20-03-2020

Referencearkitektur for indlæsning af stamdata på den Nationale Service Platform (NSP)

SUNDHEDSDATA-
STYRELSEN

1. Introduktion

Mange systemer har brug for flere forskellige typer stamdata for at kunne anvendes fyldestgørende, og de forskellige typer stamdata kommer ofte fra næsten ligeså mange forskellige kilder, som der er typer af stamdata. Det kan derfor være fordelagtigt at samle forskellige typer stamdata et fælles sted, der har en meget høj tilgængelighed, kaldet stamdata repository[1], hvorfra de anvendende systemer kan hente alle de stamdata, som de måtte have brug for, på præcis det tidspunkt, de måtte have brug for. På denne måde kan et givet anvendende system nøjes med at kommunikere på én ensartet måde med ét sted for at få alle sine relevante stamdata på de tidspunkter, der er mest opportunt for anvendersystemet, hvilket både IT-arkitekturmæssigt og kontraktmæssigt er enklere og at foretrække fremfor at skulle kommunikere med mange forskellige kilder på mange forskellige måder og tidspunkter. Stamdata repositoriet sørger for at indhente stamdata i form af stamdataudtræk fra de forskellige kilder og gemme disse stamdata i et stamdataregister i en database, kaldet stamdatadatabase, hvilket samlet set benævnes indlæsning af stamdata. Dette foretages via stamdataindlæsere – en for hver type af stamdata. Fra databasen udstilles de indlæste stamdata via services til de anvendende systemer, der således kun skal hente de relevante stamdata fra dette ene sted.

Et eksempel på en sådan stamdata arkitektur findes på den Nationale Service Platform (NSP) for sundheds-IT i Danmark, hvor der indlæses f.eks. ydere, medicinpriser, CPR data, og mange andre typer stamdata til brug på sundhedsområdet. Da to af de helt fundamentale kvaliteter for NSP er meget høj grad af tilgængelighed og datanærhed (realiseret via distribuerede instanser), kan anvendersystemer på sundhedsområdet derfor med praktisk talt 100%'s sandsynlighed hente de stamdata, de måtte have behov for på NSP, når det passer dem bedst.

Baggrunden for udarbejdelsen af nærværende referencearkitektur er ønsket om en modernisering og konsolidering af indlæsningen af stamdata på NSP. Den primære målgruppe for referencearkitekturen er derfor de udviklere og arkitekter, der skal implementere den nye generation af stamdataindlæsere af stamdata til NSP. Referencearkitekturen skal i denne sammenhæng anvendes som ramme og vejledning, når løsningsarkitekturer for konkrete nye stamdataindlæsere af stamdata til NSP skal udarbejdes.

1.1 Vision

Arkitekturvisionen for referencearkitekturen for indlæsning af stamdata på NSP er:

> *Ensartet indlæsning af stamdata på NSP via et robust rammeværk, som er baseret på velafprøvede åbne standardiserede integrationsmønstre, og er nemt at supportere og vedligeholde*

[1] Den fulde begrebsliste relevant for referencearkitekturen er givet i afsnit 3.

I de umiddelbart følgende afsnit uddybes og kvalificeres visionen via de målsætninger, kvaliteter, og principper som visionen er bygget op omkring.

1.2 Målsætninger og gevinster

Visionen kan nedbrydes i følgende målsætninger med tilhørende gevinster:

Målsætning	**Gevinst**
Indlæsning af stamdata på NSP skal foregå på en ensartet måde for alle typer af stamdata	Det gør det nemmere for både nye og eksisterende leverandører at sætte sig ind i en stamdataindlæser. Det gør vedligehold og driftssupport af stamdataindlæserne nemmere
Indlæsning af stamdata på NSP skal være så robust som muligt	Omfanget af fejlsituationer for indlæsning af stamdata reduceres
Indlæsning af stamdata på NSP skal indlæse og gemme så meget som muligt, ideelt set alt, af et givet stamdataudtræk	Så store dele af stamdataudtrækket som muligt bliver tilgængeligt hurtigst muligt for de anvendende systemer. Fejl i en enkelt afgrænset del i stamdataudtrækket forhindrer ikke at alle de øvrige dele behandles og bliver tilgængelige for de anvendende systemer
Driftssupport af indlæsning af stamdata på NSP skal være så nem og hurtig som muligt	Der kan hurtigt rettes op på fejlsituationer for indlæsning af stamdata
Vedligeholdelse af indlæsning af stamdata på NSP skal være så nem og hurtig som muligt	Der kan hurtigt udvikles ændringsønsker og rettes op på fejlsituationer for indlæsning af stamdata

1.3 Arkitekturkvaliteter

De centrale arkitekturkvaliteter som referencearkitekturen bygger på er:

- Robusthed
- Åbenhed ved åbne standarder (interoperability)
- Nemt at supportere og vedligeholde (maintainability)
- Tilgængelighed

De tre førstnævnte er alle indbygget direkte i visionens ordlyd. Tilgængelighed er delvist afledt af robusthed af stamdataindlæserne, men er yderligere sikret af at indlæsningen foregår på NSP, hvor høj tilgængelighed også garanteres via distribuering og fail-over mønstre.

1.4 Arkitekturprincipper

Følgende overordnede arkitekturprincipper ligger til grund for udformningen af referencearkitekturen. Der er naturligvis dels stor overensstemmelse imellem principperne og de ovenstående formulerede vision, målsætninger, og arkitekturkvaliteter, og dels (i nogle tilfælde) sammenhæng med de overordnede arkitekturprincipper for sundhedsområdet [1]:

Id.	Princip	Evt. reference
1	Stamdataindlæserne til de forskellige typer af stamdata på NSP skal være så ensartede som muligt	
2	Stamdataindlæserne på NSP skal være tolerante overfor fejl i enkelte informationsobjekter således at de andre ikke-fejlbehæftede informationsobjekter i et stamdataudtræk indlæses alligevel	
3	Stamdataindlæserne på NSP skal garantere så høj datakvalitet af de indlæste data som muligt indenfor de af stamdatakilden givne rammer	
4	Stamdataindlæserne på NSP skal være nemme at driftssupportere	1, 2
5	Stamdataindlæserne på NSP skal være nemme at vedligeholde	1
6	Stamdataindlæserne på NSP skal baseres på velafprøvede integrationsmønstre, komponenter, og teknologier	F6 [1], T3 [1]
7	Stamdataindlæserne på NSP skal baseres på anerkendte åbne standardiserede best practices	F2 [1], T2 [1]
8	Stamdataindlæserne på NSP skal overholde husreglerne på NSP	
9	Ved stamdataindlæsning på NSP må komponenter udenfor stamdata repository ikke skrive data i stamdata repository	6

Principperne gennemgås enkeltvis i større detaljer i det følgende.

Princip 1	Stamdataindlæserne til de forskellige typer af stamdata på NSP skal være så ensartede som muligt
Rationale	Jo mere ensartede stamdataindlæserne af de forskellige typer af stamdata er, jo nemmere er det: › at identificere og udtrække generel fælles funktionalitet på tværs af stamdataindlæserne, hvilket øger både vedligeholdelsesvenligheden og supporteringsvenligheden › at udvikle og/eller sætte sig ind i nye stamdataindlæsere, da variationerne på tværs af stamdataindlæserne minimeres
Implikationer	Alle stamdataindlæserne skal følge det samme overordnede mønster. Retningslinjer for strukturen af kildekoden givet af det overordnede mønster skal overholdes.

	Hvis det opdages at noget funktionalitet i en specifik stamdataindlæser for en given type er af generel karakter og vil være relevant for andre stamdata-indlæsere, skal kildekoden (og dokumentationen) omskrives, så den generelle funktionalitet er fælles på tværs af stamdataindlæserne
Evt. Reference	

Princip 2	Stamdataindlæserne på NSP skal være tolerante overfor fejl i enkelte informationsobjekter således at de andre ikke-fejlbehæftede informationsobjekter i et stamdataudtræk indlæses alligevel
Rationale	Jo flere dele af et stamdataudtræk, der indlæses, jo hurtigere bliver disse indlæste dele tilgængelige for de anvendende systemer
Implikationer	Indlæsningen af stamdata skal baseres på håndtering af små logiske delmængder af stamdataudtrækket ad gangen. Stamdataindlæserne skal holde styr på hvilke dele af et stamdataudtræk en indlæsning er fejlet for.
Evt. Reference	

Princip 3	Stamdataindlæserne på NSP skal garantere så høj datakvalitet af de indlæste data som muligt indenfor de af stamdatakilden givne rammer
Rationale	Datakvaliteten af indlæste stamdata på NSP skal være ligeså høj som hos stamdatakilden selv, ellers vil anvendersystemerne hente stamdata direkte hos stamdatakilden i stedet for hos NSP
Implikationer	Stamdataindlæsningen skal basere sig på en detaljeret retvisende datamodel (inklusive atomare datatyper, feltlængder etc.) leveret fra stamdatakilden. Stamdataindlæsningen skal validere de modtagne data i forhold til denne datamodel fra stamdatakilden. Stamdataindlæsningen må ikke introducere datavalideringer, der ikke er belæg for i datamodellen fra stamdatakilden
Evt. Reference	

Princip 4	Stamdataindlæserne på NSP skal være nemme at driftssupportere
Rationale	Jo nemmere indlæsningen af stamdata er at driftssupportere, jo hurtigere vil man kunne rette op på fejlsituationer
Implikationer	Stamdataindlæsernes logninger skal være så sigende og præcise som muligt. Stamdataindlæsernes status skal være så sigende og præcis som muligt. Stamdataindlæsernes logninger og status skal være så ens som mulig for alle de forskellige stamdataindlæsere. Stamdataindlæsernes logninger og status skal basere sig på den allerede eksisterende funktionalitet for disse på NSP
Evt. Reference	1, 2

Princip 5	Stamdataindlæserne på NSP skal være nemme at vedligeholde
Rationale	Jo nemmere indlæsningen af stamdata er at vedligeholde, jo hurtigere vil man kunne implementere ændringsønsker og rette fejl i stamdataindlæserne
Implikationer	Kildekoden for stamdataindlæserne skal være let forståelig, velkommenteret, ikke basere sig på kryptiske konstruktioner, og ikke være i uoverensstemmelse med de retningslinjer for kildekoden, som det overordnede mønster fastsætter. Dokumentationen af stamdataindlæserne skal være retvisende og let læselig
Evt. Reference	1

Princip 6	Stamdataindlæserne på NSP skal baseres på velafprøvede integrationsmønstre, komponenter, og teknologier
Rationale	Velafprøvede mønstre, komponenter og teknologier giver større sikkerhed, driftsstabilitet og kodekvalitet samt styrker leverandøruafhængigheden
Implikationer	Indlæsningen af stamdata skal basere sig på velkendte Enterprise Integration Patterns [2] (EIP). Kun bredt anvendte og driftsmodnede komponenter og teknologier må anvendes ved indlæsningen af stamdata. Ny-egenudviklede komponenter må således ikke anvendes ved indlæsningen af stamdata. Bredt tilgængelige ikke driftsmodnede (bleeding-edge) komponenter må således heller ikke anvendes ved indlæsningen af stamdata
Evt. Reference	F6 [1], T3 [1]

Princip 7	Stamdataindlæserne på NSP skal baseres på anerkendte åbne standardiserede best practices
Rationale	Anvendelsen af anerkendte åbne standardiserede best practices styrker leverandøruafhængigheden og giver større interoperabilitet
Implikationer	Proprietære kildekodebiblioteker må ikke anvendes. Ikke standardiserede kildekoderammeværk må ikke anvendes
Evt. Reference	T2 [1]

Princip 8	Stamdataindlæserne på NSP skal overholde husreglerne på NSP
Rationale	Da stamdataindlæserne på NSP netop skal afvikles på NSP, og NSP har nogle regler og retningslinjer, som alle programmer der afvikles på NSP skal overholde, i form af de såkaldte husregler, skal disse NSP husregler naturligvis også overholdes af stamdataindlæserne
Implikationer	Stamdataindlæserne skal overholde NSP husreglerne.

	I forbindelse med udvikling af stamdataindlæsere skal den givne leverandør sætte sig ind i NSP husreglerne, så disse kan blive overholdt. I forbindelse med QA af udviklede stamdataindlæsere skal det kontrolleres at stamdataindlæserne rent faktisk overholder NSP husreglerne
Evt. Reference	

Princip 9	Ved stamdataindlæsning på NSP må komponenter udenfor stamdata repository ikke skrive data i stamdata repository
Rationale	Af afkoblingshensyn og sikkerhedshensyn må komponenter udenfor stamdata repository aldrig skrive data direkte i stamdata repository
Implikationer	Afhængigheder til eksterne systemer (set fra stamdata repository) reduceres. De sikkerhedsregler der gælder generelt på NSP skal også overholdes ved indlæsning af stamdata. Det er altid en komponent i stamdata repository, der inde fra stamdata repository henter et stamdataudtræk. Stamdataudtræk skrives aldrig direkte til stamdata repository
Evt. Reference	6

1.5 Scope

Fokus i referencearkitekturen præsenteret i dette dokument er på selve indlæsningen af stamdata, og dermed på arkitekturen for hvordan en stamdataindlæser indlæser stamdataudtrækket fra stamdatakilden til stamdataregisteret i stamdatadatabasen. Arkitekturen for udstillingen af de indlæste stamdata til anvendersystemerne er således ikke i scope i dette dokument og ligeledes for genereringen af stamdataudtrækket. Disse to er dog medtaget på illustrationerne og i diskussionen i teksten, når det er relevant, dels for fuldstændighedens skyld og dels fordi de som kilde og anvender sætter nogle krav til indlæsningen af stamdata via deres snitflader til stamdata repositoriet.

Referencearkitekturen for indlæsning af stamdata baserer sig på en internationalt anderkendt best practice inden for området. Centrale dele af arkitekturen baserer sig på et robust velafprøvet generelt EIP, der anvendes mange steder i verden i mange forskellige IT-domæner. Selvom referencearkitekturen er udarbejdet med henblik på indlæsning af stamdata på NSP, er der ikke noget sundhedsdomæne specifikt over den og få restriktioner, der kommer fra NSP (hvor dette er tilfældet er det tydeligt indikeret). Referencearkitekturen kan således uden de store ændringer anvendes som ramme og vejledning, når løsningsarkitekturer for indlæsning af stamdata til andre konkrete stamdata repositorier skal udarbejdes uafhængigt af IT-domæne.

Alle illustrationer af referencearkitekturen i dette dokument anvender Archimate standarden [3] fra The Open Group.

2. Forretningsperspektiv

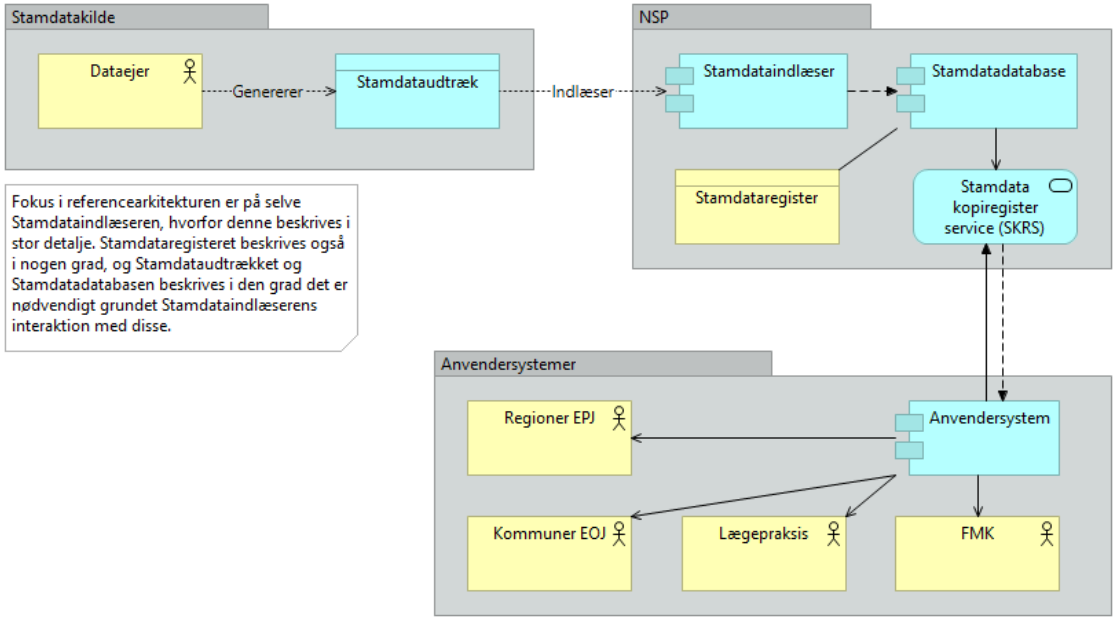
2.1 Principper

Følgende detaljerede forretningsarkitekturprincipper, afledt af de overordnede principper i afsnit 1.4, er anvendt:

Id.	Princip	Reference
F1	Forretningsarkitekturen for indlæsning af stamdata på NSP skal basere sig på indlæsning af logisk velafgrænsede delmængder af stamdataudtrækket	2
F2	Forretningsarkitekturen for indlæsning af stamdata på NSP skal anvende en detaljeret retvisende datamodel leveret af stamdatakilden.	3
F3	Forretningsarkitekturen for indlæsning af stamdata på NSP skal anvende velafgrænsede konceptuelle komponenter med hver deres klare ansvar og veldefinerede overordnede opgave	4, 5
F4	Forretningsarkitekturen for indlæsning af stamdata på NSP skal baseres på velafprøvede komponenter, der har gode referencer	6
F5	Forretningsarkitekturen for indlæsning af stamdata på NSP skal anvende åbne standardiserede komponenter, der baserer sig på anerkendte best practices	7
F6	Forretningsarkitekturen for indlæsning af stamdata på NSP skal overholde de afkoblings- og sikkerheds-principper og -regler, der gælder generelt for NSP	9

2.2 Forretningsarkitektur

Forretningsarkitekturen for indlæsning af stamdata kan illustreres som i følgende figur:



Figur 1: Forretningsarkitektur

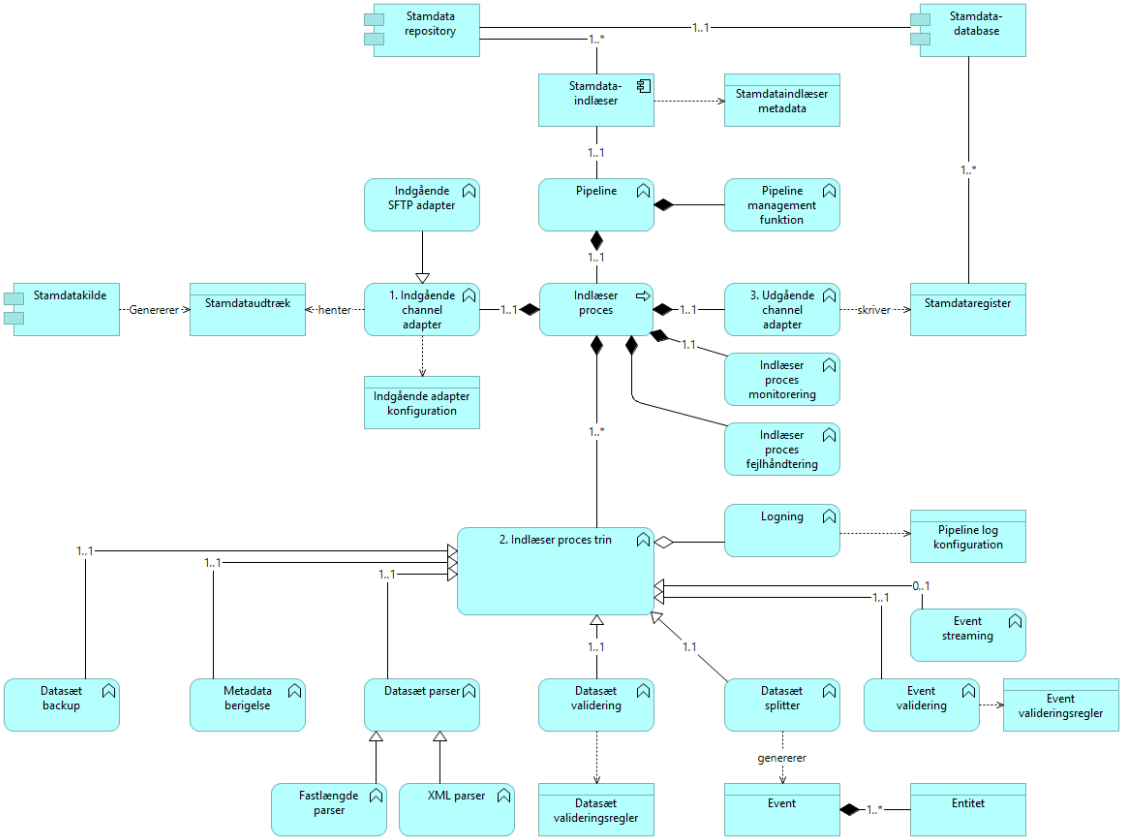
Et stamdataudtræk genereres på foranledning af dataejeren hos en stamdatakilde. Dette stamdataudtræk indlæses af en stamdataindlæser til et stamdataregister i NSP'ens stamdatadatabase. De indlæste stamdata udstilles herefter af stamdata kopiregisterservicen (SKRS) til en bred vifte af anvendersystemer fra de danske regioner (elektronisk patient journal (EPJ) systemer), kommuner (elektronisk omsorgs journal (EOJ) systemer), praktiserende læger, samt det fælles medicinkort (FMK), der dermed kan drage nytte af dem. Eksempler på stamdatatyper er ydere, medicinpriser, og CPR data.

Fokus i dette dokument er jf. afsnit 1 på arkitekturen for selve indlæsningen af stamdata, og dermed på stamdataindlæser applikationskomponenten, og de komponenter og objekter som denne interagerer direkte med (stamdataudtrækket fra stamdatakilden og stamdataregisteret i stamdatadatabasen), hvorimod arkitekturen for genereringen af stamdataudtrækket og udstillingen af de indlæste stamdata til anvendersystemerne (via SKRS) ikke er i scope, og derfor kun omtales når det er relevant, pga. de krav de stiller til forskellige dele af selve indlæsningen af stamdata.

I næste afsnit (3) introduceres alle de centrale anvendte begreber nærmere, så disse er klart definerede, inden vi vender os mod de mere tekniske arkitekturperspektiver, hvor begreberne omtales og anvendes kontinuerligt.

3. Begreber

I dette afsnit introduceres og forklares de forskellige begreber summarisk i forbindelse med indlæsning af stamdata samt disse begrebers relationer til hinanden. Først via følgende grafiske illustration og derefter i tabulær tekstform i alfabetisk rækkefølge:



Figur 2: Begrebsmodel

Begreb	Definition
Datasæt	Den logiske samling af stamdata, som skal indlæses, og som er tilstede i det modtagne stamdataudtræk
Datasæt backup	Konkret indlæser proces trin, der foretager backup af det modtagne stamdataudtræk indeholdende datasættet, der skal indlæses
Datasæt parser	Konkret indlæser proces trin, som parser det modtagne datasæt efter de givne specifikationer for den pågældende type af stamdataudtræk. Kan f.eks. være en XML parser, hvis datasættet er givet i XML

Begreb	Definition
Datasæt splitter	Konkret indlæser proces trin, der splitter datasættet op i events, der er den væsentligt mindre logiske enhed som de efterfølgende indlæser proces trin opererer på. Et datasæt indeholder et til mange events af den givne type af stamdata
Datasæt validering	Konkret indlæser proces trin, som validerer det modtagne datasæt efter specificerede datasæt valideringsregler
Datasæt valideringsregler	De valideringsregler som datasæt validering anvender
Entitet	De mindste (atomare) enheder som en given type af stamdata er opbygget af.
Event	Den mindre logiske enhed end datasættet af den givne type af stamdata, som sidste halvdel af indlæser proces trinnene opererer på. Events er bygget op af entiteter i overensstemmelse med stamdatakildens datamodel
Event streaming	Konkret indlæser proces trin, som giver mulighed for at streame events via en streaming platform til andre aftagere af stamdata end stamdataregisteret. Dette konkrete indlæser proces trin er ikke påkrævet i modsætning til alle de øvrige konkrete indlæser proces trin
Event validering	Konkret indlæser proces trin, der validerer et givet event efter specificerede event valideringsregler
Event valideringsregler	De valideringsregler som event validering anvender
Indlæser proces	Styrer trinene i proceskæden, der tilsammen udgør indlæsningen af stamdata af den givne type. Der startes altid med at hente stamdataudtrækket og afsluttes altid med at gemme de modtagne data i stamdataregistret. Derimellem er der en fast række af indlæser proces trin, der afvikles.
Indlæser proces fejlhåndtering	Fejlhåndtering i forbindelse med indlæsningen af stamdata
Indlæser proces monitorering	Monitorering af indlæsningen af stamdata
2. Indlæser proces trin	En abstraktion for de enkelte proces trin i indlæser processen. Specialiseres i de forskellige angivne konkrete trin
Indgående adapter konfiguration	Konfiguration af indgående channel adapter, der fortæller hvor og hvor ofte et stamdataudtræk skal hentes
1. Indgående channel adapter	Henter som første trin af indlæser proceskæden et stamdataudtræk i overensstemmelse med sin konfiguration. Kan f.eks. være en SFTP adapter
Logning	Anvendes af de enkelte indlæser procestrin. Anvender standard komponenter til logning på NSP
Metadata berigelse	Konkrete indlæser proces trin, der beriger henholdsvis datasæt og events med relevante metadata

Begreb	Definition
Pipeline	Kernen i en stamdataindlæser, der orkestrerer indlæsningen af en given type af stamdata. Der er en pipeline per stamdataindlæser (og dermed per type af stamdata). Kan styres af en pipeline management funktion. Orkestreringen styres i den tilhørende indlæser proces
Pipeline log konfiguration	Konfiguration af logning for den givne pipeline
Pipeline management funktion	Giver mulighed for at styre parametre for en given pipeline
Stamdatadatabase	Den database hvor de indlæste stamdata gemmes i stamdataregistre. Der er et stamdataregister per type af stamdata, og stamdatadatabasen består derfor af en til mange stamdataregistre
Stamdataindlæser	Den overordnede softwarekomponent, der effektuerer indlæsningen af en given type af stamdata via sin indbyggede pipeline. Der er en stamdataindlæser per type af stamdata
Stamdataindlæser metadata	Metadata om en stamdataindlæser, som idriftsættelsesdato, sikkerhedsklassifikation, typen af stamdata der indlæses etc.
Stamdatakilde	Det kildesystem hvorfra stamdata af en given type kommer
Stamdataregister	Det register i stamdatadatabasen, hvori stamdata af den givne type gemmes. Der er et stamdataregister per type af stamdata
Stamdata repository	Samlet betegnelse for den komponent, som både indeholder de stamdataindlæsere (software), der indlæser stamdataene, samt en stamdatadatabase, der indeholder de indlæste stamdata (data)
Stamdataudtræk	Udtræk af stamdata af en given type som stamdatakilden genererer. Typisk en fil, eventuelt komprimeret, der indeholder et datasæt af den givne type af stamdata samt eventuelle kontrol-metadata om datasættet. Kan enten indeholde samtlige stamdata af den givne type (fuldt udtræk) eller kun den delmængde af stamdata af den givne type, der er ændret siden sidste stamdataudtræk af samme type (delta udtræk)
3. Udgående channel adapter	Gemmer som afslutning af indlæser proceskæden de modtagne stamdata i det rette stamdataregister i stamdatadatabasen

Begreberne, deres relation til hinanden, samt deres anvendelse, uddybes i de resterende nedenstående afsnit af dokumentet.

4. Applikationsperspektiv

Vi zoomer nu ind på stamdataindlæser applikationskomponenten fra Figur 1, der indlæser stamdataudtrækket i stamdatadatabasen.

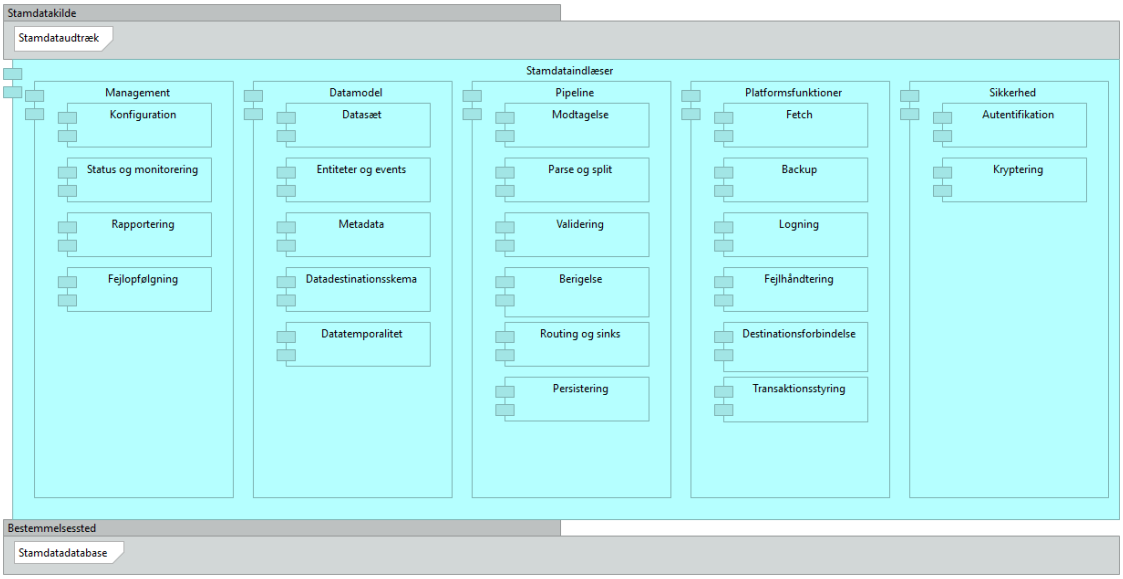
4.1 Principper

Følgende detaljerede applikationsarkitekturprincipper, afledt af de overordnede principper i afsnit 1.4, er anvendt:

Id.	Princip	Reference
A1	Alle stamdataindlæsere på NSP skal have den samme løsningsarkitektur baseret på det samme EIP	1, 6
A2	Stamdataindlæsning på NSP skal basere sig på genbrug af eksisterende NSP komponenter i den udstrækning det er muligt (f.eks. til logning)	4
A3	Applikationsarkitekturen for indlæsning af stamdata på NSP skal basere sig på små logiske komponenter, der i rækkefølge hver især udfører deres afgrænsede opgave	5, F3, A1
A4	Applikationsarkitekturen for indlæsning af stamdata på NSP skal basere sig på velafprøvede driftsmodnede komponenter med gode referenceimplementeringer	6, F4
A5	Applikationsarkitekturen for indlæsning af stamdata på NSP skal basere sig på open source komponenter, der implementerer best practices	7, F5
A6	Applikationsarkitekturen for indlæsning af stamdata på NSP skal overholde husreglerne på NSP	8

4.2 Overordnet applikationsarkitektur

Applikationsarkitekturen for stamdataindlæseren kan inddeles i fem overordnede hovedkomponenter, der hver især kan inddeles i funktionelle underkomponenter som illustreret i følgende figur:



Figur 3: Overordnet applikationsarkitektur for stamdataindlæser

Jf. begrebsmodellen i afsnit 2 er kernen af applikationsarkitekturen Pipelinen, som står for processeringen af indlæsningen af stamdata fra stamdataudtrækket er tilgængeligt ved stamdatakilden til dataene er endelig gemt i stamdatadatabasen på bestemmelsesstedet. Pipelinen foldes yderligere ud og beskrives i større detalje i nedenstående afsnit 4.3. Meget kort fortalt modtager Pipelinen stamdataudtrækket fra stamdatakilden, parser udtrækket og splitter det op i mindre dele, validerer, og beriger med metadata. Dernæst sørger pipelinen for at sende stamdataene til eventuelt andre interesserede modtagere, og endelig persisteres stamdataene i stamdatadatabasen.
Pipelinen opererer på begreberne inden for Datamodel komponenten dels via funktioner hos sig selv og dels via funktioner i Platformsfunktioner komponenten. Sikkerhedskomponenten varetager sikkerhedsaspekter og Management komponenten indeholder styringsværktøjer.
I det følgende beskrives de fem hovedkomponenter og disses underkomponenter en efter en.

4.2.1 Pipeline

Pipelinen er kernen i applikationsarkitekturen, hvor processen, der styrer indlæsningen af stamdata ligger. Den baserer sig på Pipes and Filter EIP'et [2], hvor de enkelte komponenter (filter) med hver deres veldefinerede og afgrænsede opgave er forbundne via pipelinen (pipe) og udfører deres opgave på outputtet fra den foregående komponent og sender deres eget output videre som input til den næste komponent i pipelinen. