IISc

सत्यमेव जयते

Ministry of Housing and Urban Affairs

Government of India

Smart City

MISSION TRANSFORM-NATION

Architecture and Specifications of IUDX 2.0

Table of Contents

1 Terms and Definitions **6**
- 1.1 API 6
- 1.2 Endpoint 6
- 1.3 API Server 6
- 1.4 Vert.x 6
- 1.5 API Server 6
- 1.6 Client certificate 6
- 1.7 Event Bus 7
- 1.8 Service 7

2 Solution Architecture: Catalogue and Resource Server **8**
- 2.1 API Server Architecture 8
  - 2.1.1 Service Discovery 9
  - 2.1.2 Service Proxy 9
  - 2.1.3 Circuit Breaker 9
  - 2.1.4 Metrics and Logs 9
  - 2.1.5 Configuration Retriever 9
  - 2.1.6 Request Decoding and Response Module 10
  - 2.1.7 API Definition Module 10
  - 2.1.8 Service Mesh Architecture 10
- 2.2 Catalogue Server Solution Architecture 11
  - 2.2.1 Catalogue Server Components 11
    - 2.2.1.1 API Server 11
    - 2.2.1.2 Request Decoding and Response Module 11
    - 2.2.1.3 Database Search Service 11
      - Attribute Search 12
      - Geo-Spatial Search 12
      - Text Search 12
      - Complex Search 12
      - Response Filter 12
    - 2.2.1.4 Database Item Creation Service 12
    - 2.2.1.5 Database Item Update Service 12
    - 2.2.1.6 Database Item Delete Service 12
    - 2.2.1.7 Item validation Service 13
    - 2.2.1.8 Owner validation Service 13
    - 2.2.1.9 Onboard Service 13
- 2.3 Resource Server Solution Architecture 13
  - 2.3.1 Resource Server Components 14

2.3.1.1 API Server 14
2.3.1.2 Request Decoding and Response Module 14
2.3.1.3 Database Search Service 14
2.3.2 Data Broker Service 15
2.3.2.1 Create Exchange 15
2.3.2.2 Delete Exchange 15
2.3.2.3 Create Queue 15
2.3.2.4 Delete Queue 15
2.3.2.5 Create Bindings 15
2.3.2.6 Update Bindings 15
2.3.2.7 Delete Bindings 15
2.3.3 Authentication Service 15
2.3.4 Database Connector 15
2.3.5 Provider Onboard Service 16
2.4 Multi-tenancy 16
2.5 Fault tolerance 16
2.6 Load Balancing 16
2.7 References 16

3 Authorization Service Specifications and Architecture 17
3.1 Authorization server solution architecture 17
3.1.1 API Server 17
3.1.2 Policy Engine 17
3.1.3 Database 18
3.1.4 Packet Filter 18
3.1.5 Backup Service 18
3.1.6 Notifications and Alerts 18
3.2 Authorization Service API Specifications 18
3.2.1 Access Control 20
3.2.1.1 Create a policy for a provider 20
3.2.1.2 Append to an access control policy for a provider 21
3.2.1.3 Revert to previous policy 22
3.2.1.4 List all policies of a provider. 22
3.2.2 Access Tokens 23
3.2.2.1 Get an access token 23
3.2.3 Managing tokens 24
3.2.3.1 Revoke a valid token 24
3.2.3.2 Revoke all valid tokens of a consumer 25
3.2.3.3 Introspect a token 26

3.2.3.4 Audit tokens 27
3.2.4 Managing Groups 28
3.2.4.1 Add consumer to a group 28
3.2.4.2 Delete consumer from a group 29
3.2.4.3 List consumers in a group. 29
3.2.5 Miscellaneous 30
3.2.5.1 Get certificate information 30
3.3 References 31
4 Resource Server and Catalogue API Specifications 32

Scope

IUDX 2.0 architecture and specification have been finalized and are documented herein.

There are three sections in the document that cover the various aspects of the system.

1. Solution Architecture (Catalogue and Resource Server)
2. IUDX Authorization Server (API Specifications and Solution Architecture)
3. Resource Server and Catalogue API Documentation

Disclaimer

This document is not meant to be released publicly.

1 Terms and Definitions

1.1 API

Application Programming Interface (API) is an interface with which external applications interact with the IUDX system through predefined interfaces designed based on an industry-standard such as HTTP / HTTPs to ensure interoperability.

1.2 Endpoint

An endpoint specifies how a resource can be accessed. It allows an end-user or application to interact with the IUDX API Server based on a predefined protocol and format, expressed in JSON or JSON-LD.

1.3 API Server

An API server implements the endpoint, methods, services which are publicly exposed and work as per the definition in the specification.

1.4 Vert.x

As defined in [1], Eclipse Vert.x is a set of tools for building reactive applications on the JVM. It is event-driven and non-blocking which means a server built using it can handle a lot of concurrencies using a small number of kernel threads. Vert.x lets the application scale with minimal hardware.

1.5 API Server

The API server is a faithful implementation of the API specification.

1.6 Client certificate

A client certificate is a certificate used to authenticate clients during an SSL handshake. It authenticates users who access a server by exchanging the client authentication certificate.

A SSL handshake usually involves a server certificate, where the server authenticates itself to the client. However, some APIs can also request for a client certificate during the SSL handshake in order to verify that the client is who they claim to be.

1.7 Event Bus

As defined in [2], A Vert.x event-bus is a light-weight distributed messaging system which allows different parts of your application, or different applications and services to communicate with each in a loosely coupled way. An event-bus supports publish-subscribe messaging, point-to-point messaging and request-response messaging.

1.8 Service

As defined in [3], A service is a discoverable functionality. It can be qualified by its type, metadata, and location. A service can be a database, a service proxy, an HTTP endpoint and any other resource you can imagine as soon as you can describe it, discover it and interact with it which is described by a Record. Under the hood, messages are sent on the event bus to invoke the service and get the response back. But for ease of use, it generates a proxy called a service proxy that we can invoke directly (using the API from the service interface).

2 Solution Architecture: Catalogue and Resource Server

This section describes the reference architecture for the data exchange API Server, interfaces of API server, components in this ecosystem. It also describes the Solution Architecture for the Catalogue and Resource Server where the responsibilities of various components and their interactions with other interfaces are explained.

2.1 API Server Architecture

IUDX API server will be implemented using Eclipse Vert.X framework. The generic API server architecture diagram is shown below. The same architecture is used for API components of both the Catalogue and Resource server. For a detailed description of Vert.X components and tools refer to the Vert.x documentation [4]. In this section, the components used in the IUDX API Server are briefly described.

