edustandaard

Edukoppeling

Asynchrone M2M gegevensuitwisseling binnen het onderwijs

CloudEvents profiel (Asynchrone communicatie via RESTful API's)

Edustandaard
Datum: 20 mei 2026
Versie 1.0
Status: concept

edustandaard

Inhoudsopgave

1. Status van dit document

Dit document is een conceptversie van het CloudEvents profiel voor asynchrone communicatie via RESTful API's (hierna CloudEvents profiel). Het is gebaseerd op het NL GOV profile for CloudEvents [1].

De nieuwe Edukoppeling Architectuur versie 3.0 geeft context aan synchrone en asynchrone communicatie tussen ketenpartners die het CloudEvents profiel standaardiseert. De nieuwe RESTful-API architectuur wijkt sterk af van de vorige versie[2]. In die versie onderkende we al RESTful API's, maar was nog sterk gericht op webservices en een Digikoppeling-indeling. Verder onderkende we al patronen die synchrone RESTful API's gebruikte om asynchrone communicatie te implementeren, maar werd niet beschreven hoe deze uitwisseling gestandaardiseerd en zoveel mogelijk ontkoppeld kon plaatsvinden.

Met opmerkingen [EV1]: Klopt dit wel?? We doen toch ook synchroon?

Met opmerkingen [DG2R1]: Verwerkt.

In de nieuwe opzet staat asynchrone communicatie door middel van CloudEvents centraal. Deze architectuur sluit veel meer aan op ontkoppelde architecturen zoals een Event-driven architectuur (EDA), en wordt toegepast op ketenpartnerlandschap binnen het onderwijs.

Dit is zeer relevant in een speelveld waarin geïntegreerd wordt met systemen van de Nederlandse overheid, de Nederlandse onderwijssector (sectorpartners en onderwijsinstellingen) en verschillende andere derde partijen. Hoge ontkoppeling minimaliseert effect van verkeerspieken en (tijdelijke) (on)geplande onbeschikbaarheid op de rest van de keten.

1.1. Documenthistorie

Versie	Status	Auteur	Datum	Opmerking
1.0	Concept	D. Grootendorst	Mei 2026	

[1] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/
[2] https://www.edustandaard.nl/app/uploads/2021/10/2021-02-10-Edukoppeling-Architectuur-2.0-definitief.pdf

2. Inleiding

Met opmerkingen [EV3]: Blijft dit een apart document of gaan we het opnemen binnen de EduKoppeling afspraak>

2.1. Doel van Edukoppeling

Het doel van Edukoppeling is standaardisatie van de technische afspraken op het M2M-koppelvlak waardoor het ontwikkelen van ketensamenwerkingen by design veilig en eenvoudiger wordt. De toepassing van Edukoppeling zorgt ervoor dat, op het niveau van de applicatielaag, gesloten data tijdens transport van de ene naar de andere organisatie niet ongeoorloofd kan worden ingezien of gemanipuleerd. De standaard gaat over de afhandeling van berichten (het transport) en niet over de inhoud van berichten.

Met Edukoppeling worden voor ketensamenwerking een aantal ketenfuncties in de applicatielaag geregeld (zie hoofdstuk Toepassingsscenario's).

2.2. Doelgroep

Dit document is bedoeld voor ICT-specialisten die betrokken zijn bij het ontwerpen en ontwikkelen van systeem-naar-systeem (M2M) koppelingen voor RESTful API's. Het gaat hier om werknemers (ontwikkelaars, architecten, projectmanagers, informatiemanagers etc.) werkzaam bij onderwijsgerelateerde organisaties, zowel in de publieke als private sector. De lezer van dit document willen wij vragen om zaken die ontbreken of onduidelijk zijn te melden bij de beheerorganisatie Edustandaard[3]. Edustandaard is een open platform waar partijen binnen het onderwijsveld bij elkaar komen om afspraken te maken. Hier vindt tevens de doorontwikkeling van de standaard plaats. Hiertoe is een werkgroep Edukoppeling[4] ingericht.

2.3. Organisatorisch werkingsgebied van Edukoppeling

Edukoppeling schrijft voor hoe onderwijsorganisaties, publieke uitvoeringsorganisaties, dienstverleners[5] en andere ketenpartners gegevensuitwisselingen opzetten. Edukoppeling heeft als scope alle werkingsgebieden vallend onder alle onderwijssectoren. Zie de werkingsgebieden in ROSA[6]. Hieronder vallen dus alle door de overheid erkende onderwijsorganisaties binnen de sectoren po, vo, bve en ho. Daar waar behoefte is aan technische afspraken op het M2M-koppelvlak, volgens het functioneel toepassingsgebied van Edukoppeling, zijn ketensamenwerkingen binnen deze sectoren verantwoordelijk voor de toepassing ervan. De toepassing buiten het organisatorisch werkingsgebied is toegestaan.

2.4. Functioneel toepassingsgebied

Het functioneel toepassingsgebied van Edukoppeling is de geautomatiseerde uitwisseling van gesloten data[7] tussen informatiesystemen van onderwijsorganisaties en ketenpartners

[3] https://www.edustandaard.nl/standaarden/afspraken/afspraak/edukoppeling/. Reageren kan via info@edustandaard.nl.
[4] Voor meer info over de Edukoppeling werkgroep, zie https://www.edustandaard.nl/standaard_werkgroepen/werkgroep-edukoppeling/
[5] Dienstverleners (ROSA: *Een organisatie die een dienst levert aan een organisatie of een natuurlijke persoon).*
[6] https://rosa.wikixl.nl/index.php/Werkingsgebieden
[7] Het gaat om gegevens waarvoor je geautoriseerd moet worden voor toegang. De gegevens zijn niet voor publiek hergebruik beschikbaar. Dit kan om verschillende redenen zijn, zie https://data.overheid.nl/gesloten-datasets

(onderling, met bedrijven of met de overheid). Deze uitwisseling betreft M2M point-to-point verbinding voor uitwisseling tussen een confidential client en een gesloten API.

Edukoppeling gaat over de afhandeling van berichten (het transport) en niet over de inhoud van berichten. Het is een functioneel technische standaard, maar zal ook aansluiting moeten vinden op kaders van andere architectuurlagen. Hoe Edukoppeling aansluit op bredere afspraken is aan ketensamenwerkingen[8] waarin de standaard als onderdeel van de afspraak of het afsprakenstelsel wordt gevat.

Edukoppeling regelt de volgende ketenfuncties: identificatie, authenticatie, autorisatie en routering, op de 'uitwisselingslaag'. Dit om de zorgen dat vertrouwelijke gegevens tijdens transport van de ene naar de andere organisatie, niet ongeoorloofd worden ingezien of gemanipuleerd.

2.5. Positionering van Edukoppeling in het Edustandaard vijflagen model

Met opmerkingen [EV4]: Over welke laag gaat deze technische afspraak?

Het Edustandaard 5-lagenmodel[9] onderkent de volgende lagen:

1. Grondslagenlaag: borgt de juridische basis en beleidskaders waarbinnen gegevensuitwisseling is toegestaan;
2. Organisatorische laag: ketensamenwerking afspraken over wie welke rol heeft, welke gegevensdiensten, interfaces en interactiepatronen er zijn en welke gegevens onder welke condities uitgewisseld worden;
3. Informatielaag: semantiek, waaronder gegevensdefinities, informatiemodellen en de gebruikte identifiers voor rechtspersonen en natuurlijke personen;
4. Applicatielaag: API's en hun beveiligingsprofielen, berichtspecificaties, payload beveiliging, interactiepatronen en foutafhandeling;
5. IT-infrastructuurlaag: transportprotocollen en technische beveiligingsmechanismen zoals TLS.

Het Edukoppeling-profiel heeft binnen het Edustandaard 5 lagen model met name betrekking op de applicatielaag, levert daarmee een belangrijke ondersteuning aan met name laag 2 en heeft een relatie met de IT-infrastructuurlaag.

Met opmerkingen [ER5]: Vind jij het beter om een plaat niet op te nemen?

Met opmerkingen [DG6R5]: @Erwin Geen sterke mening. In ieder geval consistent houden. In de versie van Oauth profiel die ik heb - zie ik hem niet.

2.6. Notatiewijze voorschriften

Voor elk voorschrift wordt aangegeven in welke mate hier invulling aan moet worden gegeven. Hiermee kunnen we duidelijk aangeven wat de grenzen van dit profiel zijn ten opzichte van de mogelijke externe bron(nen) waar het voorschrift eventueel van wordt overgenomen. We gebruiken hiervoor de notatiewijze van RFC2119[10]. Deze gebruikt de volgende termen: "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL".

[8] Ketensamenwerkingen zijn bijvoorbeeld OKE, Edu-V, ROD ec. (zie ook: https://rosa-begrippenkader.wikixl.nl/index.php/Begrip:27a6accf-472d-4415-bc5b-1e9de17bf288#tab=Betekenis)
[9] AMIGO-methodiek-1.1.0-1.pdf en https://rosa.wikixl.nl/index.php/Interoperabiliteit_en_het_Edustandaard_lagenmodel#Opbouw_van_het_lagenmodel
[10] https://tools.ietf.org/html/rfc2119

2.7. Leeswijzer

Hoofdstuk 2 bevat een toelichting op de CloudEvents standaard en Edukoppeling voorschriften. In hoofdstuk 3 zijn de Edukoppeling voorschriften opgenomen. Dit zijn aanscherpingen op de onderliggende internationale open industrie standaarden rond CloudEvents[11]. In hoofdstuk 4 worden scenario's beschreven die een aantal voorschriften in een bepaalde context plaatsen om ketensamenwerkingen te ondersteunen in de te maken keuzes.

2.8. Relevante ontwikkelingen

In deze paragraaf worden relevante ontwikkelingen binnen de overheid en IT beschreven die inspiratie bieden voor ontwikkeling binnen het onderwijs.

- Binnen het Groeifondsprogramma Npuls, en Edu-V worden nieuwe koppelvlakken ontwikkeld en/of koppelvlakken gemoderniseerd in lijn met Event Driven Architecturen (EDA).
- Er is een NL GOV profile for CloudEvents ontwikkeldt. Er is recent een definitieve versie 1.1[12] gepubliceerd. Deze beschrijft een patroon voor gegevensuitwisseling binnen Event Driven Architecturen. Dit verwijst o.a. naar de internationale standaard CloudEvents v1.0.1 [13], een JSON Event Format v1.0.1[14] voor CloudEvents en Web Hooks v1.0.1-[15] voor Event Delivery.
- AsyncAPI specificatie v3.1.0 initiatief voor het beschrijven van event-driven APIs [16]

Een voorgestelde IETF-standaard, om idempotency [17] te implementeren voor HTTP methoden die niet als veilig worden gezien, die toegevoegde waarde heeft voor robuuste end-to-end Event Driven Architecturen.

2.9. Uitgangspunten voor het CloudEvents profiel

1. Dit Edukoppeling-profiel is gebaseerd op de internationale open standaarden rond CloudEvents. Het fundament is de CloudEvents specificatie, opgesteld door Cloud Native Computing Foundation (CNCF).

