Klimadatastyrelsen

Brugervejledning – Transitionsguide for entitetsbaserede hændelser

Drift og modernisering af Datafordeleren

September 2025

Version 1.0 – 18-09-2025

Klimadatastyrelsen

Indholdsfortegnelse

1 Introduktion 3
1.1 Begreber 3
1.2 Afgrænsninger og forudsætninger 4
2 Hændelser på Datafordeleren 5
2.1 Hvad er hændelser? 5
2.1.1 Hvad betyder data i hændelser? 6
2.2 Overordnet arkitektur 8
2.2.1 Grundlæggende GraphQL for hændelser 8
2.3 Brugsscenarier for hændelser 9
2.3.1 Anvendelse af queries 9
2.3.2 Abonnement på hændelser 9
2.3.3 Forretningsscenarier 11
3 Udstilling af hændelser 13
3.1 Hændelser per register 13
3.2 Abonnement på hændelser 14
3.3 Status på register import 14
3.3.1 Abonnering på RegisterImportStatus 15
3.3.2 Register import metadata i fildownloads 16
3.4 Registre uden hændelser 16
4 Sådan anvender du hændelser 17
4.1 Autorisation for hændelser 17
4.1.1 Autorisation for CPR-hændelser 18
4.2 Mulige filtreringer 18
4.2.1 Hændelser 18
4.2.2 RegisterImportStatus 21
4.3 Send query 21
4.4 Opret abonnement 22
4.4.1 Retursvar 22
5 Transitionsguide 24
5.1 Datafordelerens nuværende hændelser 24
5.1.1 Ligheder med moderniserede hændelser 24
5.1.2 Fordele ved at skifte til moderne hændelser 24
5.1.3 Hændelsesabonnementer og historiske data 25
5.2 Transition af filtrering 25
5.2.1 Oversigt over forskel mellem nuværende hændelsesfiltrering og GraphQL hændelsesfiltrering 26
5.2.2 Praktisk transition 27
5.3 Eksempel på transition til moderniserede hændelser 28
5.3.1 PULL-hændelser 28
5.3.2 PUSH-hændelser 31

1 Introduktion

Organisationer kan være afhængige af opdaterede og præcise data fra offentlige registre. Når en virksomhed skifter adresse, en bygning renoveres, eller en person flytter, skal disse ændringer hurtigt reflekteres i alle relevante it-løsninger for at sikre korrekt sagsbehandling, compliance og kundeservice.

Datafordelerens moderniserede hændelsessystem løser denne udfordring ved at levere automatiske notifikationer om dataændringer i nær-realtid. I stedet for konstant at skulle forespørge registre efter opdateringer, kan organisationer nu modtage målrettede beskeder, når relevante data ændres.

Dokumentet er tiltænkt følgende læsere:

- Eksisterende anvendere af Datafordeleren, der bruger hændelser på den Nuværende Datafordeler og ønsker at bruge de moderniserede hændelser. Dokumentet afsluttes med en transitionsguide til at hjælpe eksisterende anvendere.
- Nye anvendere, der vil i gang med at bruge Datafordelerens hændelser.
- It-arkitekter og udviklere, der skal integrere Datafordelerens hændelser i organisationens egne løsninger.

1.1 Begreber

I dette dokument bruges de begreber, som findes i Tabel 1. Bemærk, at begreberne også er yderligere forklaret i løbet af dokumentet med eksempler.

Begreb	Beskrivelse
Nuværende Datafordeler	
REST-tjenester	REST-tjenester er en metode til at få data fra Datafordeleren. Man kan som anvender hente data ved at bruge en URL. REST-tjenester er beskrevet her https://confluence.sdfi.dk/pages/viewpage.action?pageId=17138461.
Moderniserede Datafordeler	
Entitet	En entitet er en fysisk repræsentation af et registerobjekt som findes i Grunddatamodellen. De entitetsbaserede fildownload, der kan hentes fra Datafordeleren, samt de entitetsbaserede GraphQL-tjenester indeholder hver især data bestående af en enkelt entitet.
GraphQL-skemaer	GraphQL-skemaer er .graphql-filer der genereres på baggrund af registrenes datamodeller og er yderligere beriget med metadata fra hvert register. Skemaerne beskriver, hvordan data er struktureret, og hvordan data kan hentes og udstilles til anvendere.
Paging	Paging er når resultater inddeles i sider således at resultater returneres én side ad gangen.
SSE	Server-Sent Events (SSE) er en web-teknologi der muliggør envejskommunikation efter at brugeren har etableret en session og sendt en forespørgsel, der åbner en forbindelse. Denne kommunikation, sker i realtid fra server til klient via en åben HTTP-forbindelse. Denne bruges til at sende hændelser fra Datafordeleren til anvendere.

Tabel 1: Begrebsliste

1.2 Afgrænsninger og forudsætninger

Hændelser udstilles via Datafordelerens entitetsbaserede GraphQL-tjenester, men dette dokument vil primært fokusere på, hvordan hændelser fungerer. Derfor vil dette dokument ikke gennemgå hvordan GraphQL-tjenesterne fungerer.

Følgende guides giver en grundlæggende forståelse for GraphQL-tjenesterne og deres anvendelse:

- Brugervejledning - Transitionsguide for entitetsbaserede GraphQL-tjenester
- Transition mellem REST-services og GraphQL-services

2 Hændelser på Datafordeleren

Dette kapitel beskriver, hvordan hændelser fungerer på Datafordeleren som et værktøj til at holde registerdata synkroniseret. Hændelser giver anvendere mulighed for at modtage automatiske notifikationer, når data ændres, så de kan holde egne løsninger opdaterede uden at skulle forespørge unødigt efter data.

2.1 Hvad er hændelser?

For et register er en hændelse en ændring af data. En hændelse kan således være resultatet af et trin i en forvaltningsproces (også omtalt som en forretningsproces), som opdaterer data i et register.

Hændelser oprettes automatisk som strukturerede notifikationer, når ændringerne i registerdata indlæses på Datafordeleren. Det fungerer som et system, der kommunikerer dataændringer, når anvendere opretter en forbindelse ved at abonnere på hændelser eller henter hændelser via queries.

Ved at bruge hændelser, kan anvendere hente opdateringer, og holde deres egne data synkroniseret med de nyeste ændringer. For eksempel hvis en virksomhed skifter navn, vil anvendere få en hændelsesbesked, hvorefter de kan opdatere deres kopiregister.

Hændelser er opdelt per register, så anvendere kan vælge kun at modtage hændelser for de enkelte registre, de har behov for. Hændelser per register er implementeret som en entitet, der fungerer på samme måde som andre entiteter, som kan hentes gennem Datafordelerens GraphQL-tjenester. F.eks. har BBR en entitet, der hedder BBR_Events, gennem hvilken alle hændelser for BBR kan hentes.

Hændelser indeholder metadata om ændringerne i registerdata, herunder:

- Hvilken handling der skete (insert, update, delete).
    - Bemærk at moderniserede hændelser modsvarer operationer i databasen, og at hændelser i den Nuværende Datafordeler var forretningshændelser, defineret af registret selv. F.eks. modsvarer en update-hændelse i den Nuværende Datafordeler en update- og to insert-hændelser i den Moderniserede Datafordeler for visse entiteter. Dette er afhængig af entitetens bitemporalitet.
- Navnet på entiteten.
- Et ID der identificerer den specifikke række, så dataene kan spores videre til entiteten med ændringerne.
- Sekvensnummer på den indlæste pakke som dataene kommer fra.

