expiration as well as a configurable limit on how many clients can be created. An initial access token can be created through the admin console [19].

Registration Access token:

When a new client is created through the Client Registration Service the response will include a registration access token. The registration access token provides access to retrieve the client configuration later, but also to update or delete the client. The registration access token is included with the request in the same way as a bearer token or initial access token. Registration access tokens are only valid once, when it's used the response will include a new token [19].

Implementation note:

"Upon OAuth 2.0 Client Registration the OAuth Client is assigned a Client_id and a Client Secret (password) by the Authorization Server.The Client_id and Client Secret is unique to the OAuth Client on that Authorization Server. Whenever the OAuth Client requests access to resources stored on that same Resource Server, the OAuth Client needs to Authenticate itself by sending the Client ID and the Client Secret to the Authorization Server." Reference [18].

Note that a client can use the Registration Access token it received during registration to be allowed to read its own confidential client metadata. See following reference for Registration Access token here.

Upon successful registration the Data Consumer Metadata is retrieved from the clients well known configuration endpoint and stored in a list of DC metadata objects in the DCC. This link is a useful reference.

3.2.1.4 Data Consumer Metadata

Some parameters of the Data Consumer client metadata will be made available to DPC module to support a Data Provider Introspection query (see section 3.2.3.2) when a Data Consumer with a registered client Id is making requests to a Data Provider.

To provide for this the DCC supports a `getClientMetadata` API with the following client metadata claims returned:

- `client_id`
- `client_name`
- `userinfo_signed_response_alg`
- `userinfo_encrypted_response_alg`
- `userinfo_encrypted_response_enc`

[1]https://www.keycloak.org/docs/latest/securing_apps/#_client_registration

- `jwks_uri`

3.2.1.5 Authentication and KYC/KYB Claims Request

This provides an external interface that will interact with the Data Consumers / Relying Parties to handle GRIDS eIDAS authentication and KYC/KYB Claims.

As of today, eIDAS legal person is only supported by one country in the EU (Netherlands[2]) and therefore the more common flow of authenticating a natural person will be given as example in the flow that is outlined in this section. In this scenario it is further needed for the client Data Consumer to provide the Legal Person Identifier and/or Legal Name used to query the Data Provider KYB data.

> Note:
>
> If a legal person is authenticated this will follow the same flow and thus should also be supported in the same way, with the only difference being that the legal claims will be requested over eIDAS instead of the natural claims.

The interface follows a standard OIDC Authorization Code flow, to request the eIDAS authentication and KYC/KYB Claims. This flow is described in low-level detail below so to fully specify its implementation in this module, and is further depicted in a sequence diagram with high level flow description in section 3.3.

1. The DCC supports a front-channel https redirection from the Data Consumer to the BAAs authorisation endpoint requesting identity assured authentication and KYC/KYB Claims. The DC will send either an HTTP GET or POST request that includes the following required and recommended parameters:
    - **scope**: OpenID Connect requests must contain the openid scope value.
    - **response_type**: Determines the authorization processing flow to be used. When using the Authorization Code Flow, this value is code.
    - **client_id**: Client identifier that is valid at the OpenID Connect Provider.
    - **redirect_uri**: Redirection URI to which the response will be sent. This value must exactly match one of the redirection URI values for the registered client at the OP.
    - **state**: Opaque value used to maintain state between the request and the callback.
    - **claims:** The Relying party uses this parameter, to request verified claims under userinfo and id_token as shown in the implementation note below.

    Additionally, in the case that the user is not able to authenticate against eIDAS as a legal person (but as a natural person) the DC should provide at least one of the following unverified claims for the Data Provider to be able to search the business claims:
    - **legal_person_identifier**: Legal Person Identifier
    - **legal_name:** Legal Name

    > Implementation note:
    >
    > A non-normative example of the OIDC IDA authentication and KYC/KYB Claims request is given below in the form of a JWT Token.
    >
    > Note it should be in a JWT Token as opposed to a request format. See OIDC standard section 6 and also this reference.
    >
    > ```
    > {
    > ```

[2] https://ec.europa.eu/cefdigital/wiki/pages/viewpage.action?pageId=74091935

```
   "iss": "https://data_consumer.example.com",
   "aud": "https://BAA.example.com",
   "response_type": "code",
   "client_id": "s6BhdRkqt3",
   "redirect_uri": "https://data_consumer.example.oom/cb",
   "scope": "openid",
   "state": "af0ifjsldkj",
   "jti": "n-0S6_WzA2Mj",
   "max_age": 86400,
   "legal_person_identifier": "375714X",
   "legal_name": "Acme Corporation",
        "claims": {
                 "userinfo": {
                          "verified_claims": {
                                  "verification": {
                                           "trust_framework": {
                                             "value": "grids_kyb"
                                           },
                                           "userinfo_endpoint": {
                                                   "value":
"www.entiyid.com"
                                           },
                                           "evidence": [
                                            {
                                             "type": {
                                                     "value":
"company_register"
                                             },
                                             "registry": {
                                                   "organisation": {
                                                     "essential": false,
                                                     "purpose": "string"
                                                   },
                                                   "country": {
                                                     "essential": true,
                                                     "purpose": "string",
                                                     "value": "ES"
                                                   }
                                             },
                                             "time": {
                                                   "max_age": 31000000,
                                                   "essential": true,
                                                   "purpose": "string"
                                             },
                                             "data": {
                                                   "essential": true,
                                                   "purpose": "string"
                                             },
                                             "extractURL": {
                                                   "essential": true,
                                                   "purpose": "string"
                                             },
                                             "document": {
                                                   "SKU": {
                                                     "value": "REX"
                                                      },
                                                   "option": {
                                                     "essential": false,
                                                     "purpose": "string"
                                                   }
                                             }
                                            }
                                           ]
                                  },
                                  "claims": {
                                          "family_name": null,
                                          "given_name": null,
                                          "birthdate": null,
                                          "legal_name": null,
                                          "legal_person_identifier": null,
                                          "lei": null,
                                          "vat_registration": null,
                                          "address": null,
                                          "tax_reference": null,
                                          "sic": null,
                                          "business_role": null,
                                          "sub_jurisdiction": null,
                                          "trading_status": null
                                  }
                          }
              },

                   "id_token":
                          "verified_claims": {
                                  "verification": {
                                          "trust_framework": {
                                            "value": "eidas"
                                          },
                                  },
                                  "claims": {
                                          "family_name": null,
                                          "given_name": null,
                                          "birthdate": null,
                                          "person_identifier": null,
                                          "place_of_birth": null,
                                          "address": null,
                                          "gender": null
                                  }
                          }
       }
     }
```

2. The received id token request for eIDAS initiates the DCC to analyse the requested verified claims and redirect the user to the ACM with an SP Request and then to the Identity Provider Connector to proceed to authenticate the user over eIDAS (via the SPHUB) for the requested eIDAS claims.

Implementation note:

A non-normative example of the SP Request is given below.

```
{
   "issuer" : "https://dataConsumer.example1.com",
   "type" : "Request",
   "recipient" : "https://baa.example1.com/dcc",
   "id" : "6c0f70a8-f32b-4535-b5f6-0d596c52813a",

   "attributes" : [
       {
           "friendlyName":"given_name,
           "isMandatory":true
       },
       {
           "friendlyName":"family_name",
           "isMandatory":true
       },
       {
           "friendlyName":"person_identifier",
           "isMandatory":true
       },
       {
           "friendlyName":"birthdate",
           "isMandatory":true
       },
       {
           "name":"address",
           "isMandatory":false
       }
   ]
   },

   "spMetadata" : {
       "entityId" : "https://dataConsumer.example1.com",
       "defaultDisplayName" : "Data Consumer Example"
       "location" : "ES|Spain",
       "protocol" : "OIDC",
       "microservice" : ["DCCms001"],
       "endpoints" : [
           {
               "type":"AssertionConsumerService",
               "method":"HTTP-POST",
               "url":"https://dataConsumer.example1.com/acs.php"
           }
       ],
       "securityKeys" : [
           {
               "keyType":"RSAPublicKey",
               "usage":"signing",
               "key":"MDAACaFgw...xFgy="
           },
           {
               "keyType":"RSAPublicKey",
               "usage":"encryption",
               "key":"MDAACaFgw...xFgy="
           }
       ],
       "encryptResponses" : false,
       "supportedEncryptionAlg" : ["AES256","AES512"],
       "signResponses": true,
       "supportedSigningAlg" : ["RSA-SHA256"]
   }

}
```

3. Once authenticated the DCC will return the authorisation code to the user agent which the DC will use to request the access token and id token.

Implementation note:

A non-normative example of the Authorisation Code HTTP response is given below.

```
HTTP/1.1 302 Found
Location:
https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA&state=xyz
```

4. The DC will now make a request to the BAA token endpoint with the authorisation code which will return the user's identity token, and also a self-describing access token to the DC. It is the DCC that will be responsible for building the identity token and access token.

Implementation note:

A non-normative example of the request to the token endpoint is given below.

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded


grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA&redirect_uri=http
s%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

A non-normative example of the response from the token endpoint is given below with a signed and base64Url encoded Id Token and also access token. The access token will be similarly signed for self-describing access tokens.

```
  HTTP/1.1 200 OK
  Content-Type: application/json
  Cache-Control: no-store
  Pragma: no-cache

  {
   "access_token": "SlAV32hkKG.yyuyyfyu6.vjh75WEdf",
   "token_type": "Bearer",
   "refresh_token": "8xLOxBtZp8",
   "expires_in": 3600,
   "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc
     yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5
     NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ
     fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz
     AKfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6q
     Jp6IcmD3HP99Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ
     NqeGpe-gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd
     QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoS
     K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4
     XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"
  }
```

5. Next the DC will query the BAAs Userinfo endpoint with the self-describing access token on the BAA (on the DCC) for the requested KYC/KYB Claims (received in the previous step with the eIDAS requested claims).

Implementation note:

A non-normative example of a signed and encrypted userinfo request with self-describing bearer access token to the BAA userinfo endpoint is given below.

```
  GET /userinfo HTTP/1.1
  Host: server.example.com
```

```
   Authorization:  Bearer ZyJhbGciOicifQ.ewogImlI1wbGUwKiOiAiMjQ4.NzYxMDA
```

6. Upon reception of a back-channel UserInfo request with self-describing Access token to the BAAs Token endpoint, the DCC decodes it and sends the "verified_claims" in the request (userinfoReq) to the ACM module to obtain the distributed KYC DP Source endpoints and corresponding self-describing access tokens, that will satisfy the KYC verified claims requested by the DC.

Implementation note:

A non-normative example of the received Access token on the BAAs userinfo endpoint is given below.

```
{
"jti":"knm34l45jl45l",
"client_id":"https://DC.example.com",
"legal_person_identifier": "375714X",
"legal_name": "Acme Corporation",
"verified_claims": {
        "verification": {
                "trust_framework":"eidas",
                "identity_assurance_level": "substantial",
                "time": "2012-04-22T11:30Z"
                },
        "claims": {
                "given_name": "John",
                "family_name": "Smith",
                "birthdate": "1971-04-17",
                "person_identifier": "X731Z219A",
                "address": {
                        "country": null,
                        "street_address": "George Street 123",
                        "locality": "Glasgow",
                        "postal_code": "G1 1QD"

                }
        }
},
"claims": {
        "userinfo": {
                "verified_claims":
                {
                        "verification": {
                                "trust_framework": {
                                  "value": "grids_kyb"
                                },
                                "userinfo_endpoint":      {
                                        "value": "www.entiyid.com"
                                },
                                "evidence": [
                                 {
                                  "type": {
                                        "value": "company_register"
                                  },
                                  "registry": {
                                        "organisation": {
                                          "essential": false,
                                          "purpose": "string"
                                        },
                                        "country": {
                                          "essential": true,
                                          "purpose": "string",
                                          "value": "ES"
                                  }
                                  },
                                  "time": {
                                        "max_age": 0,
                                        "essential": true,
                                        "purpose": "string"
                                  },
                                  "data": {
                                        "essential": true,
                                        "purpose": "string"
                                  },
                                  "extractURL": {
                                        "essential": true,
                                        "purpose": "string"
                                  },
                                  "document": {
                                        "SKU": {
                                          "value": "REX"
                                        },
                                        "option": {
                                          "essential": false,
                                          "purpose": "string"
                                        }
                                  }
                                 }]
                        },
                        "claims": {
                                "family_name": null,
                                "given_name": null,
                                "birthdate": null,
                                "legal_name": null,
                                "legal_person_identifier": null,
                                "lei": null,
                                "vat_registration": null,
                                "address": null,
                                "tax_reference": null,
                                "sic": null,
                                "business_role": null,
                                "sub_jurisdiction": null,
                                "trading_status": null
                        }
                }
 }
}
}
```

7. The self-describing UserInfo access tokens are returned to the DCC as JWTs encoded in Base64url by the ACM, and as per the security requirements in section 3.7 the DCC will both sign and encrypt the received Base64url encoded access token, with the signing performed first and then the encryption as recommended in section 11.2 of RFC7519. The encryption key to use will also be provided in the userinfoReq response from the ACM.

Implementation note:

A non-normative example of the userinfoReq response from the ACM to the DCC is given is section 3.2.4.2.

8. The DCC will then return the Userinfo response to the DC which will continue the query to the DP KYC Sources directly for the user's verified claims.

