SURFeduhub RIO Mapper Ontwerp

Remco van 't Veer, Joost Diepenmaat

2022-07-08

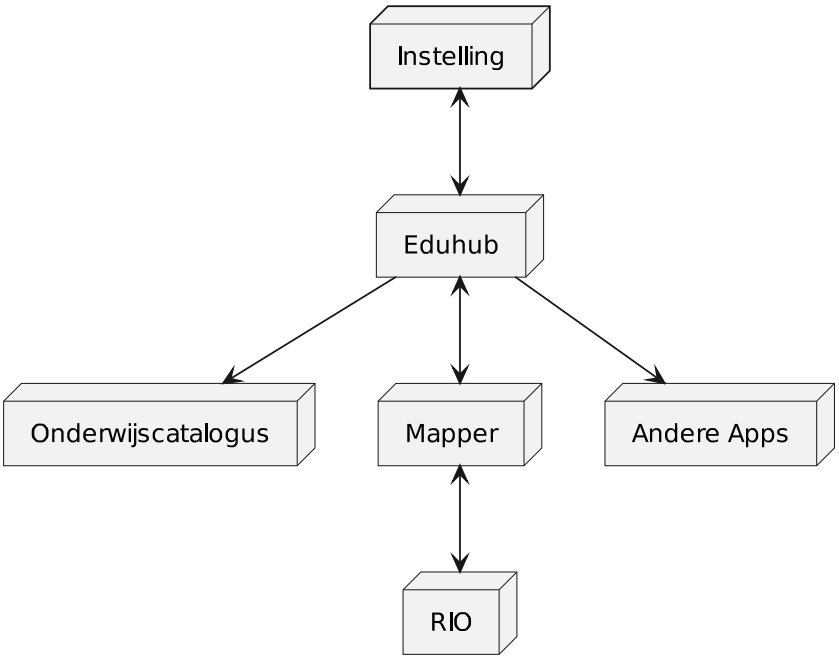
Inhoudsopgave

Geschiedenis

2022-02-17	Klad - ruwe opzet ter bespreking
2022-03-14	Beta versie - authenticatie ter bespreking
2022-03-15	Beta versie - tantoe spang
2022-03-23	Performance, eisen en specificaties toegevoegd
2022-04-01	Opmerkingen over schalen en availability verwerkt
2022-04-04	Conext autorisatie mechanisme uitgebreid
2022-07-08	API aanpassingen
2022-07-19	"delete" actie en "done" attributen toegevoegd

1 In het kort

De RIO Mapper is een koppeling tussen instellingen en RIO[1] om opleiding informatie van instellingen uit het hoger onderwijs geautomatiseerd aan te kunnen bieden aan RIO. Instellingen geven bij de SURFeduhub[2] RIO Mapper aan welke entiteit (vak, opleiding, etc.) er veranderd is en de Mapper zorgt dat deze aanpassing in RIO terecht komt. De gegevens van de instellingen worden via de OOAPI standaard uitgevraagd.



Figuur 1: Overzicht van mapper in het Eduhub netwerk

[1] Registratie Instellingen en Opleidingen
[2] Voorheen Open Onderwijs API gateway

2 Mapping van OOAPI naar RIO

2.1 Specificatie van mapping

Entiteiten in de OOAPI corresponderen met één of meerdere entiteiten in RIO. De exacte vertaling is nu nog niet bekend, ook omdat hiervoor nog een nieuwe versie van de OOAPI nodig is. Door de OOAPI werkgroep wordt er gewerkt aan deze v5 OOAPI.

2.2 Target Group (data voor RIO)

Mapper moet zorgen dat alleen data die door de instellingen voor de RIO beschikbaar gemaakt wordt, naar RIO wordt doorgestuurd. Dit kan door (nieuw filter) `TargetGroup=rio` o.i.d. te gebruiken. Hierbij is het aan de instelling om alleen de gegevens terug te geven die relevant zijn voor RIO.

Nog niet definitief in nieuwe OOAPI versie.

3 Architectuur, componenten

Het mapper systeem bestaat uit een aantal componenten.

3.1 Mapper API Server

Biedt de Mapper API aan. De Mapper API Server is een stateless dienst met als belangrijkste functie het snel afhandelen van binnenkomende verzoeken.

Vereisten

High-availability; statische applicatie waarvan meerdere instances tegelijk online kunnen en die weinig updates nodig heeft.

Low-latency; binnenkomende HTTP requests moeten zo snel mogelijk afgehandeld worden.

Interface

- `POST /job/{action}/education-specifications/{educationSpecificationId}`
- `POST /job/{action}/programs/{programId}`
- `POST /job/{action}/courses/{courseId}`

Het gebruikte endpoint is afhankelijk van het type hoofdobject. In de URL is `action` één van:

`upsert` pas een al dan niet bestaand object aan

`delete` verwijder een object

Response voor bovenstaande endpoints is:

```
200 OK
Content-Type: application/json

{"token": TOKEN}
```

Het token is een UUID string.

⚠

Om sequentieel afhandelen te garanderen, is het aan de instelling om geen parallelle aanvragen op dit eindpunt te doen. De Mapper API garandeert dat het veilig is om het volgende *request* te doen zodra er een *response* op het daaraan voorgaande *request* ontvangen is. De data is dan op de juiste topic afgeleverd - zie ook sectie 6 “Concurrency”.

`GET /status/{token}`

```
200 OK
Content-Type: application/json

{"status": STATUS}
```

Als het token niet herkend wordt:

```
404 Not Found
Content-Type: application/json

{"status": "unknown"}
```

Status is één van:

- `unknown`
- `pending`
- `in-progress`
- `time-out`
- `error`
- `done`

De "error" status heeft extra eigenschappen:

- `message`
  De foutmelding ter informatie zoals verkregen tijdens verwerking.
- `phase`
  De fase waarin de fout voorgekomen is.
  - `fetching-ooapi`
  - `resolving`
  - `fetching-rio` (misschien later geïmplementeerd)
  - `merging` (misschien later geïmplementeerd)
  - `upserting`
  - `deleting`
- `message`
  De foutmelding ter informatie zoals verkregen tijdens verwerking.