Hændelser er gjort tilgængelige for anvendere med to forskellige metoder:

- Forespørgsel til entitetsbaserede GraphQL-tjenester via query til hændelsesentiteten.
- Abonnement på hændelser via Server-Sent Events (SSE) for opdateringer i næsten realtid.

Typisk vil hændelser blive brugt i en totrinsproces:

- Hent eller modtag hændelser for at se, hvilke objekter der er ændret.
- Hent den faktiske data for de ændrede objekter i separate forespørgsler til GraphQL-tjenesterne.

2.1.1 Hvad betyder data i hændelser?

Hændelser der modtages via GraphQL kan indeholde felterne som vises i Tabel 2, hvis anvender inkluderer dem i sin query.

Feltnavn	Betydning
eventid	Identifikator for den specifikke hændelse.
entityname	Navnet på entiteten som hændelsen omhandler.
eventaction	Handlingen som er foretaget for at ændre registerdataene. Kan enten være "i" for insert, "u" for update, eller "d" for delete.
datafordelerRegisterImportSequenceNumber	Registeret dataindlæsningspakke-sekvensnummer, der indeholder ændringen.
datafordelerOpdateringstid	Tidspunktet hvor dataene (hændelsen) er blevet indsat i databasen hos Datafordeleren.
fromfailedimport	Indikator for om hændelsen er fra en fejlet import, og derfor vil være en duplikat, for at anvendere kan håndtere det i deres løsninger.
object_id	ID på registerdata-objektet som er opdateret. Genereret af registeret, hvis det findes for den specifikke entitet.
object_datafordelerRowId	Identifikator for registerdata-objektet som hændelsen omhandler. Genereret af Datafordeleren.
object_datafordelerRowVersion	Versionen for registerdata-objektet. Feltet starter med værdien 1 og stiger med 1 hver gang registeret opdaterer rækken.
object_registreringfra	Den nye værdi af "registrering fra", fra hændelsens registerdata-objekt, der er blevet opdateret – forudsat at et sådant felt findes.
object_registreringtil	Den nye værdi af "registrering til", fra hændelsens registerdata-objekt, der er blevet opdateret – forudsat at et sådant felt findes.
object_status	Den nye værdi af status, fra hændelsens registerdata-objekt, der er blevet opdateret – forudsat at et sådant felt findes.
object_virkningfra	Den nye værdi af "virkning fra", fra hændelsens registerdata-objekt, der er blevet opdateret – forudsat at et sådant felt findes.
object_virkningtil	Den nye værdi af "virkning til", fra hændelsens registerdata-objekt, der er blevet opdateret – forudsat at et sådant felt findes.

Tabel 2: Datafelter i hændelser

Data i en hændelse er i høj grad baseret på det underliggende dataobjekt, som hændelsen refererer til. En dybere forståelse af felterne kræver derfor kendskab til de specifikke entiteter, som hændelserne er baseret på, og deres struktur.

2.1.1.1 Bitemporal data

Anvendere kan filtrere entiteter baseret på bitemporale data ved hjælp af inputvariablerne object_registreringfra, object_registreringtil, object_virkningfra og object_virkningtil, hvilket muliggør mere præcis og kontekstspecifik datahentning. Nogle entiteter understøtter forespørgsler på begge dimensioner, mens andre kun understøtter én af værdierne - hvilke bitemporale dimensioner, der understøttes for en given entitet, kan ses i tjenestens GraphQL-skema. Læs mere om filtrering og tilladte operatorer i afsnit 4.2 (*Mulige filtreringer*).

2.1.1.2 Specielt om CPR-felter

CPR-registeret har specielle filtreringsmuligheder, der ikke findes i andre registre. Dette skyldes CPR's unikke behov for at kunne spore borgeres tilknytning til kommuner over tid, hvilket er relevant for mange offentlige myndigheder og private virksomheder, der arbejder med persondata. Disse vises i Tabel 3.

De to yderligere felter giver mulighed for at filtrere hændelser baseret på borgeres kommunetilhørsforhold, både nuværende og historisk. Dette er særligt nyttigt når anvendere kun har behov for at modtage hændelser om borgere i specifikke kommuner, eller når de f.eks. skal håndtere kommuneskift.

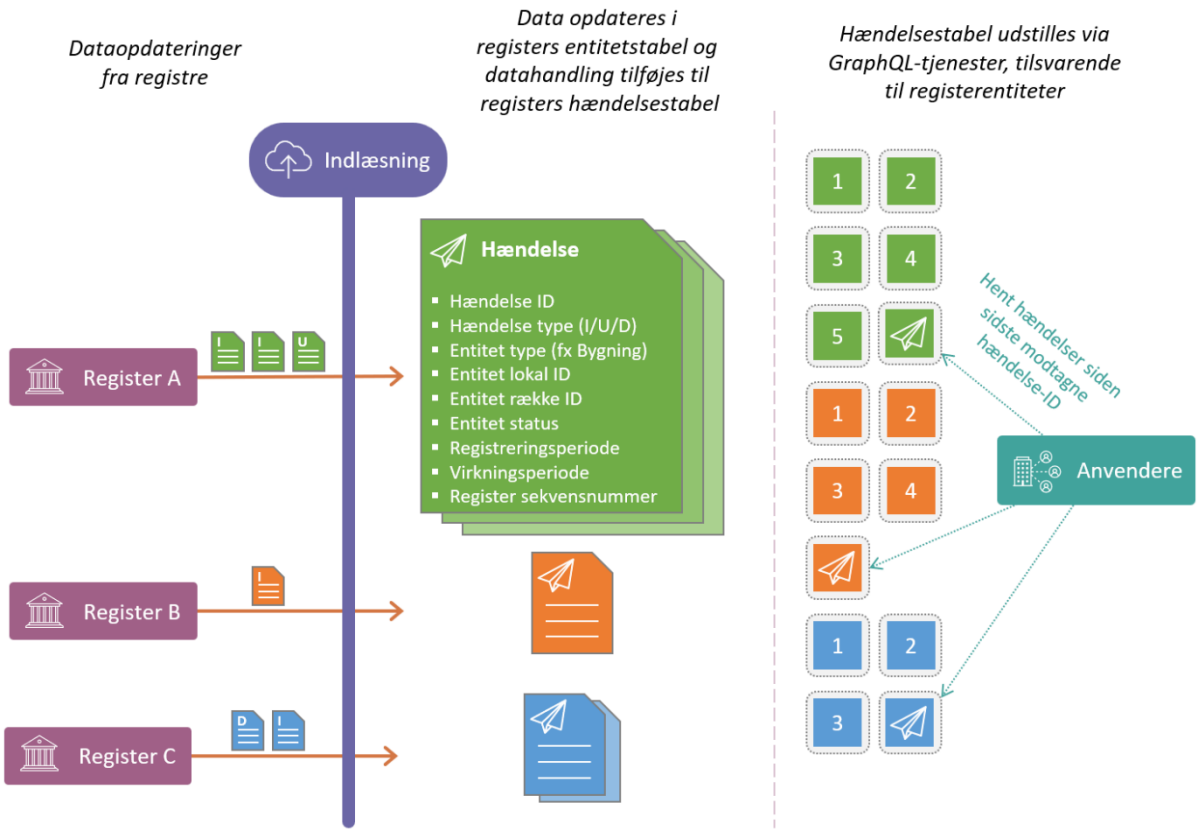
Læs mere om filtrering af hændelser baseret på kommunekoder i afsnit **Fejl! Henvisningskilde ikke fundet.** (CPR-felter).

Feltnavn	Betydning
object_kommunekodenuvaerende	Den nuværende kommunekode for et dataobjekt som har ændret sig i CPR.
object_kommunekodetidligere	Den tidligere kommunekode for et dataobjekt som har ændret sig i CPR.