Implementation note:

A non-normative example of the userinfo access token received at the Data Provider is given below.

```
{
"iss":"https://BAA.1.example.com",
"sub":"X731Z219A",
"aud":"https://DP.anotherexample.com",
"scope":"openid",
"exp":"1311281970",
"iat":"1311280970",
"jti":"knm34l45jl45l",
"txn": "1234756541114219465050879501l123",
"client_id":"https://DC.example.com",
"client_introspection_uri":"https://GRIDS.BAA1234.com/clients?id=https://DC
.example.com",
"client_introspection_access_token":"78y98yy98hyubui989y898y98yh8h8y7",
"legal_person_identifier": "375714X",
"legal_name": "Acme Corporation",
"verified_claims": {
        "verification": {
                "trust_framework":"eidas",
                "identity_assurance_level": "substantial",
                "time": "2012-04-22T11:30Z"
                },
        "claims": {
                "given_name": "John",
                "family_name": "Smith",
                "birthdate": "1971-04-17",
                "person_identifier": "X731Z219A",
                "address": {
                        "country": null,
                        "street_address": "George Street 123",
                        "locality": "Glasgow",
                        "postal_code": "G1 1QD"

                }
        }
 },
"claims": {
        "userinfo": {
                "verified_claims": {
                        "verification": {
                                "trust_framework": {
                                  "value": "grids_kyb"
                                },
                                "evidence": [
                                 {
                                  "type": {
                                          "value": "company_register"
                                  },
                                  "registry": {
                                        "organisation": {
                                          "essential": false,
                                          "purpose": "string"
                                        },
                                        "country": {
                                          "essential": true,
                                          "purpose": "string",
                                          "value": "ES"
                                        }
```

```
                                    },
                                    "time": {
                                          "max_age": 0,
                                          "essential": true,
                                          "purpose": "string"
                                    },
                                    "data": {
                                          "essential": true,
                                          "purpose": "string"
                                    },
                                    "extractURL": {
                                          "essential": true,
                                          "purpose": "string"
                                    },
                                    "document": {
                                          "SKU": {
                                            "value": "REX"
                                          },
                                          "option": {
                                            "essential": false,
                                            "purpose": "string"
                                          }
                                    }
                                   }
                                  ]
                         },
                         "claims": {
                                  "given_name": null,
                                  "family_name": null,
                                  "birthdate": null,
                                  "legal_name": null,
                                  "legal_person_identifier": null,
                                  "lei": null,
                                  "vat_registration": null,
                                  "address": null,
                                  "tax_reference": null,
                                  "sic": null,
                                  "business_role": null,
                                  "sub_jurisdiction": null,
                                  "trading_status": null
                         }
                 }
   }
}
}
```

Additionally, it is possible that a DC has already authenticated a user and performed KYC queries to distributed userinfo endpoints and proceeds to request to the BAA to do more KYC queries with identity claims obtained from the initial DP KYC sources. In this case, the above process is repeated, but the DC can specify the "prompt" parameter set to "none" in the new authentication and claims request, and the user will not have to reauthenticate as long as the session is found on the BAA not expired.

For more detailed information on the flow and the interaction with the DPC module please see section: 3.3.

Implementation note:

The existing ESMO GW is used for the basis of the GRIDS microservices platform and the OIDC SP microservice will be used as the base code for the DCC. The spRequest will be used to handle the eIDAS authentication request with set of claims included in the attributeSet object and the verification properties, as specified in the OIDC IDA 1.0 specification [7], will be included inside the properties object.

Implementation note:

Distributed access token generated by the BAA will include not just the userinfo claims being requested to the DP but also the identity claims with the values obtained by the eIDAS authentication. As the access token may therefore contain personal identity claims, it should be considered to be encrypted as agreed with the DP it is destined for. This security hardening feature will be considered in section 3.7.

This link is a useful reference.

Implementation note:

The self-describing distributed Access token is also used by the ID4me architecture, and GRIDS should learn from their implementation.

Useful ID4ME references given here, here and here.

Implementation note:

This link is a useful reference for building an Id Token.

3.2.2 Identity Provider Connector (IdPC)

3.2.2.1 Registration

The IdPC will manually configure the eIDAS metadata manually in the Configuration Manager. See a reference example of eIDAS Metadata in section 7.5.

3.2.2.2 Authentication

The IdPC provides an external interface that will interact with sources of authentication. BAA deployments will connect identity providers as required for each deployment e.g. eIDAS, national IdPs etc. As the IdPs require the user to respond to security challenges, they will all be user-centred and front-channel.

The IdP Connector module also implements a generic internal interface to the Session Manager and Attribute Collection Manager modules respectively for receiving Authentication requests and returning responses to the corresponding modules of the BAA.

3.2.2.3 IdPC Events

When an authentication is attempted it is reported the success or failure of the event and capturing non personal data such as success or fail and country authenticated.

The IdPC will forward the event to the platforms Centralised Event Logging & Reporting as described in the high-level design.

3.2.3 Data Provider Connector (DPC)

3.2.3.1 Registration

DPs apply to register out of band and supply a configuration end point to a GRIDS BAA operator to become part of the GRIDS Network. This process is handled in the configuration Manager described in section 3.2.5.1.

Any DPC instance will be able to support any Data Provider registered with the BAA.

The Data Provider will in turn query the BAAs well known OIDC configuration endpoint to obtain its list of trusted BAA issuers (verified_claims_trusted_issuers) in the GRIDs network, so to be able to validate Userinfo requests issued by any of the entities in the trusted entity list. The DP will access the JWKS URI of the trusted BAA issuer to verify the signature of the userinfo request it received.

3.2.3.2 Data Consumer Client Introspection

The BAA will support the DPC primarily to offer a Data Consumer client introspection end point where the Data Provider can receive information on the Data Consumer so to confirm how it responds with the requested KYC/KYB Claims, as per specified signing and encryption algorithms and access to the DCs public key.

When the ACM previously created the distributed request for the Data Provider it made a generateClientIntrospectionAccessToken request to the DPC module for a specific client Id & transaction Id (TXN) so to restrict the access to the Data Consumer Client Introspection endpoint and obtain an authorisation token to query it. The DPC saves the Introspection access token with the client Id and TXN.

When the DP later queries the DC Client Introspection endpoint it will include the token in the body. The token is burned upon use, after a configurable timer times-out.

Implementation note: A non-normative example of a DP query and response to the client introspection point with the supplied URI is shown below.

```
POST /http://localhost:8080/dpc/dcIntrospection
Host: baa.example1-server.com
Authorization: Bearer vF9dft4qmT
Content-Type: application/json
"client_id": " f87674bhb ",
```

A non-normative response is given below:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "client_id": " f87674bhb ",
    "client_name": "Data Consumer XYZ",
    "userinfo_signed_response_alg": "RS256",
    "userinfo_encrypted_response_alg": "RSA-OAEP-256",
    "userinfo_encrypted_response_enc": "A128CBC-HS256",
    "jwks_url": "https://dataconsumer.xyz.com/jwks.json"
}
```

Note: The DPC obtains the DC Client claims from the DCC microservice using the `getClientMetadata` API (see section 3.2.1.4). To obtain the DCC microservice routing information for this API, the DPC will contact the Configuration Manager to obtain the Data Consumer Connector endpoint to which to forward this metadata request.

3.2.3.3 DP Events

When a DP queries the DC Client Introspection endpoint, the DPC generates an event reporting the DC Client Introspection took place on the BAA for a specific Client Id and TXN Transaction Id. The TXN was associated with the Introspection Access token when it was generated in the DPC.

The DPC will forward the event to the platforms Centralised Event Logging & Reporting which is described in the high-level design.

3.2.4 Attribute Collection Manager (ACM)

3.2.4.1 Authentication + KYC Request handling

The role of the ACM module in conjunction with the Session Manager module is to provide a generic internal interface to the DC, IdP, DP Connector modules. In this way, KYC Requests received from the DC in any protocol are able to be redirected to the ACM over a generic interface for processing requests from DCs and routing them to locally connected IdPs for requested authentication claims and to DPs for KYC/KYB Claims.

As previously described in the high level design, the GRIDS BAA supports the OIDC IDA v1.0 specification [3] and is designed to support both aggregated and distributed claims approaches. However, for this version of GRIDS BAA only the latter distributed approach is implemented.

As illustrated by the sequence diagram in section 3.3 the ACM supports authentication for the eIDAS claims requested towards eIDAS and returning the authenticated claims with the required eIDAS trust framework in the attribute set of the SP Response.

The KYC/KYB Claims requested are supported as per the OIDC IDA 1.0 specification through distributed claims and described in the next section.

OpenID support standard claim names (see section 5.1 of the OIDC Core specification [6]) and some of these are not directly equivalent to eIDAS claims supported in GRIDS and therefore the ACM must do some mapping of the OpenID claim names to eIDAS friendly names as follows (Table 1):

OpenID / GRIDS	**eIDAS**	**Format (Any Difference?)**
given_name	FirstName	string
family_name	FamilyName	string

address	CurrentAddress	OpenID specification is found here.
		
		It is needed to map between this and eIDAS specification. An example of OpenID address is:
		
		"address":{
		  "locality":"Maxstadt",
		  "postal_code":"12344",
		  "country":"DE",

		"street_address":"Anderson Avenue 22" } The corresponding example in eIDAS is as follows: “CurrentAddress”: ["LocatorDesignator", "Thoroughfare", "PostName", "PostCode"] “CurrentAddress”: ["22", " Anderson Avenue ", " Maxstadt ", "12344"]
birthdate	DateOfBirth	string ISO 8601:2004 [ISO8601-2004] YYYY-MM-DD
gender	Gender	string
lei	LEI	string
birth_family_name	BirthName	string eIDAS gives both given and family names and so is not compatible with OpenID
birth_given_name	BirthName	string eIDAS gives both given and family names and so is not compatible with OpenID
place_of_birth	PlaceOfBirth	json object of type strings vs eIDAS string eIDAS will only return place of birth e.g. “London” and will add this as a string inside the place_of_birth object.
person_identifier	PersonIdentifier	string
legal_person_identifier	LegalPersonIdentifier	string
legal_name	LegalName	string
sic	SIC	string
lei	LEI	string
vat_registration	VATRegistration	string

Table 1 OIDC / GRIDS Claims to eIDAS Attributes mapping

Implementation note:

eIDAS friendly names and format are specified here.

3.2.4.2 Userinfo Request handling

The ACM supports the end userinfo request to the DCC by receiving the forwarded request on `userinfoRequest` API with a decoded self-describing Access token including the verified claims being requested and the previously authenticated eIDAS attributes as well as the requested trust framework.

If the actual DP KYC Source(s) have been specified in the trust framework[3] (using the `"userinfo_endpoint"` claim) it will be used to generate the self-describing Access tokens to each of the requested distributed DP KYC Sources.

Otherwise, the ACM will return, the distributed claims for all DP KYC Sources that match or partially match the requested claims as per the identity assurance being requested by the Data Consumer. This is actually currently an issue[4] raised by GRIDS and is being addressed by the OIDC IDA WG in the next release of the specification. The example given in the implementation note below follows the proposed update to the OIDC IDA 1.0 specification.

The DP metadata and verified claims that are supported for all DPs is obtained by querying the configuration manager.

In the distributed claims approach, there is no direct communication between the DP module and the ACM. Once the ACM module has prepared the distributed claims it will create the distributed self-describing tokens with corresponding DP userinfo endpoints and return this information in the response to the userinfo request and encoded in Base64url.

The ACM also interfaces with the Session Manager module so to access and update the session and attribute information objects which are stored in the one secure Session Manager module.

Implementation note:

ACM will process the received verified claims in the UserInfo access token (see example in section 3.2.1.5) as per the IDA Specification to identify the available DP KYC Sources that meeting the requested verification request.

The Standard specification of the IDA Verified Claims Schema used as reference is specified here.

A non-normative example of the response to the `userinforeq` API from the ACM to the DCC with distributed userinfo claims is given below:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
  "_claim_names": {
    "verified_claims":
      {
      "src1":  {
        "verified_claims_available": {
            "verification": {
                "trust_framework": {
                  "value": "grids_kyb"
                },
                "evidence": [
                  {
                   "type": {
                      "value": "company_register"
                   },
                   "registry": {
                      "country": {
                        "value": "ES"
                      }
                    },
```

[3] Note to specify the actual KYC Sources inside a KYC request is a GRIDS specific extension of the OIDC IDA v1.0 Specification.
[4] https://bitbucket.org/openid/ekyc-ida/issues/1242/look-ahead-for-provided-claims-in

```
                 "document": {
                    "SKU": {
                      "value": "REX"
                    }
                  }
               }
             ]
          },
          "claims": {
              "legal_name": null,
              "legal_person_identifier": null,
              "lei": null,
              "vat_registration": null,
              "address": null,
              "tax_reference": null,
              "sic": null,
              "business_role": null,
              "sub_jurisdiction": null,
              "trading_status": null
          }
        }
   }
  },
   {
    "src2":  {
      "verified_claims_available": {
          "verification": {
              "trust_framework": {
                "value": "grids_kyb"
              },
              "evidence": [
                {
                 "type": {
                    "value": "company_register"
                  },
                  "registry": {
                    "country": {
                      "value": "ES"
                     }
                   },
                 "document": {
                    "SKU": {
                      "value": "REX"
                    }
                  }
               }
             ]
          },
          "claims": {
              "legal_name": null,
              "legal_person_identifier": null,
              "sic": null,
              "business_role": null,
              "sub_jurisdiction": null,
              "trading_status": null
          }
        }
},
"_claim_sources": {
  "src1": {
      "endpoint": "https://data_provider1.example.com/claim_source",
      "access_token": "eyJhbGciOIkpXVCJ9.eyJpcI6ImU4MTQ4NjAzNDI0N.S04MjViLWMxMDhiOGI2",
      "Key": {
        "kty": "EC",
        "crv": "P-256",
        "x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
        "y": "4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
        "use": "enc",
        "kid": "1"
       }
  },
  "src2": {
        "endpoint": "https://data_provider2.example2.com/claim_source",
        "access_token": "Btui76787pXVCJ9.zxc45ReyJpcI6zLTg5MzQtNDI0N.ViLWMxMDhi323",
        "Key": {
          "kty": "EC",
          "crv": "P-256",
          "x": "FFKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
          "y": "GGtl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
          "use": "enc",
          "kid": "1"
         }
    }
  }
}
```

Note: Access token is returned in userinforeq response to DCC in Base64url

3.2.4.3 ACM events

When the ACM creates and responds to DC KYC Requests with Access tokens, the DC client Id and DP Userinfo endpoint and TXN are captured in the event for the requested claims.

The ACM will forward the event to the platforms Centralised Event Logging & Reporting.

3.2.5 Configuration Manager (CM)

This module will be responsible for the configuration and provisioning of all functional modules in the BAA concerning e.g. configuration of optional features and provisioning of trusted entity metadata.