2. Dit Edukoppeling-profiel volgt de richtlijnen zoals beschreven in het NL GOV profile for CloudEvents [18]:
   a. Context attributen (H3);
   b. Event Data (H4);
   c. Size Limits (H5);

Met opmerkingen [EV7]: EDA wordt binnen bepaalde ketenprcessen toegepast. Dus het gaat om enkele koppelvlakken. Trouwens naast deze EDA wordt ook het gebruik van synchrone transacties toegestaan. Denk aan set-up van data etc.

Met opmerkingen [DG8]: In de werkgroepbijeenkomst van 30 maart hebben we 2 dingen benoemd:

1. Het is ongewenst om grote bestanden via dit patroon mee te leveren en dat een referentie naar het bestand/data dan gewenst is. H5 beschrijft dit al. Voldoet dit voor de werkgroep?
2. Het is wenselijk om duidelijk te beschrijven welke data waar ligt, zoals contextuele data, wat zit in headers, wat niet. H3 is vrij duidelijk wat wél in context zit (en daarmee ook wat er niet in zit). Voldoet dit voor de werkgroep?

Met opmerkingen [EV9R8]: Graan nader toelichten. Want eens met opmerking om we een keuze te maken waarbij H5 niet wordt voorgeschreven.

Gewijzigde veldcode

Gewijzigde veldcode

Gewijzigde veldcode

Gewijzigde veldcode

Gewijzigde veldcode

Gewijzigde veldcode

Gewijzigde veldcode

Gewijzigde veldcode

[11] https://cloudevents.io/
[12] NL GOV profile for CloudEvents 1.1
[13] https://github.com/cloudevents/spec/tree/v1.0.1
[14] https://github.com/cloudevents/spec/blob/v1.0.1/json-format.md
[15] https://github.com/cloudevents/spec/blob/v1.0.1/http-webhook.md
[16] https://www.asyncapi.com/docs/reference/specification/v3.1.0
[17] https://datatracker.ietf.org/doc/draft-ietf-httpapi-idempotency-key-header/
[18] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/1.1/

d. Gebruik van JSON, HTTP en webhook (bijlage A) zoals ook beschreven in de Guidelines for NL-GOV profile CloudEvents [19].

Met uitzondering van:

e. CloudEvent Security Options (H6);
f. Abuse protection as described in the guidelines [20].

In paragraaf 3.6 CloudEvent beveiligingsopties worden de beveiligingsopties binnen dit profiel beschreven.

3. Dit Edukoppeling-profiel is alleen van toepassing in de context van confidential clients.

4. Dit Edukoppeling-profiel volgt het OAuth client credentials profiel voor RESTful API's voor machine-to-machine (M2M) gegevensuitwisseling binnen het onderwijs.

5. Dit Edukoppeling-profiel schrijft het gebruik van Transport Layer Security (TLS) voor conform UBV TLS.

Met opmerkingen [EV10]: Edu-V zegt JSON, HTTPS, webhooks + api + event-notificatie. Met we kijken hierbij wel naar het patroon die we willen volgen. Waarbij we bewust iets flexibeler zijn. :
- concept = event-driven
- Patroon = publish/subscribe
- Implementatie = open (webhook. Poling, streaming etc.)

Met opmerkingen [EV11]: Definitie uitwerken, voorbeelden?

Met opmerkingen [EV12R11]: Is deze basis niet te smal?

Met opmerkingen [GG13]: Dat is voor DUO niet acceptabel als DUO degene is die een service aanbiedt voor gemandateerden. Wij werken aan een ander profiel (ook obv OAUTH) waarbij mandaatvlidatie plaats vindt als in het huidige Edukoppeling.

Met opmerkingen [EV14R13]: Nader bespreken

[19] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/guidelines/1.0/
[20] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/guidelines/1.0/

Gewijzigde veldcode

Gewijzigde veldcode

3. Toelichting voorschriften

3.1. Inleiding

Dit hoofdstuk beschrijft de aanvullende voorschriften op de CloudEvents standaard die de basis vormt voor asynchrone communicatie binnen het Edukoppeling-profiel. Het CloudEvents-profiel biedt op een aantal punten keuzemogelijkheden. Hiermee beogen we een profiel dat een verplichte basisset van voorschriften bevat, maar ook genoeg ruimte biedt voor passende configuraties om voor ketenpartners binnen een bepaalde ketensamenwerking niet onoverkomelijke drempels op te werpen. Het is aan een ketensamenwerking om te bepalen welke opties van toepassing zijn.

heeft opmaak toegepast: Markeren

Dit profiel moet worden toegepast wanneer er binnen een ketensamenwerking *asynchrone* gegevensuitwisseling plaatsvindt tussen confidential clients en RESTful API's. Dit geldt niet alleen voor simpele notificaties en signalen, maar ook voor het bereiken van ontkoppelde gegevensuitwisselingen.

De CloudEvents standaard is gebaseerd op het principe van het niet opleggen van meer eisen op de betrokken partijen dan noodzakelijk. Het gebruik van zorgt voor ontkoppeling van aflevering en inhoud. Voor asynchrone communicatie is daarom de voorkeur om deze vorm van uitwisseling uit het aanbod te gaan gebruiken.

Met opmerkingen [EV15]: Is dit niet een algemeen principe die voor heel Edustandaard/Edukoppeling geldt?

Het onderwijsveld bestaat uit verschillende ketenpartners die ieder verantwoordelijk zijn voor een deel van de keten en zijn systemen. De CloudEvents standaard moet daarom in de context van deze ketensamenwerking worden geplaatst.

Met opmerkingen [EV16]: En ketenproces?

De rollen en interacties worden hieronder verder toegelicht.

3.1.1. Rollen

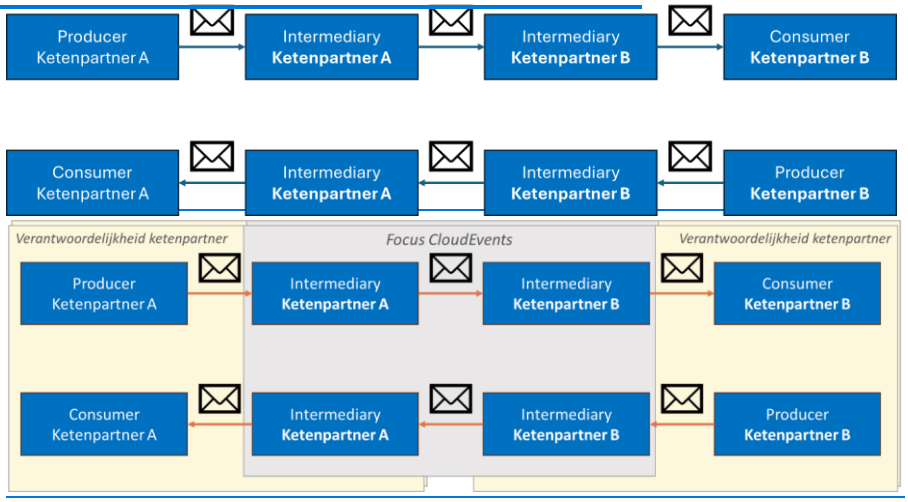
Het basispatroon beschrijft een applicatie in de rol van ‘producer’ die ‘events’ publiceert: dataregistraties die een gebeurtenis en de bijbehorende context vastleggen. Gepubliceerde events kunnen worden geconsumeerd door applicaties in de rol van ‘consumer’. Consumers abonneren zich op bepaalde type events. Er kunnen één of meerdere applicaties in de rol van ‘intermediary’ zijn die zorgdragen voor het routeren van events naar consumers op basis van contextuele informatie. Dit is vergelijkbaar met het publish-subscribe-patroon:

Figuur 1 - Publish-subscribe patroon

In dit patroon is het duidelijk dat de producer wordt beheerd door de ketenpartner die berichten produceert en de consumer wordt beheerd door de ketenpartner die de berichten consumeert, en asynchroon verwerkt. Voor de Intermediary is dit niet direct duidelijk en is er momenteel ook binnen de (semi-)overheid niet één centrale partij toe te wijzen die logischerwijs dit beheer op zich kan nemen. We kiezen hiervoor om binnen de CloudEvents patronen *per ketenpartner* een Intermediary te implementeren. Dit zorgt ervoor dat elke dienstverlener deze Intermediary naar eigen inzicht kan inrichten en beheren.

De CloudEvents specificatie beschrijft de berichtuitwisseling tussen Intermediary A en Intermediary B en plaatst dit in context van een producer en consumer. Echter, de ketenpartner bepaalt zelf hoe de interactie van eigen Intermediary naar Consumer/Producer plaatsvindt en hoe de scheiding van rollen daarin plaatsvindt. Dit is schematisch weergegeven in Figuur 2.

Met opmerkingen [EV17]: Binnen Edu-V gebeurt het op dit moment zo:
https://edu-v.atlassian.net/wiki/x/FACN
Waar zit notificatie en abonnement in dit model



Figuur 2 - Intermediary per ketenpartner

Deze aanpassing aan het patroon betekent dat elke partij een API-endpoint definieert waar CloudEvents naar verstuurd kunnen worden. Het uitgangspunt voor dit API-endpoint is technologie agnostisch; losstaand van ketenpartner specifieke technologie keuzes voor implementatie van een producer / intermediary / consumer.

De rollen worden respectievelijk in paragrafen 3.3, 3.4.1, 3.4.2 en 3.4.3 nader toegelicht.

3.1.2. Interacties

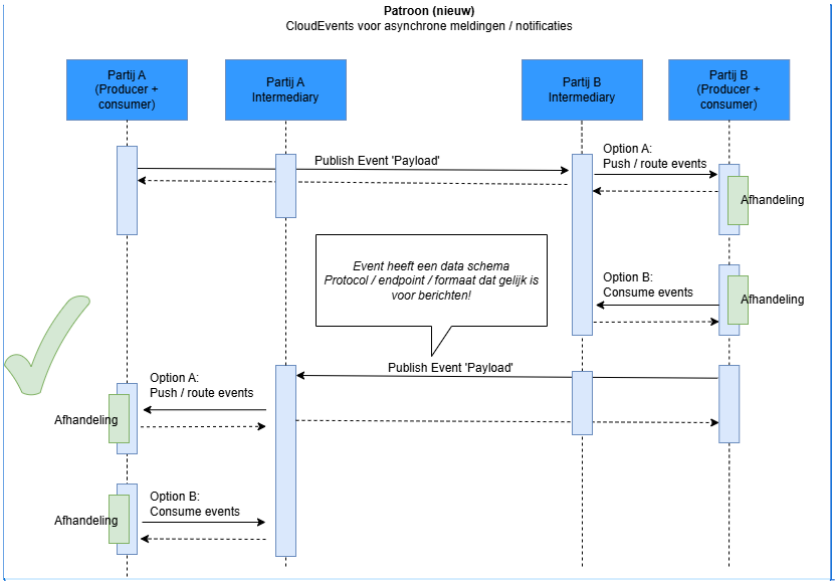
Deze paragraaf beschrijft de interacties tussen de rollen op hoofdlijnen. In paragraaf 3.4 Interacties~~Interacties~~ worden de interacties en met name vereisten van elke rol verder uitgediept. Dit biedt een eerste inleiding en biedt een beeld hoe ketenpartners via CloudEvents communiceren.