Tabel 3: CPR-felter

2.2 Overordnet arkitektur

Figur 1 illustrerer den overordnede arkitektur for distribution af hændelser via Datafordeleren. Data oprettes og vedligeholdes i registre og kopieres løbende til Datafordeleren gennem replikeringskanaler. Herefter indlæses data i Datafordelerens databaser, hvor der genereres hændelser baseret på den indlæste data. Datafordeleren distribuerer både registerdata og hændelser til anvendere, som kan tilgå data gennem forskellige tjenester, eksempelvis de entitetsbaserede GraphQL-tjenester.



Figur 1: Overordnet arkitektur for hændelser

2.2.1 Grundlæggende GraphQL for hændelser

Hændelser udstilles via GraphQL-tjenester, som følger de samme principper som der er beskrevet i Transition mellem REST-services og GraphQL-services og Brugervejledning - Transitionsguide for entitetsbaserede GraphQL-tjenester. De vigtige koncepter for moderniserede hændelser er:

- **Skemastruktur -** Hændelser følger samme skemaprincipper som andre GraphQL-entiteter.
- **Filtrering -** Anvender samme filtreringsoperatorer (eq, gt, in, osv.) som beskrevet i Transition mellem REST-services og GraphQL-services.
- **Paging -** Hændelser følger standard GraphQL-pagingsmønstre.

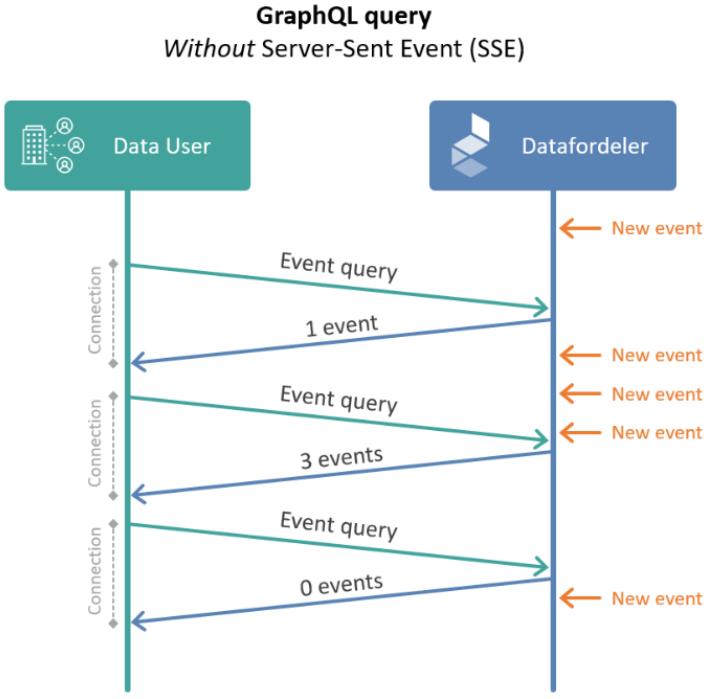
2.3 Brugsscenarier for hændelser

Der kan være forskellige scenarier for, hvordan en anvender ønsker at modtage hændelser. Har man ikke behov for at modtage opdateringer med det samme, kan man hente hændelser med queries mod GraphQL-tjenesterne, beskrevet i afsnit 2.3.1 (*Anvendelse af queries*). Har man derimod behov for at modtage hændelser i nær realtid, fordi man for eksempel skal igangsætte en proces baseret på ændringer i registerdata, kan man i stedet abonnere på hændelser, beskrevet i afsnit 2.3.2 (*Abonnement på hændelser*).

Autentifikation og autorisation for at kunne sende forespørgsler til GraphQL-tjenesterne er beskrevet i Brugervejledning - Transitionsguide for entitetsbaserede GraphQL-tjenester

2.3.1 Anvendelse af queries

Ved anvendelse af queries for at modtage hændelser, sendes en query afsted mod endepunktet for det ønskede registers GraphQL-tjeneste. Når anvenderen sender en query, får anvenderen hændelser tilbage, der indeholder metadata om ændringer i registerets data. Hvor mange hændelser der bliver returneret, og hvilke entiteter det omhandler kommer an på den sendte query. Queries returnerer alle hændelser for et register, inklusive historiske. EventID kan bruges for at filtrere på nye hændelser gennem at lave et "greater than" filter på den sidste kendte EventID. Filtrering beskrives yderligere i afsnit 4.2 (*Mulige filtreringer*). Dette flow er afbilledet i Figur 2.



Figur 2: Flow for modtagelse af hændelser via queries

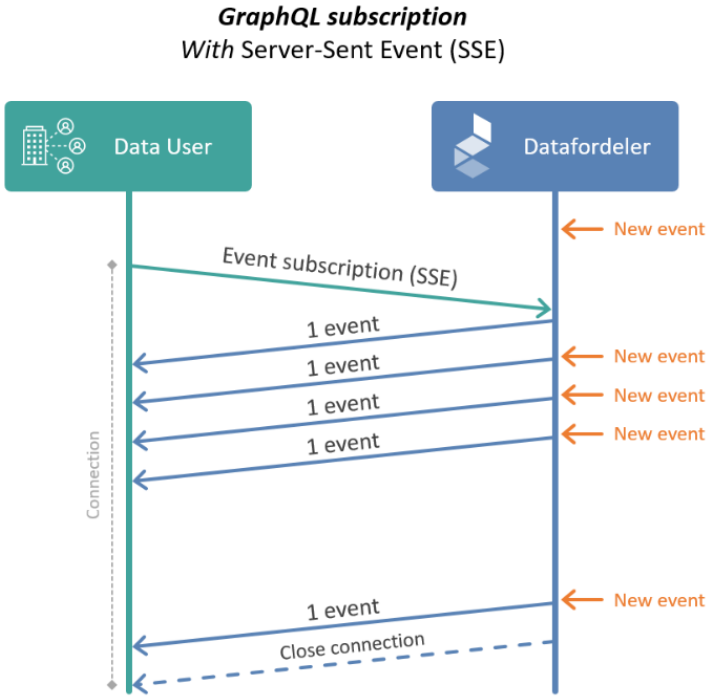
2.3.2 Abonnement på hændelser

GraphQL-abonnementer på hændelser benytter Server-Sent Events (SSE) teknologi, som muliggør en vedvarende forbindelse mellem server og klient (anvenders egne løsninger). Når en klient sender en GraphQL forespørgsel for at abonnere på hændelser, oprettes der til serveren, en åben forbindelse, som venter på nye data.

Hvis der er hændelser tilgængelige der opfylder de kriterier opsat i abonnementet, returneres de næsten øjeblikkeligt via SSE. Ellers holdes forbindelsen åben, indtil forbindelsen rammer tidsgrænsen på 10 minutter, eller

brugeren selv lukker den. Klienten kan derefter straks åbne en ny forbindelse og fortsætte med at vente på opdateringer, hvilket sikrer næsten realtids-opdateringer. Afsendelsesfrekvensen for hændelser vil som udgangspunkt ligge på 30 sekunder for alle registre, dvs. at der maksimum vil gå 30 sekunder, fra en hændelse er oprettet i Datafordelerens database, til den publiceres til anvendere. Dette flow er afbilledet i Figur 3.

Abonnementsforespørgsler ligner almindelige queries, men uden pagingsegenskaber, da kun én hændelse returneres ad gangen, når nye data detekteres. Abonnementer understøtter de samme filtreringsmuligheder som queries, hvilket giver anvender kontrol over hvilke hændelser, der skal modtages. Filtrering beskrives i afsnit 4.2 (*Mulige filtreringer*), og oprettelse af abonnement på hændelser beskrives i afsnit 4.4 (*Opret abonnement*).