It is inherited from the ESMO project and is adapted with the description in the following sub-sections.

The CM holds configuration metadata on all external entities in the BAA and a complete list of what is used in GRIDS is given below:

- **EIDAS:** eIDAS IDP Metadata populated in the local BAA for redirecting authentication requests towards eIDAS.
- **DP List:** Array of DPs well known configuration endpoints that are provisioned for that local BAA
- **BAA (Local):** BAA metadata containing all locally supported DP OP IDA Metadata as read from the DPs well known configuration endpoints
- **BAA List:** Array of GRIDS BAA metadata endpoints to obtain the OP IDA metadata supported by all remote BAAs in a GRIDS network (future support).
- **BAA (External):** Array of all supported remote BAA metadata containing all locally supported DP OP IDA Metadata (future support)
- **GRIDS:** BAA metadata including all supported DP OP Metadata and list of trusted BAAs in the GRIDS network

3.2.5.1 Trusted DP list

Data Providers that want to register their KYC Sources to the BAA Network need only supply their configuration endpoint. No other trust relationship is needed as it is enough for a Data Provider to request the BAA to read its metadata and include it in the BAA trusted KYC DPs.

Data Provider configuration endpoint url will be configured manually in the Configuration Manager and is used to retrieve the DP Metadata and store it in the local BAA Metadata list.

An example of the Trusted DP list is given in section 7.4.

3.2.5.2 Local BAA Metadata

The local BAA Metadata is periodically generated and primarily includes its entity identifier (issuer address), a summary of all DP Metadata for all DPs configured to that BAA, the individual DPs and their IDA supported and the time it was last successfully read.

The CM will periodically query the Data Provider well-known configuration end points that are populated in the Trusted DP List as described in section 3.2.5.1 (set by deployment variable). This will read the DP Metadata and obtain the KYC/KYB Claims and trust framework it supports along with the encryption and signing algorithms the DP supports. An example of the well-known DP Metadata read by the CM is given in section 7.9.

Additionally, the CM will query the JWKS uri and append the DP´s supported public keys to the DP Metadata.

Implementation note:

The BAA will support encoding of the userinfo to the DP according to the first public encryption JWK read from the DP JWKS and added to the DP metadata by the CM (with RSA and EC keys supported).

The reading of the JWKS will follow https://tools.ietf.org/html/rfc7517 with non-normative examples given below.

```
{"keys":
  [
    {"kty":"EC",
     "crv":"P-256",
     "x":"MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
     "y":"4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
     "use":"enc",
     "kid":"1"},

    {"kty":"RSA",
     "n": "0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx
4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebWKRXjBZCiFV4n3oknjhMs
tn64tZ_2W-5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-65YGjQR0_FDW2
QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91CbOpbI
SD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-G_xBniIqb
w0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",
     "e":"AQAB",
     "alg":"RS256",
     "use":"enc",
     "kid":"2011-04-29"}
  ]
}
```

This information obtained for each DP, is added to a local BAA metadata file containing the IDA metadata for all DPs configured in that BAA along with the time it was successfully retrieved ("successfullyQueried") and the public encryption key of the DP.

Therefore, the list of local DPs and the KYC Source metadata, claims and trust framework they support are able to be queried by the ACM module as part of the process in determining the DP to request claims from using API `/metadata/internal/{confId}` with confId equal to "BAA".

A reference example is given in section 7.1.

3.2.5.3 Trusted BAA List

GRIDS supports the possibility that there is a GRIDS Network of multiple BAAs, so that DPs registered with one DP are also visible by all DPs in the GRIDS Network.

To support this, each BAA makes available the local BAA Metadata on a publicly available endpoint so to be read by each BAA, as it is registered in the Trusted BAA List.

A reference example is given in section 7.3.

Note: This feature is desirable but not mandatory for the project.

3.2.5.4 External BAA Metadata

To obtain the OP IDA Metadata supported by all other BAAs in the GRIDS network the CM will query the BAAs in the trusted BAA list as per section 3.2.5.3, and add each into an external BAA json file, as per the example given in section 7.2 .

Therefore, the list of remote DPs and the KYC Source metadata, claims and trust framework they support are able to be queried by the ACM module as part of the process in determining the DP to request claims from using API `/metadata/externalEntities/{collectionId}` with collectionId equal to "BAA".

Note: This feature is desirable but not mandatory for the project.

3.2.5.5 GRIDS Metadata

The GRIDS Metadata file is dynamically generated in the CM from the local and remote BAA JSON files to include:

- GRIDS entity identifier
- Summary of all DP Metadata for all DPs registered to all BAAs in the GRIDS Network
- Individual DPs and their IDA supported (locally and remotely) and the time it was last successfully read.

The GRIDS Metadata will thus compile the trusted BAA nodes in the entire GRIDS network and all verified claims supported over the GRIDS network as required by the OP Metadata.

Additionally, GRIDS extends the OIDC IDA specification by including in the OP Metadata the verified claims (KYC/KYB Claims and Trust framework) supported by each DP KYC source available over GRIDS and this is also included in the GRIDS Metadata file.

The GRIDS Metadata file is primarily read by the DCC module to add the additional OP Metadata to include the verified claims available over the GRIDS Network.

An example of the GRIDS Metadata is given in section 7.6.

3.2.5.6 eIDAS Metadata

The eIDAS Metadata is configured on the CM so to be able to be read by the ACM to be able to identify the sources that support requested eIDAS claims and can re-direct the authentication request to the IdP module that supports it.

An example of the EIDAS Metadata is given in section 7.5.

3.2.5.7 CM Events

When a CM is making a metadata query to an entity such as another BAA or to a DP and it does not receive a response an event is raised to the platforms Centralised Event Logging & Reporting.

3.2.6 Session Manager

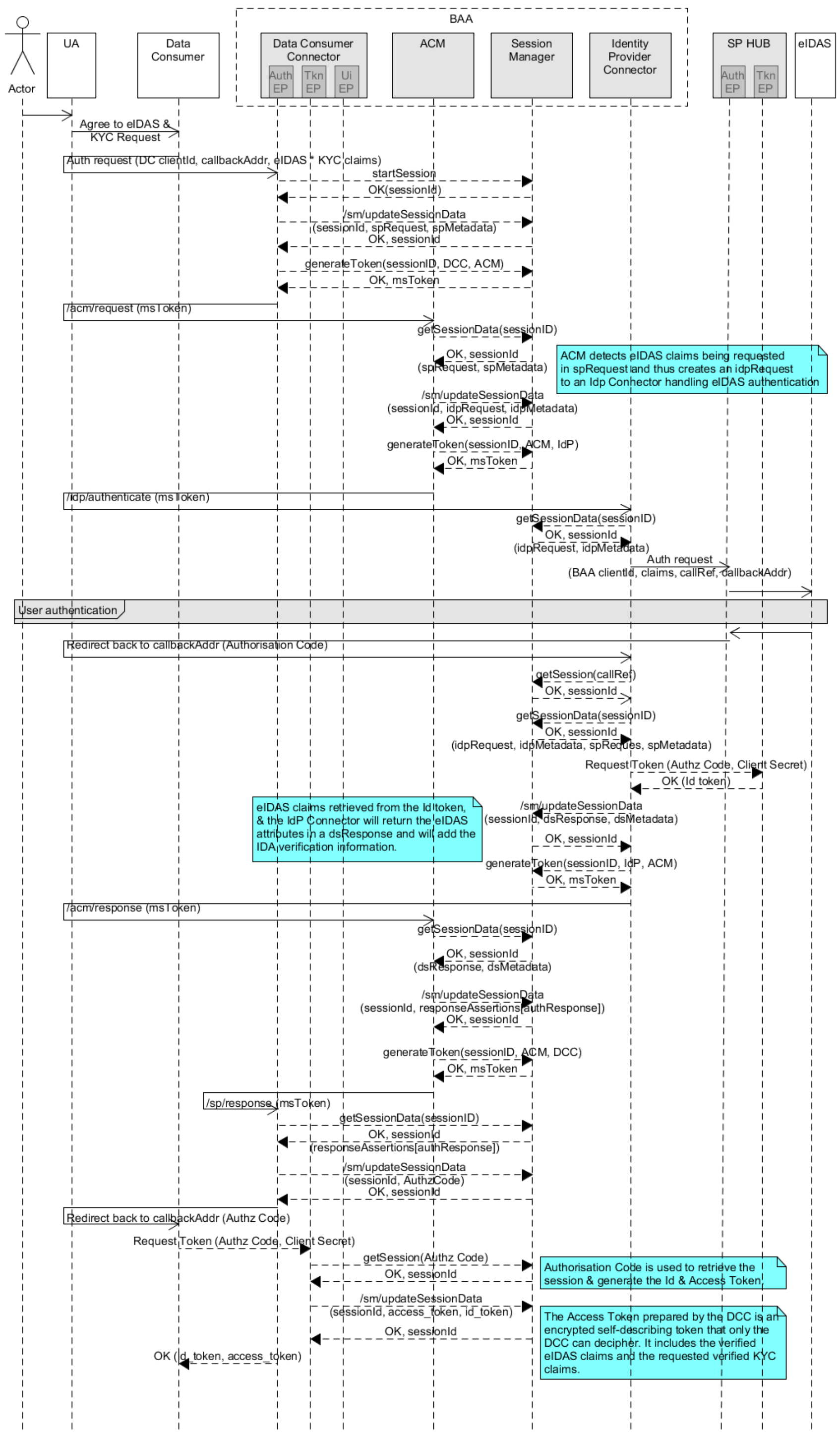
This is inherited from ESMO without any changes.

This provides a generic internal interface to all modules in the BAA for caching the Authentication and KYC Request information and returning a session token to the modules, which in turn can be used to later access and manipulate the session data. The request information, metadata, and session state are cached until its associated response is received, or a time out occurs. The associated response information is matched with the cached request and is also cached in the Session Manager module until delivered to the requesting DC module, or a time out occurs.

3.3 Sequence diagrams

This section provides a sequence diagram and description demonstrating the OIDC Authorization and Distributed Claims flow following the OIDC IDA 1.0 specification.

3.3.1 BAA Authentication and Verified KYC/KYB Claims request


BAA
Actor
UA
Data Consumer
Data Consumer Connector
Auth EP
Tkn EP
Ui EP
ACM
Session Manager
Identity Provider Connector
SP HUB
Auth EP
Tkn EP
eIDAS
Agree to eIDAS & KYC Request
Auth request (DC clientId, callbackAddr, eIDAS + KYC claims)
startSession
OK(sessionId)
/sm/updateSessionData (sessionId, spRequest, spMetadata)
OK, sessionId
generateToken(sessionID, DCC, ACM)
OK, msToken
/acm/request (msToken)
getSessionData(sessionID)
OK, sessionId (spRequest, spMetadata)
/sm/updateSessionData (sessionId, idpRequest, idpMetadata)
OK, sessionId
generateToken(sessionID, ACM, IdP)
OK, msToken
/idp/authenticate (msToken)
getSessionData(sessionID)
OK, sessionId (idpRequest, idpMetadata)
Auth request (BAA clientId, claims, callRef, callbackAddr)
User authentication
Redirect back to callbackAddr (Authorisation Code)
getSession(callRef)
OK, sessionId
getSessionData(sessionID)
OK, sessionId (idpRequest, idpMetadata, spReqdes, spMetadata)
Request Token (Authz Code, Client Secret)
OK (Id token)
/sm/updateSessionData (sessionId, dsResponse, dsMetadata)
OK, sessionId
generateToken(sessionID, IdP, ACM)
OK, msToken
/acm/response (msToken)
getSessionData(sessionID)
OK, sessionId (dsResponse, dsMetadata)
/sm/updateSessionData (sessionId, responseAssertions[authResponse])
OK, sessionId
generateToken(sessionID, ACM, DCC)
OK, msToken
/sp/response (msToken)
getSessionData(sessionID)
OK, sessionId (responseAssertions[authResponse])
/sm/updateSessionData (sessionId, AuthzCode)
OK, sessionId
Redirect back to callbackAddr (Authz Code)
Request Token (Authz Code, Client Secret)
getSession(Authz Code)
OK, sessionId
/sm/updateSessionData (sessionId, access_token, id_token)
OK, sessionId
OK (id_token, access_token)

ACM detects eIDAS claims being requested in spRequest and thus creates an idpRequest to an Idp Connector handling eIDAS authentication

eIDAS claims retrieved from the Id token, & the IdP Connector will return the eIDAS attributes in a dsResponse and will add the IDA verification information.

Authorisation Code is used to retrieve the session & generate the Id & Access Token

The Access Token prepared by the DCC is an encrypted self-describing token that only the DCC can decipher. It includes the verified eIDAS claims and the requested verified KYC claims.

Figure 3 BAA Authentication and Verified KYC/KYB Claims request

The flow in the above sequence diagram (Figure 3) is described in detail below:

1) User authentication request is received at the BAA DCC from the DC with identity-token and userinfo claims request.

   > Implementation Note
   >
   > The authentication request should be in a JWT Token as opposed to a Request format: see OIDC standard section 6.

2) The DCC stores the user-info claims in a session variable, and proceeds to generate an SP Request message based on the received identity-token claims and also captures the SP Metadata. The DCC proceeds to forward the SP Request to the ACM microservice to handle the request of these identity claims on behalf of the DC.
3) Analysis of the Identity-token claims request in the ACM determines that the attributes being requested are for eIDAS (requested trust_framework indicates e.g. "eidas_ial_substantial" or if not specified the requested claims are resolved to eIDAS), and the ACM routes these to the IdP Connecter microservice that handles eIDAS authentication (based on configuration data). If the trust framework or claims are not recognised an error response is returned.
4) This results in redirecting the user to the eIDAS network to perform authentication at their national IdP, and the user's authenticated attributes are returned to the IdP Connector in the BAA, based on the call-back address previously provided by the IdP Connector.
5) The IdP Connector next returns the Data Source response to the ACM in an attributeSet object.
6) The ACM returns the response to the DCC module, which is handling the request on behalf of the DC, and proceeds to generate an authorisation code for the session and returns it to the DC Relying Party.
7) The DC queries the DCC token endpoint on the BAA, with the authorisation code and client secret for the retrieval of the identity-token and access token. The DCC generates:
   a. an identity-token from the authenticationSet stored in the session with the previously eIDAS authenticated attributes,
   b. an encrypted and signed (nested JWE[5] JWS) self-describing access token, based on the userinfo claims that were previously stored for the session (in the initial authentication request). The claims included in the token are the eIDAS claims, with the identity of the authenticated user, as well as the requested KYC/KYB Claims including IDA trust framework.

   The DC can now close the session as the access token will now hold all the information for it to process the subsequent userinfo request from the DCC.
8) The DC receives both the Identity-token and access token. This is considered in the next section 3.3.2.

[5] In the initial development just a JWT Access Token will be generated for test and debug, and then it will be added later the signing and encryption with nested JWS/JWE.

3.3.2 BAA KYC Source Distributed Claims Flow

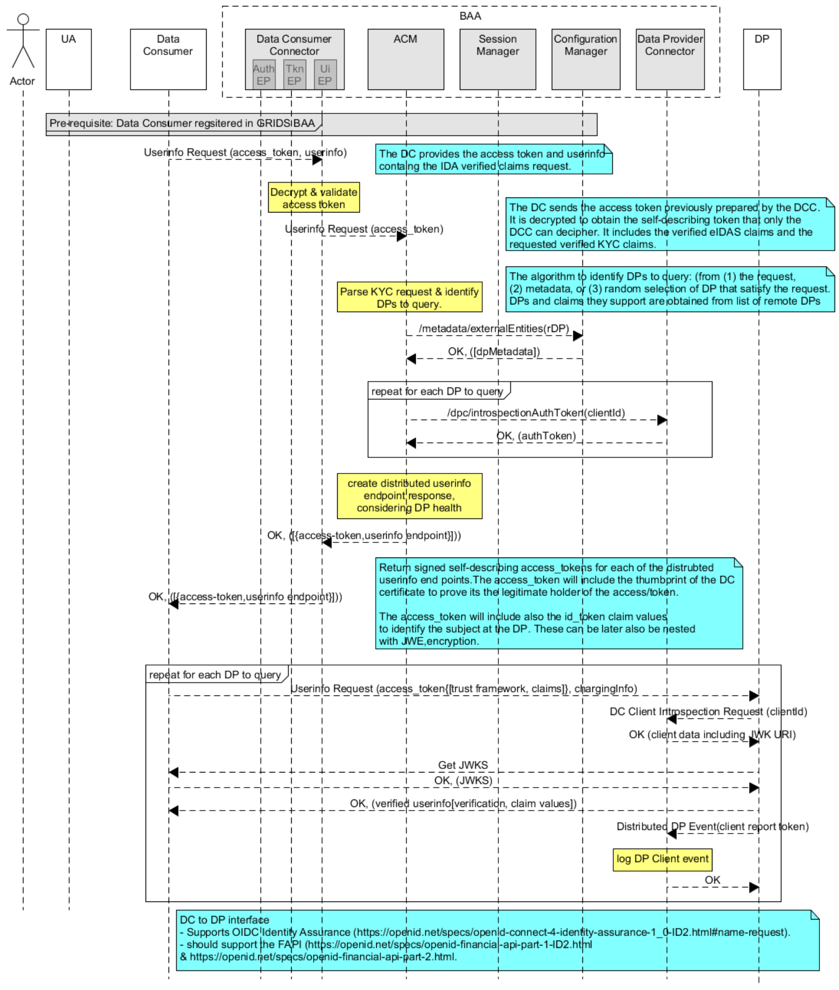

BAA
Actor
UA
Data Consumer
Data Consumer Connector
Auth EP
Tkn EP
Ui EP
ACM
Session Manager
Configuration Manager
Data Provider Connector
DP
Pre-requisite: Data Consumer regsitered in GRIDS BAA
Userinfo Request (access_token, userinfo)
The DC provides the access token and userinfo containg the IDA verified claims request.
Decrypt & validate access token
Userinfo Request (access_token)
The DC sends the access token previously prepared by the DCC. It is decrypted to obtain the self-describing token that only the DCC can decipher. It includes the verified eIDAS claims and the requested verified KYC claims.
Parse KYC request & identify DPs to query.
The algorithm to identify DPs to query: (from (1) the request, (2) metadata, or (3) random selection of DP that satisfy the request. DPs and claims they support are obtained from list of remote DPs
/metadata/externalEntities(rDP)
OK, ([dpMetadata])
repeat for each DP to query
/dpc/introspectionAuthToken(clientId)
OK, (authToken)
create distributed userinfo endpoint response, considering DP health
OK, ([{access-token,userinfo endpoint}]))
Return signed self-describing access_tokens for each of the distrubted userinfo end points.The access_token will include the thumbprint of the DC certificate to prove its the legitimate holder of the access/token.
The access_token will include also the id_token claim values to identify the subject at the DP. These can be later also be nested with JWE,encryption.
OK, ([{access-token,userinfo endpoint}]))
repeat for each DP to query
Userinfo Request (access_token{[trust framework, claims]}, chargingInfo)
DC Client Introspection Request (clientId)
OK (client data including JWK URI)
Get JWKS
OK, (JWKS)
OK, (verified userinfo[verification, claim values])
Distributed DP Event(client report token)
log DP Client event
OK
DC to DP interface
- Supports OIDC Identity Assurance (https://openid.net/specs/openid-connect-4-identity-assurance-1_0-ID2.html#name-request).
- should support the FAPI (https://openid.net/specs/openid-financial-api-part-1-ID2.html
& https://openid.net/specs/openid-financial-api-part-2.html.

Figure 4 BAA KYC Source Distributed Claims Flow

The flow in the above sequence diagram (Figure 4), follows on from the Authentication Flow in Figure 3, and is described in detail below:

1) After successful user authentication, and receiving the identity token and access token, the DC next sends the access token to the userinfo endpoint on the DCC in the BAA, for the requested verified claims, in order to receive the distributed userinfo endpoints and associated access tokens.

Implementation Note

GRIDS extends the IDA spec trust framework to optionally include the DP KYC Source userinfo endpoint in the userinfo request from the DC to the BAA. This is made possible, as the GRIDS BAA includes in its BAA OP Metadata a catalogue of all available DP KYC Services (their verified claims and associated trust frameworks), of the GRIDS network. The DC will be able to read the BAA OP Metadata at any time. The DC could also request the same verified claims to be requested from multiple DP KYC Sources.

In the case that the DC does not provide the DP KYC Source in the userinfo request, the ACM will match the request against all DP KYC/KYB sources that are seen to be available.

The non-normative example below for the userinfo request received from the Data Consumer includes a distributed KYC/KYB source *"`userinfo_endpoint`"* as given in the example below:

```
{
   "userinfo":{
      "verified_claims":{
         "verification": {
            "trust_framework": null,
             "userinfo_endpoint": "www.entiyid.com"
         },
         "claims":{
            "given_name":null,
            "family_name":null,
            "birthdate":null
         }
      }
   }
}
```

2) The DCC decrypts and forwards the userinfo request to the ACM, which then determines the DP KYC/KYB Source endpoints to query that match the verified KYC request and are available in the GRIDS network. The distributed DP KYC Source userinfo endpoints are obtained either from:
   a. the userinfo request itself with KYC/KYB Sources given by `"userinfo_endpoint"` or
   b. matching the request against all KYC/KYB Sources to return all possible options, so that the DC will decide which DP source(s) to use

   The ACM proceeds to generate the distributed Access tokens including the following:
   a. The previously verified identity claims obtained from the userinfo so that the DPs will have assured eIDAS identity claims on the subject being queried.
   b. The verified claims with trust framework to be queried from this DP KYC Source.
   c. A DC Client Introspection Point URI
      a. This enables the DP to query the Data Consumer client Id and so to obtain the DCCs JSON Web Key Set which will specify how the DP should return the requested KYC/KYB Claims to the client e.g. signed and/or encrypted.
   d. Finally, the ACM generates and signs the associated self-describing access tokens (with verified identity claims) for each of the IDA queries in the userinfo request. Any previously specified userinfo endpoints in the original userinfo request are removed.

Implementation Note

An example of an unencoded userinfo access token to be sent to a DP KYC Source for distributed verified claims is given below. Refer to this link.

```
{
"iss":"https://BAA.1.example.com",
"sub":"X731Z219A",
"aud":"https://DP.anotherexample.com",
"scope":"openid",
"exp":"1311281970",
"iat":"1311280970",
"jti":"knm34l45jl45l",
"txn": "12347565411142194650508795011123",
"client_id":"https://DC.example.com",
"client_introspection_uri":"https://GRIDS.BAA1234.com/clients?id=https://DC
.example.com",
"client_introspection_access_token":"78y98yy98hyubui989y898y98yh8h8y7",
"legal_person_identifier": "375714X",
"legal_name": "Acme Corporation",
"verified_claims": {
        "verification": {
                "trust_framework":"eidas",
                "identity_assurance_level": "substantial",
                "time": "2012-04-22T11:30Z"
                },
        "claims": {
                "given_name": "John",
                "family_name": "Smith",
                "birthdate": "1971-04-17",
                "person_identifier": "X731Z219A",
                "address": {
                        "country": null,
                        "street_address": "George Street 123",
                        "locality": "Glasgow",
                        "postal_code": "G1 1QD"

                }
        }
 },
"claims": {
        "userinfo": {
                "verified_claims": {
                        "verification": {
                                "trust_framework": {
                                  "value": "grids_kyb"
                                },
                                "evidence": [
                                 {
                                  "type": {
                                          "value": "company_register"
                                  },
                                  "registry": {
                                        "organisation": {
                                          "essential": false,
                                          "purpose": "string"
                                        },
                                        "country": {
                                          "essential": true,
                                          "purpose": "string",
                                          "value": "ES"
                                        }
                                  },
                                  "time": {
                                        "max_age": 0,
                                        "essential": true,
                                        "purpose": "string"
                                  },
                                  "data": {
                                        "essential": true,
                                        "purpose": "string"
                                  },
                                  "extractURL": {
                                        "essential": true,
                                        "purpose": "string"
                                  },
                                  "document": {
                                        "SKU": {
                                          "value": "REX"
                                          "essential": false,
                                          "purpose": "string"
```

```
                                                    },
                                                    "option": {
                                                      "essential": false,
                                                      "purpose": "string"
                                                    }
                                             }
                                           }
                                          ]
                                },
                                "claims": {
                                          "given_name": null,
                                          "family_name": null,
                                          "birthdate": null,verification
                                          "legal_name": null,
                                          "legal_person_identifier": null,
                                          "lei": null,
                                          "vat_registration": null,
                                          "address": null,
                                          "tax_reference": null,
                                          "sic": null,
                                          "business_role": null,
                                          "sub_jurisdiction": null,
                                          "trading_status": null
                                }
                      }
    }
}

}
```

3) The Data Consumer receives the userinfo response in a JWT containing the distributed access tokens and DP userinfo endpoint from the BAA. An example of the userinfo response received from the BAA with the distributed claims is given in section 3.2.4.2.
4) The Data Consumer proceeds to request for the KYC information and sends a GET with the received access token to the specified DP KYC Source userinfo endpoint.

Implementation Note

An example of the userinfo request is given below:

```
GET /userinfo HTTP/1.1
Host: https://DP.anotherexample.com
Authorization: Bearer ksj3n283dkey889yhhiuafb76cdefhuk455445jk45jkk45
```

5) The DP receives the request and decodes the self-describing token. It the processes the KYC request as follows:
   a. Identifies the subject eIDAS identity claims and any other identity claims provided.
   b. Processes the requested verified claims based on given identity information.
   c. If it is able to provide the KYC/KYB Claims requested, it verifies the received charging details are correct.
   d. The BAA Client introspection point is queried and returns the DCs JWK URI which the DP then queries to obtain the DCs JWKS.
   e. The DP prepares the token response as per the DCs JWKS either, plain, signed and/or encrypted.

6) The requested verified KYC and identity claims are returned to the Data Consumer in line with the DCs JWKS.
7) The client is charged as per the charging info provided in the KYC request to the DPs userinfo endpoint.
8) Finally, the DP captures the transaction with the DCC, in an event token that is sent to the BAA Event endpoint. This will indicate success or fail, and the verified KYC/KYB Claims requested with a hash of the subject identifier.

If the DC resultantly needs to make a new changed KYC/KYB Claim request it will repeat the above flow from step 1, but with the identity token previously issued to the RP by the BAA included and the new userinfo claims being requested and with any new identity claims provided and also with the "prompt" parameter set to "none". So that if the session is still available the user does not need to be prompted to login again by the BAA, however it is important that the DC does ensure to get the user authorisation for requesting new claims.

3.4 BAA Component Diagram

The following component diagram (Figure 5) shows the BAA implementation and interfaces supported in this release.