Deze componenten hebben de volgende interacties:

- Berichtenverkeer partij A naar partij B:
  - De producer van partij A produceert events (via zijn eigen Intermediary van partij A) naar de Intermediary van partij B
  - De Intermediary B persisteert het event zodanig dat consumer partij B deze kan consumeren óf stuurt het event direct door naar consumer partij B.
  - Consumer partij B verkrijgt het event van Intermediary B gerouteerd/gepushed óf consumeert de events van Intermediary B.
- Berichtenverkeer partij B naar partij A:

- De producer van partij B produceert events (via zijn eigen Intermediary van partij B) naar de Intermediary van partij A.
- De Intermediary A persisteert het event zodanig dat consumer partij A deze kan consumeren óf stuurt het event direct door naar consumer partij A.
- Consumer partij A verkrijgt het event van Intermediary A gerouteerd/gepushed óf consumeert de events van Intermediary A.

De componenten en interacties worden weergegeven in Figuur 3 - CloudEvents uitwisseling.



Figuur 3 - CloudEvents uitwisseling

Met opmerkingen [DG18]: Nav overleg 30 maart heb ik deze plaat aangepast (oftewel een intermediary aan beide kanten). Ik heb de tekst daar ook op aangepast.

Met opmerkingen [ER19]: Kun je de bronnen van de plaatjes opleveren?

Met opmerkingen [DG20R19]: Bronnen plaatjes CloudEvents

3.2. Event levering semantiek

In een gedistribueerd landschap heb je traditioneel te maken met de uitdaging van *atomiciteit*. Dit geldt ook voor ~~het~~ de CloudEvents~~event~~-uitwisseling. In de ketencontext heb je het over de communicatie tussen twee intermediaries waarbij één intermediary als verzender optreedt en één intermediary als ontvanger optreedt. Echter, robuuste end-to-end verwerking vereist garanties van elke hop in de uitwisseling, dus van producer naar Intermediary, van Intermediary naar Intermediary, en van Intermediary naar consumer.

3.2.1. Aflevergaranties

Als het gaat om aflevergaranties richting een Intermediary (van Producer naar Intermediary, of van Intermediary naar Intermediary) zijn er drie opties:

- At most once: een bericht kan worden afgeleverd, maar nooit meer dan één keer. Dit kan leiden tot verlies van berichten en wordt daarom zelden, zo niet nooit, gebruikt.
- At least once: een bericht wordt afgeleverd, maar kan meer dan één keer worden afgeleverd. Dit kan leiden tot dubbele berichten.

- Exactly once: een bericht wordt precies één keer afgeleverd.

De reikwijdte van deze garanties ligt echter alleen binnen de event-broker en niet daarbuiten, zoals in een gedistribueerd IT-landschap. In een volledig robuust IT-landschap moeten we rekening houden met end-to-end robuuste aflevering en uitgaan van het worstcasescenario.

Een worstcasescenario is wanneer een producer crasht / herstart / stopt tijdens de verwerking van een event. Met name in een schaalbare infrastructuur is dit een realistisch scenario. Afhankelijk van het exacte moment van de crash kunnen zich verschillende situaties voordoen:

- Als de producer het event nog niet heeft geproduceerd: de producer zal het event opnieuw proberen te leveren. Dit vormt geen probleem.
- Als de producer het event heeft geproduceerd en geen andere status hoeft bij te houden (meestal wanneer de event-broker de enige bron van waarheid is): dit vormt geen probleem.
- Als de producer het event heeft geproduceerd, maar het event nog niet als afgeleverd heeft gemarkeerd: de producer zal het event opnieuw proberen te produceren, wat leidt tot dubbele events. Zonder mitigerende maatregelen aan de consumer-zijde kan dit tot problemen leiden.

3.2.2. Consumer garanties

Wat betreft consumer-garanties vanuit een event-broker zijn er drie opties:

- At most once: een bericht kan worden geconsumeerd, maar nooit meer dan één keer. Dit kan leiden tot verlies van berichten en wordt daarom zelden, zo niet nooit, gebruikt. Dit treedt typisch op wanneer een bericht als verwerkt wordt gemarkeerd vóór de verwerking, maar de verwerking daarna faalt.
- At least once: een bericht wordt geconsumeerd, maar kan meer dan één keer worden geconsumeerd. Dit treedt typisch op wanneer een bericht pas ná de verwerking als verwerkt wordt gemarkeerd, maar het markeren faalt en het bericht opnieuw wordt geconsumeerd. Zonder aanvullende maatregelen kan dit leiden tot dubbele verwerking.
- Exactly once: een bericht wordt exact één keer geconsumeerd én verwerkt (oftewel in één transactie de verwerking plaatsvinden en markeren dat het bericht is geconsumeerd). In onze gedistribueerde opzet is dit per definitie niet mogelijk, vanwege de uitdagingen rond atomiciteit.

3.2.3. End-to-end exactly-once verwerking in een EDA

In deze sectie bekijken we hoe echte end-to-end exactly-once verwerking kan worden gerealiseerd binnen een Event-Driven Architecture (EDA). Daarbij zorgen we ervoor dat:

A. Een **producer** **of intermediary** events **at least once** produceert. In normale omstandigheden produceert een producer altijd exactly once, maar in foutscenario's kan dit vaker gebeuren.
B. Een **consumer** events **at least once** consumeert. In normale omstandigheden consumeert een consumer altijd exactly once, maar in foutscenario's kan dit vaker gebeuren.
C. De verwerking **exactly once** wordt uitgevoerd door middel van idempotentie (zie bijlage B) in de ontvangende service.

Dit zorgt voor een minimale extra impact op de performance aan de verwerkingskant en waarborgt uiteindelijke atomiciteit zonder dataverlies of duplicatie.

3.2.4. Trade-offs

We beogen een moderne EDA. We bekijken in deze paragraaf hoe traditionele patronen voor transactionaliteit (Two-phase commit en Saga, zie bijlage A) en moderne patronen (idempotentie, zie bijlage B) bijdragen aan end-to-end exactly-once verwerking in een EDA setup.

	2PC	**Saga**	**Idempotentie**
Schaalbaarheid	---	+	+++
Complexiteit	- (coördinatie)	--- (compenserende acties, orkestratie)	+
Latency	---	-	+++
EDA-vriendelijk	---	+++	+++
Impact op leveranciers/keten	---	---	-
Eindscore	---	+-	++

Het implementeren van 2PC of een Saga patroon vereist dat interfaces en systemen met oog op deze patronen ontwikkeld worden. Voor bestaande systemen / leveranciers in de keten kan dit daarmee flinke impact hebben. Daarnaast geldt dat significant meer berichtuitwisselingen nodig zijn om exactly-once verwerking te bewerkstelligen.

Bij idempotency geldt dit niet. Uitgangspunt is dat er altijd 1 bericht op de lijn gaat zoals dit ook plaatsvindt; er is dus 1 bericht nodig voor datauitwisseling. Met name in foutsituaties bij producer of consumers kunnen ervoor zorgen dat berichten meermaals respectievelijk geproduceerd of geconsumeerd worden. In bijlage B worden richtlijnen voor het implementeren van idempotentie beschreven.

Met opmerkingen [EV21]: Er wordt hier een keuze gemaakt. Edu-V ziet dit meer als een default, maar niet exclusief. De keuze willen we graag binnen de ketensamenwerking in relatie tot het de bijbehorende ketenproces de afspraak willen of kunnen maken.

3.3. Registratie

Met opmerkingen [EV22]: Set-up proces? In hoeverre is deze anders dan bij M2M?

De ketenpartner die verantwoordelijk is voor de producer (hierna: producer) is ervoor verantwoordelijk dat een portaal en/of proces beschikbaar gesteld voor registratie van de ketenpartners die verantwoordelijk is voor de consumer (hierna: consumer).

De registratie volgt een aantal stappen:

- Ketenpartners spreken het voornemen uit naar elkaar om berichtuitwisseling via CloudEvents op te zetten.
- De consumer registreert zich bij de producer.
- De producer accepteert de registratie van de consumer.

Na registratie abonneert de consumer zich bij de producer op de ontvangen berichten. Het is de verantwoordelijkheid van de consumer om zich te abonneren bij de producer voor de berichten die gegevensleveringen die noodzakelijk zijn. De consumer levert hiervoor een webhook endpoint aan.

Bij abonneren is het van belang dat de producer alleen gegevensleveringen naar consumers mogelijk maakt die toegang mogen hebben tot de data die wordt uitgewisseld. Dit stelt de

producer zeker via autorisaties. De producer mag het mogelijk maken via portaal/proces om een afleverlocatie te configureren / aan te leveren.

Het niveau waarop abonnementen (en daarmee de scope van een abonnement) worden vastgelegd hangt nauw samen met verschillende aspecten waaronder de data die wordt uitgewisseld, de beoogde type berichten die moeten worden uitgewisseld en het niveau van autorisatie op de data die gewenst is. Ter illustratie zou je kunnen abonneren op 'events' van individuele personen of alle personen; dit maakt het verschil tussen respectievelijk een miljoen+ abonnement of één abonnement. Dit is erg use-case afhankelijk en we benoemen daarom hier het belang om afstemming hierover te laten plaatsvinden tussen ketenpartijen. Het is buiten de scope van dit document om dit op een use-case basis uit te werken.

Met opmerkingen [DG23]: Nav overleg 30 maart: Ik heb het belang van niveau/scope abonnementen hier expliciet benoemd nav het vorige werkgroep overleg. Ik denk dat net zoals het ontwerpen van een API dat dit erg use-case afhankelijk is en richtlijnen lastig zijn om op te stellen. Ziet de werkgroep:
1. hier nog aanvullende richtlijnen die we hiervoor moeten opstellen?
2. het logisch om dit op te nemen in de standaard?

Met opmerkingen [EV24R23]: Ben voor om te blijven denken in scope.

3.4. Interacties

In deze paragraaf worden de interacties en met name vereisten van elke rol verder uitgediept.

3.4.1. Producer (incl. intermediary producer kant)

Een producer is een systeem dat voor één of meer gegevensuitwisselingen (in verschillende ketensamenwerkingen) CloudEvents produceert en aflevert (via zijn eigen gekoppelde Intermediary) bij geregistreerde Intermediaries. Voor simplificatie beschouwen we beide componenten hier als de producer waarin de ketenpartner zelf vorm kan geven aan de implementatie qua scheiding producer / intermediary, uiteraard zonder de richtlijnen uit deze paragraaf uit het oog te verliezen.