Figur 3: Flow for modtagelse af hændelser via abonnement

2.3.3 Forretningsscenarier

For at illustrere de praktiske anvendelsesmuligheder af Datafordelerens hændelsessystem, præsenteres her en række konkrete forretningsscenarier fra forskellige brancher og sektorer. Disse eksempler viser, hvordan organisationer kan udnytte hændelser til at automatisere forretningsprocesser, sikre datakvalitet og optimere deres it-systemer.

Scenarierne spænder fra offentlige myndigheder, der skal sikre regeloverholdelse og effektiv sagsbehandling, til private virksomheder der bruger realtidsdata til at forbedre kundeservice og risikostyring. Hvert scenarie beskriver både det forretningsmæssige behov og den konkrete tekniske implementering ved hjælp af moderniserede hændelser.

Disse eksempler kan bruges som inspiration til at identificere lignende anvendelsesmuligheder i din egen organisation og som vejledning til implementering af hændelsesbaserede løsninger.

2.3.3.1 Kommune – Byggesagsbehandling

Scenarie: En kommune skal automatisk opdatere deres byggesagssystem, når nye bygninger registreres, eller eksisterende bygninger ændres i BBR.

Implementering:

- Abonner på BBR_Bygning hændelser med eventaction: ["i", "u"]
- Ved modtagelse af hændelse, hent bygningsdata via GraphQL ved hjælp af object_datafordelerRowId fra hændelsen
- Automatisk opret eller opdater byggesag i kommunens system
- F.eks. ved en bitemporal update af en Byggesag, skal både en ”Insert” og en ”Update” hændelse hentes, for at opdatere anvenderens byggesagsystem
- Hvis kommunen har behov for at holde sig opdateret på personflytninger:
    - Abonner på CPR_Hændelse med kommunekodefiltrering
    - Ved modtagelse af hændelse, hent den faktiske data via GraphQL ved hjælp af object_id for CPR Public Sector

2.3.3.2 Selskabsskat - Virksomhedsovervågning

Scenarie: SKAT skal overvåge virksomhedsændringer for korrekt selskabsbeskatning og momsregistrering.

Implementering:

- Abonner på CVR_Virksomhed hændelser for alle virksomhedsændringer
- Ved modtagelse af hændelse, hent den faktiske virksomhedsdata via GraphQL ved hjælp af object_datafordelerRowId fra hændelsen
- Automatisk opdater skatteregistre ved ændring af:
    - Virksomhedsform (personligt/aktieselskab)

- Aktivitetsområde (branchekoder)
- Adresse

2.3.3.3 It-serviceudbyder – Datasynkronisering

Scenarie: En it-leverandør skal holde flere kundesystemer synkroniserede med centrale registre.

Implementering:

- Indhent oprindeligt kopiregister via fildownload
- Brug RegisterImportStatus til at sikre dataintegritet, ved at tjekke status for registret, gennem RegisterImportStatus, inden indhentning af data
- Abonner på relevante hændelser fra modsvarende sekvensnummer i fildownload for at sikre kontinuerlig synkronisering efter den initiale etablering af kopiregisteret

3 Udstilling af hændelser

Hændelser udstilles via GraphQL-tjenesterne, som en entitet for hvert register. Dvs. man kun vil modtage hændelser for et enkelt register per kald, og der vil kun være hændelser for de entiteter man har adgang til. Adgangssystemet for adgang til entiteter og hvordan adgang tildeles er beskrevet i Brugervejledning - Transitionsguide for entitetsbaserede GraphQL-tjenester**Fejl! Henvisningskilde ikke fundet.**.

I disse eksempler bruges API key som autentifikationsmetode til at hente hændelser fra BBR:

- https://graphql.datafordeler.dk/BBR/v1?apiKey={noeglePlaceholder}

3.1 Hændelser per register

Følgende query i Figur 4 vil sende en forespørgsel til BBR om at få hændelser for den specifikke entitet ”Bygning”:

```
query {
  BBR_Events(
    where: {
        entityname: { eq: "Bygning" }
    }
  ) {
    nodes {
      eventid
      entityname
      eventaction
      datafordelerRegisterImportSequenceNumber
      datafordelerOpdateringstid
      fromfailedimport
      object id
      object_datafordelerRowId
      object_datafordelerRowVersion
      object_registreringfra
      object_registreringtil
      object status
      object_virkningfra
      object_virkningtil
    }
  }
}
```

Figur 4: BBR eksempel query

3.2 Abonnement på hændelser

Forespørgsler til at abonnere på hændelser er næsten ens med almindelige GraphQL queries, med den forskel at der kommer et ”subscription”-præfix på i stedet for ”query”. Følgende forespørgsel i Figur 5 til BBR abonnerer på hændelser for den specifikke entitet ”Bygning”:

```
subscription {
  BBR_Events(
    where: {
        entityname: { eq: "Bygning" }
    }
  )
     {
      eventid
      entityname
      eventaction
      datafordelerRegisterImportSequenceNumber
      datafordelerOpdateringstid
      fromfailedimport
      object_id
      object_datafordelerRowId
      object_datafordelerRowVersion
      object_registreringfra
      object_registreringtil
      object_status
      object_virkningfra
      object_virkningtil
    }
  }
```

Figur 5: BBR-hændelse abonnement query

3.3 Status på register import

Registre importerer data til Datafordeleren i pakker. Information om tilstanden og aktualiteten af denne dataimport bliver udstillet til anvendere gennem RegisterImportStatus entiteten. RegisterImportStatus er en åben entitet, som alle anvendere har adgang til.

RegisterImportStatus opdateres kun når en hel pakke af hændelser er færdigimporteret, mens individuelle hændelser løbende tilføjes til hændelsestabellen under importen. Dette sikrer, at anvendere kan:

- Modtage komplette datapakker: En pakke kan indeholde relaterede hændelser (f.eks. både oprettelse og sletning for samme række) som skal behandles samlet
- Undgå usammenhængende tilstande: Ved at sammenligne RegisterImportStatus sekvensnummer med hændelsernes sekvensnummer kan anvendere sikre, at de kun behandler komplette pakker
- Garantere dataintegritet: Kritiske forretningsscenarier hvor rækkefølgen og sammenhængen mellem hændelser er vigtig bliver håndteret korrekt

Anvendere bruger RegisterImportStatus til at sikre, at de er opdaterede med alle hændelser indenfor et givent register ved at sammenligne dets sekvensnummer med sekvensnummeret på de hændelser de processerer.

Query i Figur 6 vil hente data fra RegisterImportStatus:

```
query {
  DAF_RegisterImportStatus {
      lastSequenceNumber,
      lastEventId,
      lastUpdated
  }
}
```

Figur 6: Eksempel query for RegisterImportStatus

Forespørgslen sendes til URL'en som tilhører det registers entitetsbaserede GraphQL-tjeneste, som man vil have oplysninger fra, se for eksempel BBR i afsnit 3 (*Udstilling af hændelser*). RegisterImportStatus udstiller de 3 felter som vises i Tabel 4.

Feltnavn	Betydning
lastSequenceNumber	Sekvensnummeret på senest indlæste pakke for registeret.
lastEventId	ID'et på den seneste hændelse genereret for et givent register.
lastUpdated	Tidspunkt for seneste opdatering.

Tabel 4: RegisterImportStatus felter

3.3.1 Abonnering på RegisterImportStatus

Da RegisterImportStatus er eksponeret gennem GraphQL på samme måde som hændelser, er det muligt at abonnere på ændringer til "DAF_RegisterImportStatus".

Bemærk at Datafordeleren kun gemmer én række per replikeringskanal i databasen. Dette har den konsekvens, at hvis flere opdateringer forekommer inden for det interval der udsendes hændelser, vil kun de seneste data blive sendt til anvendere gennem abonnementer. Se Figur 7 for eksempel på forespørgsel til at oprette abonnement på RegisterImportStatus.

```
subscription {
  DAF_RegisterImportStatus {
      lastSequenceNumber,
      lastEventId,
      lastUpdated
  }
}
```

Figur 7: Eksempel på subsciption for RegisterImportStatus

3.3.2 Register import metadata i fildownloads

Sekvensnummeret fra RegisterImportStatus udstilles som metadata for fildownloads under GetAvailableFileDownloads, hvis en anvender ønsker at kombinere hændelser med fildownloads, til at vedligeholde egen data. Mere information om vedligeholdelse af kopiregister med fildownloads kan findes i Brugervejledning - Transitionsguide for fildownload samt etablering og vedligeholdelse af kopiregister - UL3.

3.4 Registre uden hændelser

Der er 3 registre, som er tilgængelige i GraphQL, men som ikke har nogen funktionalitet relateret til moderniserede hændelser. Disse registre er:

- Historiske Kortbladsinddelinger: Indeholder udelukkende historiske data
- DHM Højdekurver: Opdateres sjældent (ca. halvårligt)
- DHM Oprindelse: Opdateres sjældent (ca. halvårligt)

For registre med historiske data eller sjældne opdateringer er hændelsesfunktionalitet ikke relevant. Derfor kan hverken Events eller RegisterImportStatus tilgås i GraphQL-tjenesterne for disse registre.

4 Sådan anvender du hændelser

Nu hvor du har fået overblik over, hvad hændelser er og hvilke brugsscenarier de dækker, er næste skridt at forstå, hvordan du konkret implementerer og anvender hændelser i din løsning. Dette afsnit guider dig gennem de praktiske aspekter af at arbejde med hændelser på Datafordeleren. Derudover, som supplement til denne guide findes Brugervejledning - Vedligeholdelse af kopiregister med hændelser, der indeholder kodeeksempler på hvordan du vedligeholder dit eget kopiregister med hændelser.

Som tidligere beskrevet, findes der to primære tilgange til at anvende hændelser, afhængigt af dine specifikke behov:

Query-baseret tilgang - Velegnet når du ønsker at hente hændelser på forespørgsel, fx som del af en planlagt synkroniseringsproces eller når realtidsopdateringer ikke er kritiske.

Abonnement-baseret tilgang - Optimal når du har behov for næsten realtidsnotifikationer om dataændringer, fx til at udløse automatiske processer eller holde kritiske systemer opdateret.

Hvordan begge tilgange implementeres i en anvenders it-system eller via klient software, vises i Brugervejledning - Vedligeholdelse af kopiregister med hændelser. Her findes der forskellige kodeeksempler og en vejledning til hvordan det kan implementeres i et .NET-projekt med C#.

I de følgende afsnit gennemgår vi begge tilgange med konkrete eksempler på hvordan hændelser kan kaldes på Datafordeleren, og bedste praksis.

4.1 Autorisation for hændelser

Autorisation for hændelser sker på entitetsniveau (”entityname” i GraphQL), hvilket betyder at adgang kontrolleres baseret på de specifikke entiteter anvenderen forsøger at tilgå.

For eksempel er det muligt for en anvender med adgang til "Ejendomsadministrator"-entiteten i EJF at filtrere på og modtage hændelser for denne entitet med Query i Figur 8.

```
query {
 EJF_Events(where: { entityname: { eq: "Ejendomsadministrator" } }) {
   nodes {
     entityname
   }
 }
}
```

Figur 8: EJF-Ejendomsadministrator eksempel query

Dog er det ikke muligt for den samme anvender at modtage data for en anden entitet i EJF, for eksempel "Ejerskab" som det ses i Figur 9, hvis anvenderen ikke har adgang til specifikt den entitet.

```
query {
  EJF_Events(where: { entityname: { eq: "Ejerskab" } }) {
    nodes {
      entityname
    }
  }
}
```

Figur 9: EJF-Ejerskab eksempel query

For vejledning og dybdegående forklaring af autentifikation og autorisation til hændelser, henvises der til afsnittet "Autentifikation & autorisation for GraphQL-tjenester" i Brugervejledning - Transitionsguide for entitetsbaserede GraphQL-tjenester. Denne guide gennemgår detaljeret hvordan hver metode implementeres, herunder oprettelse af credentials, håndtering af Access Tokens, IP-begrænsninger for adgangsbegrænset data, samt praktiske eksempler på opsætning i Postman. Derudover beskrives det hvordan OAuth-flow fungerer med både shared secrets og certifikater, inklusiv token-validitet og sikkerhedsovervejelser.

4.1.1 Autorisation for CPR-hændelser

CPR udstilles med Custom tjenester og giver ikke adgang til sine entiteter som andre register (undtaget CprHændelseskode). Dette påvirker hvordan hændelser tilgås for CPR, hvor dette gælder for deres entiteter:

- AdministrativEnhed og AdministrativEnhedType er ikke beskyttet og hændelser for disse kan tilgås af alle anvendere
- CustomPrivateSectorPerson giver IKKE adgang til nogle hændelser
- CustomPublicSectorPerson giver adgang til hændelser for alle beskyttede entiteter
- CprHændelseskode hændelser kan tilgås ved godkendt dataadgang

CPR godkender ikke adgang til nogen anden entitet end dem ovenfor. Hvis du vil tilgå hændelser for beskyttede entiteter, skal du altså have CustomPublicSectorPerson dataadgang, eller CprHændelseskode adgang, hvis du kun er interesseret i denne entitet.

Bemærk: CustomPublicSectorPerson giver også adgang til at kunne se hændelser for CprHændelsekode, men ikke til at kunne lave generelle queries til CprHændelsekode entiteten.

4.2 Mulige filtreringer

Filtreringsmuligheder varierer for GraphQL-entiteterne. Hændelser er ens for alle registre, med undtagelse af CPR-registeret der indeholder ekstra felter, som tidligere nævnt i afsnit 2.1.1.2 (Specielt om CPR-felter).

4.2.1 Hændelser

GraphQL-tjenesterne understøtter filtreringsmuligheder på hændelser, både på hændelsesspecifikke felter og dataobjektrelaterede felter. Alle filtre kan kombineres ved hjælp af "and"-operatoren for at skabe avancerede forespørgsler. Specifikationerne kan findes i GraphQL-skemaet for hvert register der udstiller hændelser, under {registernavn}_EventsFilterInput. For eksempel for BBR vil det findes under BBR_EventsFilterInput.

4.2.1.1 Tilladte operatorer

Filtreringsoperatorerne følger samme principper som beskrevet i Transition mellem REST-services og GraphQL-services, i afsnittet ”Analyse filters”. Her findes der eksempler på mere komplekse filtreringsscenarier. Nedenstående Tabel 5 viser de specifikke operatorer tilgængelige for hændelser.