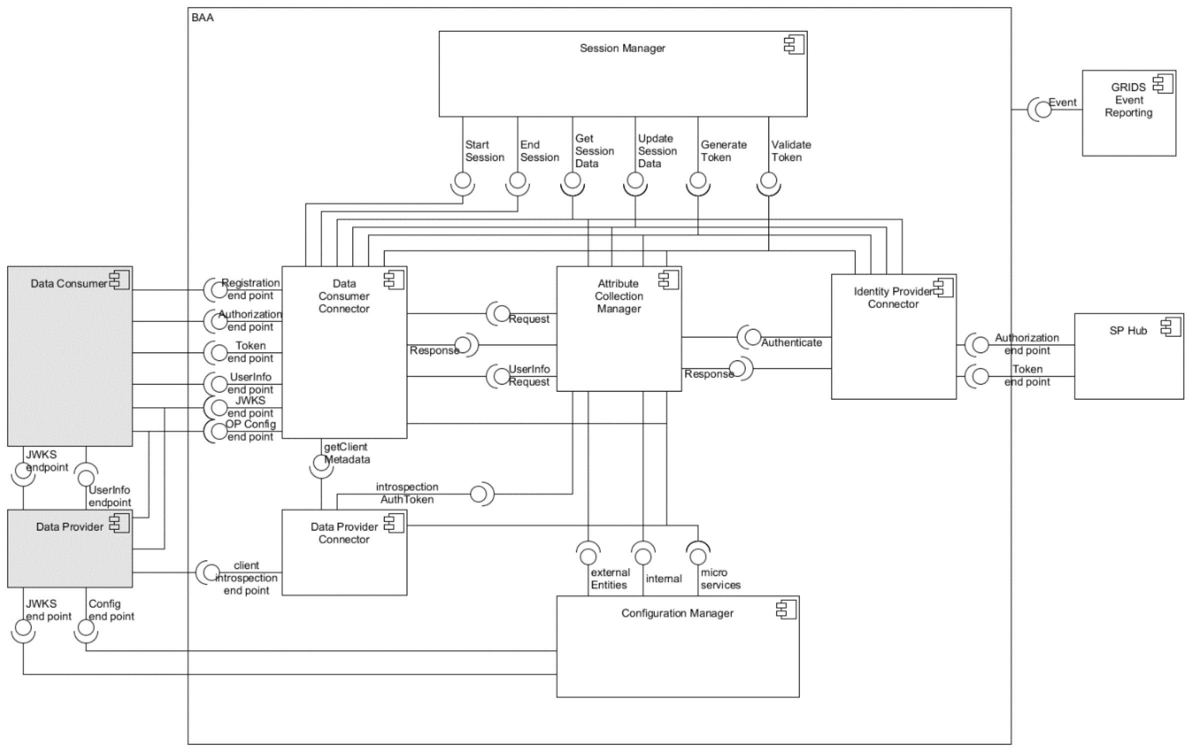


Figure 5 BAA Component Diagram

3.5 Data Model

The OIDC IDA 1.0 specification specifies certain KYC trust framework and claims and is captured in this section. The claims specific to GRIDS

3.5.1 OIDC IDA 1.0

3.5.1.1 Predefined identifier values

Identifiers for various element types used in the verified data representation of OpenID Connect 4 Identity Assurance are specified here.

3.5.1.2 Verified Claims

A JSON is specified for the OIDC IDA verified claims response here.

3.5.1.3 Verified Claims Request

A JSON is specified for the OIDC IDA verified claims request here.

3.5.2 GRIDS

3.5.2.1 GRIDS Identity Assurance Claims

GRIDS has specified a specific IDA trust framework for the Data Provider being piloted and is captured below.

```
"verified_claims": {
                              "verification": {
                                        "trust_framework": {
                                         "value": "grids_kyb"
                                        },
                                        "evidence": [
                                         {
                                         "type": {
                                                  "value": "company_register"
                                         },
                                         "registry": {
                                                 "organisation": {
                                                  "essential": false,
                                                  "purpose": "string"
                                                 },
                                                 "country": {
                                                  "essential": true,
                                                  "purpose": "string",
                                                  "value": "ES"
                                                 }
                                         },
                                         "time": {
                                                 "max_age": 0,
                                                 "essential": true,
            "purpose": "string"
    },
    "data": {
            "essential": true,
            "purpose": "string"
    },
    "extractURL": {
            "essential": true,
            "purpose": "string"
    },
    "document": {
            "SKU": {
              "value": "REX"
              "essential": false,
              "purpose": "string"
            },
            "option": {
              "essential": false,
              "purpose": "string"
            }
     }
    }
   ]
```

The values for country are specified according to ISO 3166 country codes.

The values of the SKU are as follows:

- "REX",
- "AA",
- "AOA",
- "SL"

Potentially, any KYC/KYB claims can be supported over GRIDS as long as the Data Consumer and Data Provider are able to consume and process them. However, a default list should be advertised by the GRIDS Operator towards the Data Consumers that want to connect to GRIDS.

For example, the default KYC/KYB claims handled by GRIDS could be as given below and these should be declared and definition referenced by the GRIDS Operator so that all Data Consumers are able to process them in a standardized format supported over GRIDS.

- "legal_name",
- "legal_person_identifier",
- "lei",
- "vat_registration",
- "address",

- "sic",
- "business_role",
- "sub_jurisdiction",
- "trading_status",
- "family_name",
- "given_name",
- "birthdate",
- "person_identifier"

3.6 Internal interfaces

The internal interfaces between BAA microservices are specified in the Open API yaml specification in Annex 7.10.

3.7 Security Guidelines

High level security guidelines for the implementation of GRIDS are as outlined below.

Security Guidelines:

1. All external interfaces should support https to prevent man in the middle attacks.
2. All communications between the client Data Consumer and BAA OIDC Provider should follow the authorisation code flow with standard security guidelines as indicated in the OIDC Core specification [6].
3. Id Tokens issued by the BAA are signed by the BAA and the signature is validated by the clien DCs.
4. Self-describing Access tokens issued by the BAA, to be consumed by the BAAs own end userinfo point are at signed and encrypted and decrypted with its own private key and validated upon its consumption by its own private signing key.
5. Self-describing Access tokens issued by the BAA, to be consumed by external Data Provider userinfo endpoints are both signed by the BAA OP and encrypted with the public encryption key of the Data Provider.
6. Communication from the Data Provider to the client Data Consumer with the response of the distributed userinfo request will be according to the client's publication of support for signed and encrypted userinfo responses.

Implementation Note 3.3.1.A

Useful references are given below:

Building an access token reference is found here.

FAPI Cert binding in the Access token with the DC thumbprint CERT reference s found here.

Use the same DC cert in mutual TLS authentication to the token end point on the BAA authorisation server as with the DC mutual TLS authentication on the DP Infouser endpoint, and DP checks the cert is the same so authenticating the DC is the same as was issued with the access token. Useful references are found here, here, here and here.

3.8 Microservices chassis framework

It would be advantageous to update the ESMO Microservices architecture [17] to an Open Source Microservices chassis framework, such as Spring Cloud, so to make use of the OS community tools available for improved service discovery and load balancing mechanisms.

However, this is not possible to implement in this project and is recommended as a future action.

4 Development Modules, Process and Tools

This section aims to outline the development process and tools used in the project for the modules that were developed.

4.1 BAA Microservice modules

The partners agreed to develop the Task 3.1 modules as per the table below (Table 2).

BAA Modules	Microservice Development	Partner
SM	Lifted from ESMO - No dev	UAEGEAN
CM	Lifted from SEAL with GRIDS specific development for managing BAA & DP OP metadata	Atos
ACM	Handles requests from DCC with redirection to eIDAS and handles KYC/KYB request for verified claims with DPs in GRIDS trust network before preparing distributed claims access tokens	Atos
DCC	Connector to Data Consumer OIDC Authorisation Server	UAEGEAN
DC Tester	Provide a dummy test DC container to test the whole flow	UAEGEAN
IdPC	Lifted from SEAL base for OIDC interwork towards eIDAS proxy with GRIDS specific interwork and events added	ADACOM
DPC	Provides Introspection endpoint on the BAA to the DPs for DC metadata queries and provides a n access token for the distributed userinfo to the DPs	InfoCert
ELK	Deployed & configured Elastic Search and Kibana so to support monitoring of events on the BAA	Kompany

Table 2 BAA microservice module development

4.2 Development tools

The GRIDS development process supported by Atos consists of the following tools:

- GitLab (v11.6.3)
- Nexus (v3.29.2-02)
- SonarQube (v8.2)
- GRIDS VM (Ubuntu 20.04.1) where the BAA node will be deployed.

Domain name: **example-xyz.eu**

Every partner who signed the related ATOS form should access those without problems, by means of the user Id & password provided by ATOS.

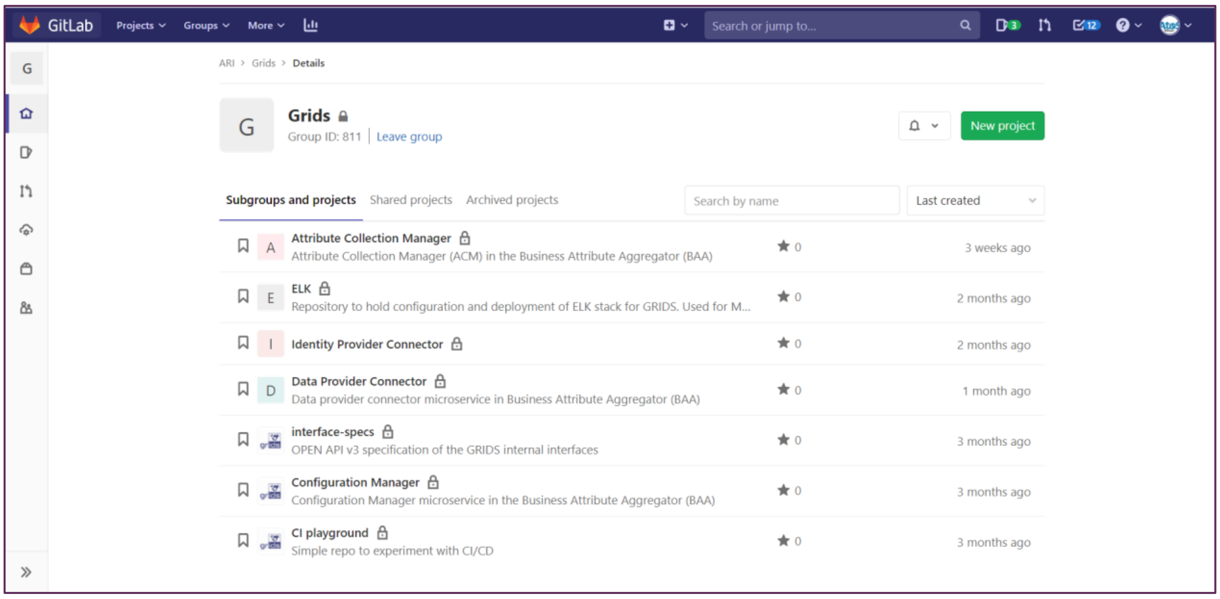
Only CI practices are described in this document by the moment. CD practices like the automation of the docker-compose deployments, will be considered in the future

4.3 Development process

This document sets the guidelines for the CI/CD flow during the development of the GRIDS project.

The aim of our CI/CD approach is to allow the GRIDS partner design teams to work together more efficiently to quickly and automatically test, package, and deploy the GRIDS software modules.

For that, the following GitLab Group (Figure 6) has been created: partners are adding with the related permissions to store code, build CI pipelines, etc.



Grids
Group ID: 811 | Leave group

A	Attribute Collection Manager Attribute Collection Manager (ACM) in the Business Attribute Aggregator (BAA)	★ 0	3 weeks ago
E	ELK Repository to hold configuration and deployment of ELK stack for GRIDS. Used for M...	★ 0	2 months ago
I	Identity Provider Connector	★ 0	2 months ago
D	Data Provider Connector Data provider connector microservice in Business Attribute Aggregator (BAA)	★ 0	1 month ago
	interface-specs OPEN API v3 specification of the GRIDS internal interfaces	★ 0	3 months ago
	Configuration Manager Configuration Manager microservice in the Business Attribute Aggregator (BAA)	★ 0	3 months ago
	CI playground Simple repo to experiment with CI/CD	★ 0	3 months ago

Figure 6: GRIDS GitLab Group

Note the repository *CI playground* in the GRIDS group is used as a sample of how to use the GRIDS CI framework.

General documentation on GitLab can be found at [10].

4.3.1 Contribution Guidelines

A sample of general information on pull requests, code conventions, commit conventions, language etc. is specified in the GRIDS Repo.

4.3.2 Basic steps

Next steps should be followed after creating a new repository:

Select the License

The *CI playground* project uses EUPL 1.2.

README file

LET'S WRITE IT before starting to upload any code!

Use of Headers

Every source file must have a header. This is an example for Atos code:

```
/**
Copyright © 2021  Atos Spain SA. All rights reserved.
```

```
This file is part of GRIDS Configuration Manager (GRIDS CM).
GRIDS CM is free software: you can redistribute it and/or modify it under the terms of EUPL
1.2.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT ANY WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT,
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

See README file for the full disclaimer information and LICENSE file for full license
information in the project root.

@author Atos Research and Innovation, Atos SPAIN SA

<Short description of the class>
*/
```

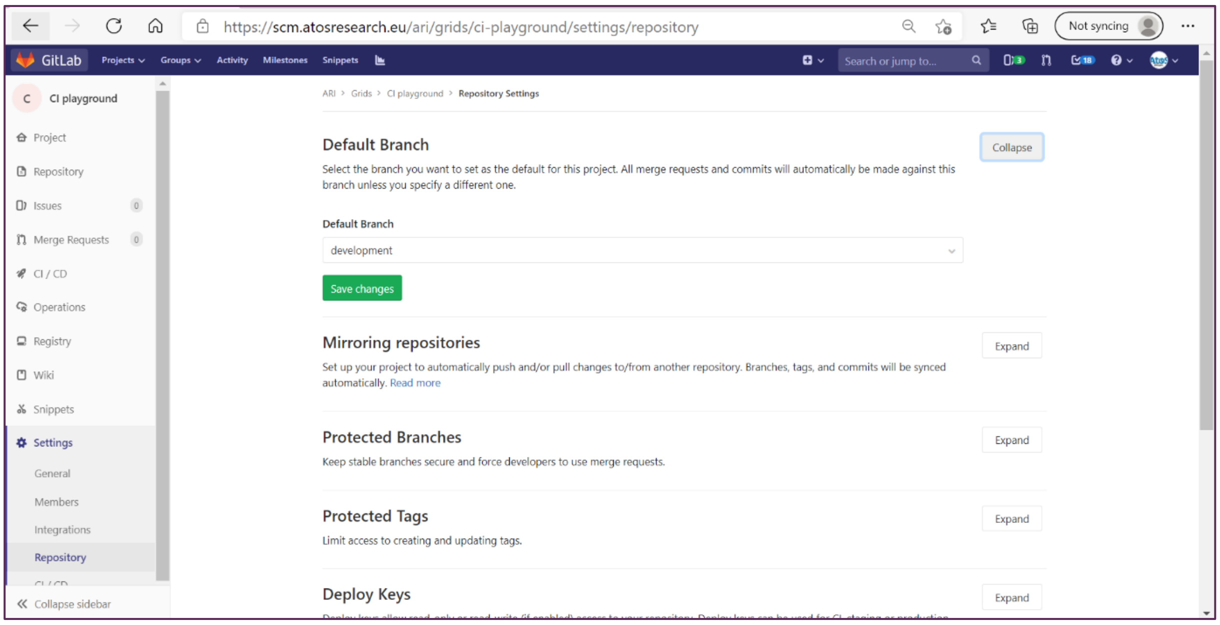
4.3.3 Java package naming

“eu.grids.xxxxx”.