De producer houdt bij het opstellen van requests rekening met aan welke geregistreerde partijen berichten dienen te worden aangeleverd (conform registratie) en stuurt de berichten naar het registreerde afleverpunt (i.e. een webhook) van de Intermediary (consumer kant). Hierbij houdt de producer rekening met eventuele verschillende versies die een bericht kan hebben.

Indien de producer eigenaar is van de berichtspecificatie, registreert de producer de technische contracten bij het contract register (zie 3.5 Technische contracten).
Voor de producer gelden aanvullend de volgende richtlijnen:

- Een producer *MOET* best-practices in event-driven architecturen ondersteunen om at-least once produceren van events en exactly-once verwerking mogelijk te maken, door middel van bijv. het transactioneel outbox patroon (paragraaf 7.2.3) en idempotentie (hoofdstuk 8).
- Een producer *MOET* rekening houden met rate-limiting van de intermediary en past throttling toe waar nodig.
- Een producer *MAG* CloudEvents bufferen en als batch aanleveren, indien de ontvanger CloudEvents in batch ondersteunt.
- Een producer *MAG* circuit breaking toepassen bij ongeplande niet-beschikbaarheid.

~~3.5.~~3.4.2. Intermediary (consumer kant)

De intermediary uit zich naar de buitenwereld als een beveiligd endpoint. Het endpoint verwacht een payload in lijn met de CloudEvents standaard. Dit betreft daardoor twee varianten:

1. Endpoint voor individuele CloudEvents berichten (verplicht)
2. Endpoint voor CloudEvents in batch (optioneel)

De intermediary registreert het technische contract van de webhook bij het contract register (zie 3.5 Technische contracten).

De intermediary heeft een aantal taken:

- voert authenticatie en autorisatie uit;
- accepteert het verzoek;
- valideert het verzoek (minimaal de CloudEvents structuur);
- persisteert het verzoek;
- initieert verwerking bij de consumer (optioneel). Dit vindt plaats in geval van optie A uit Figuur 3 - CloudEvents uitwisseling;
- geeft een technische bevestiging dat het bericht is ontvangen. Functionele verwerking vindt asynchroon plaats aan de kant van de consumer.

De intermediary *mag* rate-limiting toepassen om zijn systemen weerbaarder te maken. In het algemeen wordt aangeraden om rate-limiting toe te passen per producer.

~~3.6.~~3.4.3. Consumer

De consumerende partij bepaalt hoe de ~~ontvanger~~ consumer geïmplementeerd wordt. Hier kunnen verschillende patronen gebruikt worden zoals beschreven in Figuur 3 - CloudEvents uitwisseling:

A. De consumer wacht op berichten voor verwerking die actief door de Intermediary aangeleverd kunnen worden (PUSH).
B. De consumer haalt de berichten actief bij de intermediary (PULL).

~~Een~~ De ~~consumer~~ ontvanger *MOET* best-practices in event-driven architecturen ondersteunen om at-least once consumeren van events en exactly-once verwerking mogelijk te maken, door middel van bijv. idempotentie (hoofdstuk 8), retries, DLQ en replays. De ontvanger MOET dit waarborgen in de intermediary óf consumer; deze vrijheid ligt bij de consumerende partij. In het algemeen is het aan te raden dit soort maatregelen te treffen zo dicht mogelijk bij de plek waar de verwerking plaatsvindt.

~~3.7.~~3.5. Technische contracten

Ketenpartner bepalen altijd gezamenlijk wie welke contracten opstelt. Desondanks, stellen we belangrijk richtlijnen op die kunnen worden gehanteerd. We hanteren de volgende richtlijnen:

- De OpenAPI-specificatie (OAS) van de webhook wordt gedefinieerd door de ontvangende partij.
- De AsyncAPI-specificatie (AAS), i.e. de business data, wordt gedefinieerd door de producer, omdat deze eigenaar is van data en het gedrag. Hierbij stemt de producer

Met opmaak: Kop 3

Met opmerkingen [RR25]: Ik verwacht dat dit geen volgorde lijst is? In High volume omgevingen zullen e en f bijvoorbeeld omgedraaid zijn.

Met opmerkingen [DG26R25]: Verwerkt. Nu een unordered list.

Met opmaak: Kop 3

Met opmerkingen [RR27]: Ik heb grote moeite met het grote aantal "MOET" voorschriften t.a.v. de consumer. De intermediaries aan zowel producer als consumer kant kunnen veel van de event driven afhandeling implementeren en vooral bij legacy applicaties ook afschermen van die applicaties. Daarom stel ik voor om de eisen vooral neer te leggen bij de intermediaries aan producer en consumer kant en richting producer en consumer vooral adviezen te geven

Met opmerkingen [DG28R27]: Ik kan deze beredenering goed volgen voor deze eis. Het lijkt mij goed om in de voorschriften de impact of legacy applicaties te beperken.

Wat mij betreft pas we dat dan ook aan. In de basis ben ik het eens met het voorstel, waarbij ik het liever nét iets anders insteek:

1. De eisen leggen we bij verzender óf ontvanger (i.e. een ketenpartner); waarbij de ketenpartner zelf de vrijheid heeft over wat bij consumer / producer / intermediary wordt geïmplementeerd.

2. We adviezen dit soort best-practices zo dicht mogelijk bij de eindverwerker (de consumer te implementeren); dit is géén MUST.

Ik heb een tekstvoorstel gedaan.

Met opmerkingen [DG29R27]: Bespreken: Aangepaste tekst.

Met opmerkingen [ER30]: Is het niet zo dat Async ook de client spec (gaat) opnemen? Ook voor OAS en AAS moeten we dan expliciete versie opnemen. In AR en OAuth is dit nog niet opgenomen. Aandachtspunt

de specificatie af op de eisen van de mogelijke consumers. Datacontracten hanteren idealiter waar mogelijk sectorstandaarden voor gegevensstructuren.

Zoals elke richtlijn kan daar met goede argumentatie van afgeweken worden, indien noodzakelijk. Zo kan het wenselijk zijn dat een partij die een centrale voorziening levert afspraken maakt over welke gegevens ontvangen kunnen worden. Daarom bepalen ketenpartners altijd gezamenlijk definitief *wie* de specificatie opstelt en onderhoudt.

~~3.7.1.~~3.5.1. Contract-register

Contracten worden op één plek geregistreerd door de partij die de specificatie definieert; elke ketenpartner zal zo'n contract-register aanbieden. Deze plek, een contract-register, is de bron van waarheid voor de betreffende specificatie.

Met opmerkingen [EV31]: Graag toelichten. Edu-V beheert de standaard. Worden zijn dan ook verantwoordelijk voor het contract-register. En wat is hier het contract?

Een contractregister maakt het mogelijk om afspraken over schemastructuren, eigenaarschap en kwaliteit te definiëren en centraal op te slaan en versioneren. Het zorgt ervoor dat systemen veilig met elkaar kunnen communiceren door afspraken over berichtenformaten, API-specificaties en events te beheren. Een contractregister maakt dit mogelijk voor synchrone en asynchrone APIs.

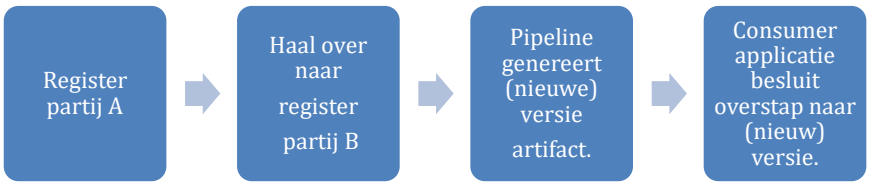
Het is mogelijk om vanuit beveiligingsoogpunt eerst de specificatie over te zetten naar het 'eigen' intern register. Dit is weergegeven hieronder:

Figuur 4: Interactie tussen registers partijen.

Andere partijen kunnen deze specificatie dan handmatig en/of geautomatiseerd ophalen om de andere kant van de integratie op te zetten. Automatisering kan helpen om de berichtspecificatie op te halen uit de bron en daaruit direct, of via een intern register dat een kopie opslaat, automatisch code te generen.

Bijvoorbeeld als volgt:

- Register partij A bevat het AsyncAPI contract.
- Partij B haalt het contract over register A naar een eigen immutable register B.
- Een pipeline draagt zorg voor het ophalen van dit contract uit het interne register, genereren, compileren, packaging en publicatie van het artifact naar een interne repository.
- Een automatische dependency updater kan detecteren dat er een nieuwe versie beschikbaar is, de versie ophogen en een ontwikkelaar laten besluiten of dit direct doorgevoerd kan worden. Belangrijk: Een consumer applicatie beslist zelf wanneer een upgrade plaatsvindt.



Figuur 5: Voorbeeld proces over rol register en automatisering.

~~3.8.~~3.6. CloudEvent beveiligingsopties

De beveiliging vindt plaats op applicatielaag en IT-infrastructuurlaag (verplicht). Hiervoor wordt gebruikt gemaakt van ~~twee niveaus (verplicht):~~

- ~~Applicatielaag: Maakt gebruik van~~ het OAuth client credentials profiel voor RESTful API's.
- ~~IT-infrastructuurlaag: Maakt gebruik van Transport Layer Security (TLS) conform UBV TLS.~~

Met opmaak: Standaard, Geen opsommingstekens of nummering

Dit Edukoppeling-profiel definieert geen aanvullende beveiligingsmechanismen. Zo geldt:

- De payload wordt niet aanvullend beveiligd. Dat is niet nodig omdat het uitgangspunt machine-to-machine (M2M) is. Vooralsnog wordt de payload alleen beveiligd in transport (TLS).
- Abuse protection zoals beschreven in de webhook standaard[21] is niet nodig, omdat OAuth2 al de toegang regelt tot het webhook-endpoint. Indien de consumer toegang verleent aan de producer via OAuth2, wordt daarmee ook de producer goedgekeurd om berichten naar de consumer te sturen.

De volgende voorschriften gelden wel voor CloudEvents:

- Context-attributen: Sensitieve informatie ZOU niet opgenomen moeten worden in context-attributen aangezien producers, consumers en intermediairs deze attributen mogen loggen.
- Domein specifieke data (data-attribuut): Domein specifieke event data wordt niet versleuteld.

Aanvullende beveiliging afspraken worden, waar noodzakelijk, afgestemd tussen de producers en consumers. Zo geldt:

a. Versleuteling is alleen noodzakelijk in scenario's waar (niet vertrouwde) tussenliggende componenten voor datalekken kunnen zorgen. Versleutelen is rekenintensief en het maakt het bovendien moeilijker voor beveiligingsmechanismen, zoals API-gateways, de payload te valideren en transformeren (indien nodig).
b. Ondertekening van gegevens is alleen nodig indien er een risico is dat de integriteit van de gegevens aangetast kan worden of als onweerlegbare overdracht vereist wordt.

[21] https://github.com/cloudevents/spec/blob/v1.0.1/http-webhook.md

Gewijzigde veldcode

4. Voorschriften CloudEvents

Met opmerkingen [ER32]: Levert de nieuwe EK AR en info rond Asycnhroon / EDA nog aanvullende voorschriften op?