Voorbeeld:

```
{
  "status": "error" ,
  "phase": "deleting",
  "message": "404 Not Found"
}
```

De "done" status bevat (sleutel-)attributen van het gemanipuleerde RIO object. Voorbeeld:

```
{
  "status": "done" ,
  "attributes": {
    "opleidingseenheidcode": "..."
  }
}
```

Oplossing

Er wordt een JVM-based web server gebruikt voor het implementeren van deze API, geprogrammeerd in Clojure. De EduHub gateway zorgt voor autorisatie afhandeling en voor load balancing, rate limiting etc.

3.2 Worker Queue

We gebruiken een queue om update berichten veilig te stellen (zie paragraaf 3.1 “Mapper API Server”) en in de juiste volgorde af te handelen. De afhandeling van een update is afhankelijk van de snelheid en beschikbaarheid van de RIO API, en de binnenkomende berichten mogen niet verloren gaan als de RIO API tijdelijk niet of minder beschikbaar is.

Vereisten

Berichten van een instelling moeten in opgegeven volgorde één voor één worden afgehandeld (zie sectie 6 “Concurrency”).

Berichten van verschillende instellingen mogen in willekeurige volgorde of gelijktijdig worden afgehandeld als dit de totale doorvoer verbetert.

Durable: berichten moeten beschikbaar blijven als de queue herstart of geüpdatet wordt.

Queue info: voor het worker proces moet het mogelijk zijn om de beschikbare topics op te vragen zodat er nieuwe worker threads gestart kunnen worden.

Interface

`Push(topic_id, Msg)`

Zet `Msg` op de topic met `topic_id` (gebaseerd op instelling *SchacHomeOrganization* zoals gebruikt in OOAPI Gateway).

`Pop(topic_id) => Msg`

Lees oudste `Msg` van topic `topic_id`. Zodra er over dit bericht een `Ack` ontvangen wordt, wordt het bericht verwijdert.

`Ack(msg_id)`

Bevestig verwerking van `Msg`, vervolgens kan het volgende bericht van de topic gehaald worden.

`Nack(msg_id)`

Een expliciete niet-bevestiging; `Msg` kon niet verwerkt worden en mag bij de volgende `Pop` opnieuw van de topic gehaald worden.

`List_topics => topics...`

Geeft een lijst met beschikbare topics terug. Deze lijst is nodig om te bepalen welke Worker threads er opgestart moeten worden. Alternatief kan deze informatie opgehaald worden bij de Edugateway (lijst van gekoppelde instellingen opvragen), maar deze API call is nog niet geïmplementeerd.

Oplossing

Omdat berichten van een instelling in opgegeven volgorde één voor één moeten worden afgehandeld (zie sectie 6 “Concurrency”) gebruiken we één topic per instelling. Verder wordt afgedwongen dat per topic slechts één consumer actief mag zijn, en dat berichten pas van de topic verwijderd worden als deze actief "acknowledged" zijn.

We kiezen voor RabbitMQ omdat dit een betrouwbare implementatie is die beschikt over de gewenste functionaliteiten voor deze constructie.

Redis zou eventueel ook gebruikt kunnen worden als queue functionaliteit, maar beschikt niet over de ingebouwde synchronisatie mechanismen waarmee deze constructie eenvoudig te implementeren is.

3.3 Mapper Worker

Is verantwoordelijk voor het verzamelen van relevante data uit de OOAPI en het vertalen en doorsturen van gegevens naar de RIO API. De OOAPI wordt benadert via de SURFeduhub gateway.

Vereisten

Updates per instelling worden verwerkt op volgorde van binnenkomende updates. Maximum aantal aanroepen per seconde van RIO instelbaar.

Deploys en restarts veilig uit te voeren door het ondersteunen van een "gracefull shutdown" door, bijvoorbeeld, het ontvangen van een *SIGTERM* signaal waarna workers hun werk afrond en dan afsluiten.

Interface

Spreekt andere interfaces aan; heeft zelf geen API:

`Pop` / `Ack` / `Nack` van queue, om werk te verkrijgen en te bevestigen

`POST` naar Status DB, om resultaten te communiceren

OOAPI `GET` requests via EduHub Gateway, om instellingen gegevens op te halen

Bij het opvragen van OOAPI entiteiten wordt ook gecontroleerd of deze entiteiten geschikt zijn voor mappen naar het RIO, dit gebeurt door middel van tag / targetGroup matching. Entiteiten die niet voldoen worden genegeerd. Zie ook paragraaf 2.2 “Target Group (data voor RIO)”.

Resolver `GET` requests, om OOAPI identifiers naar RIO identifiers te vertalen

RIO API `GET` en `PUT` requests, om updates uit te voeren

Oplossing

Één multi-threaded JVM proces voor de Worker, geprogrammeerd in Clojure. RabbitMQ connectie per worker thread met "single active consumer" per topic. Zie sectie 6 “Concurrency”.

Periodiek controleert het worker proces welke topics er beschikbaar zijn en start / stopt de corresponderende worker threads.

3.4 Status DB

Hierin wordt per entity-updated request de data voor update status opgeslagen. Deze database gebruikt altijd een `update_token` (zie paragraaf 3.1 “Mapper API Server”) voor het opvragen en opslaan van data en kan dus een eenvoudige key-value store zijn.

Vereisten

- Key based lookup
- Verwijderen van data na vervaltijd (liefst automatisch)
- Durable: data moet beschikbaar blijven bij herstart / update van mapper services of database.
- Lookup latency moet laag genoeg zijn voor verwachte hoeveelheid verkeer. Zie ook sectie 7 “Performance”.