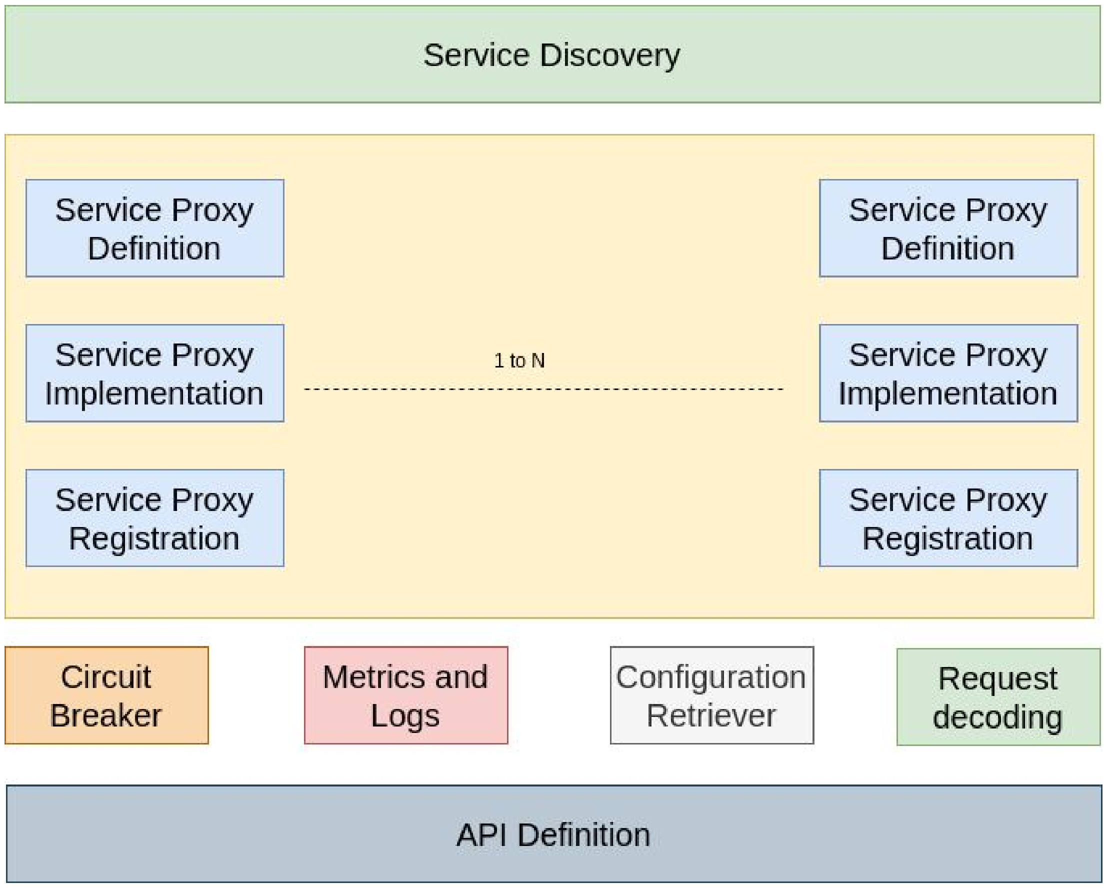


Figure 1 : API Server Reference Architecture

2.1.1 Service Discovery

A Service Discovery provides an infrastructure to publish and discover various services using the service proxy. In the API server, we have database service, data broker service, request decoding service, onboard service etc. Using the service discovery interface, each service can be exposed to the consumer using a service record. A service record contains the name, endpoint, status of the service. A service provider will update the record and the consumer will use the record to understand the ways to connect to the provider.

2.1.2 Service Proxy

A Service Proxy will allow us to isolate a functionality (a service) somewhere and make it available to be used from the rest of the application. For example, we can expose a database service on the event bus, and the API Server can consume it, as soon as it knows the address on which the database service is published.

2.1.3 Circuit Breaker

A circuit breaker can handle faults that might take a variable amount of time to recover when connecting to a remote service. For example, if the database is down, any call to the API Server to search for data in the database will have an issue. When we do not have a circuit breaker, all the calls from the API Server will be routed to the database service which will fail due to a timeout since the database is down. When we have a circuit breaker in place, the API server will know the state of the database service before routing the call and any issues related to the service can be handled by the API server itself. This can improve the stability and resiliency of an application. It will also allow API server to operate with few services even when one or few of the microservices are down. This limits the downtime of the Server and handles server faults effectively.

2.1.4 Metrics and Logs

A Metrics service allows monitoring of various components of the server when running in production. Vert.x Metrics Service Provider Interface (SPI) allows reporting metrics to open source components such as Dropwizard, Micrometer from components like HttpServer, NetServer etc. Since we are using the HttpServer for developing the API server, we can use the SPI for understanding the runtime metrics of the server.

2.1.5 Configuration Retriever

A Vert.x Configuration Retriever provides multiple ways to configure the server.

- For syntaxes, it provides options such as JSON, properties, Yaml (extension), Hocon (extension).
    - We use properties as a way to configure the services

    - Information such as endpoint, username, password etc will be served through the properties file
- For stores, it provides options such as files, directories, HTTP, git (extension), Redis (extension), system properties and environment properties.
    - We use HTTP to download the properties file to be used to set up the service.

2.1.6 Request Decoding and Response Module

The query decoding module implements the API decoding, service discovery, interacts with the event bus to execute the service, responds to the application after executing the request.

2.1.7 API Definition Module

The API definition module implements the endpoints as defined in the IUDX specification and also allows an end-user or application to interact with the API Server based on a predefined protocol and format.

2.1.8 Service Mesh Architecture

As a design choice on the architecture, keeping scalability for microservices in mind we chose the Service Mesh Architecture. In a service mesh architecture, each microservice is a well-defined module that can be containerized and discovered using service discovery. The orchestration of the services can be such that data-intensive modules are residing closer to the database which also helps in better response times, limits the bandwidth and reduces the cost. Also, it helps in scaling of a specific microservice at ease.
Other advantages include :

- ***Faster delivery***: Smaller modules, frequent releases with less risk
- ***Isolation***: Single service cannot crash the entire system
- ***Scaling***: We can scale individual services based on the use
- ***Culture***: Well defined ownership

2.2 Catalogue Server Solution Architecture

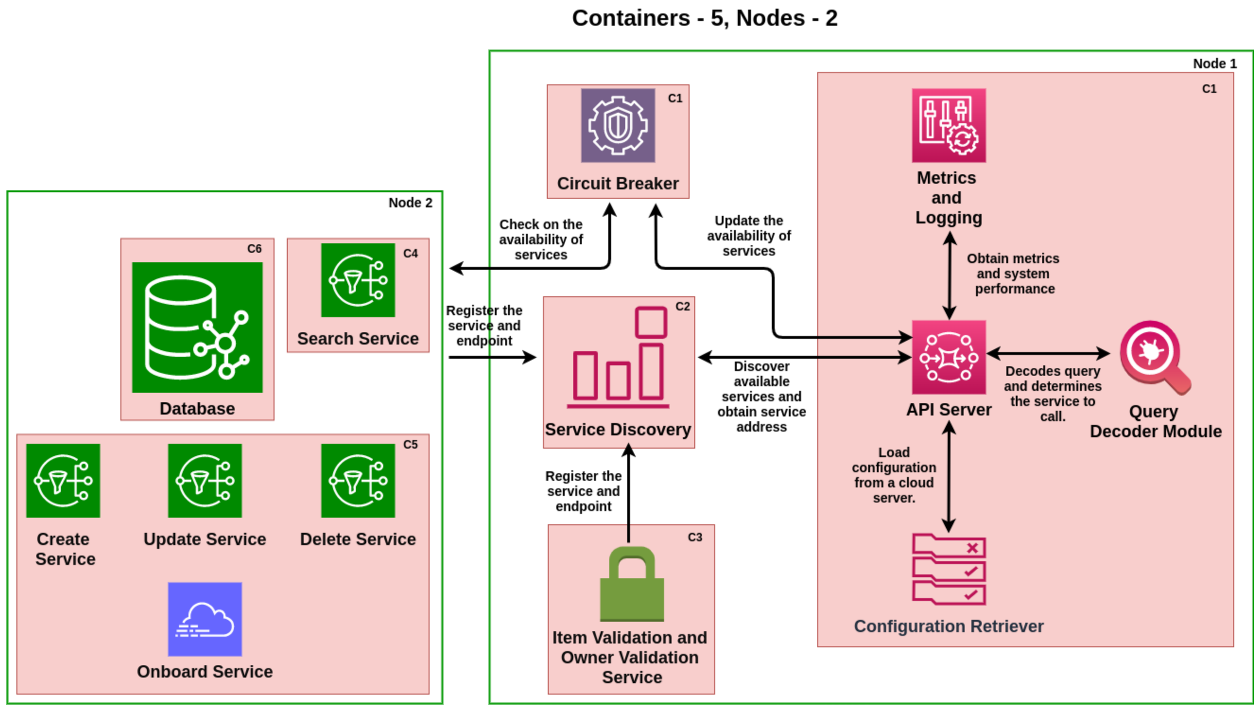

Containers - 5, Nodes - 2
Node 1
Node 2
C1
Circuit Breaker
C1
Metrics and Logging
Check on the availability of services
Update the availability of services
Obtain metrics and system performance
C6
Database
C4
Search Service
Register the service and endpoint
C2
Service Discovery
Discover available services and obtain service address
API Server
Decodes query and determines the service to call.
Query Decoder Module
Load configuration from a cloud server.
Configuration Retriever
C5
Create Service
Update Service
Delete Service
Onboard Service
Register the service and endpoint
C3
Item Validation and Owner Validation Service

Figure 2 : Catalogue Server Solution Architecture

A minimalistic setup of a catalogue server will consist of 5 docker containers in 2 nodes. Having a setup like this will allow specific high load containers to scale in ease.

2.2.1 Catalogue Server Components

2.2.1.1 API Server

An API server is a server-side web API implementation of the endpoint, methods, services which are publicly exposed and works as per the definition of the request-response messaging system. The APIs of the catalogue server are implemented as per the specification of IUDX.

2.2.1.2 Request Decoding and Response Module

The query decoding module implements the API decoding, service discovery, interacts with the event bus to execute the service, responds to the application after executing the request.

2.2.1.3 Database Search Service

A search service interacts with the Database, searches for documents and serves the request as per the API Definition.

Attribute Search

An attribute search is a module of search service that interacts with the Database, searches for documents using attribute queries and serves the request as per the API Definition.

Geo-Spatial Search

A geospatial search is a module of search service that interacts with the Database, searches for documents using geospatial queries and serves the request as per the API Definition.

Text Search

A text search is a module of search service that interacts with the Database, searches for documents using text queries and serves the request as per the API Definition.

Complex Search

A complex search is a module of search service that interacts with the Database, searches for documents using attribute and geospatial queries or text and geospatial queries and serves the request as per the API Definition.

Response Filter

A response filter is a module of search service that interacts with the Database, searches for documents using any of the above queries and serves the request as per the API Definition.

2.2.1.4 Database Item Creation Service

The items in a catalogue can be classified into the following item types

- Resource
- ResourceGroup
- ResourceServer
- Provider

Based on the itemType to be created, the database item creation service validates the provider using a certificate, validates the item based on the item type structure and connects with the database to create an item of itemType.

2.2.1.5 Database Item Update Service

Based on the itemType to be deleted, the database item update service validates the provider using a certificate, validates the item based on the item type structure and connects with the database to update an item of itemType.

2.2.1.6 Database Item Delete Service

Based on the itemType to be deleted, the database item deletion service validates the provider using a certificate, validates the provider and provider rules of the item based on the certificate and connects with the database to delete an item of itemType.

2.2.1.7 Item validation Service

An item validation service validates the item of an itemType for conformance to a specific document type. The catalogue does not accept in-valid documents and it is assured through this service.

2.2.1.8 Owner validation Service

An owner validation service verifies an onboarding user using their certificate or token. During an item update or deletion, this service plays a major role in access control.

2.2.1.9 Onboard Service

An onboard service helps in setting up the data publication pipeline by interacting with the IUDX Resource Server to create or delete exchanges, queues and bindings based on the operation in the catalogue.

2.3 Resource Server Solution Architecture

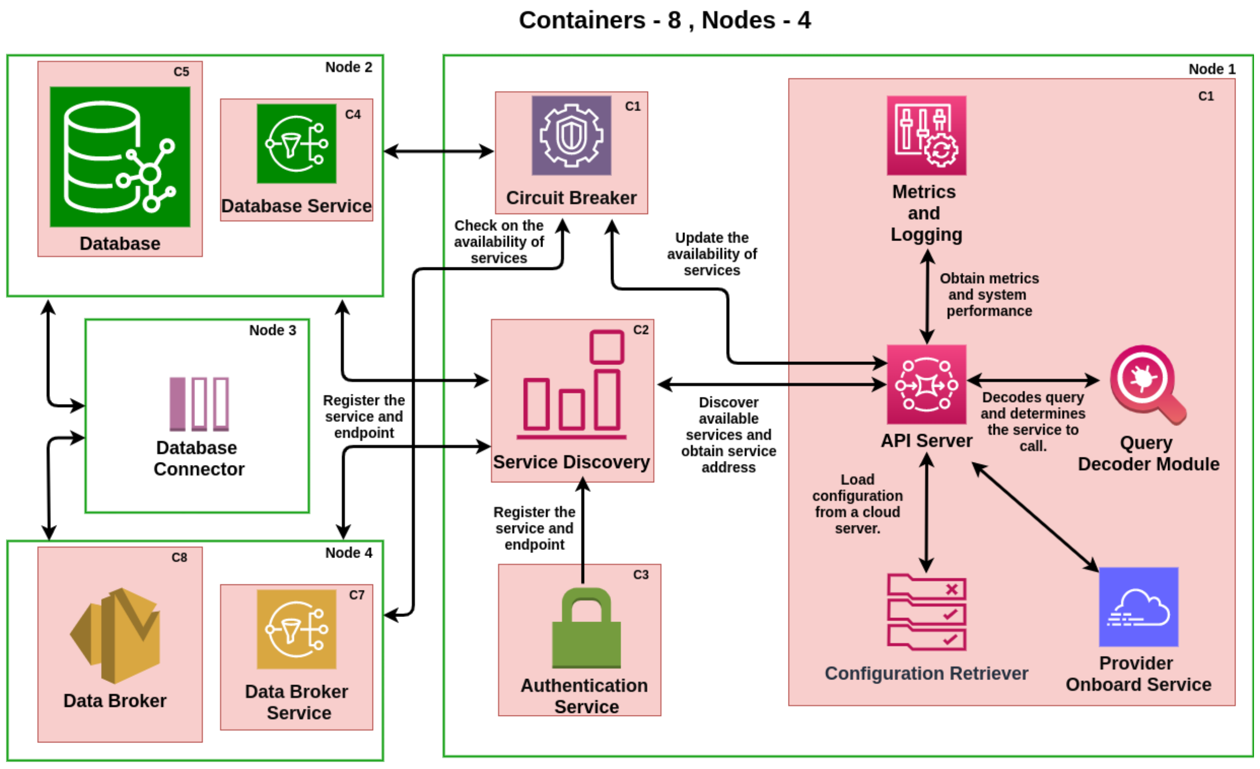

Containers - 8 , Nodes - 4
Node 2
C5
Database
C4
Database Service
Node 3
Database Connector
Register the service and endpoint
Node 4
C8
Data Broker
C7
Data Broker Service
Node 1
C1
Circuit Breaker
Check on the availability of services
Update the availability of services
C2
Service Discovery
Discover available services and obtain service address
Register the service and endpoint
C3
Authentication Service
C1
Metrics and Logging
Obtain metrics and system performance
API Server
Decodes query and determines the service to call.
Query Decoder Module
Load configuration from a cloud server.
Configuration Retriever
Provider Onboard Service

Figure 3 : Resource Server Solution Architecture

A minimalistic setup of a resource server will consist of 8 docker containers in 4 nodes. Having a setup like this will allow specific high load containers to scale in ease.

2.3.1 Resource Server Components

2.3.1.1 API Server

An API server is a server-side web API implementation of the endpoint, methods, services which are publicly exposed and works as per the definition of the request-response messaging system. The APIs, wherever applicable, of the resource server, is implemented as per the specification of NGSI-LD [5]. In scenarios where there is a requirement of new APIs, IUDX will recommend those which will be then adopted as NGSI-LD APIs.

2.3.1.2 Request Decoding and Response Module

The query decoding module implements the API decoding, service discovery, interacts with the event bus to execute the service, responds to the application after executing the request.

2.3.1.3 Database Search Service

A search service interacts with the Database, searches for documents and serves the request as per the API Definition.

Attribute Search
An attribute search is a module of search service that interacts with the Database, searches for documents using attribute queries and serves the request as per the API Definition.

Geo-Spatial Search
A geospatial search is a module of search service that interacts with the Database, searches for documents using geospatial queries and serves the request as per the API Definition.

Text Search
A text search is a module of search service that interacts with the Database, searches for documents using text queries and serves the request as per the API Definition.

Complex Search
A complex search is a module of search service that interacts with the Database, searches for documents using attribute and geospatial queries or text and geospatial queries and serves the request as per the API Definition.

Response Filter
A response filter is a module of search service that interacts with the Database, searches for documents using any of the above queries and serves the request as per the API Definition.

2.3.2 Data Broker Service

The Data Broker Service interacts with the data broker to create, update, delete exchanges, queues and bindings.

2.3.2.1 Create Exchange

A create exchange is a module of data broker service which enables the creation of exchange to which adaptors, sensor-gateway or sensors can publish data.

2.3.2.2 Delete Exchange

A delete exchange is a module of data broker service which enables deletion of an exchange.

2.3.2.3 Create Queue

A create queue is a module of data broker service which enables the creation of queues to which data published by adaptors, sensor-gateway or sensors can reside for the subscription.

2.3.2.4 Delete Queue

A delete queue is a module of data broker service which enables deletion of a queue.

2.3.2.5 Create Bindings

A create binding is a module of data broker service which enables binding between a sender (exchange) and a receiver (queue) based on a topic or ID.

2.3.2.6 Update Bindings

An update binding is a module of data broker service which updates an existing binding between a sender (exchange) and a receiver (queue) based on a topic or ID.

2.3.2.7 Delete Bindings

A delete binding is a module of data broker service which deletes or unbinds a topic or ID in the consumer [a receiver (queue)] from a sender (exchange).

2.3.3 Authentication Service

An authentication service connects with the IUDX Authentication server to validate a token using the Token Introspection Point (TIP). It also has a cache layer, taking token validity into consideration to avoid frequent calls to the Authentication Server.

2.3.4 Database Connector

The database connector interacts with the data broker to fetch data pushed by sensors/adaptors from the respective queues and writes it into the database collection as per the tenant policy.

2.3.5 Provider Onboard Service

A provider onboard service will help in setting up the policies on the pipeline for data ingestion. For eg. when an adaptor needs to publish data into the resource server, certain policies need to be enforced to allow a publication to exchanges.

2.4 Multi-tenancy

In HTTPs with every API request, by default, an instance ID is passed as a Host header field. Every document in the catalogue and every database collection or a data broker virtual host in the resource server has an instance ID field associated with it. With instance ID as the tenant ID, multi-tenancy is achieved.

2.5 Fault tolerance

Circuit Breaker can prevent an application from repeatedly trying to execute an operation that's likely to fail. Allowing it to continue without waiting for the fault to be fixed will incur unnecessary resource utilisation in the server. Circuit Breaker will enable a faulty service to be tracked, temporarily disables the service from accepting new tasks and allows faults in the service to be fixed. Once the fault has been resolved it will open the closed circuit and enable application modules to connect with the service which was temporarily down / faulty. More information on this can be found at [6].

2.6 Load Balancing

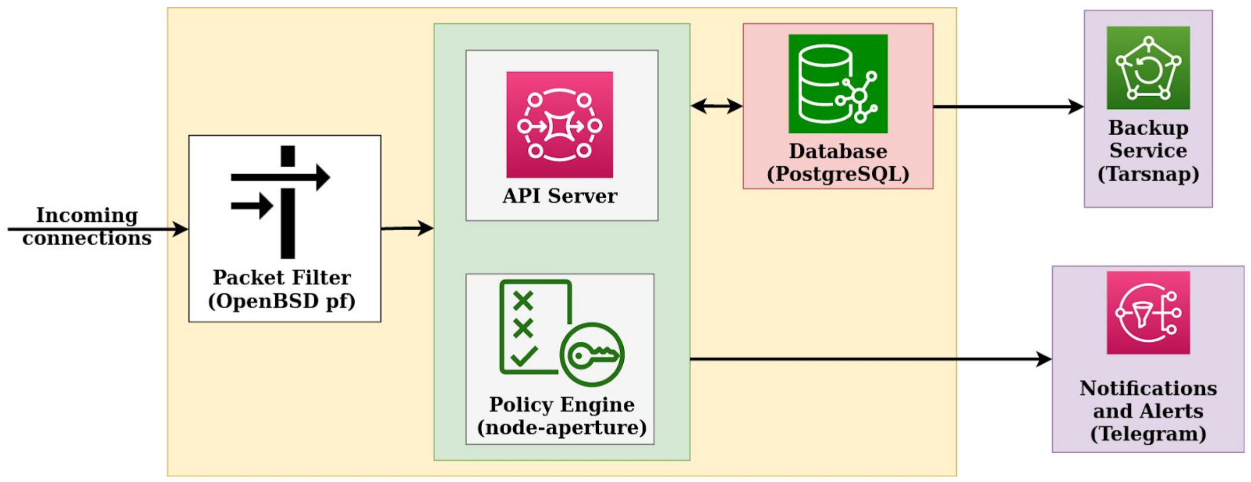
In a service mesh architecture, each service can be scaled as required and the reason why we chose to containerize each service was to take this design advantage. Though the design of the system plays a key role, it is the overlying docker layer which keeps the load of the system under control with predefined policies as per the system principles.

2.7 References

[1] Vert.x available at https://vertx.io/
[2] Vert.x Event Bus available at
https://vertx.io/preview/docs/kdoc/vertx/io.vertx.core.eventbus/-event-bus/
[3] Vert.x Service Discovery available at https://vertx.io/docs/vertx-service-discovery/java/
[4] Vert.x Documentation available at https://vertx.io/docs/
[5] NGSI-LD APIs available at
https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.01.01_60/gs_CIM009v010101p.pdf
(Need to update to latest specification)
[6] Circuit Breaker Pattern available at
https://docs.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker

3 Authorization Service Specifications and Architecture

3.1 Authorization server solution architecture


Incoming connections
Packet Filter (OpenBSD pf)
API Server
Policy Engine (node-aperture)
Database (PostgreSQL)
Backup Service (Tarsnap)
Notifications and Alerts (Telegram)

The Authentication, Authorization and Accounting (Auth-Server) system is built upon the OpenBSD operating system. The various components are described below.

3.1.1 API Server

The API server serves all the Auth related APIs. It is currently written in NodeJS. The server spawns a single master process and several worker processes to handle requests. The server makes use of OpenBSD security mechanisms to reduce the damage an attack can cause.

3.1.2 Policy Engine

The policy engine is used to evaluate requests based on policies set by the provider. It is based on node-aperture [0], which allows for policies to be written in a near plain-text syntax. A provider will set policies for their resources based on their requirements and conditions. Policies are evaluated when a client makes requests for certain resources owned by a provider.

3.1.3 Database

The database stores all token-related and policy-related information. A single instance of PostgreSQL has been used. Token-related information includes token hashes (only the has of the token is stored), resources related to the token and identity information of consumers. Policy-related information includes the policies set by various providers as well as groups of consumers that are managed by providers.

3.1.4 Packet Filter

A packet filter is used to perform rate-limiting on incoming connections and serves as a preliminary protection from spammed requests and denial of service attacks. The pf [1] tool provided by OpenBSD is used for this. If a client is making too many requests per second, the IP address of that client will be blocked for some period of time.

3.1.5 Backup Service

A backup of the database is taken daily and is pushed to Tarsnap [2]. Tarsnap allows for differential backups to be taken, i.e. only new data will be stored.

3.1.6 Notifications and Alerts

Telegram [3] is used as a notification channel to inform the admin of crashes or errors in functioning of the system. Telegram was chosen as it was easy to integrate with the system.

3.2 Authorization Service API Specifications

The Auth APIs expect the **METHOD** to be **POST**. Inputs/outputs if any, are expected to be in **JSON** format. The APIs return:

- **200** on success
- **400** on bad request
- **403** on unauthorized request

And on exceptions APIs return:

- **402** on payment required (due to insufficient credits)
- **429** on too many requests
- **500** on internal error

IUDX Authorization server APIs can be called by:

- Data providers (the resource owners)
- Data consumers (client/users who wish to access one or more data provider's data)
- Resource servers (which hosts the data provider's data)

through HTTPS using a valid client-side X.509 certificate.

The complete list of APIs is as follows:

#	Endpoint	Description	Can be called by
1	/auth/v1/acl/set	Set access control policies	Data provider
2	/auth/v1/acl/revert	Revert to previous access control policy	Data provider
3	/auth/v1/acl/append	Append to existing access control policies	Data provider
4	/auth/v1/acl	Get the current list of access control policies	Data provider
5	/auth/v1/token	Request for an access token	Data consumer
6	/auth/v1/token/introspect	Verify a token	Resource server
7	/auth/v1/token/revoke	Revoke a list of tokens	Both data providers and consumers
8	/auth/v1/token/revoke-all	Revoke all tokens associated with a certificate	Both data providers and consumers
9	/auth/v1/audit/tokens	Audit tokens	Both data providers and consumers
10	/auth/v1/group/add	Add a consumer to a group	Data provider
11	/auth/v1/group/delete	Delete a consumer from a group	Data provider
12	/auth/v1/group/list	List all valid members of a group	Data provider

13	/auth/v1/certificate-info	Get user's certificate details	Anyone

Certain APIs can be accessed only by a certain “class” of certificate. The IUDX Certificate Authority [4] currently issues 3 classes of certificates:

- **Class-1** : For resource servers (can only introspect tokens)
- **Class-2** : For consumers (can only request for tokens)
- **Class-3** : For providers, data officers, and consumers

The Auth server accepts certificates issued by the IUDX CA. It also accepts certificates issued by licensed CAs in India [5], and other trusted CAs. It categorises them into classes based on the certificate:

- **Class-1** : Any valid certificate
- **Class-2** : Any valid certificate with an email address

(Currently, Class-3 certificates can only be issued by IUDX CA)

The following sections provide details of the authorization server APIs.

3.2.1 Access Control

Access control list (acl) is a list of data sharing policies for a provider.

3.2.1.1 Create a policy for a provider

Endpoint	**/auth/v1/acl/set**
Called by	Data Provider (Class-3)
Method	POST
Status Code	`200 OK` - If policy has been set successfully `400 Bad Request` - If the policy contains syntax errors
API help	http://auth.iudx.org.in/acl-set.html

Request	`https://auth.iudx.org.in/auth/v1/acl/set`
	
	Header:
	`content-type: application/json`
	
	Body:
	`{"policy":"barun@iisc.ac.in can access pune.iudx.org.in/streetlight-1 for 10 days"}`
	
	(Multiple rules can be added with ';' e.g
	`{"policy":"barun@iisc.ac.in can access pune.iudx.org.in/streetlight-1 for 10 days;* can access pune.iudx.org.in/aqm for 1 hour"})`

Response	`200 OK`
	`{"success":true}`

3.2.1.2 Append to an access control policy for a provider

Endpoint	**/auth/v1/acl/append**
Called by	Data Provider (Class-3)
Method	POST
Status Code	`200 OK` - If policy has been appended successfully `400 Bad Request` - If the policy contains syntax errors
API help	http://auth.iudx.org.in/acl-append.html

Request	`https://auth.iudx.org.in/auth/v1/acl/append`
	
	Header:
	`content-type: application/json`
	
	Body:
	`{"policy":"barun@iisc.ac.in can access pune.iudx.org.in/streetlight-1 for 10 days"}`
	
	(Multiple rules can be added with ';' e.g

	`{"policy":"barun@iisc.ac.in can access pune.iudx.org.in/streetlight-1 for 10 days;* can access pune.iudx.org.in/aqm for 1 hour"})`

Response	`200 OK`
	`{"success":true}`

3.2.1.3 Revert to previous policy

Endpoint	**/auth/v1/acl/set**
Called by	Data Provider (Class-3)
Method	POST
Status Code	`200 OK` - If policy has been reverted successfully `400 Bad Request` - If there are no saved "`previous`" policy
API help	http://auth.iudx.org.in/acl-revert.html
Request	`https://auth.iudx.org.in/auth/v1/acl/revert`

Response	`200 OK`
	`{"success":true}`

3.2.1.4 List all policies of a provider.

Endpoint	**/auth/v1/acl**
Called by	Data Provider (Class-3)
Method	GET
Status Code	`200 OK` - If policy has been retrieved successfully `400 Bad Request` - If no policy has been set by provider
API help	http://auth.iudx.org.in/acl.html

Request	`https://auth.iudx.org.in/auth/v1/acl`
Response	`200 OK` `{ "policy": [<The list of policies>], "previous-policy": [<List of policies before most recent update>], "last-updated": <Timestamp when policies were last updated>, "api-called-from": <Site from which policy was updated> }`

3.2.2 Access Tokens

3.2.2.1 Get an access token

Endpoint	**/auth/v1/token**
Called by	Data Consumer (Class-2 and above)
Method	POST
Status Code	`200 OK` - If access is allowed, and token issued `400 Forbidden` - If access to requested resource denied `429 Too Many Requests` - If client makes too many requests
API help	http://auth.iudx.org.in/token.html
Request	`https://auth.iudx.org.in/auth/v1/token` **Header:** `content-type: application/json` **Body:** The request body can be structured in two ways 1.Simple `{` `"request" : <resource id(s)>                // required`

```
"token-time": <requested_token_validity>    // optional
}
```

2.Complex

```
{
  "request" :
  {
    "id"          : <resource id>            // required
    "apis"        : <array of APIs>          // optional
    "methods" : <array of methods>          // optional
    "body"        : <dictionary of body variables to be
called with the API>                        // optional
  },
  "token-time" : <requested_token_validity>   //optional
}
```

The "`request`" field could also be an "array" of strings/objects.

Response

```
200 OK

{
  "access_token":  <token>,
  "token_type": " IUDX",
  "expires_in": <expiry-time-in-seconds>,
  “server_token” : {<server>:<token>,...}
}
```

3.2.3 Managing tokens

3.2.3.1 Revoke a valid token

End-Point	**/auth/v1/token/revoke**
Called by	Data Providers and Data Consumers (Class-3)
Method	POST
Status Code	`200 OK` - If token is successfully revoked `403 Not Found` - If token is invalid
API help	http://auth.iudx.org.in/token-revoke.html

Request	`https://auth.iudx.org.in/auth/v1/token/revoke`
	
	Body:
	`{"tokens": [<list of-tokens(or token hashes)-to-be-revoked>]}`

Response	`200 OK`
	
	`{"success":true}`

3.2.3.2 Revoke all valid tokens of a consumer

End-Point	**/auth/v1/token/revoke-all**
Called by	Data Providers and Data Consumers (Class-3)
Method	POST
Status Code	`200 OK` - If token is successfully revoked `400 Bad Request` - If inputs are invalid
API help	http://auth.iudx.org.in/token-revoke-all.html

Request	`https://auth.iudx.org.in/auth/v1/token/revoke-all`
	
	Body:
	`{"serial": "<certificate-serial-of-consumer>", "fingerprint" : "<sha1-fingerprint-of-the-certificate>"}`

Response	`200 OK`
	`{"success":true}`

3.2.3.3 Introspect a token

End-Point	**/auth/v1/token/introspect**
Called by	Resource Server (Class-1)
Method	POST
Status Code	`200 OK` - If the token is valid `403 Not Found` - If the token is invalid/expired OR The "`CN`" of the resource-server certificate does not match the IP address of the machine which is calling this API `429 Too Many Requests` - If the resource-server makes too many requests
API help	http://auth.iudx.org.in/token-introspect.html

Request	`https://auth.iudx.org.in/auth/v1/token/introspect`
	
	Body:
	`{`
	`"token”: <token presented by the consumer>// required`
	`"server-token": <server token presented by the consumer>  // optional`
	`"request": <the request that was sent by the consumer> // optional`
	`}`

Response	`200 OK`
	`{`
	`   "consumer":"<consumer-email>",`
	`   "expiry":"<expiry-timestamp>",`
	`   "request":[`
	`      {      "id":"<id>",`
	`         "apis":[<apis>],`
	`         "body":<body>,`
	`         "methods":[<methods>]}`
	`,...],`
	`"consumer-certificate-class":<cert-class-of-consumer>`

	`}`

3.2.3.4 Audit tokens

End-Point	**/auth/v1/audit/tokens**
Called by	Data Providers and Data Consumers (Class-3)
Method	POST
Status Code	`200 OK` - If audit report has been successfully fetched `403 Not Found` - If the certificate class is < 3
API help	http://auth.iudx.org.in/audit-tokens.html

Request	`https://auth.iudx.org.in/auth/v1/audit/tokens`
	
	Body:
	`{"hours":`
	`<number-of-hours-for-which-audit-report-has-to-be-gene`
	`rated>}`

Response	`200 OK`
	`{`
	`  "consumer": "<consumer>",`
	`  "token-hash": "<token-hash>",`
	`  "token-issued-at": "<time>",`
	`  "introspected": false,`
	`  "revoked": false,`
	`  "expiry": "<time>",`
	`  "expired": true,`
	`  "certificate-serial-number": "<serial-num>",`
	`  "certificate-fingerprint": "<fingerprint>",`
	`  "request": [<ids, methods, apis requested for this`
	`token> ],`
	`  "geoip": {`
	`      "ll": [<lat,long>],`
	`      "city": "<city>",`
	`      "region": "<region like KA>",`
	`      "country": "<country like IN>",`

	`"timezone": "<timezone like Asia/Kolkata>"`
	`},`
	`"paid": true,`
	`"api-called-from": null`
	`}`

3.2.4 Managing Groups

3.2.4.1 Add consumer to a group

End-Point	**/auth/v1/group/add**
Called by	Data Provider (Class-3)
Method	POST
Status Code	`200 OK` - If group is created `400 Bad Request` - If inputs are invalid
API help	http://auth.iudx.org.in/group-add.html

Request	`https://auth.iudx.org.in/auth/v1/group/add`
	
	Body:
	`{`
	`"consumer": <the name of the consumer>`
	`"group": <the name of the group to which the consumer has to be added>`
	`"valid-till": <the number of hours for which the group membership is valid>`
	`}`

Response	`200 OK`
	`{"success":true}`

3.2.4.2 Delete consumer from a group

End-Point	**/auth/v1/group/delete**
Called by	Data Provider (Class-3)
Method	POST
Status Code	`200 OK` - If consumer is deleted `400 Bad Request` - If consumer does not exist in group
API help	http://auth.iudx.org.in/group-delete.html

Request	`https://auth.iudx.org.in/auth/v1/group/delete`
	
	Body:
	`{`
	`"consumer": <the name of the consumer>`
	`"group": <the name of the group from which the consumer has to be deleted>`
	`}`
	
	If “`consumer`” is `*`, then group will be deleted

Response	`200 OK`
	`{`
	`“num-consumers-deleted" : <no.-of-consumers-deleted>`
	`}`

3.2.4.3 List consumers in a group.

End-Point	**/auth/v1/group/list**
Called by	Data Provider (Class-3)
Method	POST
Status Code	`200 OK` - If list of consumers retrieved

API help	http://auth.iudx.org.in/group-list.html

Request	`https://auth.iudx.org.in/auth/v1/group/list`
	
	Body:
	`{"group": “<name-of-the-group>”}`

Response	`200 OK`
	`[`
	`   {"consumer":"<consumer-name>",`
	`“valid-till”:”<till-when-consumer-is-part-of-group>”`
	`}, … ]`

3.2.5 Miscellaneous

3.2.5.1 Get certificate information

Endpoint	**auth/v1/certificate-info**
Called by	Anyone with a valid certificate
Method	POST
Status Code	`200 OK` - If request successful `400 Bad Request` - If access denied
API help	http://auth.iudx.org.in/certificate-info.html
Request	`https://auth.iudx.org.in/auth/v1/certificate-info`

Response	`200 OK`
	
	`{`
	`"id": <email-id>,`
	`"certificate-class": <certificate-class>`
	`"serial": <serial>,`
	`"fingerprint": <fingerprint>`
	`}`

3.3 References

[0] https://github.com/rbccps-iisc/node-aperture, forked from https://github.com/joyent/node-aperture
[1] https://www.openbsd.org/faq/pf/
[2] https://www.tarsnap.com/
[3] https://core.telegram.org/bots
[4] https://ca.iudx.org.in/
[5] http://cca.gov.in/licensed_ca.html

4 Resource Server and Catalogue API Specifications

Resource Server API Documentation

The resource server provides data access through search, count, subscription APIs. The APIs for these functionalities can be constructed using (1) Resource Server ID, (2) Resource Group ID, and (3) Verbs, Queries and Filters.

Authentication and Authorization for the resource server interface is through the use of IUDX tokens issued by the IUDX Authorization Server. All the APIs of the resource server interface accept the IUDX auth token using the “token” header. If a token is not provided then the APIs operate only on publicly available data sets or service-offerings. However, when a token is supplied, the resource server interface discerns the scope of the token after contacting the IUDX auth server and performs operations on all those resources instead of only restricting the operation to public resources.

Table of Contents

ComplexSearchAPIs

- `POST /entities/search`

SimpleSearchAPIs

- `GET /entities`
- `GET /entities/{entityId}`

SubscriptionAPIs

- `POST /subscriptions`
- `DELETE /subscriptions/{subscriptionId}`
- `GET /subscriptions/{subscriptionId}`
- `PATCH /subscriptions/{subscriptionId}`

ComplexSearchAPIs

`POST /entities/search`

(**complexSearch**)

Obtain results for a complex search.

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body **standardQuery** **(required)**

Body Parameter —

Return type
standardDataResponse

Example data
Content-Type: application/json

```
[ {
  "id" : "string",
  "type" : "string",
  "location" : {
    "type" : "GeoProperty",
    "value" : { },
    "observedAt" : "2020-06-03T09:37:20.049Z"
  },
  "observationSpace" : {
    "type" : "GeoProperty",
    "value" : { },
    "observedAt" : "2020-06-03T09:37:20.049Z"
  },
  "operationSpace" : {
    "type" : "GeoProperty",
    "value" : { },
    "observedAt" : "2020-06-03T09:37:20.049Z"
  },
  "@context" : "Unknown Type: string,object,array",
  "createdAt" : "2020-06-03T09:37:20.049Z",
  "modifiedAt" : "2020-06-03T09:37:20.049Z"
} ]
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Query successful standardDataResponse
400
Bad query. standardBadQueryResponse
404
No query can be performed. standardBadQueryResponse

SimpleSearchAPIs

GET /entities

(**search**)

Search for data which matches the given query.

Query parameters

id (required)
Query Parameter — ID of the entity

type (optional)
Query Parameter — Comma separated list of Entity type names to be retrieved

idPattern (optional)
Query Parameter — Regular expression that must be matched by Entity ids

attrs (optional)
Query Parameter — Comma separated list of attribute names (properties or relationships) to be retrieved

q (optional)
Query Parameter — Query

geoRel (optional)
Query Parameter — Geo Relationship

geometry (optional)
Query Parameter — Geometry type, for e.g, linestring, bbox, polygon.

coordinates (optional)
Query Parameter — Coordinates for the given geometry. It should be a string encoded multi dimensional array.

geoproperty (optional)
Query Parameter — The geoproperty on which the geo query is to be performed.

Return type

standardDataResponse

Example data

Content-Type: application/json

```
[ {
  "id" : "string",
  "type" : "string",
  "location" : {
    "type" : "GeoProperty",
    "value" : { },
    "observedAt" : "2020-06-03T09:37:20.049Z"
  },
  "observationSpace" : {
    "type" : "GeoProperty",
    "value" : { },
    "observedAt" : "2020-06-03T09:37:20.049Z"
  },
  "operationSpace" : {
    "type" : "GeoProperty",
    "value" : { },
    "observedAt" : "2020-06-03T09:37:20.049Z"
  },
  "@context" : "Unknown Type: string,object,array",
  "createdAt" : "2020-06-03T09:37:20.049Z",
  "modifiedAt" : "2020-06-03T09:37:20.049Z"
} ]
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200
Query is successful. standardDataResponse
400
Bad query. standardBadQueryResponse

GET /entities/{entityId}

(**simpleSearch**)

Search for data which matches the given query.

Path parameters

entityId (required)
Path Parameter — Id of the entity.

Query parameters

attrs (optional)
Query Parameter — Comma separated list of attribute names (properties or relationships) to be retrieved

Return type

standardDataResponse

Example data

Content-Type: application/json

```
[ {
  "id" : "string",
  "type" : "string",
  "location" : {
    "type" : "GeoProperty",
    "value" : { },
    "observedAt" : "2020-06-03T09:37:20.049Z"
  },
  "observationSpace" : {
    "type" : "GeoProperty",
    "value" : { },
    "observedAt" : "2020-06-03T09:37:20.049Z"
  },
  "operationSpace" : {
    "type" : "GeoProperty",
    "value" : { },
    "observedAt" : "2020-06-03T09:37:20.049Z"
  },
  "@context" : "Unknown Type: string,object,array",
  "createdAt" : "2020-06-03T09:37:20.049Z",
  "modifiedAt" : "2020-06-03T09:37:20.049Z"
} ]
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200
Query is successful. standardDataResponse
400
Bad query. standardBadQueryResponse
404
No entity with the id is found. standardBadQueryResponse

SubscriptionAPIs

POST /subscriptions

(**subscriptionsPost**)

Register a subscription.

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body subscriptionDescription (required)
Body Parameter —

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

201
Subscription has been registered.
400
Bad subscription request. standardBadQueryResponse
409
Subscription already exists.

DELETE /subscriptions/{subscriptionId}

(**subscriptionsSubscriptionIdDelete**)

Delete subscription.

Path parameters

subscriptionId (required)
Path Parameter — Id of the subscription.

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

204
Subscription is deleted.
400
Bad request. standardBadQueryResponse
404
No such subscription found. standardBadQueryResponse

GET /subscriptions/{subscriptionId}

(**subscriptionsSubscriptionIdGet**)

Subscribe to data from a resource server by matching a query. If subscription exists, it retrieves the subscriptions available in the system.

Path parameters

subscriptionId (required)
Path Parameter — Id of the subscription.

Return type

subscriptionDescription

Example data

Content-Type: application/json

```
{
  "id" : "string",
  "name" : "string",
  "description" : "string",
  "watchedAttributes" : [ "string" ],
  "timeInterval" : 0,
  "geoQ" : {
    "geometry" : "string",
    "coordinates" : [ 0 ],
    "georel" : "string"
  },
  "notification" : {
    "attributes" : [ "string" ],
    "format" : "string",
    "endpoint" : {
      "uri" : "string",
      "accept" : "application/json"
    },
    "timesSent" : 0,
    "lastNotification" : "2020-06-03T09:51:42.022Z",
    "lastFailure" : "2020-06-03T09:51:42.022Z",
    "lastSuccess" : "2020-06-03T09:51:42.022Z"
  },
  "expires" : "2020-06-03T09:51:42.022Z",
  "status" : "active",
  "throttling" : 0,
  "q" : "string",
  "@context" : "Unknown Type: string,object,array",
  "type" : "Subscription",
  "entities" : [ {
    "id" : "string",
    "idPattern" : "string",
    "type" : "string"
  } ]
}
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200
List of subscriptions. subscriptionDescription

PATCH /subscriptions/{subscriptionId}

(**subscriptionsSubscriptionIdPatch**)

Updates a specific Subscription within a system. The additional subscription parameters must include the id.

Path parameters

subscriptionId (required)
Path Parameter — Id of the subscription.

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body subscriptionDescription (required)
Body Parameter —

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

204
Subscription updated successfully.
400
Bad subscription update request. standardBadQueryResponse
404
No subscription exists to update. standardBadQueryResponse

Models

Table of Contents

1. `standardBadQueryResponse` - Root Type for standardBadQueryResponse
2. `standardDataResponse` - Root Type for standardDataResponse
3. `standardDataResponse_inner`
4. `standardQuery` - Root Type for standardQuery
5. `subscriptionDescription` - Root Type for subscriptionDescription
6. `subscriptionDescription_entities`
7. `subscriptionDescription_geoQ`
8. `subscriptionDescription_notification`
9. `subscriptionDescription_notification_endpoint`

`standardBadQueryResponse` - Root Type for standardBadQueryResponse

Response for bad queries.

type (optional)
String

title (optional)
String

detail (optional)
String

`standardDataResponse` - Root Type for standardDataResponse

Standard data response for an entities query

`standardDataResponse_inner`

id (optional)
String

type (optional)
String

location (optional)
Object

observationSpace (optional)
Object

operationSpace (optional)
Object

@context (optional)
String

createdAt (optional)
Date format: date-time

modifiedAt (optional)
Date format: date-time

`standardQuery` - Root Type for standardQuery

A standard complex query body.

id (optional)
String

type (optional)
String

idPattern (optional)
String

attrs (optional)
String

q (optional)
String

georel (optional)
String

geometry (optional)
String

coordinates (optional)
String

geoproperty (optional)
String

`subscriptionDescription` - Root Type for subscriptionDescription

Payload describing information required to register a subscription.

id (optional)

String

name (optional)
String

description (optional)
String

watchedAttributes (optional)
array[String]

timeInterval (optional)
Integer format: int32

geoQ (optional)
subscriptionDescription_geoQ

notification (optional)
subscriptionDescription_notification

expires (optional)
Date format: date-time

status (optional)
String

throttling (optional)
Integer format: int32

q (optional)
String

@context (optional)
String

type (optional)
String

entities (optional)
array[subscriptionDescription_entities]

`subscriptionDescription_entities`

id (optional)
String

idPattern (optional)
String

type (optional)
String

`subscriptionDescription_geoQ`

geometry (optional)
String

coordinates (optional)
array[Integer] format: int32

georel (optional)
String

`subscriptionDescription_notification`

attributes (optional)

array[String]

format (optional)
String

endpoint (optional)
subscriptionDescription_notification_endpoint

timesSent (optional)
Integer format: int32

lastNotification (optional)
Date format: date-time

lastFailure (optional)
Date format: date-time

lastSuccess (optional)
Date format: date-time

`subscriptionDescription_notification_endpoint`

uri (optional)
String

accept (optional)
String

Catalogue API Documentation

The information resource catalogue contains the meta-data of resources along with auxiliary descriptions, API endpoints, data models and other meta-information like discovery hints, location details, providers etc. More details can be found in the IUDX specification

Authentication and Authorization for the catalogue interface is achieved through tokens which is obtained using the client side certificates, issued by the IUDX Certificate Authority. Apart from search APIs, all the other APIs requires a token.

The API documentation is written using swagger, where the defenitions, methods, endpoints, query and path parameters are described.

Table of Contents

ConfigurationAPIs

- `GET /cities`
- `GET /ui/config/`
- `GET /ui/cities`
- `DELETE /ui/config/`
- `POST /ui/config/`
- `PUT /ui/config/`

MetaInformationOnboardingAPIs

- `DELETE /items/{id}`
- `POST /items`

SearchAPIs

- `GET /items/{id}`
- `GET /search`

ConfigurationAPIs

`GET /cities`

(**citiesGet**)

Request headers

`token (required)`

Return type

listOfCities

Example data

Content-Type: application/json

```
[ "id1", "id2" ]
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
List of cities listOfCities

GET /ui/config/

(**getconfig**)

Get a UI configuation file for an instance id.

Request headers

```
token (required)
```

Query parameters

instanceid (required)
Query Parameter — Instance id for which config is required.

Return type
citiesResponseBody

Example data
Content-Type: application/json

```
{
  "status" : "success",
  "results" : [ {
    "__instance-id" : "pudx.catalogue.iudx.org.in",
    "configurations" : {
      "smart_city_name" : "PSCDCL",
      "map_default_view_lat_lng" : [ 18.5644, 73.7858 ]
    }
  } ]
}
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
The query yielded results. citiesResponseBody
204
Configuration was not loaded. invalidSearchResponesBody
400
Bad query.

GET /ui/cities

(**uiCitiesGet**)

Get all cities registered to a catalogue.

Request headers

```
token (required)
```

Return type

citiesResponseBody

Example data

Content-Type: application/json

```
{
  "status" : "success",
  "results" : [ {
    "__instance-id" : "pudx.catalogue.iudx.org.in",
    "configurations" : {
      "smart_city_name" : "PSCDCL",
      "map_default_view_lat_lng" : [ 18.5644, 73.7858 ]
    }
  } ]
}
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200
The query yielded results. citiesResponseBody
204
No cities were registered. invalidSearchResponesBody
400
Bad query.

DELETE /ui/config/

(**uiConfigDelete**)

Delete a configuration.

Request headers

```
token (required)
```

Query parameters

instanceid (required)
Query Parameter — Instance id whose config should be deleted.

Responses

204
Config deleted.

POST /ui/config/

(**uiConfigPost**)

Insert a configuration for an instance.

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body [configBody](#) (required)
Body Parameter —

Request headers

Query parameters

instanceid (required)
Query Parameter — Instance id for which config is to be inserted.

Responses

201
Config created.

`PUT /ui/config/`

(**uiConfigPut**)

Update a configuration.

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body [configBody](#) (required)
Body Parameter —

Request headers

`token (required)`

Query parameters

instanceid (required)
Query Parameter — Instance id for which config needs to be updated.

Responses

201
Successfully updated.
400
Invalid body.

MetaInformationOnboardingAPIs

`DELETE /items/{id}`

(**itemsIdDelete**)

Delete an item.

Path parameters

id (required)
Path Parameter — id of the item

Request headers

Responses

204
Successfully deleted.
401
Not allowed to delete item.
404
Didn't find the item.

POST /items

(**itemsPost**)

Insert an item into the catalogue. The item's body specifies the type of item.

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body item (required)
Body Parameter —

Request headers

`token (required)`

Return type

item

Example data

Content-Type: application/json

```
{
  "id" : "hash/group/resourceId"
}
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

201
Successfully inserted. item
400
Schema of item didn't match.
401
Don't have sufficient permission to insert items into the catalogue.

SearchAPIs

`GET /items/{id}`

(**getItem**)

Get the item.

Path parameters

id (required)
Path Parameter — id of the item

Return type

item

Example data

Content-Type: application/json

```
{
  "id" : "hash/group/resourceId"
}
```

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200
Found the item. item
404
Didn't find the item.

`GET /search`

(**searchGet**)

Search items in the catalogue

Query parameters

property (optional)
Query Parameter — Array of properties on which query is to be made. The mapping between a property and a value is one-to-one.

value (optional)
Query Parameter — Values of the one-to-one mapped properties.

geoproperty (optional)
Query Parameter — Name of the property if the query is on a geospatial property.

georel (optional)
Query Parameter — Type of geoquery, for e.g, within, near, outside.

geometry (optional)
Query Parameter — Geometry of the geo-query, for e.g, linstring, bbox, polygon.

coordinates (optional)
Query Parameter — Coordinates for the specific query type. For e.g, [[1,2], [3,4]]. Note: The data-type of this field is a string, therefore the coordinates are a string encoded multidimensional array.

q (optional)
Query Parameter — query for text/fuzzy search.

limit (optional)
Query Parameter — Limit number of records in search response.

offset (optional)
Query Parameter — Offset from where the next batch of results should be returned.

Return type
searchResponseBody

Example data
Content-Type: application/json

```
{
  "status" : "success",
  "totalHits" : 200,
  "limit" : 1,
  "offset" : 100,
  "results" : [ { } ]
}
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
All the query parameters yielded results. searchResponseBody
204
None of the query parameters yielded results.
206
Some of the query parameters yielded results. searchResponseBody
400
Bad query.

Models

Table of Contents

1. `citiesResponseBody` - Root Type for citiesResponseBody
2. `citiesResponseBody_configurations`
3. `citiesResponseBody_results`
4. `configBody` - Root Type for configBody
5. `configBody_configurations`
6. `configResponse`
7. `invalidSearchResponesBody` - Root Type for invalidSearchResponesBody
8. `item` - Root Type for itemResponse
9. `listOfCities` - Root Type for listOfCities
10. `searchResponseBody` - Root Type for searchResponseBody

`citiesResponseBody` - Root Type for citiesResponseBody

Response body for a ui/cities query

status (optional)
String

results (optional)
array[citiesResponseBody_results]

`citiesResponseBody_configurations`

smart_city_name (optional)
String

map_default_view_lat_lng (optional)
array[Double] format: double

`citiesResponseBody_results`

__instance-id (optional)
String

configurations (optional)
citiesResponseBody_configurations

`configBody` - Root Type for configBody

Body of the configuration request

instanceId (optional)
String

configurations (optional)
configBody_configurations

`configBody_configurations`

cat_base_URL (optional)
String

`configResponse`

Response of the configuration request.

`invalidSearchResponesBody` - Root Type for invalidSearchResponesBody

Response body for invalid requests.

status (optional)
String

results (optional)
array[null]

`item` - Root Type for itemResponse

A catalogue item.

id (optional)
String

`listOfCities` - Root Type for listOfCities

List of cities.

`searchResponseBody` - Root Type for searchResponseBody

Response body of a search request.

status (optional)
String

totalHits (optional)
Integer format: int32

limit (optional)
Integer format: int32

offset (optional)
Integer format: int32

results (optional)
array[Object]