Feltnavn	Datatype	Understøttede operatorer	Begrænsninger
datafordelerOpdateringstid	DateTime	eq, gt, gte, lt, lte	-
datafordelerRegisterImportSequenceNumber	Int	eq, gt, gte, lt, lte, in	MinValue: 1, MaxListSize: 100
entityname	String	eq, in	MinLength: 1, MaxLength: 3999, MaxListSize: 100 Værdien er begrænset til de tilgængelige entiteter i registret.
eventaction	String	eq, in	Kun tilladt: "i", "u", "d", MaxListSize: 100
eventid	Long	eq, gt, gte, lt, lte, in	MinValue: 1, MaxListSize: 100
object_id	String	eq, in	MinLength: 1, MaxLength: 3999, MaxListSize: 100
object_status	String	eq, in	MinLength: 1, MaxLength: 3999, MaxListSize: 100

Tabel 5: Tilladte operatorer

Yderligere bemærkninger**:**

- Numeriske felter med MinValue-begrænsning skal være >= den angivne værdi

4.2.1.2 CPR-felter

CPR-registeret indeholder to yderligere felter i hændelserne, som kan bruges til filtrering på kommunetilhørsforhold. Disse vises i Tabel 6. Bemærk, at disse felter hentes fra CprHændelse-entiteten, og er altså ”null” for alle hændelser hvor ”entityname” ikke er ”Cprhaendelse”.

CprHændelse indeholder også de hændelser som CPR klassificerer som en hændelse. På grund af dette er det en anbefaling, at man altid filtrerer hændelser for denne entitet, hvis man vil tilgå beskyttet data (se afsnit 4.1.1 (*Autorisation for CPR-hændelser*)), eller har behov af at filtrere med kommunekode. Se afsnit 4.2.1.3.1 (*Eksempel på filtrering af kommunekode i CPR*) for eksempel. Det object_id der udstilles gennem hændelser kan derefter bruges som input i CprPublicSector-tjenesten for at hente den nye data. Se Brugervejledning - Transitionsguide for entitetsbaserede GraphQL-tjenester for mere information om CPRs custom tjenester.

Feltnavn	Datatype	Understøttede operatorer	Begrænsninger
object_kommunekodetidligere	String	eq, in	MaxLength: 3999 MaxListSize: 100
object_kommunekodenuvaerende	String	eq, in	MaxLength: 3999 MaxListSize: 100

Tabel 6: Tillade operatorer for CPR

4.2.1.3 Eksempel på filtrering af hændelser

4.2.1.3.1 Eksempel på filtrering af kommunekode i CPR

Eksemplet i Figur 10 viser filtrering på begge kommunekode-felter for hændelser på Personer i CPR-registeret.

```
where: {
entityname: { eq: "Cprhaendelse" }
object_kommunekodetidligere: { eq: "0461" }
object_kommunekodenuvaerende: { eq: "0101" }
}
```

Figur 10: Eksempel på filtrering med kommunekode

4.2.1.3.2 Eksempel på filtrering i BBR

I Figur 11 er et eksempel på en query med filtrering af hændelser mod BBR-registeret.

```
query {
  BBR_Events(
    where: {
      and: [
        { eventaction: { eq: "i" } }
        { datafordelerOpdateringstid: { gte: "2025-01-01T00:00:00Z" } }
        { entityname: { eq: "Bygning" } }
        { object_status: { eq: "Aktiv" } }
      ]
    }
    first: 100
  ) {
    nodes {
      eventid
      eventaction
      entityname
      object_id
      object_status
      datafordelerOpdateringstid
    }
    pageInfo {
      hasNextPage
      endCursor
    }
  }
}
```

Figur 11: Eksempel på filtrering af hændelser

4.2.1.4 Optimeringsstrategier

Effektiv filtrering er afgørende for både performance og ressourceforbrug når du arbejder med hændelser på Datafordeleren. Ved at anvende præcise filtreringsparametre kan du betydeligt reducere den datamængde der overføres og processeres, hvilket resulterer i:

- Reduceret netværkstrafik da kun relevante hændelser sendes over netværket
- Hurtigere responstider da mindre datasæt processeres hurtigere af både server og klient
- Lavere ressourceforbrug da der vil være mindre CPU- og hukommelsesbelastning i dit system

4.2.2 RegisterImportStatus

DAF_RegisterImportStatus entiteten har ingen filtreringsmuligheder, og tilgås uden filtrering som querien fra eksemplet i afsnit 3.3 (*Status på register import*) viser.

4.3 Send query

Ved afsendelse af queries fungerer hændelser på samme måde som de entitetsbaserede tjenester, derfor kan Brugervejledning - Transitionsguide for entitetsbaserede GraphQL-tjenester bruges som vejledning til hvordan queries sendes afsted.

4.4 Opret abonnement

Oprettelse af abonnementer til hændelser vil ligne almindelige queries på forespørgselsniveau. Kodeeksempler på at implementere forespørgslerne i en klient kan findes i Brugervejledning - Vedligeholdelse af kopiregister med hændelser.

4.4.1 Retursvar

Når en klient har oprettet forbindelse til at abonnere på hændelser for et register, vil dataene som en hændelse indeholder i retursvaret, ligner eksemplet i Figur 12 der kommer fra BBR.


```
{"data":
   {"BBR_Events":
      {
      "datafordelerOpdateringstid":"2025-06-24T13:46:02.424335Z",
      "datafordelerRegisterImportSequenceNumber":99999999,
      "entityname":"Bygning",
      "eventaction":"i",
      "eventid":1177475,
      "object_datafordelerRowId":"3b1040ef-ca52-4cb4-9788-909102a44155",
      "object_datafordelerRowVersion":1,
      "object_id":"49b2aafe-5fb4-42b5-b22d-7104a616804f",
      "object_registreringfra":"2025-06-24T13:46:02.424335Z",
      "object_registreringtil":"2025-06-24T13:46:02.424335Z",
      "object_status":"6",
      "object_virkningfra":"2025-06-24T13:46:02.424335Z",
      "object_virkningtil":"2025-06-24T13:46:02.424335Z"
      }
   }
}
```


Figur 12: Eksempel på svar fra BBR_Events

4.5 Specielle fejlmeddelelser

Hændelser og Register Import Status kan returnere unikke fejlmeddelelser i GraphQL, der ikke findes for andre entiteter. Se figur 13 for et eksempel.

```
{
  "errors": [
    {
      "message": "eventaction filtered value is not valid. It only allows the following values: i,
u, d",
      "locations": [
        {
          "line": 2,
          "column": 42
        }
      ],
      "path": [
        "MAT_Events"
      ],
      "extensions": {
        "traceId": "00-8d74c84ec80437d8a0ad3d716bc7b6da-e19e2140e0a1d526-01",
        "code": "DAF-GQL-0016"
      }
    }
  ],
  "data": {
    "MAT Events": null
  }
}
```


Figur 13: Eksempel på fejlbesked fra MAT_Events for ikke tilladt filtrering

Tabel nedenfor beskriver alle specielle fejlmeddelelser der kan returneres i GraphQL for Register Import Status og Hændelser, og hvordan disse kan håndteres:

Beskrivelse	Code	Løsning	Entitet
Ikke tilladt filtrering, f.eks. filtrering på eventaction med en værdi der ikke er i, u eller d. Se figur 13 for et eksempel	DAF-GQL-0016	Fejlmeddelelsen indeholder de værdier der er tilladt. Opdatere filter i din query til at være indenfor disse grænseværdier	Hændelser
Manglende adgang, eller ukendt navn for den filtrerede entitet	DAF-GQL-0016	Tjek filter for ”entityname” i din query. Sikre at du har adgang og at navn er for en rigtig entitet (case sensitive)	Hændelser
Ingen tilgængelig import status for registret	DAF-GQL-0023	Registret har ikke indlæst data efter udstilling af moderniserede hændelser. Prøv igen senere for at hente status	Register Import Status

Tabel 7: Specielle fejlmeddelelser

5 Transitionsguide

Denne transitionsguide beskriver forskelle mellem Datafordelerens nuværende hændelser og de moderniserede hændelser der udstilles i GraphQL-tjenesterne, samt hvordan du som anvender af de nuværende hændelser, kan komme i gang med at bruge de moderniserede hændelser der udstilles på GraphQL-tjenester.