Interface

`SET ${update_token} ${data}` Zet de data van de update met het opgegeven `update_token`.

Data is een geserialiseerd object met één of meer van de volgende attributen:

Attribuut	Omschrijving
`status`	Een van {Ok, Error}
`ooapi_path`	Het OOAPI pad van het bijgewerkte record
`institution_id`	Het instelling *SchacHomeOrganisation* zoals gebruikt in OOAPI Gateway
`rio_id`	De RIO code van het opgegeven record (alleen in geval van status `Ok`)

`GET ${update_token}`

Vraagt de data van de update met het opgegeven `update_token` op.

Response is `nil` als er geen data beschikbaar is. Anders is de response de data zoals gezet.

Oplossing

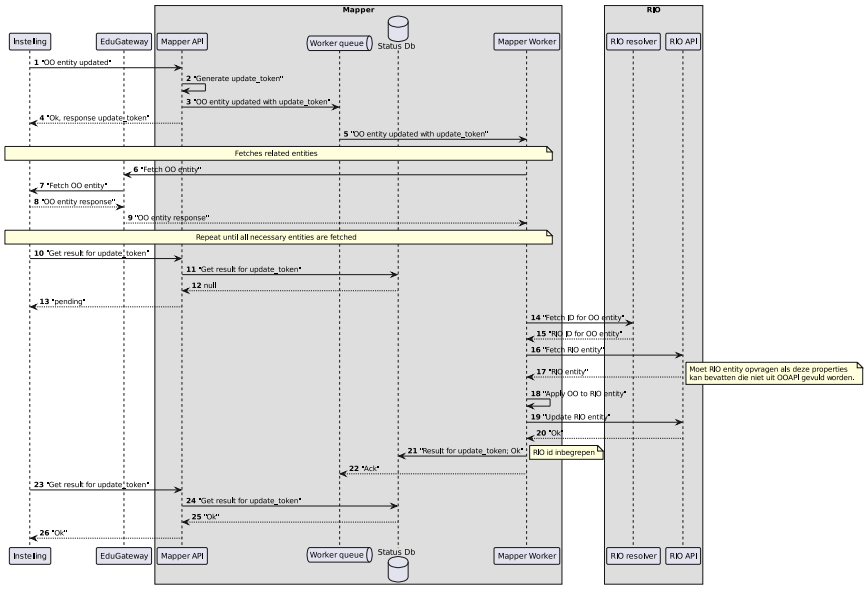
Redis wordt nu ook naar tevredenheid gebruikt voor caching van credentials van de EduHub gateway. We verwachten dat we het al beschikbare Redis cluster ook kunnen gebruiken als status DB. Dit beperkt de hoeveelheid operationele capaciteit (kennis en tijd) die nodig is om dit component te realiseren.

Voor het automatisch laten vervallen (zie ook paragraaf 9.1 “Configuratie van Mapper”) kan Redis EXPIRE/TTL gebruikt worden.

4 Sequence diagrammen

4.1 Happy flow

In figuur 2 “Happy flow” wordt geschetst welke acties er uitgevoerd worden bij een foutloze update.



Figuur 2: Happy flow

4.2 Fouten afhandeling

4.2.1 Retryable

Situaties:

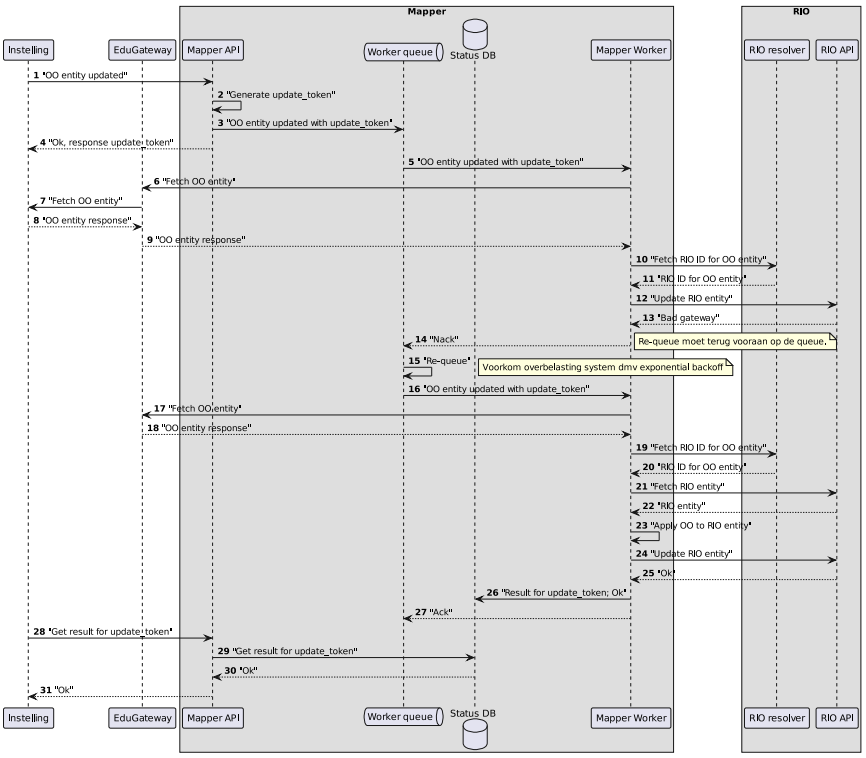
- HTTP statussen "502 Bad Gateway", "503 Service Unavailable", etc.
- Uitzonderlijke fouten; exceptions

Afhandeling:

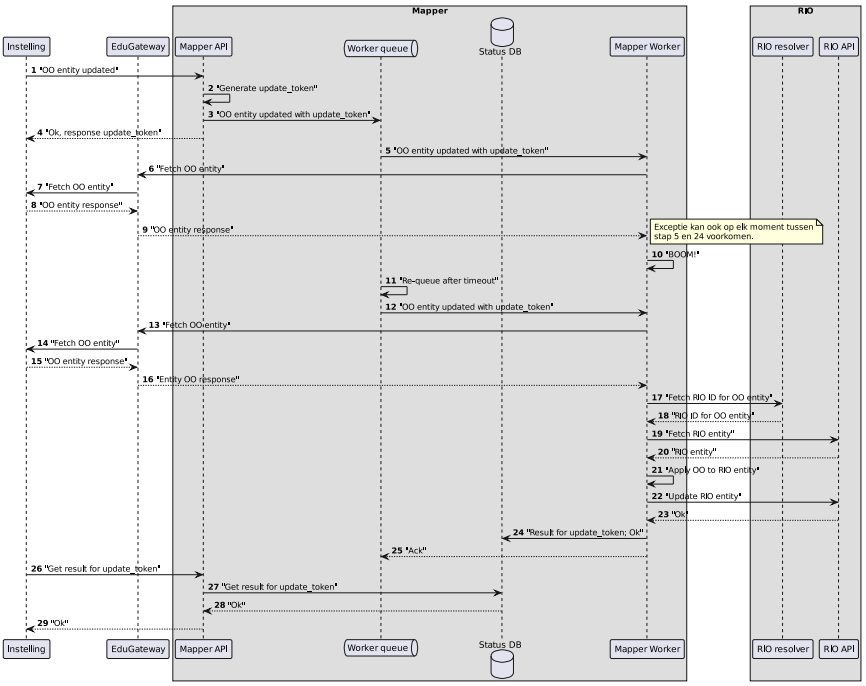
- terug op queue; retry
- niet te snel opnieuw; delay

Zie figuur 3 “Bad Gateway van RIO” de situatie waarin RIO een "Bad Gateway" response geeft.

Zie figuur 4 “Exception in mapper” waarbij de mapper crasht of herstart wordt.



Figuur 3: Bad Gateway van RIO



Figuur 4: Exception in mapper

4.2.2 Definitieve fout

Situaties:

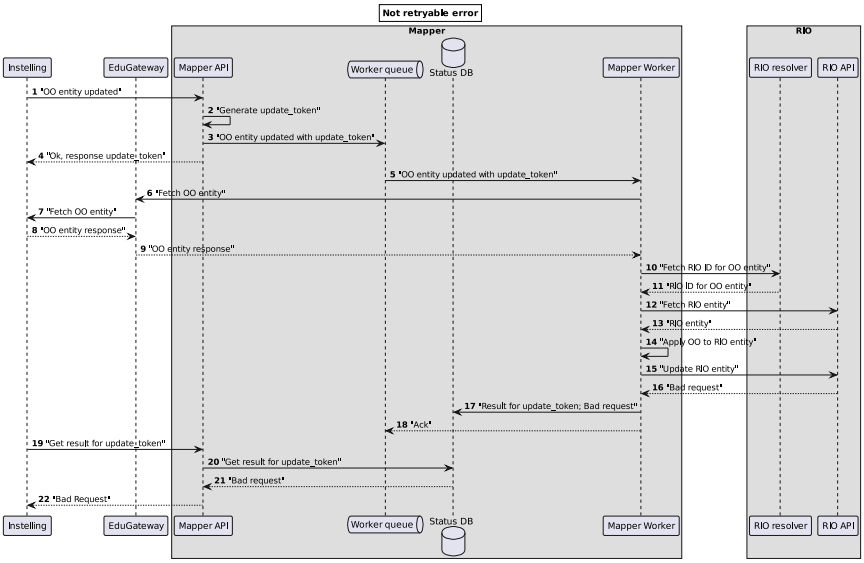
- data incompleet;
- "400 Bad Request" response en dergelijken van RIO

Definitieve fouten zijn fouten waarbij een herhaling van de instructie zinloos is. De worden daarom niet opnieuw uitgevoerd maar gemarkeerd als mislukt; "Error".

Afhandeling:

- Deadletter topic. Voor mislukte updates wordt een `Msg` op de deadletter topic geplaatst.
- "Error" resultaat in Status DB (zie ook paragraaf 3.4 “Status DB”). Hiermee kan de instelling via de Mapper API inzien dat de betreffende update niet verwerkt kan worden.

Zie figuur 5 “Exception in mapper”.

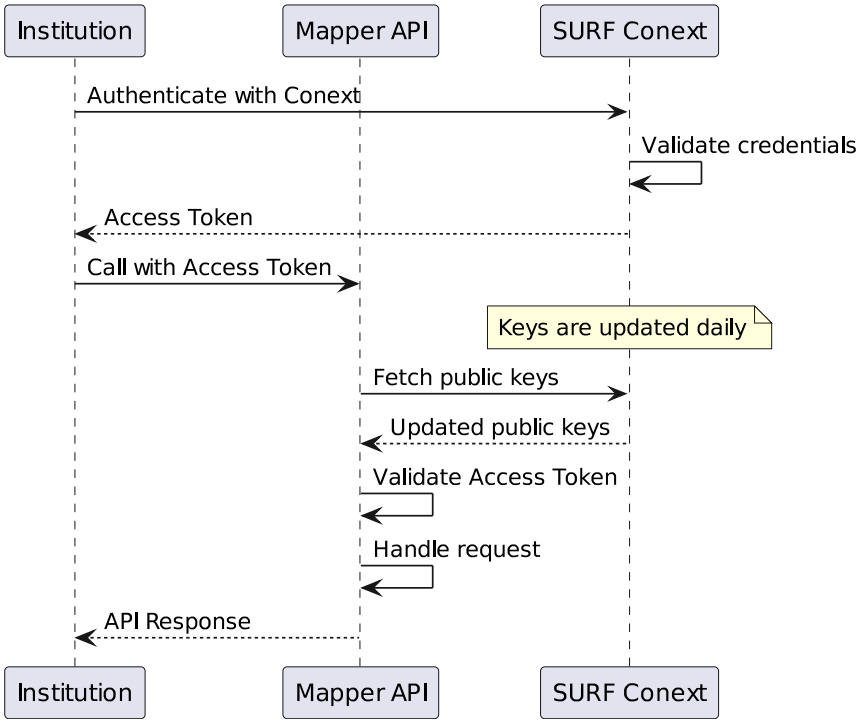


Figuur 5: Exception in mapper

5 Authenticatie

5.1 Van instelling bij mapper

De instellingen kunnen zich bij de Mapper API authenticeren door middel van de Oauth2 / OpenID Connect Client Credentials flow[3]. Hiervoor wordt SURF Conext gebruikt als autorisatie server. Zie ook figuur 6 “OAuth2 Client Credentials flow met SURF Conext”.