Dit hoofdstuk beschrijft de aanvullende voorschriften op het NL GOV for CloudEvents versie 1.1. Het vormt de basis voor een Edukoppeling CloudEvents profiel (asynchrone communicatie via RESTful APIs).

De voorschriften in dit hoofdstuk zijn aanscherpingen op het NL GOV profile for CloudEvents. Het geheel aan voorschriften biedt een Edukoppeling CloudEvents profiel die kan worden toegepast wanneer een confidential client via een M2M point-to-point verbinding interacteert met een gesloten API op asynchrone wijze. Bij asynchrone communicatie stuurt een computersysteem een bericht of event en ontvangt hoogstens een *bevestiging* van ontvangst. De *verwerking* vindt losgekoppeld en later plaats en levert geen direct antwoord op.

Het Edukoppeling CloudEvents profiel is bewust in context van een producer en consumer geplaatst, maar dicteert niet hoe elke ketenpartner communicatie tussen de producer/consumer en hun intermediair vormgeven; het profiel focust op de interactie tussen de ketenpartners en hun intermediairs. Het is aan elke ketenpartner om die scheiding zelf goed te leggen. Wel geven de voorschriften aan waar een ketenpartner aan moet voldoen; in welke component dat precies wordt geïmplementeerd is aan de ketenpartner. Voor simplificatie, binnen de voorschriften wordt geschreven over ‘verzender’ en ‘ontvanger’: respectievelijk de combinatie van een intermediair, en producer of consumer.

De voorschriften zijn verdeeld in algemene voorschriften, voorschriften voor de verzender, ontvanger en een contractregister.

4.1. Algemene voorschriften

1. MUST: Dit profiel moet worden toegepast wanneer er binnen een ketensamenwerking asynchrone gegevensuitwisseling plaatsvindt tussen confidential clients en RESTful API's. Dit geldt niet alleen voor simpele notificaties en signalen, maar ook voor het bereiken van ontkoppelde gegevensuitwisselingen; een event bevat naast de gebeurtenis ook een referentie naar het object. De volgende richtlijnen gelden hier voor:
    a. Voor grote dataobjecten (lees: bestanden groter dan ‘size limits’ – zie voorschrift nummer 3) geldt dat een referentie naar de locatie van het object wordt opgenomen.
    b. Events zijn van nature geschikt om te informeren over individuele gebeurtenissen / entiteiten. Goed ontwerp van events zorgt dat niet volledige geaggregeerde objecten hoeven worden meegeleverd (bijv. een school event bevat informatie over de school, maar niet alle gekoppelde studenten en daaronder gekoppelde data). Het ontwerp van events en richtlijnen daarvoor is verder niet in scope van dit profiel.

2. MUST: Dit Edukoppeling CloudEvents-profiel is alleen van toepassing in de context van confidential clients. De client moet de vertrouwelijke gegevens voor authenticatie (client secret) veilig kunnen opslaan en hier veilig over kunnen beschikken.

Met opmerkingen [RR33]: Dit zou betekenen dat dit verplicht zou moeten zijn voor OKE. Wat mij betreft is dat een te grote stap.

Met opmerkingen [DG34R33]: Bespreken.

Wat ondersteunende vragen:
1. Wanneer moet dit profiel toegepast worden, zeker in relatie tot bestaande systemen en asynchrone interacties?
2. Wat betekent de introductie van nieuwe versie van Edukoppeling voor bestaande systemen?

Met opmerkingen [EV35R33]: Eens met René. Edu-V zal het binnen bepaalde scenario’s toepassen maar niet bij alles wat confidential is.

3. MUST: Dit Edukoppeling-profiel volgt de richtlijnen zoals beschreven in het NL GOV profile for CloudEvents v1.1 [22]:
    a. Context attributen (H3);
    b. Event Data (H4);
    c. Size Limits (H5);
    d. Gebruik van JSON, HTTP en webhook (bijlage A) zoals ook beschreven in de Guidelines for NL-GOV profile CloudEvents v1.0 [23].

    Met uitzondering van:
    e. CloudEvent Security Options (H6);
    f. Abuse protection as described in the guidelines[24].

4. MUST: Beveiliging tussen intermediairs op basis van het Edukoppeling OAuth-profiel[25]. Dit Edukoppeling CloudEvents-profiel volgt het OAuth client credentials profiel voor RESTful API's voor machine-to-machine (M2M) gegevensuitwisseling binnen het onderwijs en schrijft daarmee ook het gebruik van Transport Layer Security (TLS) voor conform UBV TLS.

5. MUST: Een ketenpartner implementeert een Intermediair voor ontvangst en versturen van Events. De ketenpartner bepaalt zelf hoe de koppeling met het achterliggende landschap (de producer / consumer) wordt gerealiseerd en gescheiden, uiteraard zonder de overige voorschriften uit het oog te verliezen.

6. MUST: Een ketenpartner implementeert een Contract-register voor registratie van technische contracten. Hiervoor geldt aanvullend:

    a. MUST: Een ketenpartner stelt het contract-register voor ketenpartners beschikbaar met een OpenAPI-specificatie (OAS).
    b. MUST: Een contract-register API maakt het mogelijk om contracten uit de lijsten, versie van contracten uit te lijsten én een individuele versie van een contract op te halen. Het contract bestaat minimaal uit de API-specificatie.
    c. SHOULD: Een contract-register API geeft de mogelijkheid om aanvullende metadata rondom status, eigenaarschap en contactpunt voor een contract op te halen.
    d. MAY: Het is mogelijk om vanuit beveiligingsoogpunt de specificatie over te zetten van het 'eigen' intern register naar het contract-register dat extern beschikbaar gesteld worden.
    e. MAY: Ketenpartners mogen specificaties uit het contract-register ophalen als onderdeel van handmatige en geautomatiseerde processen.

Met opmerkingen [EV36]: Edu-V snapt dit maar wil wij binnen het hele stelsel op implementatie vrijheid houden. Dus structuur MUST, implementatie SHOULD/MAY

Met opmerkingen [RR37]: Beveiliging tussen intermediairs

Met opmerkingen [DG38R37]: Verwerkt.

Met opmerkingen [RR39]: Waarom niet dezelfde aanpak als bij EDU-koppeling? Wel een contractregister maar niet per se via een API? Een OpenAPI pagina zou in basis genoeg moeten zijn.

Met opmerkingen [DG40R39]: Bespreken: Willen we dit wel/niet afwakken?

[22] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/1.1/
[23] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/guidelines/1.0/
[24] https://github.com/cloudevents/spec/blob/v1.0.1/http-webhook.md#4-abuse-protection
[25] Link opnemen naar gepublic doc

Gewijzigde veldcode

Gewijzigde veldcode

Gewijzigde veldcode

4.2. Verzender voorschriften

7. MUST: De verzender biedt een mogelijkheid voor ontvangers om te registreren en abonneren. Bij abonneren is het van belang dat de verzender alleen gegevensleveringen naar ontvangers mogelijk maakt die toegang mogen hebben tot de data die wordt uitgewisseld. Dit stelt de verzender zeker via autorisaties.

    a. MAY: De verzender mag het mogelijk maken via portaal/proces om een afleverlocatie te configureren / aan te leveren.

8. MUST: De verzender houdt bij het opstellen van requests rekening met aan welke geregistreerde partijen berichten dienen te worden aangeleverd (conform abonnementen) en stuurt de berichten naar het registreerde afleverpunt (i.e. een webhook) van de Intermediary (ontvanger kant). Hierbij houdt de verzender rekening met eventuele verschillende versies die een bericht kan hebben.

9. MUST: Een verzender moet best-practices in event-driven architecturen ondersteunen om at-least once events te produceren en exactly-once verwerking mogelijk te maken, door respectievelijk (bijvoorbeeld) het transactioneel outbox patroon (paragraaf 7.2.3) en idempotentie (Bijlage C: Richtlijnen voor idempotentie).

10. MUST: Een ~~producer~~ verzender moet rekening houden met rate-limiting van de ontvanger en past throttling toe waar nodig.

11. MAY: Een ~~producer~~ verzender mag CloudEvents bufferen en als batch aanleveren, indien de ontvanger CloudEvents in batch ondersteunt.

12. MAY: Een ~~producer~~ verzender mag circuit breaking toepassen bij ongeplande niet-beschikbaarheid.

4.3. Ontvanger voorschriften

13. MUST: Een ontvanger registreert zich bij de verzender die het event produceert.

14. MUST: Een ontvanger abonneert zicht bij de producerende ketenpartner op de berichten die de ontvanger wenst te ontvangen. Na registratie abonneert de ontvanger zich bij de verzender op de ontvangen berichten. Het is de verantwoordelijkheid van de ontvanger om zich te abonneren bij de verzender voor de berichten die gegevensleveringen die noodzakelijk zijn. De ontvanger levert hiervoor een webhook endpoint en edukoppeling credentials aan.

15. MUST: De ~~Intermediair~~ ontvanger implementeert een endpoint voor individuele CloudEvents berichten.

16. MAY: De ~~Intermediair~~ ontvanger implementeert een endpoint voor CloudEvents in batch.

Met opmerkingen [RR41]: en edukoppeling credentials

Met opmerkingen [DG42R41]: Verwerkt.

17. MUST: Een ontvanger consumeert events at least once, via PUSH óf PULL vanuit de Intermediair. In normale omstandigheden consumeert een ontvanger altijd exactly once, maar in foutscenario's kan dit vaker gebeuren.

18. MUST: Een ~~consumer~~ ontvanger verwerkt events exactly once door middel van idempotentie, retries, DLQ en/of replays in de intermediary en/of consumer.~~.~~

    a. SHOULD: Het is aan te raden bovenstaande maatregelen te treffen zo dichtbij het doelsysteem waar de verwerking plaatsvindt (i.e. consumer).

    ~~18.~~b. MAY: Om moderne flows i.c.m. legacy applicaties voldoende ruimte te geven mag dit ook anders geïmplementeerd worden, bijv. in de Intermediary.

19. MUST: Een ontvanger geeft een technische bevestiging aan de verzender dat het bericht is ontvangen. Functionele verwerking vindt asynchroon plaats aan de kant van de ontvanger.

20. MAY: Een ontvanger mag rate-limiting toepassen om zijn systemen weerbaarder te maken.

    a. SHOULD: Indien rate-limiting wordt toegepast, wordt ~~i~~In het algemeen ~~wordt~~ aangeraden om rate-limiting toe te passen per producer.

    ~~20.~~b. SHOULD: Indien rate-limiting wordt toegepast, wordt in het algemeen afgeraden om rate-limiting toe te passen op ip-adres. Verschillende producers kunnen immers vanaf hetzelfde ip-adres geserviced worden.