Det skal bemærkes at moderniserede hændelser og de nuværende hændelser driftes parallelt i en periode, så anvendere af Datafordeleren har mulighed for at skifte fra REST-hændelser til GraphQL-hændelser.

5.1 Datafordelerens nuværende hændelser

Datafordelerens nuværende hændelsessystem giver anvendere mulighed for at blive informeret om ændringer i registerdata gennem to forskellige leveringsmetoder. Afhængigt af teknisk behov og systemarkitektur kan anvendere vælge mellem PUSH-hændelser, hvor Datafordeleren automatisk sender notifikationer, eller PULL-hændelser, hvor anvender selv henter opdateringer efter behov.

Læs om de nuværende hændelser på **Hændelser på Datafordeleren** og **Guide til hændelser på Datafordeleren**.

5.1.1 Ligheder med moderniserede hændelser

De nuværende hændelser kan relateres til moderne hændelser, da de har konceptuelle ligheder.

Queries kan sammenlignes med PULL-hændelser:

- Anvender laver et API kald
- Manuel forespørgsel
- Enkelt request-response
- Præcis specificering af ønskede data

Subscriptions kan sammenlignes med PUSH-hændelser:

- Datafordeleren sender kontinuerlige beskeder
- Beskeder sendes når data opdateres
- Leverer én hændelse ad gangen

5.1.2 Fordele ved at skifte til moderne hændelser

Transitionen fra Datafordelerens nuværende hændelsessystem til de moderniserede hændelser bringer flere betydelige fordele for anvendere. Den mest markante forbedring ligger i den forbedrede tekniske arkitektur, hvor moderne hændelser anvender moderne, industrielle standarder som Server-Sent Events (SSE) i stedet for proprietær PUSH-løsninger. Dette sikrer et ensartet API-design, hvor alle hændelser følger samme GraphQL-skemastruktur, hvilket reducerer kompleksiteten i integration og giver mere robuste fejlmeddelelser gennem standardiserede HTTP-statuskoder.

Derudover, skal anvendere i det nuværende system anmode om PUSH-hændelser gennem en ofte langvarig proces, hvilket kan forsinke adgang til kritiske dataopdateringer. De moderniserede hændelser eliminerer denne barriere ved at tilbyde adgang gennem GraphQL-grænsefladen, såfremt anvender har de nødvendige adgange.

Moderne hændelser tilbyder også betydeligt øget fleksibilitet og præcision sammenlignet med det nuværende system. Anvendere får nu mulighed for:

- Granulær filtrering på specifikke felter som eventaction, entityname og eventid
- Præcis specificering af hvilke felter der ønskes returneret i hændelser gennem GraphQL queries
- Kombinering af flere filtre ved hjælp af ”and”-operator til målrettede forespørgsler

Fra et ydeevne- og skalerbarhedsperspektiv medfører moderne hændelser reduceret netværkstrafik, da kun relevante data returneres baseret på query-specifikationen. Paging håndterer store datasæt mere effektivt, mens SSE-baserede abonnementer sikrer næsten øjeblikkelige notifikationer om dataændringer.

5.1.3 Hændelsesabonnementer og historiske data

I de nuværende hændelser kan et abonnement kun modtage hændelser, der sker efter oprettelsen af abonnementet. Det er ikke muligt at hente hændelser, som er indkommet til datafordeleren inden oprettelsen af abonnementet. I de moderniserede hændelser er det muligt at hente ældre hændelser ved at lave en GraphQL query med den startdato (datafordelerOpdateringstid) eller eventid, hvorfra hændelser ønskes modtaget.

5.2 Transition af filtrering

Filtreringsmulighederne i moderne hændelser adskiller sig markant fra de nuværende hændelser både i struktur og funktionalitet.

5.2.1 Oversigt over forskel mellem nuværende hændelsesfiltrering og GraphQL hændelsesfiltrering

I Tabel 8 ses en oversigt over forskellene mellem den nuværende hændelsesfiltrering og GraphQL hændelsesfiltrering.

Aspekt	Nuværende hændelsesfiltrering	GraphQL hændelsesfiltrering
Filterstruktur	• Betingelser består af: **Felt + Operator + Værdi** • Gruppering med logiske operatorer (OG/ELLER) • Opsætning via webportal med grafisk interface	• Defineret direkte i GraphQL query-syntaks • Kombinering kun med AND-operator • Programmatisk opsætning i kode
Tilgængelige felter	• Beskedtype, BeskedansvarligAktør, TværgåendeProces • ObjektID, ObjektType, Registreringsaktør • Registreringstid, Status, RegistreringsID • Objekthandling, OpgaveEmne • RelateretObjektID, RelateretObjekttype	• eventid, entityname, eventaction • datafordelerOpdateringstid, datafordelerRegisterImportSequenceNumber • object_id, object_datafordelerRowId, object_datafordelerRowVersion • object_registreringfra/til, object_virkningfra/til, object_status • CPR-specifikke: object_kommunekodeNuvaerende, object_kommunekodeTidligere
Operatorer	• = (Lig med), <> (Ikke lig med) • <, <=, >, >= (Sammenligning) • I, IKKE I (Liste-medlemskab) • INDEHOLDER (kun for Objekthandling)	• eq (equals), in (liste-medlemskab) • gt, gte, lt, lte (sammenligning) • Ingen negation eller "indeholder" operatorer

Tabel 8: Sammenligning mellem nuværende hændelsesfiltrering og GraphQL hændelsesfiltrering

5.2.2 Praktisk transition

Tabel 9 viser nuværende felter og deres GraphQL-ækvivalent, hvilket kan bruges til at oversætte hændelser, der hentes via REST, til queries der kan hentes moderne hændelser med.

Nuværende felter	GraphQL-ækvivalent
ObjektType	entityname
Objekthandling	eventaction
Registreringstid	object_registreringfra / object_registreringtil
ObjektID	object_id
Status	object_status

Tabel 9: GraphQL-ækvivalenter af nuværende REST-felter

Det skal bemærkes, at Objekthandling og eventaction ikke har fuldstændig samme betydning. I REST-hændelserne kunne registrene frit definere Objekthandling, som derfor ikke nødvendigvis stemte overens med de faktiske databaseoperationer. For eksempel kunne en forretningslogisk "opdatering" af en person godt resultere i en INSERT-operation i databasen, men stadig have Objekthandling sat til "Update".

I moderniserede hændelser bliver eventaction derimod defineret ud fra den faktiske databaseoperation (INSERT/UPDATE/DELETE) der er foretaget. Dette betyder, at alt efter hvordan et register vælger at opdatere data med indlæsningspakker, for eksempel, hvis registeret laver INSERT operationer i stedet for UPDATE operationer, kan det resultere i at anvendere er nødt til at lave yderligere kald til den pågældende entitet for at tjekke hvordan de potentielt flere rækker ser ud for et bestemt ObjektID.

5.2.2.1 Eksempler på filtrering

Følgende eksempler viser hvordan almindelige filtreringsscenarier fra de nuværende hændelser kan oversættes til GraphQL-syntaks. Eksemplerne er organiseret efter filtreringstype og viser både den nuværende betingelsesstruktur og den tilsvarende GraphQL-implementering.

Eksemplerne kan bruges som reference ved migrering fra nuværende til moderniserede hændelser, og tager udgangspunkt i BBR. For hver filtreringstype vises først betingelsen som den opsættes i det nuværende system, efterfulgt af den ækvivalente GraphQL query.