Figuur 6: OAuth2 Client Credentials flow met SURF Conext

Conext gebruikt JWT als access tokens. Deze worden getekend met een *RS256 signature*, waarvan de sleutels dagelijks vervangen worden. Deze sleutels worden door de Mapper ververst zodra er een onbekende sleutel-id (`kid`) gezien wordt.

Hiermee zijn Mapper API aanroepen te autoriseren met minimale communicatie met de Conext servers.

[3] https://auth0.com/docs/get-started/authentication-and-authorization-flow/client-credentials-flow

5.2 Van mapper bij Eduhub

De RIO Mapper wordt op de SurfEduhub aangesloten als "normale" Eduhub applicatie.

5.3 Van mapper bij RIO

De communicatie met RIO verloopt via de EduKoppeling 1.4 standaard [4]. De belangrijkste praktische punten zijn:

Autorisatie van de Mapper verloopt via een PKI certificaat dat uitgegeven is voor SURF.

Bij het aanroepen van de RIO API wordt het OIN[5] van de instelling waar de gegevens op betrekking hebben doorgegeven in de request door middel van een `http://www.w3.org/2005/08/addressing:From` element met een `anonymous + OIN` adres:

```
<soapenv: Header xmlns:wsa="http://www.w3.org/2005/08/addressing" >
<wsa:To>
  http://www.intermediairx.nl/services
  ?oin=00000007000011BB00001 /* OIN
  */
</wsa:To>
<wsa:Action>
  http://www.intermediairx.nl/services/ontvangenLeerlinginformatie_V2
  /* de WSDL-operatie */
</wsa:Action>
<wsa:MessageID>
  urn:uuid:ad47792d-d518-499b-a516-4182b344e18b
</wsa:MessageID>
<wsa:From>
  <wsa:Address>
    http://www.w3.org/2005/08/addressing/anonymous
    ?oin= 00000007000011BB00000 /* OIN */
  </wsa:Address>
</wsa:From>
</soapenv: Header>
```

[4] https://www.edustandaard.nl/standaard_afspraken/edukoppeling-transactiestandaard/edukoppeling-februari-2021/

[5] Organisatie-identificatienummer

6 Concurrency

Overwegingen:

Updates moeten op binnenkomende volgorde per instelling afgehandeld worden om te voorkomen dat eerdere aanpassingen latere overschrijven. Dit betekent dat ook dat instellingen er zelf zorg voor moeten dragen dat updates naar de mapper API in een "zinnige" volgorde opgegeven worden.

⚠

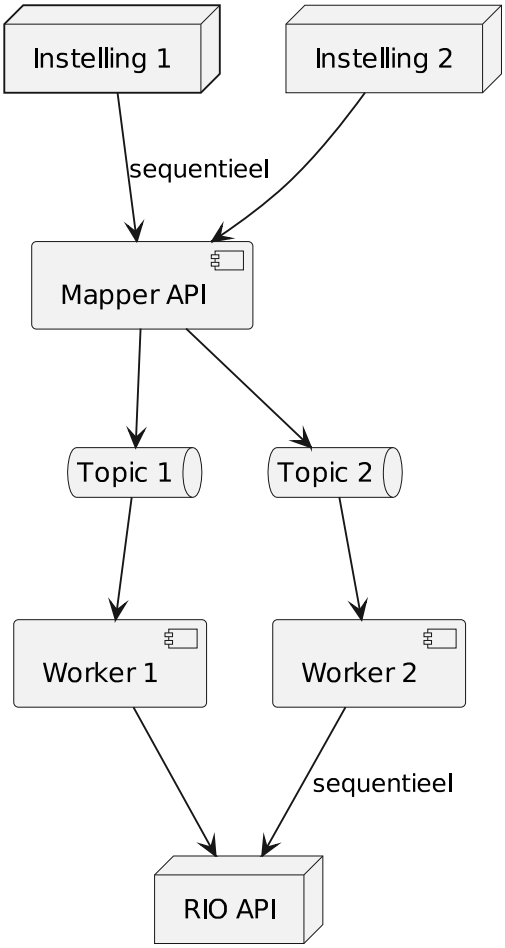
Dit moet duidelijk gedocumenteerd worden voor instellingen!

Aangenomen dat updates in de juiste volgorde aangeleverd worden bij de Mapper HTTP API, is het problematisch als er meerdere Mapper Worker processen tegelijk kunnen lopen. Hierbij moet gezorgd worden dat alle updates van 1 instelling door hetzelfde worker proces afgehandeld worden of door maximaal 1 worker proces tegelijkertijd.

We gaan uit van van één worker thread per topic - met één topic per instelling. Alle worker threads lopen in hetzelfde proces.

Bij het uitvoeren van requests naar RIO wordt er door middel van een lock / semafoor gezorgd dat het maximum aantal gelijktijdige requests niet overschreden wordt. Zie figuur 7 “Topics en volgorde van afhandeling”.