4.4 CI Flow

4.4.1 GitLab Project

When a new project is created, add the project members first (from Setting Members), and then specify the settings for branches (from Settings Repository). For each repository, the default branch (see Figure 7 below) represents the “base” branch in that repository. All the pull requests and code commits are automatically made on it, unless you specify a different branch.



Default Branch

Select the branch you want to set as the default for this project. All merge requests and commits will automatically be made against this branch unless you specify a different one.

Default Branch: development

Mirroring repositories

Set up your project to automatically push and/or pull changes to/from another repository. Branches, tags, and commits will be synced automatically. Read more

Protected Branches

Keep stable branches secure and force developers to use merge requests.

Protected Tags

Limit access to creating and updating tags.

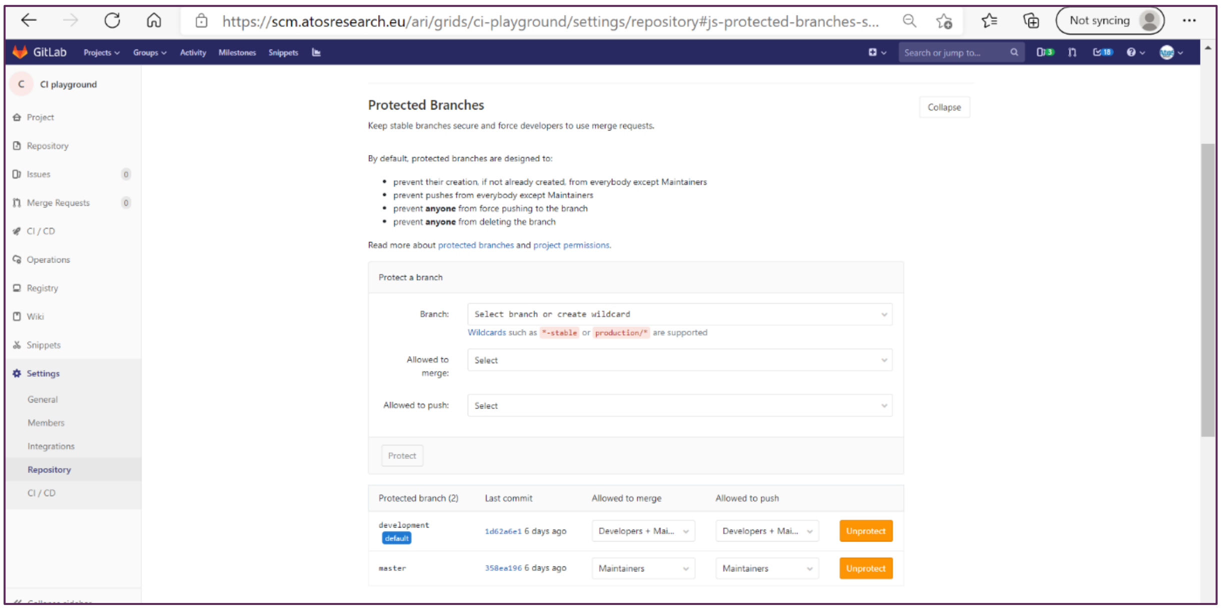
Deploy Keys

Figure 7: Set the default branch

You can specify how, and which branches are protected (from Settings Repository). In this way, certain branches cannot be deleted, or pushed, according to the specified roles.

The objective is the master branches of all the GRIDS projects to be aligned. For that, the most restrictive rule is to be added to the master in each repository. For the development branches (default branch), minor restriction; other branches created, the less restrictive.

In the *CI playground* project, the default branch is "development", and the protected branches are "master" and "development", with different restrictive rules, as we can see in the next figure:


https://scm.atosresearch.eu/ari/grids/ci-playground/settings/repository#js-protected-branches-s...

Protected Branches

Keep stable branches secure and force developers to use merge requests.

By default, protected branches are designed to:

- prevent their creation, if not already created, from everybody except Maintainers
- prevent pushes from everybody except Maintainers
- prevent **anyone** from force pushing to the branch
- prevent **anyone** from deleting the branch

Read more about protected branches and project permissions.

Protect a branch

Branch: Select branch or create wildcard

Wildcards such as `*-stable` or `production/*` are supported

Allowed to merge: Select

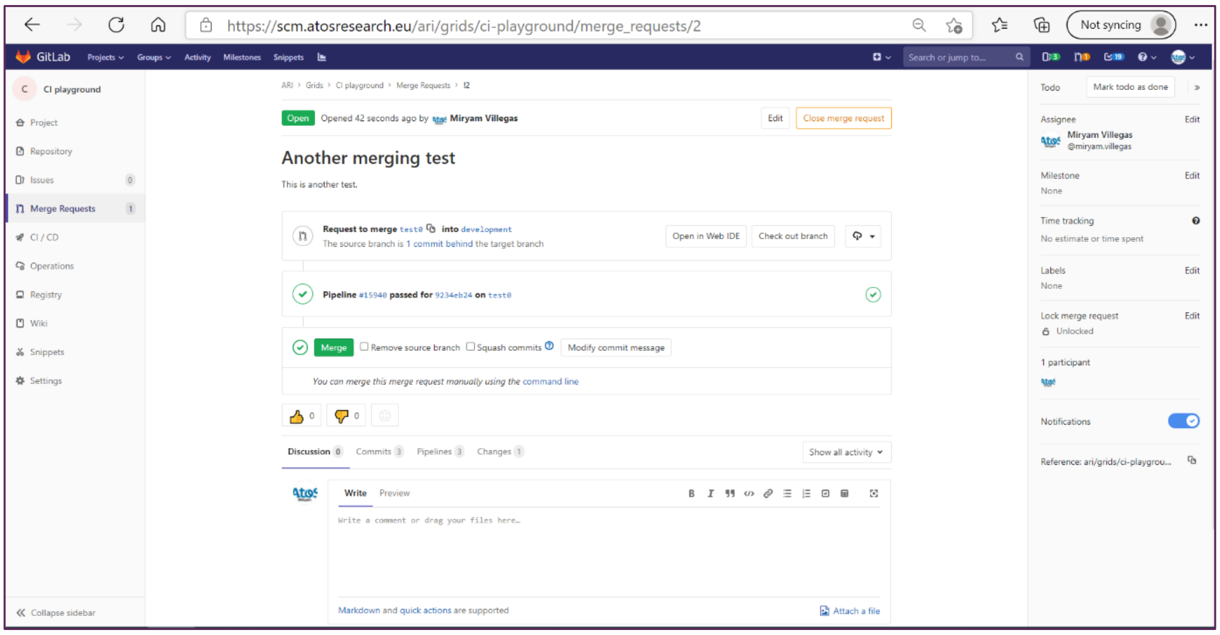
Allowed to push: Select

Protected branch (2)	Last commit	Allowed to merge	Allowed to push
development default	1d62a6e1 6 days ago	Developers + Mai...	Developers + Mai...
master	358ea196 6 days ago	Maintainers	Maintainers

Figure 8: Set the protected branches

From the Settings General menu, Permissions and Merge options can be expanded in order to adjust those settings.

Two important functionalities are to be used along the development of the project: Issues and Merge Requests.


https://scm.atosresearch.eu/ari/grids/ci-playground/merge_requests/2

ARI > Grids > CI playground > Merge Requests > !2

Open Opened 42 seconds ago by Miryam Villegas

Another merging test

This is another test.

Request to merge `test0` into `development`

The source branch is 1 commit behind the target branch

Pipeline #15940 passed for 9234eb24 on test0

☐ Remove source branch ☐ Squash commits

You can merge this merge request manually using the command line

Assignee: Miryam Villegas @miryam.villegas

Milestone: None

Time tracking: No estimate or time spent

Labels: None

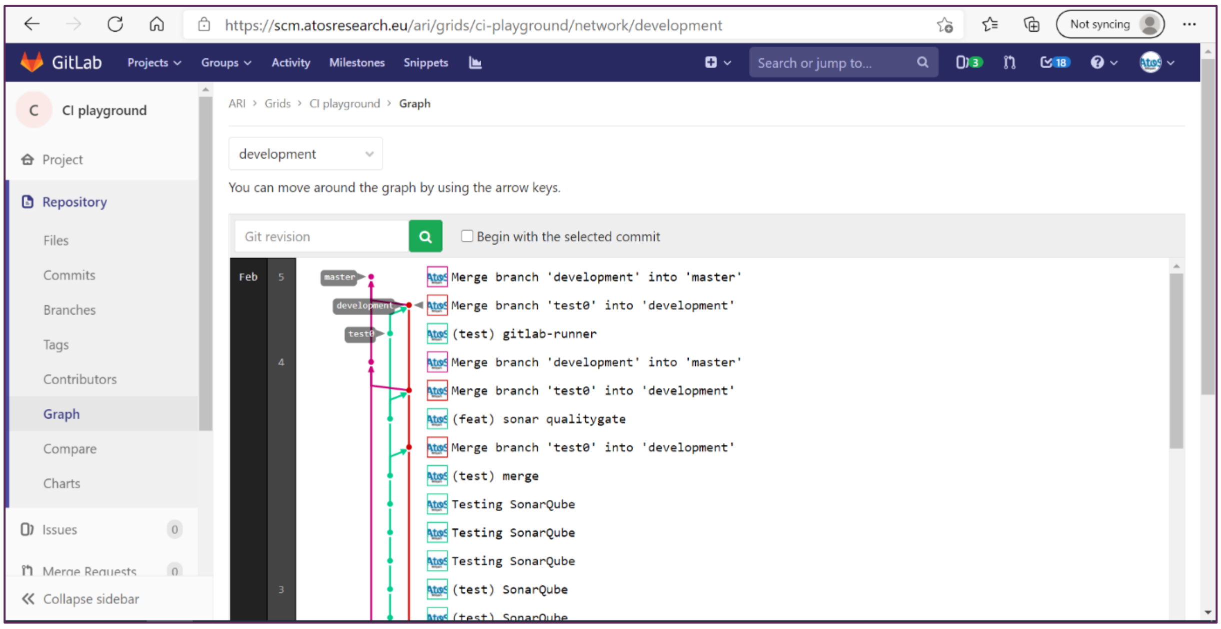
Lock merge request: Unlocked

1 participant

Reference: ari/grids/ci-playgrou...

Figure 9: Approving a merge request

Finally, it is interesting to access the Graph of the project, to see its status in relation to the branches and merges done:



You can move around the graph by using the arrow keys.

```
Merge branch 'development' into 'master'
Merge branch 'test0' into 'development'
(test) gitlab-runner
Merge branch 'development' into 'master'
Merge branch 'test0' into 'development'
(feat) sonar qualitygate
Merge branch 'test0' into 'development'
(test) merge
Testing SonarQube
Testing SonarQube
Testing SonarQube
(test) SonarQube
(test) SonarQube
```

Figure 10: Repository Graph

4.4.2 Continuous Integration

The file where the CI/CD is defined is the **.gitlab-ci.yml** file at the root of your repository. The runners are the software agents in charge of executing the jobs specified in that file.

When any commit (or any action you have considered in the file) happens in your repository, the runner executes the jobs detailed in that file. A set of jobs is called pipeline.

(Refer to [11] for more information.)

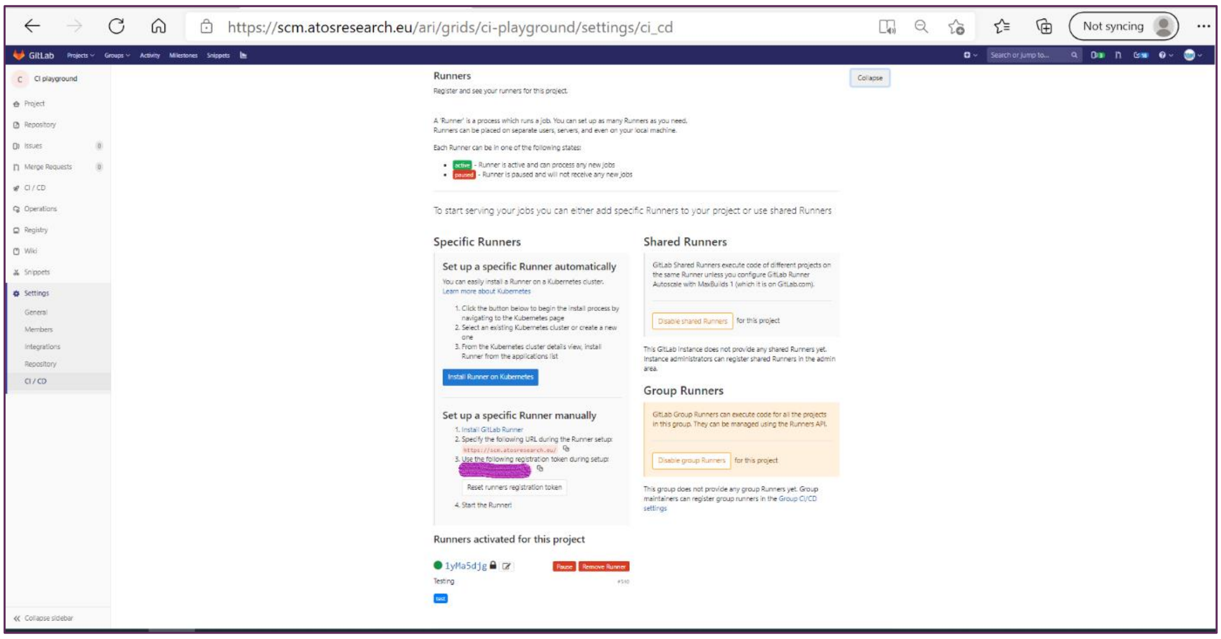
Our goal is to have available a .gitlab-ci.yml file per repository to scan the code with SonarQube, build and push images to the Nexus. The runners can be installed whatever machine you want, but if you prefer to run in the GRIDS VM, ATOS team will install and register them for you.