4.4. Contractregister voorschriften

21. MUST: De technische contracten worden geregistreerd in een contract-register door de ketenpartner die de specificatie opstelt en onderhoudt (zie punt 23 en 24). We onderkennen de volgende contractspecificaties:
    - De OpenAPI-specificatie v3.2.0[26] (OAS) van de webhook, oftewel van contextuele data, conform de CloudEvents specificatie.
    - De AsyncAPI-specificatie v3.1.0[27] (AAS) van de business data.

22. MUST: Ketenpartners bepalen altijd gezamenlijk definitief wie de specificatie opstelt en onderhoudt.

23. SHOULD: De OAS van de webhook wordt gedefinieerd door de ontvangende partij.

24. SHOULD: De AAS wordt gedefinieerd door de ~~producer~~verzender, omdat deze eigenaar is van data en het gedrag. Hierbij stemt de verzender de specificatie af op de eisen van de mogelijke ontvanger(s).

25. SHOULD: Datacontracten hanteren idealiter waar mogelijk sectorstandaarden voor gegevensstructuren.

[26] https://spec.openapis.org/oas/v3.2.0.html
[27] https://www.asyncapi.com/docs/reference/specification/v3.1.0

Met opmerkingen [RR43]: Dit is een MAY: Je kan van legacy applicaties achter een intermediair niet verwachten dat ze moderne flows aankunnen

Met opmerkingen [DG44R43]: Verwerkt, conform comment in 3.4.3.

Met opmaak: Inspringing: Links: 1,27 cm, Geen opsommingstekens of nummering

Met opmaak

Met opmaak: Inspringing: Links: 1,27 cm, Geen opsommingstekens of nummering

Met opmerkingen [RR45]: Ik zou dit zelfs sterker willen uitdrukken. Bijvoorbeeld ook: Het wordt afgeraden om rate-limiting toe te passen op ip adres (omdat bv bij SaaS omgevingen verschillende producers vanaf hetzelfde ip adres geserviced worden)

Met opmerkingen [DG46R45]: Verwerkt en ook opgenomen.

Met opmaak

Met opmerkingen [RR47]: Volgens mij conflicteert dit met 21. Het enige wat de ontvangende partij zelfstandig kan definiëren is het endpoint en de credentials (binnen de vrijheidsgraden die de keten legt op het gebruik van Edukoppeling)

Met opmerkingen [DG48R47]: Bespreken. Ik zie het conflict nog niet; wellicht mis ik een subtiliteit.

- 21 specificeert de 'wat': dat contracten worden opgenomen in een contract-register (2 soorten worden onderkend)

- 23 specificeert een 'wie': dat de OAS (endpoint + definitie) wordt gedefinieerd door de ontvanger (in lijn met de Edukoppeling standaard).
- 24 specificeert een 'wie': dat de AOS wordt gedefinieerd door de verzender.

Met opmerkingen [RR49]: Zelfde als bij punt 23 maar dan aan verzender kant. Zou ook voor de partijen verzender en ontvanger gebruiken en termen als producer en consumer reserveren voor de specifieke componenten.

Met opmerkingen [DG50R49]: 1.Punt 23: Zie vorige comment.
2. Verwerkt: Ik heb producer vervangen door verzender.

26. SHOULD: Sensitieve informatie zou niet opgenomen moeten worden in context-attributen aangezien producers, consumers en intermediairs deze attributen mogen loggen.

5. Toepassingsscenario's

5.1. Probleemstelling - wat moet er met het profiel geregeld worden?

Voor onderwijsorganisaties, sectorvoorzieningen en ketenpartners die binnen een ketensamenwerking met elkaar gesloten data uitwisselen is het van belang dat er standaarden zijn die uitwisselingen van gegevens op een betrouwbare en robuuste manier mogelijk maakt zonder dat de systemen te nauw gekoppeld zijn aan elkaar. Hoge koppeling tussen systemen zorgt voor meer afhankelijkheid, meer afstemming en daarmee minder flexibiliteit en snelheid. Voor synchrone uitwisseling biedt een REST-profiel al de basis voor een gemeenschappelijke taal en voor asynchrone uitwisselingen beoogt het Edukoppeling CloudEvents deze rol te vullen en daarmee te voorkomen dat partijen bij elke koppeling opnieuw discussie voeren over logistieke details.

Binnen een ketensamenwerking is controle en afstemming over uitwisseling noodzakelijk. Zo geldt dat pieken bij één ketenpartner gecontroleerd door de keten moeten bewegen. Het Edukoppeling CloudEvents profiel moet handvaten bieden die gebruikt kunnen worden om controle uit te oefenen binnen operationele afspraken die ketensamenwerkingen met elkaar maken.

Om binnen een ketensamenwerking betrouwbare en robuuste manier gegevens uit te wisselen zijn er richtlijnen nodig die dit ondersteunen zodat ook bij *tijdelijke storingen* het resultaat deterministisch is. Dit is niet mogelijk in een gedistribueerde ketensamenwerking zonder richtlijnen. Asynchrone uitwisseling die vaak worden gehanteerd bij producer – consumer patronen met events zoals in event driven architecturen werken van nature meer asynchrone, herhaalbare en mogelijke dubbele gebeurtenissen als gevolg van *at-least*-once *delivery*. Het toepassen van de juiste event levering semantiek, met de juiste aflever- en consumergaranties en bijbehorende richtlijnen voor idempotentie zijn daarom essentieel voor het juiste resultaat.

Het Edukoppeling CloudEvents profiel gaat ketensamenwerkingen daarmee helpen richting betrouwbare uitwisselingen via een moderne architectuur.

5.2. Oplossing - wat biedt het profiel?

We bieden ketens een standaard voor asynchrone uitwisseling van gegevens met een hoog niveau van ontkoppeling. Ketenpartners behouden de flexibiliteit om hun eigen landschap naar eigen inzicht in te richten, maar worden óók gestuurd op interoperabiliteit binnen de ketensamenwerking. De interoperabiliteit wordt bereikt via duidelijke voorschriften die toegepast dienen te worden. Deze voorschriften sturen op betrouwbaarheid, ontkoppeling (flexibiliteit) en controle binnen de ketensamenwerking.

Het Edukoppeling CloudEvents profiel biedt structuur in de vorm van duidelijke scheiding tussen contextuele metadata en business data, en daarmee duidelijkheid en een stabiele API voor het transport van berichten. Hiermee kan een grote verscheidenheid aan berichten via één RESTful API overgebracht worden; hier is niet per berichttype afstemming nodig. Het Edukoppeling CloudEvents profiel specificeert *niet* de business data, onderliggende

gegevensmodel en/of events die uitgewisseld dienen te worden; dit dient binnen de ketensamenwerking via andere (Edustandaard) standaarden geregeld te worden.

Voorkeuren van richting van uitwisseling tussen producer en consumer worden niet door individuele ketenpartners opgelegd. Het profiel biedt juiste de flexibiliteit voor elke ketenpartner om dat binnen het eigen landschap naar wens in te richten. Zo geldt dat binnen een ketenpartner een consumer berichten aangeleverd kan krijgen óf kan ophalen.

6. Bijlage A: Begrippen

Met opmerkingen [DG51]: Veel begrippen zouden ook onder 'architectuur' thuis kunnen horen.

Antwoord: Een antwoord is de inhoudelijke reactie op een verzoek: het resultaat van een operatie of vraag.

API aanbieder (ook wel API provider): Ketenpartner die binnen een ketensamenwerking een RESTful API aanbiedt.

API afnemer (ook wel API provider): Ketenpartner die binnen een ketensamenwerking met een systeem een RESTful API afneemt.

Asynchroon: Bij asynchrone communicatie stuurt een computersysteem een bericht of event en ontvangt hoogstens een *bevestiging* van ontvangst. De *verwerking* vindt losgekoppeld en later plaats en levert geen direct antwoord op.

Atomiciteit: Binnen de grenzen van één enkel computersysteem zorgen transacties ervoor dat óf alle wijzigingen óf geen van de wijzigingen worden opgeslagen. Dit concept van atomiciteit zorgt ervoor dat een transactie "alles of niets" is.

Bevestiging: Een bevestiging van ontvangst geeft alleen aan dat het bericht technisch correct is ontvangen, niet dat het al is verwerkt.

Business data: Domeinspecifieke informatie (oftewel de payload). Dit kan informatie bevatten over de gebeurtenis zelf, details over gewijzigde gegevens of andere relevante gegevens.

Client secret: Een vertrouwelijke sleutel die alleen bekend is bij de client en afhankelijk van de vorm (symmetrisch (wachtwoord) of asymmetrisch (PKI)) ook bij de Authorization Server. Het wordt gebruikt om de client bij de Authorization Server te kunnen authenticeren via het token endpoint.

Confidential client[28]: Een confidential client is een client die draait in een omgeving waar vertrouwelijke gegevens (waaronder het client secret) veilig bewaard kunnen worden (bijvoorbeeld server-side webapplicaties).

Consumer: Een "consumer" ontvangt het event en onderneemt actie op basis daarvan. De consument gebruikt de context en de data om logica uit te voeren, wat kan leiden tot het optreden van nieuwe events.

Consumeren: Het lezen/verkrijgen van een event vanuit de Intermediary.

Contextuele metadata: Contextmetadata zijn vastgelegd in de contextattributen. Tools en applicatiecode kunnen deze informatie gebruiken om de relatie van events met onderdelen van het computersysteem of met andere events te identificeren.

[28] https://datatracker.ietf.org/doc/html/rfc6749#section-2.1

Contract: Een (data) contract beschrijft hoe systemen met elkaar communiceren. Een data contract kan naast de berichtspecificaties van velden/datatypes/verplichte velden bestaan uit metadata zoals eigenaarschap, classificatie, ondersteuning, status (draft, stable), governance regels en eventueel schema evolutie voor migratie tussen versies.

Contractregister: Een contractregister maakt het mogelijk om afspraken over schemastructuren, eigenaarschap en kwaliteit te definiëren en centraal op te slaan en versioneren. Het zorgt ervoor dat systemen veilig met elkaar kunnen communiceren door afspraken over berichtenformaten, API-specificaties en events te beheren. Dit geldt dus voor synchrone en asynchrone APIs.

~~Data: Domeinspecifieke informatie (oftewel de payload). Dit kan informatie bevatten over de gebeurtenis zelf, details over gewijzigde gegevens of andere relevante gegevens.~~

DLQ (Dead Letter Queue): een aparte queue waarin events terechtkomen die na meerdere retries niet succesvol verwerkt konden worden.

Deterministisch: Het resultaat is hetzelfde, ongeacht hoe vaak de operatie wordt herhaald. Niet aan toeval overgelaten.

Idempotentie: Het meerdere keren verwerken van een identiek verzoek veroorzaakt geen schadelijke of onbedoelde neveneffecten of wijzigingen in de systeemtoestand en garandeert een deterministisch resultaat.

Intermediary: Een “intermediary” ontvangt een bericht dat een event bevat met als doel dit door te sturen naar de volgende ontvanger. Dit kan een andere intermediair of een consument zijn. Een typische taak van een intermediair is het routeren van events naar ontvangers op basis van de informatie in de context.