Deze opzet betekent dat er bij updates / service restarts opgelet moet worden dat er niet 2 worker processen tegelijk lopen. Hiervoor biedt RabbitMq een oplossing; er kan een `exclusive` flag opgegeven worden, waarbij een worker kan opgeven dat deze de enige afnemer van de topic moet zijn. Wellicht nog beter is bij de topic te configureren met de `single-active-consumer` setting. Deze helpen ook met restarts (als workers herstart worden, heb je een timeout voordat een message gezien wordt als NACK, en dus risico dat nieuwe worker alvast de volgende message oppakt). We gaan afdwingen dat berichten expliciet ge-ACKt moeten worden.



Figuur 7: Topics en volgorde van afhandeling

7 Performance

Gedraging van het complete systeem hand is sterk af van de manier waarop instellingen deze gebruiken en OOAPI toegang bieden. Door het gebruik van queues is het relatief gemakkelijk het verkeer naar RIO te reguleren. De drukte op deze queues is daarmee een goede indicator voor performance problemen (zie ook sectie 8 “Logging, metrics en alert”).

7.1 Limieten van externe diensten

De maximale doorvoersnelheid (het aantal updates dat in een periode afgehandeld kan worden) is gelimiteerd door de reactiesnelheid van de OOAPI endpoints van de instellingen en door de reactiesnelheid van de RIO API.

Omdat updates van één instelling sequentieel afgehandeld worden (zie sectie 6 “Concurrency”) en bij deze afhandeling requests op de OOAPI uitgevoerd worden, is dit deel van de performance voornamelijk afhankelijk van de snelheid van de OOAPI van de instelling.

Als één instelling een langzame API heeft, heeft dit in principe geen invloed op de verwerking van updates van andere instellingen.

Daarnaast resulteren alle updates ook in aanroepen van de RIO API, deze aanroepen worden gelijktijdig uitgevoerd als er meerdere updates (van verschillende instellingen) gelijktijdig verwerkt worden. Het maximum aantal updates per periode wordt dus beperkt door de snelheid van de RIO API.

De doorloopsnelheid van zowel RIO API als de OOAPI endpoint van de instellingen wordt bij gehouden, zie ook sectie 8 “Logging, metrics en alert”.

Limieten op de Mapper API zijn noodzakelijk ter voorkomen van problemen door runaway processen. Hiervoor kan op de HTTP API rate limiting ingesteld worden en op de queue een maximum queue lengte worden afgesproken (overschreiden van de deze limieten resulteert in HTTP Response fout codes).

7.2 Schalen van Mapper

Sharding per instelling kan gebruikt worden om de doorvoersnelheid van de Mapper te verhogen. Hierbij wordt één of meer van de componenten (zie sectie 3 “Architectuur, componenten”) direct toegewezen aan één instelling of een groep instellingen, als het totaal aantal gelijktijdige requests toeneemt. Sharding helpt als de betreffende component de bottleneck is en de te verwerken berichten betrekking hebben op verschillende instellingen.

Omdat er per instelling updates sequentieel afgehandeld worden, is het niet zinvol om meer dan één eenheid per component per instelling te schalen. De verwachting is dat voor er zover opgeschaald moet worden de limieten van de externe diensten bereikt zijn (zie paragraaf 7.1 “Limieten van externe diensten”.

Afhankelijk van de performance metrics van de componenten kunnen de volgende componenten geshard worden:

- Mapper API; afhandelsnelheid API aanroepen
- Queue; afhandelsnelheid update requests en workers
- Workers; geheugen gebruik en afhandelsnelheid workers
- Status db; afhandeling status rapportage en bevraging

8 Logging, metrics en alert

De logs en metrics van de componenten in de RIO Mapper worden centraal opgeslagen en geïndexeerd in de al gebruikte infrastructuur van EduHub. Waar beschikbaar wordt er in ieder geval gelogd:

- Trace context (`trace-id`, `parent-id`) zodat log entries aan elkaar gerelateerd kunnen worden
- Het `update_token` dat bij het verzoek hoort
- Instelling *SchacHomeOrganization* of OIN[5]

Metrics om bij te houden, over alle verkeer en per instelling:

- Queue lengte
- Tijd in queue
- HTTP Request status / tijd / URL etc.
- Verwerksnelheid RIO API calls
- Verwerksnelheid OOAPI calls

Alerts en dashboards kunnen worden ingesteld in Prometheus alert manager door het operationeel team. Hieronder een aantal suggesties:

- Laatste "goede" update per instelling
- Laatste "foute" update per instelling
- aantal goede/foute updates in periode per instelling
- fout en succes rate over tijd
- Verwerksnelheid van API calls - als bijvoorbeeld 10% van de requests voor een bepaald endpoint langer dan 2 seconden duurt.

9 Deployment

Via Docker, te integreren met al bestaande componenten via docker compose:

- API image
- Mapper image
- DB image
- Queue image

9.1 Configuratie van Mapper

Credentials van instellingen om Mapper API (zie ook paragraaf 5.1 “Van instelling bij mapper”) aan te roepen worden aanpasbaar in de Gateway Configurator. De gateway controleert de authenticatie.

De Mapper workers krijgen credentials om de OOAPI Gateway aan te roepen, net als andere applicaties die de OOAPI Gateway gebruiken, zie ook paragraaf 5.2 “Van mapper bij Eduhub”.

De Mapper workers krijgen één certificaat waarmee ze de RIO API aan kunnen spreken in naam van alle instellingen, zie ook paragraaf 5.3 “Van mapper bij RIO”. Bij deze aanroep wordt het OIN van de betreffende instelling doorgegeven; het OIN wordt afgeleid uit de SchacHome van de instelling en de Worker configuratie.

Verdere configuratie (redeploy / herstart nodig bij aanpassingen):

- maximum gelijktijdige interacties met RIO (zie ook sectie 6 “Concurrency”)
- maximum queue lengte
- endpoints, netwerk ports, paden ivm staging, productie e.d.
- resultaat retentietijd
- configuratie mapping van SchacHome + OIN (voor identificatie van instelling bij aanroep van RIO API).

In de toekomst is het mogelijk om deze configuratie in een centrale registry bij te houden om fragmentatie te voorkomen. Bijvoorbeeld via etcd.

9.2 Deploys, updates en high-availability

Om ervoor te zorgen dat instellingen altijd updates kunnen afleveren, en dat deze updates ook altijd worden afgehandeld, worden updates per component uitgevoerd.

De gebruikelijke aanpak van programmeren in een netwerkomgeving uiteraard zijn van toepassingen.

Als het niet mogelijk is om een maintenance window te reserveren voor een upgrade van de RIO Mapper, kunnen de verschillende componenten via onderstaande scenario's bijgewerkt worden zonder downtime.

9.2.1 Upgrade van Mapper API Server

De Mapper API Server moet altijd beschikbaar zijn voor het afleveren van updates. Er wordt bij een upgrade nieuwe Mapper API Server of cluster online gezet, en nieuwe requests worden via een front proxy (bijvoorbeeld HAProxy) doorgestuurd naar de nieuwe servers. Zodra lopende requests op de oude servers zijn afgehandeld kan deze worden gestopt.

9.2.2 Upgrade van Mapper Workers

De mapper workers worden bij een upgrade helemaal worden gestopt, waarbij indien mogelijk eerst lopende berichten worden afgehandeld. Zodra de workers gestopt zijn worden de nieuwe workers gestart.

Dit veroorzaakt tijdelijke vertraging in het afhandelen van updates, maar uiteindelijk worden alle updates verwerkt omdat deze in de tussentijd op de queue geparkeerd staan.

9.2.3 Upgrade van Status DB

Status DB records worden alleen aangemaakt door de workers, en gelezen door de Mapper API Server. Updates aan de Redis status DB verlopen als volgt:

- Stop workers - dit voorkomt wijzigingen aan de DB;
- Maak een backup van de DB;
- Zet een nieuwe DB versie online op basis van de backup;
- Upgrade de Mapper API Server (zie subparagraaf 9.2.1 "Upgrade van Mapper API Server") met een verwijzing naar de nieuwe DB;

- Start nieuwe workers met verwijzing naar nieuwe DB (zie subparagraaf 9.2.2 “Upgrade van Mapper Workers”);
- Verwijder oude DB

9.2.4 Upgrade van Queue

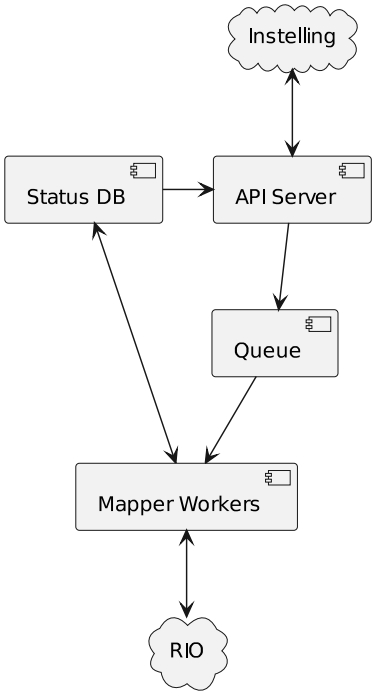
Bij een upgrade van RabbitMQ is de veilige optie om eerst een nieuw (leeg) RabbitMQ cluster online te brengen. Dan wordt een nieuwe Mapper API Server uitgerold (zie subparagraaf 9.2.1 “Upgrade van Mapper API Server”) die binnenkomende updates naar de nieuwe queue doorgeeft.

Zodra de oude queue leeg is, kunnen nieuwe workers worden uitgerold die updates van de nieuwe queue verwerken (zie subparagraaf 9.2.2 “Upgrade van Mapper Workers”. Daarna kunnen de oude queue, workers en api servers worden verwijderd.

9.2.5 Data formaat wijzigingen

Als wijzigingen aan de data formaten plaatsvinden moeten deze ook stapsgewijs uitgerold worden, en zal er in ieder geval tijdelijk twee formaten geaccepteerd moeten worden.

De gefaseerde uitrol begint dan met de componenten die zich "stroomafwaards" bevinden; eerst worden de ontvangers aangepast zodat deze het nieuwe formaat ondersteunen, daarna de verzenders, zodat het nieuwe formaat ook wordt aangemaakt. In figuur 8 “Data Flow tussen componenten” geven we een overzicht van de verzender/ontvanger relaties tussen componenten.



Figuur 8: Data Flow tussen componenten

10 Plan van aanpak

Alle broncode van de onderdelen van de API Mapper worden in een enkele repository ondergebracht omdat ze heel nauw samenhangen.

10.1 Voorwaarden

OOAPI Gateway is geüpdatet naar OOAPI versie 5. Aan te sluiten instellingen bieden ook v5 OOAPI aan.

Het is waarschijnlijk dat er een aantal V5 release-candidate versies ontwikkelt en getest moeten worden voordat een complete mapping gemaakt kan worden. We kunnen met deze release candidates al beginnen met stap paragraaf 10.2 “Gegevens conversie” om deze API versies te valideren.

Hieronder volgen de implementatie stappen in, grofweg, de uit te voeren volgorde.

10.2 Gegevens conversie

Benodigd: OOAPI entiteiten en bijbehorende RIO objecten (volledige data). Van ieder type entiteit in ieder geval één voorbeeld. Hiervan maken we unit tests.

Maken van een RIO object op basis van OOAPI entiteit.

Gegeven een aantal entiteiten in OOAPI met bijbehorende data in RIO OOAPI attributen toepassen op RIO entiteiten.

10.3 Ophalen data uit RIO

Benodigd: RIO test API inclusief RIO Resolver om RIO data uit op te halen, liefst als docker image o.i.d. zodat we deze in ontwikkel en test omgevingen kunnen gebruiken.