4.4.2.1 GitLab Runners

The gitlab runner installed for the *CI playground* project was created and registered in this way:

```
docker pull gitlab/gitlab-runner:v11.7.0
docker volume create gitlab-runner-config
docker run -d --name gitlab-runner --restart always \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v gitlab-runner-config:/etc/gitlab-runner \
    gitlab/gitlab-runner:v11.7.0
docker exec -it gitlab-runner gitlab-runner register --docker-privileged
```

You can see from Settings CI/CD, in the Runners section in Figure 11 below.



https://scm.atosresearch.eu/ari/grids/ci-playground/settings/ci_cd

Runners

Register and see your runners for this project.

A 'Runner' is a process which runs a job. You can set up as many Runners as you need. Runners can be placed on separate users, servers, and even on your local machine.

Each Runner can be in one of the following states:

- active - Runner is active and can process any new jobs
- paused - Runner is paused and will not receive any new jobs

To start serving your jobs you can either add specific Runners to your project or use shared Runners

Specific Runners

Set up a specific Runner automatically

You can easily install a Runner on a Kubernetes cluster. Learn more about Kubernetes

1. Click the button below to begin the install process by navigating to the Kubernetes page
2. Select an existing Kubernetes cluster or create a new one
3. From the Kubernetes cluster details view, install Runner from the applications list

Set up a specific Runner manually

1. Install GitLab Runner
2. Specify the following URL during the Runner setup: `https://scm.atosresearch.eu/`
3. Use the following registration token during setup: [redacted]
4. Start the Runner!

Shared Runners

GitLab Shared Runners execute code of different projects on the same Runner unless you configure GitLab Runner Autoscale with MaxBuilds 1 (which it is on GitLab.com).

Disable shared Runners for this project

This GitLab instance does not provide any shared Runners yet. Instance administrators can register shared Runners in the admin area.

Group Runners

GitLab Group Runners can execute code for all the projects in this group. They can be managed using the Runners API.

Disable group Runners for this project

This group does not provide any group Runners yet. Group maintainers can register group runners in the Group CI/CD settings

Runners activated for this project

1yMa5djg

Testing

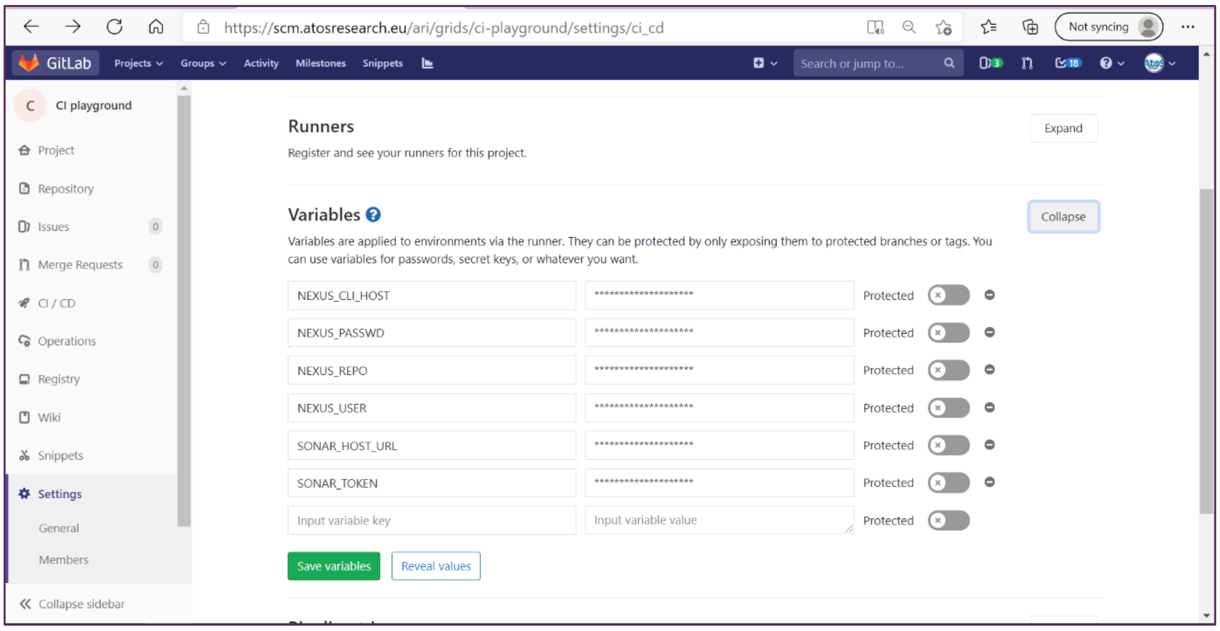
#510

Figure 11: GitLab runner for the *CI playground* repository

See more information in reference [12].

4.4.3 The .gitlab-ci.yml file

If you read the .gitlab-ci.yml file of the *CI playground* repository, several variables are used. Such variables have to be defined previously from Settings CI/CD in the **Variables** section in Figure 12 below.



https://scm.atosresearch.eu/ari/grids/ci-playground/settings/ci_cd

Runners

Register and see your runners for this project.

Variables

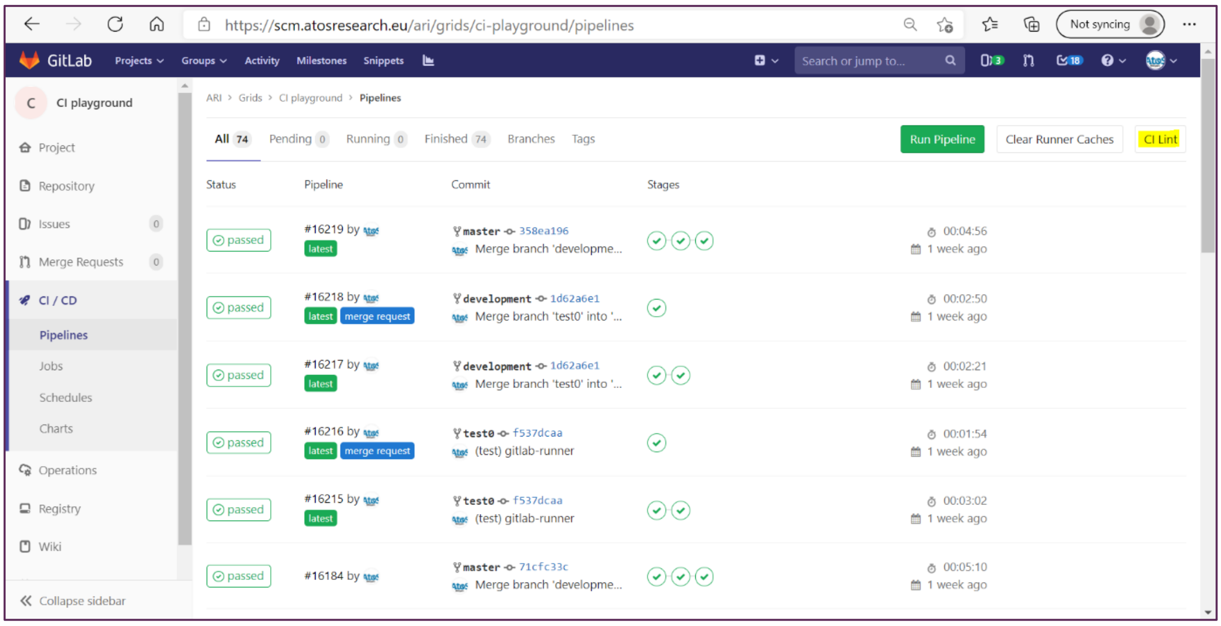
Variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. You can use variables for passwords, secret keys, or whatever you want.

NEXUS_CLI_HOST	********************	Protected
NEXUS_PASSWD	********************	Protected
NEXUS_REPO	********************	Protected
NEXUS_USER	********************	Protected
SONAR_HOST_URL	********************	Protected
SONAR_TOKEN	********************	Protected
Input variable key	Input variable value	Protected

Figure 12: *CI playground* CI/CD variables

Some of those variables are used along the specification of the pipeline of the sample project. In our case, we have defined several jobs to be executed (or not) depending on the branch we are working.

It is highly recommended to use the lint utility to validate the syntax of your .gitlab-ci.yml. It can be found at CI/CD Pipelines in Figure 13 below.



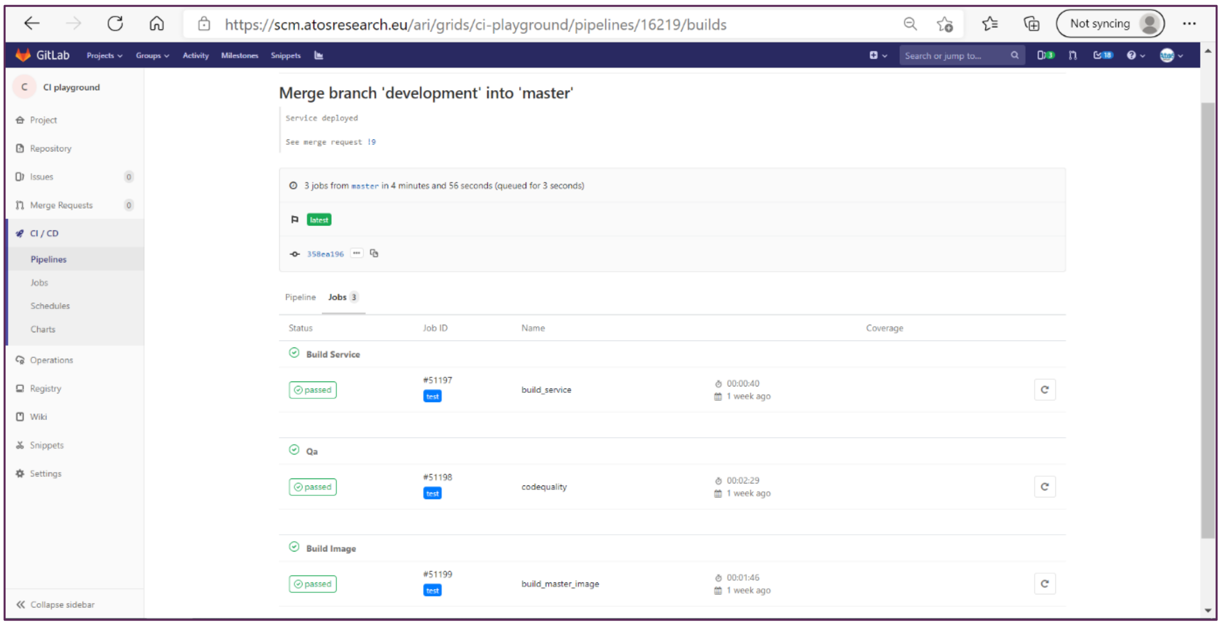
Status	Pipeline	Commit	Stages	
passed	#16219 by latest	master 358ea196 Merge branch 'developme...	✓ ✓ ✓	00:04:56 1 week ago
passed	#16218 by latest merge request	development 1d62a6e1 Merge branch 'test0' into '...	✓	00:02:50 1 week ago
passed	#16217 by latest	development 1d62a6e1 Merge branch 'test0' into '...	✓ ✓	00:02:21 1 week ago
passed	#16216 by latest merge request	test0 f537dcaa (test) gitlab-runner	✓	00:01:54 1 week ago
passed	#16215 by latest	test0 f537dcaa (test) gitlab-runner	✓ ✓	00:03:02 1 week ago
passed	#16184 by	master 71cfc33c Merge branch 'developme...	✓ ✓ ✓	00:05:10 1 week ago

Figure 13: CI Lint utility

4.4.3.1 Status

To check the status of your project CI, check the pipeline status as shown in Figure 13.

By clicking on a giving pipeline, you can get more detailed information, and the jobs taking part in that pipeline. Access the jobs logs selecting the job you want to analyse as in Figure 14 below.



Merge branch 'development' into 'master'

```
Service deployed

See merge request !9
```

3 jobs from master in 4 minutes and 56 seconds (queued for 3 seconds)

latest

358ea196

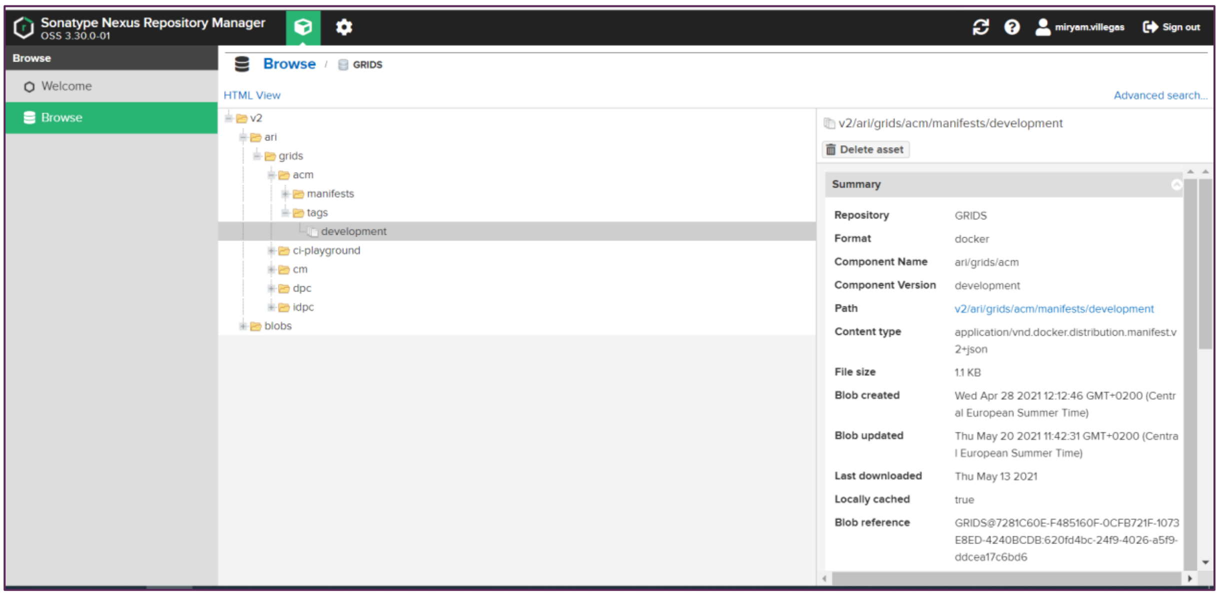
Status	Job ID	Name		Coverage
Build Service				
passed	#51197 test	build_service	00:00:40 1 week ago	
Qa				
passed	#51198 test	codequality	00:02:29 1 week ago	
Build Image				
passed	#51199 test	build_master_image	00:01:46 1 week ago	

Figure 14: CI status

4.4.4 Nexus registry

The container images generated are to be stored in a Nexus registry as depicted in Figure 15 below.