Ontvanger: Een ontvangende ketenpartner; de combinatie van Intermediary en Consumer.

Order guarantee: Order guarantee is het concept dat berichten worden verwerkt in de volgorde waarin ze zijn geproduceerd. Dit voorkomt dat een status wordt teruggezet naar een oude waarde terwijl een nieuwere waarde al is verwerkt.

Producer: De “producer” is een specifieke instantie, proces of apparaat dat de datastructuur aanmaakt die het CloudEvent beschrijft.
Produceren: Het creëren van een event en het afleveren ervan bij de Intermediary.

Replay: Het opnieuw aanbieden van eerder opgeslagen events om ze (opnieuw of alsnog) te verwerken.

Retry: Het opnieuw proberen te verwerken van een event nadat de eerste verwerking is mislukt, vaak vanuit een apart gezette queue.

RESTful API: Een RESTful API is een application programmable interface (API) die HTTP-methoden, zoals GET, POST, PUT, PATCH en DELETE, gebruikt om resources te beheren. Resources worden geïdentificeerd via URIs (Uniform Resource Identifiers) en worden

doorgaans geretourneerd in JSON- of XML-formaat. We noemen ze RESTful omdat ze niet aan alle REST[29] principes[30] hoeven te voldoen.

Synchroon: Bij synchrone communicatie stuurt een systeem een verzoek en wacht op het *antwoord* (response). Dit antwoord wordt pas teruggestuurd nadat de *verwerking* is afgerond.

Tijdelijke storing: Een tijdelijke, kortdurende, vaak vanzelfherstellende fout, bijv. als gevolg voor een korte onderbreking in het netwerkverkeer (engels: transient failures).

Verwerking: Verwerking is het daadwerkelijk uitvoeren van de businesslogica op basis van het ontvangen bericht of verzoek.

Verzender: Een verzendende ketenpartner; de combinatie van Intermediary en Producer.

[29] REST - Wikipedia
[30] Dit Edukoppeling-profiel gaat uit van vertrouwelijke gegevens waarbij ketenpartners weten wat ze van elke vragen binnen een bepaalde ketensamenwerking. Ondersteuning van HATEOAS lijkt niet noodzakelijk.

7. Bijlage B: Uitdagingen in een gedistribueerd landschap

Een gedistribueerd IT-landschap bestaat uit meerdere systemen die met elkaar communiceren door het uitwisselen van berichten. In die zin is er altijd sprake van een zender en een ontvanger van een bericht. We moeten ervoor zorgen dat er geen berichten verloren gaan of dubbel worden verwerkt. Dit zou namelijk leiden tot informatieverlies of dubbele data.

Bijvoorbeeld: in een IT-landschap met een ordersysteem en een betaalsysteem moeten we ervoor zorgen dat een order die in het ordersysteem wordt aangemaakt exact één keer financieel wordt verwerkt.

Robuuste afhandeling betekent dat alle berichten (verzonden door een zender) exact één keer door de ontvanger worden verwerkt, zonder aanvullende neveneffecten.

7.1. Traditionele uitdagingen in een gedistribueerd IT-landschap

Binnen de grenzen van één enkel systeem zorgen transacties ervoor dat óf alle wijzigingen óf geen van de wijzigingen worden opgeslagen. Dit concept van atomiciteit zorgt ervoor dat een transactie “alles of niets” is.

In een gedistribueerd IT-landschap is het echter, zonder aanvullende maatregelen die een gedistribueerde transactie over systeemgrenzen heen waarborgen, niet mogelijk om twee (of meer) systemen atomair bij te werken, omdat er in wezen sprake is van twee afzonderlijke transacties.

Consistentie over gedistribueerde systemen kan wel worden geïmplementeerd, maar dit gaat ten koste van de schaalbaarheid. In de volgende paragrafen verkennen we traditionele patronen om deze uitdagingen op te lossen binnen een microservices-architectuur of andere systeem-tot-systeemkoppelingen.

In paragraaf beschrijven we traditioneel gebruikte patronen in een traditioneel gedistribueerd IT-landschap.

7.2. Traditioneel gebruikte patronen in een gedistribueerd IT-landschap

7.2.1. Patroon: Two-phase commit (2PC)

Het two-phase commit-patroon wordt gebruikt om data op meerdere nodes in een gedistribueerd systeem atomair, volgens het alles-of-niets-principe, op te slaan. De twee fasen in dit patroon worden gecoördineerd door één centrale coördinator:

- Voorbereidingsfase (preparation phase): De coördinator stuurt een verzoek naar alle nodes en vraagt elke deelnemer te controleren of de transactie kan worden voltooid.
- Commitfase (commit phase): Als alle deelnemers positief hebben geantwoord, stuurt de coördinator een commit naar alle deelnemers. Als ten minste één deelnemer negatief heeft geantwoord, stuurt de coördinator een abort naar alle deelnemers.

Belangrijk in dit patroon is dat elke deelnemer in de voorbereidingsfase de duurzaamheid (durability) van de beslissing waarborgt.

7.2.2. Patroon: Saga-patroon

Het Saga-patroon is in essentie een reeks lokale transacties in afzonderlijke systemen. Deze reeks kan op twee manieren worden gecoördineerd:

- Elke lokale transactie publiceert berichten die lokale transacties in andere services activeren; een choreografie van berichtensequenties.
- Een orchestrator instrueert de deelnemers welke lokale transacties zij in welke volgorde moeten uitvoeren.

Binnen dit patroon moeten maatregelen worden genomen om een transactie “ongedaan te maken” wanneer een vervolgstap door een deelnemer niet succesvol kan worden verwerkt. Het ongedaan maken van een transactie is een compenserende actie, en geen strikte rollback.

7.2.3. Patroon: Transactioneel outbox-patroon

Tijdens de verwerking van een businessoperatie schrijft een microservice het ‘uitgaande bericht’ weg in een outbox-tabel binnen dezelfde transactie als de overige datawijzigingen. Een achtergrondproces leest de outbox en publiceert de berichten op betrouwbare wijze naar een message broker.

Dit zorgt ervoor dat er geen berichten verloren gaan wanneer de volledige businesslogica niet succesvol wordt afgerond. Hierdoor worden atomaire writes mogelijk binnen het systeem waarin de businessoperatie plaatsvond.

8. Bijlage C: Richtlijnen voor idempotentie

In deze bijlage kijken we naar de praktische richtlijnen voor het implementeren van idempotentie. Eerst bespreken we *wanneer* dit moet worden toegepast en vervolgens *hoe*. Als laatst worden technische richtlijnen beschreven voor de implementatie.

8.1. Wanneer implementeren

Idempotentie zorgt ervoor dat het meerdere keren verwerken van hetzelfde bericht geen schadelijke of onbedoelde neveneffecten of wijzigingen in de systeemtoestand veroorzaakt en dat een deterministisch resultaat wordt gegarandeerd. Dit betekent ook dat niet alle service-operaties hierdoor worden beïnvloed. Sommige service-operaties zijn per definitie idempotent. Enkele voorbeelden:

- Elke GET-operatie is idempotent, omdat deze geen toestand wijzigt.
- Elke PUT-operatie (volgens de specificaties) vervangt of wijzigt een resource. *Voorbeeld:* PUT /account/123 met {balance: 100} moet het saldo van de rekening op 100 zetten. Opnieuw toepassen levert hetzelfde resultaat op. Order guarantee is hierbij belangrijk.
- Elke DELETE-operatie is idempotent, omdat een entiteit niet meer dan één keer kan worden verwijderd. Aandachtspunt is de response – zie sectie 8.2.1.
- POST-operaties kunnen wel of niet idempotent zijn, afhankelijk van de functionaliteit. *Voorbeeld:* POST /transactions met { amount: 100, account: 123 } kan bij herhaling opnieuw geld toevoegen aan rekening 123, wat leidt tot een onjuist saldo. *Voorbeeld:* POST /email/send zal opnieuw een e-mail versturen (er is dus een neveneffect).

8.2. Algemene richtlijnen

De aanbevolen manier om idempotentie te implementeren is door middel van een uniek event-ID in elk bericht. De ~~consumer~~ ontvanger kan dan de volgende eenvoudige logica toepassen:

- Kent de ~~consumer~~ ontvanger het message-ID al? → Verwerp het bericht.
- Kent de ontvanger ~~consumer~~ het message-ID nog niet? → Verwerk het bericht en sla het message-ID op.

Om volledige atomiciteit te garanderen, wordt het message-ID opgeslagen in dezelfde transactie als de daadwerkelijke businessverwerking.

Een goede message-ID moet globaal uniek, stabiel, deterministisch en compact genoeg zijn. Dit message-ID moet door de producer worden gegenereerd.

8.2.1. Response

Elke idempotente service zorgt ervoor dat de response bij de eerste, tweede en derde poging identiek is. Dit is met name relevant voor services die:

- Data aanmaken: de client wil doorgaans het identificatienummer van de aangemaakte entiteit ontvangen.
- Data verwijderen: de client moet correcte foutafhandeling kunnen implementeren. Dit zou niet mogelijk zijn als bij een tweede DELETE-aanroep een foutresponse wordt teruggegeven.

8.3. Technische implementatierichtlijnen: Idempotentie

Idempotentie zorgt ervoor dat meerdere identieke verzoeken hetzelfde effect hebben als één enkel verzoek. Dit is cruciaal voor scenario's waarin clients verzoeken opnieuw proberen te versturen vanwege time-outs, netwerkfouten of onherstelbare fouten aan de clientzijde.

Deze technische richtlijnen zijn gebaseerd op een bestaand Internet Engineering Task Force (IETF)-concept [31] en bieden aanvullende richtlijnen om consistente implementatie en efficiënte systeemintegratie binnen de onderwijssector te ondersteunen. Eerst worden de idempotentieconventies binnen de sector beschreven, daarna volgt een gedetailleerd implementatievoorbeeld op basis van deze conventies.

8.3.1. Idempotentieconventies

Alle operaties die resources aanmaken of muteren, typisch POST- en PATCH-operaties, *moeten* een Idempotency-Key-header gebruiken. Hoewel DELETE-operaties per definitie [32] idempotent zijn, kunnen clients in een gedistribueerd systeem met retries inconsistente responses krijgen. Daarom *zouden* DELETE-operaties ook een idempotency key moeten gebruiken.

[31] The Idempotency-Key HTTP Header Field
[32] RFC 9110: HTTP Semantics

Met opmerkingen [RR52]: Ook hier, zou ontvanger gebruiken ipv consumer omdat je hiermee iets oplegt aan het applicatielandschap van de ontvanger

Met opmerkingen [DG53R52]: Verwerkt.

Gewijzigde veldcode

Elke operatie die niet voldoet aan de standaarden voor het gebruik van HTTP-methoden [33] en resources aanmaakt of wijzigt, *moet* eveneens worden meegenomen.

8.3.2. Uniekheid idempotentie sleutel

Een UUIDv4 [34] (RFC4122) *moet* worden gebruikt als idempotentie-sleutel.