Dit ook voor adressen tbv OOAPI `Buildings`.

Ophalen specifieke RIO entiteiten (op basis van RIO identifier, dus zonder resolver).

Gegeven aantal entiteiten in OOAPI, ophalen van bijbehorende entiteiten in RIO ophalen, gebruikmakend van Resolver.

Bij foutcondities moet duidelijk zijn of deze fout retryable is of niet.

10.4 Indienen van data in RIO

Benodigd: test API zoals bij paragraaf 10.3 “Ophalen data uit RIO”.

Op basis van voorbeeld entiteiten in RIO, indienen van deze entiteiten in de API implementeren en testen.

Bij foutcondities moet duidelijk zijn of deze fout retryable is of niet.

10.5 Ophalen data uit OOAPI

Benodigd: test API, maken we zelf.

Bij foutcondities moet duidelijk zijn of deze fout retryable is of niet.

10.6 Worker functie

Gegeven een OOAPI pad en instelling *SchacHomeOrganization*, haal de betreffende OOAPI data op, bevraag RIO resolver, zoek de RIO data erbij, pas de OOAPI toe op de RIO data, en dien de nieuwe RIO data in.

Bij foutcondities moet duidelijk zijn of deze fout retryable is of niet.

Op dit moment kunnen er metingen gedaan worden aan de doorvoersnelheid van de RIO Mapper.

10.7 Basis API

Een endpoint welke een update bericht aanneemt van een instelling en de worker functie opstart met het OOAPI pad en instelling *SchacHomeOrganization*. Merk op: in deze opstelling is een queue nog niet nodig.

Inclusief credentials afhandelen via SURF Conext.

10.8 Met Status DB

Het update API endpoint uitbreiden met `update_token` generatie welke teruggegeven aan de instelling en aan de worker functie door wordt gegeven. De worker functie registreert het resultaat van de actie in de Status DB (zie paragraaf 3.4 “Status DB”). Een nieuw API endpoint maakt het mogelijk deze Status DB entries op te halen op basis van het verkregen `update_token`.

10.9 Queue

Ontkoppel basis API (zie paragraaf 10.7 “Basis API”) van worker functies door er een queue tussen te zetten. Per instelling wordt een topic gemaakt, een worker proces zorgt dat er per topic een worker thread draait welke abonneert op dat topic en update aanvragen sequentieel afhandelt, zie sectie 6 “Concurrency”.

10.10 Artifacts en configuratie

We maken build scripts voor het opzetten van Docker images. De artifacts zijn configureerbaar zoals gedocumenteerd in paragraaf 9.1 “Configuratie van Mapper”.

10.11 Bestaande data in RIO koppelen

Voordat de RIO Mapper in gebruik genomen wordt is er al data over entiteiten aanwezig in RIO die ook in de OOAPI beschikbaar worden. Deze bestaande entiteiten moeten gekoppeld worden zodat de RIO mapper geen duplicaten aanmaakt in het RIO (dit kan problemen opleveren met de STAP[6] registraties).

Om dit te faciliteren kan er per instelling een rapport gemaakt worden van de entiteiten in OOAPI (bijvoorbeeld in MS-Excel), waarin te zien is welke entiteiten er zijn - met herkenbare eigenschappen: naam, etc en het OOAPI pad.

Op basis van dit overzicht kan dan in RIO, handmatig gecontroleerd worden of een entiteit al bestaat en deze als sourceId het OOAPI pad geven. Hiermee zijn de entiteiten gekoppeld.

⚠

Omdat we volledige "standen" doorgeven, moet alle al bestaande data van een instelling in het RIO gekoppeld zijn (ter voorkoming van dubbelen), en in de OOAPI beschikbaar zijn voor export naar RIO (anders wordt de bestaande RIO data potentieel verwijderd, zie ook paragraaf 2.2 “Target Group (data voor RIO)”).

10.12 Bestaande data in OOAPI synchroniseren

Verder in dit ontwerp worden OOAPI entiteiten overgezet op basis van een "update" bericht. De RIO data wordt daarmee bijgewerkt wanneer een OOAPI entiteit wordt aangemaakt, bijgewerkt of verwijdert.

Bij ingebruikname van de RIO Mapper moet ook al bestaande data in OOAPI overgezet worden naar RIO zonder dat voor alle entiteiten een update bericht verstuurd moet worden. Dit kan door een spider actie van de OOAPI of door een aparte bulk import. We schetsen hier een spider proces.

[6] Stimulans ArbeidsmarktPositie

Een spider proces wordt voor één instelling gestart op een voor de instelling geschikt moment. Hierbij wordt voor alle entiteiten die geschikt zijn voor RIO (zie paragraaf 2.2 “Target Group (data voor RIO)”) een conversie maakt naar RIO.

Dit is een los te starten proces welke ook in "dry run" modus gedraaid kan worden. Dat wil zeggen dat de updates niet naar RIO gestuurd worden maar opgenomen worden in een rapportage ter controle. In dit rapport kunnen zaken worden opgenomen als:

- resolver fouten, bijvoorbeeld adres van gebouwen nog niet ingevoerd in RIO
- opvoeren van nieuwe entiteiten in RIO
- aanpassingen aan entiteiten in RIO
- verwijderde van entiteiten in RIO

11 Gebruikte termen

RIO Registratie Instellingenen en Opleidingen

OOAPI Open Onderwijs Application Programming Interface

SURFeduhub OOAPI Gateway

OIN Organisatie-identificatienummer

SchacHomeOrganization SCHema for ACademia eigenschap om een instelling aan te duiden; een domeinnaam

Trace Context W3C Trace Context HTTP headers voor het doorgeven van correlatie ids; `trace-id` en `parent-id` hiermee kunnen bewerkingen over gedistribueerde systemen gerelateerd worden.