5.2.2.1.1 Entitetsbaseret-filtrering

Eksempel ses i Figur .

- Betingelse: ObjektType = ”Bygning”

```
where: { entityname: { eq: "Bygning" } }
```

Figur 14: Eksempel på filtrering i BBR på bygningsentitet

5.2.2.1.2 Handlingsbaseret-filtrering

Eksempel ses i Figur 15.

- Betingelse: Objekthandling = ”Create”

```
where: {  eventaction: { eq: "i" } }
```

Figur 15: Eksempel på filtrering for "Create" hændelser

5.2.2.1.3 Kombineret filtrering

Eksempel ses i Figur .

- Betingelse: ObjektType = ”Bygning”
- Betingelse: Objekthandling = ”Create,Update”

```
where: {
entityname: { eq: "Bygning" }
eventaction: { in: ["i", "u"] }
}
```

Figur 16: Eksempel på kombineret filtrering af Bygning og "Create" og "Insert"

5.3 Eksempel på transition til moderniserede hændelser

Afsnittet giver et eksempel på hvordan en eksisterende anvender på Datafordeleren kan overgå fra at bruge PUSH- og PULL-hændelser, til at bruge moderniserede hændelser. Eksemplerne her i transitionsguiden tager udgangspunkt i en anvender som ønsker hændelser for entiteten BBR_Bygning fra BBR-registeret.

5.3.1 PULL-hændelser

Her henter anvenderen hændelsesbeskeder ved at kalde Datafordeleren, svarende til abonnering på hændelser beskrevet i afsnit 2.3.2 (*Abonnement på hændelser).* Afsnittet påbegyndes med en situationsbeskrivelse, efterfulgt af konkrete trin der beskriver transitionen.

5.3.1.1 Situationsbeskrivelse

Anvenderen har været inde og foretage filtrering på de hændelser, som Datafordeleren returnerer (se afsnittet om filtrering på **Guide til hændelser på Datafordeleren**). Filtrering er sat op så der kun returneres hændelser for BBR_Bygning.

Anvenderen benytter følgende parametre:

- DateFrom = 2025-06-21
- DateTo = 2025-07-21
- Format = json
- Page = 1
- PageSize = 100

Anvender har allerede en tjenestebruger, som abonnementet er relateret til. Det er vigtigt at bemærke, at anvenderen kun kan modtage hændelser der er opstået efter tidspunktet hvor abonnementet blev oprettet – hændelser fra før dette tidspunkt er ikke tilgængelige. Anvender bruger sin tjenestebruger til at kalde REST-tjenesterne på URL'en:

- https://services.datafordeler.dk/system/EventMessages/1.0.0/custom?datefrom=2025-06-21&dateto=2025-07-21&format=json&page=1&pagesize=100

5.3.1.2 Transition

1. Anvender tilgår følgende tjeneste med sin API-nøgle for at hente GraphQL-skemaet for BBR:
    a. https://graphql.datafordeler.dk/BBR/schema/v1?apiKey={noeglePlaceholder}
2. Da anvender har dannet sig et overblik over GraphQL-skemaet og derfor ved hvilke queries der kan sendes for BBR, sender anvender nu query fra Figur 17 til URL'en nedenfor for at hente hændelser for BBR_Bygning med de ønskede oplysninger fra situationsbeskrivelsen:
    a. https://graphql.datafordeler.dk/BBR/v1?apiKey={noeglePlaceholder}

```
query {
  BBR_Events(
    first: 100
    where: {
      and: [
        { entityname: { eq: "Bygning" } },
        { datafordelerOpdateringstid: { gte: "2025-06-21T00:00:00Z" } },
        { datafordelerOpdateringstid: { lte: "2025-07-21T23:59:59Z" } }
      ]
    }
  ) {
    pageInfo { hasNextPage endCursor }
    nodes {
      eventid entityname eventaction datafordelerRegisterImportSequenceNumber
      datafordelerOpdateringstid fromfailedimport object id object datafordelerRowId
      object_datafordelerRowVersion object_registreringfra object_registreringtil
      object_status object_virkningfra object_virkningtil
    }
  }
}
```

Figur 17: Eksempel på query for transition af BBR-hændelse

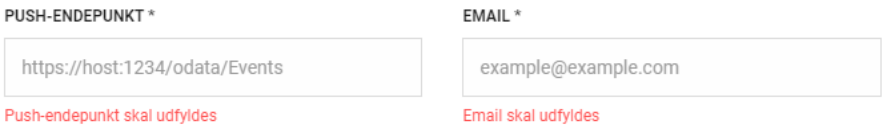
3. Anvender får nu et respons tilbage fra Datafordeleren der indeholder hændelser indenfor de specificerede kriterier i JSON-format som vist i Figur (resultatet er kortet ned til kun at indeholde et eksempel på en hændelse).


```
{
  "data": {
    "BBR_Events": {
      "pageInfo": {
        "hasNextPage": false,
        "endCursor": "TVE9PTs0L1lSQUFBQUFBQT0="
      },
      "nodes": [
        {
          "eventid": 1177315, "entityname": "Bygning", "eventaction": "i",
          "datafordelerRegisterImportSequenceNumber": 3453517,
          "datafordelerOpdateringstid": "2025-06-23T13:57:37.016852Z",
          "fromfailedimport": false,
          "object_id": "85209566-b4a4-4769-a016-13f4b7d4ce1d",
          "object_datafordelerRowId": "04816f75-71a4-4648-b978-5faf7fb0fb18",
          "object_datafordelerRowVersion": 1,
          "object_registreringfra": "2023-10-23T04:53:19.487369Z",
          "object_registreringtil": null,
          "object_status": "3",
          "object_virkningfra": "2023-02-28T07:54:30.700555Z",
          "object_virkningtil": "2023-10-23T04:53:19.487369Z"
        }
      ]
    }
  }
}
```


Figur 18: Eksempel på svar fra en query til BBR_Events

5.3.2 PUSH-hændelser

PUSH-hændelser kan ikke oversættes direkte til GraphQL, da de forudsætter al opsætningen til PULL-hændelser og endvidere konfiguration af et PUSH-endepunkt og en e-mail. Denne konfiguration skal foretages i Datafordeler administrationen, men eftersom PUSH hændelser udfases i takt med moderniserede hændelser gøres tilgængelig, findes den opsætning ikke længere.



Figur 19: Konfigurering af PUSH-endepunkt og e-mail

Eftersom abonnering på moderniserede hændelser foregår via SSE, er dette unødvendigt, og derfor findes muligheden for at konfigurere et PUSH-endepunkt som vist i Figur ikke i Datafordelerens nye brugerportal. Der kan derfor tages udgangspunkt i querien fra afsnit 5.3.1 (*PULL-hændelser*), sammen med vejledningen fra afsnit 3.2 (*Abonnement på hændelser*). Dette vil resultere i query i Figur , der kan bruges til at abonnere på hændelser via SSE.

```
subscription {
  BBR_Events(
    where: {
      and: [
        { entityname: { eq: "Bygning" } }
      ]
    }
  ) {
    eventid entityname eventaction datafordelerRegisterImportSequenceNumber
    datafordelerOpdateringstid fromfailedimport object_id object_datafordelerRowId
    object_datafordelerRowVersion object_registreringfra object_registreringtil
    object_status object_virkningfra object_virkningtil
  }
}
```

Figur 20: Eksempel på subscription query til BBR_Events

Filtreringen på datafordelerOpdateringstid er fjernet i dette eksempel, da det ikke vil give mening at filtrere på ved abonnering, eftersom hændelserne sendes til anvender løbende efter ny data er indlæst i Datafordeleren.