You have to sign in with the user/password provided by Atos.



```
v2
  ari
    grids
      acm
        manifests
        tags
          development
      ci-playground
      cm
      dpc
      idpc
  blobs
```

v2/ari/grids/acm/manifests/development

Repository	GRIDS
Format	docker
Component Name	ari/grids/acm
Component Version	development
Path	v2/ari/grids/acm/manifests/development
Content type	application/vnd.docker.distribution.manifest.v2+json
File size	1.1 KB
Blob created	Wed Apr 28 2021 12:12:46 GMT+0200 (Central European Summer Time)
Blob updated	Thu May 20 2021 11:42:31 GMT+0200 (Central European Summer Time)
Last downloaded	Thu May 13 2021
Locally cached	true
Blob reference	GRIDS@7281C60E-F485160F-0CFB721F-1073E8ED-4240BCDB:620fd4bc-24f9-4026-a5f9-ddcea17c6bd6

Figure 15: *CI playground* images stored in Nexus

The CLI URL to be used and in this way, you would make:

```
docker login registry.host.xyz:18460
docker build -t registry.host.xyz:18460/ari/grids/your-repo .
docker push registry.host.xyz:18460/ari/grids/your-repo:0.1
sudo docker pull registry.host.xyz:18460/ari/grids/your-repo:latest
```

Note: Please be aware of the two different URLs, the web and the CLI.

A CI playground example is given on how to automate the uploading to Nexus. The jobs build_image and build_master_image show the commands used:

- Note the use of environment variables previously defined in our project like NEXUS_CLI_HOST or NEXUS_CLI_REPO. (See please at Figure 12: *CI playground* CI/CD variables.)
- Other variables belong to GitLab: CI_COMMIT_REF_SLUG and others will allow to tag the image.

4.4.5 SonarQube

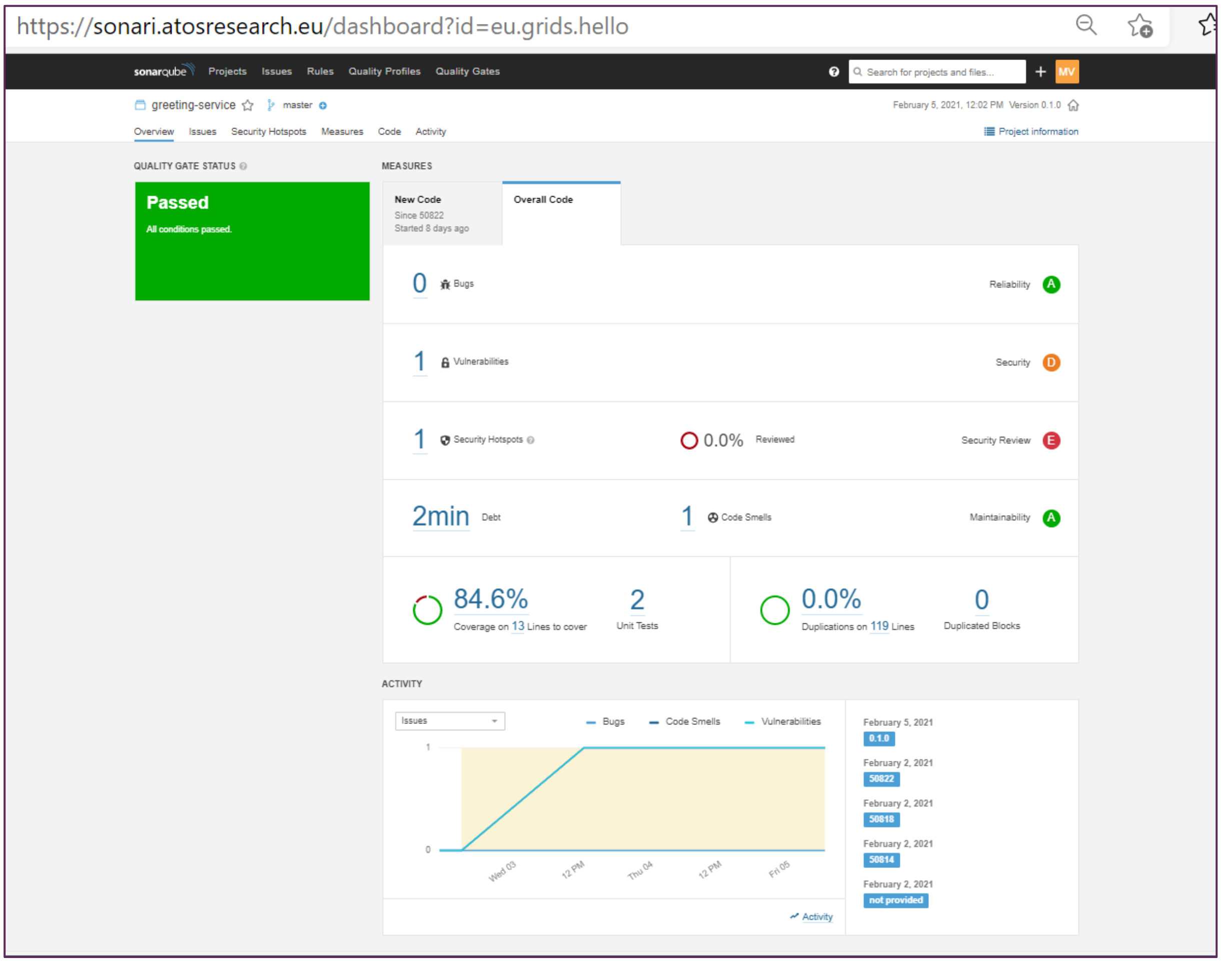
To gain access to the tool sign in with your GitLab user/password provided by ATOS, where a list of projects is displayed.

Firstly, you have to add/create a new project in order to specify the language of your code. A SonarQube *token* will be generated, necessary when analysing the codeThe commands to scanner the code are provided at the end of the creation also. Such commands are the ones to be added to your .gitlab-ci.yaml file. You can try them in a manual way before automating.

For the *CI playground* repository:

```
mvn sonar:sonar \
  -Dsonar.projectKey=eu.url.hello \
  -Dsonar.host.url=https://host.xyz \
  -Dsonar.login=the_token
```

You can see, for example, the Sonar analysis performed for the *CI Playground* project as depicted in Figure 16 below.



https://sonari.atosresearch.eu/dashboard?id=eu.grids.hello

greeting-service · master · February 5, 2021, 12:02 PM Version 0.1.0

QUALITY GATE STATUS: Passed — All conditions passed.

MEASURES — New Code (Since 50822, Started 8 days ago) | Overall Code

0 Bugs		Reliability A
1 Vulnerabilities		Security D
1 Security Hotspots	0.0% Reviewed	Security Review E
2min Debt	1 Code Smells	Maintainability A
84.6% Coverage on 13 Lines to cover	2 Unit Tests	
0.0% Duplications on 119 Lines	0 Duplicated Blocks	

ACTIVITY

February 5, 2021	0.1.0
February 2, 2021	50822
February 2, 2021	50818
February 2, 2021	50814
February 2, 2021	not provided

Figure 16: Sonar analysis for *CI playground* project

In the file yml file can see how to automate the SonarQube analysis. The job “codequality” is in charge of executing the SonarQube scanner:

- Some environment variables are used (SONAR_HOST_URL and SONAR_HOST_TOKEN). (See please at Figure 12: *CI playground* CI/CD variables.)
- Note the qualitygate option (true/false): it is the way to say the pipeline to stop or not if the scanner fails (the result is below the quality gate specifications).

Find more information at [13].

4.5 Deployment

We are going to use docker-compose utility to deploy GRIDS microservice in our development VM.

GRIDS partners shall provide a docker-compose installation in order to be included in the GRIDS deployment. They will be able to access the VM in order to see the ms deployed, the containers and the images, and read the logs generated by such ms if they want. The deployment will be done by ATOS team.

The docker-compose file used for deploying the *CI playground* service is available on the GRIDS repo.

5 Conclusions

The report first recaps the high-level design as applicable to the implementation of the Business Attribute Aggregator before fully specifying its low-level design.

It is seen that the BAA has aligned with the OIDC IDA specification and has been active in its WG so to include GRIDS use case requirements in the next release of the specification.

The design process and tools used by the partners' in developing the BAA microservice modules is also captured.

6 References

[1] ESMO Project, D2.1 Cross-border mechanisms and Technical Design for the effective use of eID DSI

[2] **Websites:** Elasticsearch, Logstash , Kibana OSS Event Monitoring https://www.elastic.co/elastic-stack , retrieved 2020-09-24

[3] **Websites:** https://www.javainuse.com/spring/springboot-microservice-elk , retrieved 2020-09-24

[4] **Websites:** https://openid.net/specs/openid-connect-4-identity-assurance-1_0.html , retrieved 2020-10-19

[5] **Websites: https://openid.net/specs/openid-connect-discovery-1_0.html,** retrieved 2020-10-19

[6] **Websites: https://openid.net/specs/openid-connect-registration-1_0.html#ClientRegistration,** retrieved 2020-10-19

[7] **Websites, https://openid.net/specs/openid-connect-core-1_0.html,** retrieved 2020-10-19

[8] **Websites,** https://openid.net/specs/openid-connect-4-identity-assurance-1_0.html, retrieved 2020-10-19

[9] **Websites,** https://tools.ietf.org/html/rfc7662, retrieved 2020-10-19

[10] **Websites,** GitLab Docs, GitLab Docs, https://docs.gitlab.com/ee/README.html#new-to-git-and-gitlab. retrieved 2021-02-11

[11] **Websites**, GitLab Docs, Get started with GitLab CI/CD, https://docs.gitlab.com/ee/ci/quick_start/, retrieved 2021-02-11

[12] **Websites**, GitLab Docs, Install GitLab Runner, https://docs.gitlab.com/runner/install/. retrieved 2021-02-12

[13] **Websites**, SonarQube Docs, Overview, https://docs.sonarqube.org/latest/analysis/overview/, retrieved 2021-02-11

[14] GRIDS Project, D2.1 Business Services and Technical Architecture report

[15] **Websites**, RFC6749 The OAuth 2.0 Authorization Framework, https://datatracker.ietf.org/doc/html/rfc6749, retrieved 2021-05-19

[16] **Websites**, OIDC Discovery 1.0, https://openid.net/specs/openid-connect-discovery-1_0.html#ProviderConfig.

[17] **Websites**, EC ESMO Project Git Hub Repo, https://github.com/ec-esmo

[18] **Websites**, https://ldapwiki.com/wiki/OAuth%202.0%20Client%20Registration, retrieved 2021-05-31

[19] **Websites**, https://www.keycloak.org/docs/7.0/securing_apps/, retrieved 2021-05-31

7 Annexes

7.1 Local BAA Metadata (CM)

This is a non-normative example of the BAA metadata dynamically configured on the CM module to include the local BAA metadata and verified claims available over the local DP nodes and their last successful refresh time.

This is made available to the ACM for internal BAA metadata query and also other trusted BAA nodes that query the BAA.

baaIDAmetadata_lo
cal v0.2.json

7.2 External BAA Metadata (CM)

This is a non-normative example of how all external BAAs and their supported DP metadata are queried on the CM by the ACM for external BAA metadata.

baaIDAmetadata_ex
ternal v0.2.json

The CM populates this file by querying each BAA Metadata endpoint populated in the trusted BAA list.

7.3 Trusted BAA List (CM)

This is a list of trusted BAA nodes the administrator has approved and configured as trusted BAA nodes in GRIDS.

GRIDS BAA
endpoints list.json

7.4 Trusted DP List (CM)

This is a list of Data Providers that the BAA administrator has approved and configured as trusted KYC Sources of verified credentials.

DP OIDC
Configuration endp

This is used by the CM to query locally supported Data Providers OIDC Configuration Metadata.

7.5 EIDAS Metadata (CM)

This is a non-normative example of an EIDAS metadata json file configured by the CM, and queried by the ACM for EIDAS metadata.

GRIDS IDPmetadata
v02.json

7.6 GRIDS Metadata (CM)

Includes the verified claims supported by the BAA over the whole GRIDS Network and to be added to the BAA OP Metadata well-known configuration by the DCC. This is queried by the DCC for GRIDS metadata.

GRIDSmetadata
v0.8.json

7.7 Basic BAA Metadata (DCC)

This is a non-normative example of the basic well-known OP Metadata configured on the DCC module.

GRIDS basic BAA
metadata json exam

7.8 BAA OP Metadata well-known configuration (DCC)

This is the well-known OIDC configuration publicly available and used by all Data Consumers and Data Providers in the GRIDS ecosystem. This is generated by the DCC module.

GRIDS BAA OIDC
configuration endp

7.9 DP OIDC IDA Metadata (DP)

This is a non-normative example of the well-known OP Metadata provided by the Data Providers.

DPOIDCmetadata
v2.json

7.10 BAA YAML

The OPEN API 3 specification of all BAA APIs between its internal microservices are specified in the yaml included below.

GRIDS
openapi3.yml

7.11 eIDAS Natural and Legal Claims

The natural and legal person claims supported in respective eIDAS natural persons and legal persons requests are specified in the file included below.

eIDAS.json

7.12 JWKS

A non-normative example of a JWKS is given below.

jwks.json