8.3.3. Geldigheid en verloop van idempotentie sleutel

De geldigheid en vervaldatum van de idempotency key *zouden* rekening moeten houden met de tijd die nodig is voor automatische en/of handmatige retry-mechanismen en met de mogelijkheid van herhalingen (worstcasescenario na een restore door de client).

De time-to-live (TTL) voor een idempotency-entry *zouden* gebaseerd moeten zijn op het maximaal verwachte replay-venster: met andere woorden, het maximale tijdsvenster waarin een bericht opnieuw kan worden aangeboden. Een vervalperiode van 7 dagen wordt *aanbevolen*.

8.3.4. Idempotentie fingerprint

Voor extra veiligheid *moet* een idempotentie fingerprint worden aangemaakt. Aanbevolen wordt om de volledige request-payload te gebruiken om deze fingerprint te genereren.

8.3.5. Verantwoordelijkheden client

Clients *moeten* de idempotentie sleutel opslaan voor de volledige levensduur van de operatie (oftewel de TTL die door de resource is gespecificeerd). Dit garandeert dat de sleutel uniek, stabiel, deterministisch en constant blijft gedurende de operatie. De client MAG hiervoor het transactional outbox-patroon gebruiken (zie bijlage A).

Clients *moeten* voor één idempotency key dezelfde request-payload versturen om HTTP 422-fouten van de resource te voorkomen.

Clients *zouden* automatische retries moeten stoppen nadat de TTL van de resource is verlopen. Daarna kan de client er niet langer van uitgaan dat de resource de idempotentie sleutel nog heeft opgeslagen. Handmatige bevestiging binnen de clientapplicatie *zou* dan moeten worden gestart en een nieuw verzoek *moet* een nieuwe sleutel bevatten.

8.3.6. Verantwoordelijkheden resource

De resource *moet* bij een duplicaatverzoek de response retourneren van de eerder voltooide operatie. Met name DELETE-verzoeken *zouden* ook dezelfde response moeten retourneren als eerder.

Als de resource meerdere clients ondersteunt, *moet* de opslag van idempotency keys per client worden gescheiden om het risico op botsingen te elimineren.

Om volledige atomiciteit te waarborgen, *zou* de idempotency key moeten worden opgeslagen in dezelfde transactie als de daadwerkelijke businessverwerking.

[33] RFC 9114: HTTP/3
[34] RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace

Gewijzigde veldcode
Gewijzigde veldcode

8.3.7. Foutafhandeling

De resource *moet* de idempotency key valideren voordat verdere verwerking plaatsvindt. Als een ongeldige idempotency key wordt aangeleverd, *zou* de resource moeten antwoorden met een HTTP 400-statuscode.

8.3.8. Client side implementatie (voorbeeld)

De volgende informatie kan in een outbox worden vastgelegd volgens het transactional outbox-patroon:

- event_id
- event_type
- payload
- schema_version
- created_at
- expires_at (moment waarop het event niet langer geldig wordt verklaard, bijvoorbeeld na het verstrijken van de TTL van de resource)
- idempotency_key (de UUIDv4)
- status (pending, acknowledged, failed)

De logica aan de clientzijde kan er als volgt uitzien:

1. Een business-event wordt vastgelegd inclusief alle metadata.
2. Zoek business-events met status *pending*.
3. Als expires_at > now(), zet de status op *failed*.
4. Anders:
   a. Lees het business-event.
   b. Exporteer het business-event en wacht op de response.
   c. Als de response een 4XX-fout aangeeft, los het probleem in de applicatie op.
   d. Als de response een 5XX-fout aangeeft, zet de status op *failed*.
   e. Als de response succesvolle verwerking aangeeft, verwijder het business-event of zet de status op *acknowledged*.

8.3.9. Resource implementatie (voorbeeld)

Alle API's die resources aanmaken of muteren, typisch POST, DELETE en PATCH-operaties, MOETEN een Idempotency-Key-header ondersteunen.

De volgende informatie kan worden opgeslagen in een idempotency key-tabel:

- idempotency_key
- idempotency_fingerprint (hash van het request)
- resource_id (indien van toepassing)
- client_id (indien van toepassing)
- response_status_code
- response_body
- created_at
- expires_at

De validatie aan de resourcezijde kan er als volgt uitzien:

1. Valideer de aanwezigheid én het formaat van de idempotency key. Indien deze ontbreekt of ongeldig is, retourneer een HTTP 400-fout.
2. Zoek de idempotency_key op voor resource_id + client_id in de tabel.
   a. Indien aanwezig én de idempotency fingerprint niet overeenkomt, retourneer een HTTP 422-fout.
   b. Indien aanwezig, retourneer de opgeslagen response.
3. Probeer een idempotency-lockrecord te verkrijgen inclusief vervaltijd. Als dit mislukt, verwerkt een ander proces deze key op dat moment en wordt een HTTP 409-fout geretourneerd.
4. Start een transactie:
   a. Verwerk de businesslogica.
   b. Voeg het idempotency-record inclusief response toe aan de tabel.
5. Commit de transactie en geef de idempotency-lock vrij.
6. Retourneer de response van de businesslogica.

Daarnaast kan een automatisch proces de idempotency key-entries opschonen waarvoor expires_at > now().

8.4. Voorschriften idempotentie

Dit hoofdstuk beschrijft de voorschriften voor het toepassen van idempotentie. De voorschriften zijn verdeeld in algemene voorschriften, voorschriften voor de client en resource.

8.4.1. Algemene voorschriften

1. MUST: Idempotency voorschriften worden toegepast voor synchrone en asynchrone uitwisselingen.

2. MUST: Alle asynchrone uitwisselingen bevatten een Idempotency-Key-header.

3. MUST: Voor synchrone operaties die resources aanmaken of muteren, typisch POST- en PATCH-operaties, moet een Idempotency-Key-header worden meegegeven.

4. MUST: Elke synchrone HTTP operatie die niet voldoet aan de standaarden voor het gebruik van HTTP-methoden [35] en resources aanmaakt of wijzigt, moet eveneens worden meegenomen.

5. SHOULD: Hoewel DELETE-operaties per definitie [36] idempotent zijn, kunnen clients in een gedistribueerd systeem met retries inconsistente responses krijgen. Daarom *zouden* DELETE-operaties in synchrone uitwisselingen ook een idempotency key moeten gebruiken.

6. MUST: Een UUIDv4 [37] (RFC4122) moet worden gebruikt als idempotentie-sleutel.

[35] RFC 9114: HTTP/3
[36] RFC 9110: HTTP Semantics
[37] RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace

Gewijzigde veldcode
Gewijzigde veldcode
Gewijzigde veldcode

7. SHOULD: De geldigheid en vervaldatum van de idempotency key *zouden* rekening moeten houden met de tijd die nodig is voor automatische en/of handmatige retry-mechanismen en met de mogelijkheid van herhalingen (worstcasescenario na een restore door de client). De time-to-live (TTL) voor een idempotency-entry is 7 dagen.

8. MUST: Voor extra veiligheid moet een idempotentie fingerprint worden aangemaakt. Aanbevolen wordt om de volledige request-payload te gebruiken om deze fingerprint te genereren.

8.4.2. Voorschriften client

9. MUST: Clients *moeten* de idempotentie sleutel opslaan voor de volledige levensduur van de operatie (oftewel de TTL die door de resource is gespecificeerd). Dit garandeert dat de sleutel uniek, stabiel, deterministisch en constant blijft gedurende de operatie.

10. MAY: De client mag hiervoor het transactional outbox-patroon gebruiken (zie paragraaf 7.2.3).

11. MUST: Clients moeten voor één idempotency key dezelfde request-payload versturen. Bij synchrone operaties ontvangt de client anders een HTTP422-fout van de resource.

12. SHOULD: Clients zouden automatische retries moeten stoppen nadat de TTL van de resource is verlopen. Daarna kan de client er niet langer van uitgaan dat de resource de idempotentie sleutel nog heeft opgeslagen. Handmatige bevestiging binnen de clientapplicatie *zou* dan moeten worden gestart.

13. MUST: Clients moeten een nieuwe idempotentie sleutel genereren na het verzoek van de TTL; een nieuw verzoek moet een nieuwe sleutel bevatten.

8.4.3. Voorschriften resource

14. MUST: Bij synchrone operaties moet de resource bij een duplicaatverzoek de response retourneren van de eerder voltooide operatie. Met name DELETE-verzoeken *zouden* ook dezelfde response moeten retourneren als eerder.

15. SHOULD: Bij asynchrone operaties neemt de ontvanger het duplicate event in de Intermediary wel aan en negeert het event later in de keten als de consumer de verwerking doet.

16. MUST: Als de resource meerdere clients ondersteunt, moet de opslag van idempotency keys per client worden gescheiden om het risico op botsingen te elimineren.

17. SHOULD: Om volledige atomiciteit te waarborgen, zou de idempotency key moeten worden opgeslagen in dezelfde transactie als de daadwerkelijke businessverwerking.

18. MUST: De resource *moet* de idempotency key valideren voordat verdere verwerking plaatsvindt. Als een ongeldige idempotency key wordt aangeleverd, antwoord de resource met een HTTP 400-statuscode.

9. Bijlage D: Referenties

- **NL GOV profile for CloudEvents 1.1 (LOGIUS STANDARD)**
  https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/1.1/
- **CloudEvents v1.0.1 (CNCF SPECIFICATION STANDARD)**
  https://github.com/cloudevents/spec/blob/v1.0.1/cloudevents/spec.md
- **HTTP 1.1 Web Hooks for Event Delivery v1.0.1 (CNCF SPECIFICATION STANDARD)**
  https://github.com/cloudevents/spec/blob/v1.0.1/http-webhook.md
- **Guidelines for NL-GOV profile CloudEvents 1.0 (LOGIUS GUIDELINES)**
  https://gitdocumentatie.logius.nl/publicatie/notificatieservices/guidelines/1.0/
- **AsyncAPI specification for defining asynchronous APIs v3.1.0 (LINUX FOUNDATION SPECIFICATION)**
  https://www.asyncapi.com/docs/reference/specification/v3.1.0
- **OpenAPI specification for defining synchronous APIs v3.2.0 (LINUX FOUNDATION SPECIFICATION)**
  https://spec.openapis.org/oas/v3.2.0.html

Idempotence:

- **Idempotency-key HTTP Header Field (IETF PROPOSED STANDARD)**
  https://www.ietf.org/archive/id/draft-ietf-httpapi-idempotency-key-header-07.txt
- **RFC 9110: HTTP Semantics, idempotent methods (IETF STANDARD)**
  https://www.rfc-editor.org/rfc/rfc9110#section-9.2.2
- **RFC 9114: HTTP/3 (IETF PROPOSED STANDARD)**
  https://www.rfc-editor.org/rfc/rfc9114
- **RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace (IETF PROPOSED STANDARD)**
  https://www.rfc-editor.org/rfc/rfc4122

Gewijzigde veldcode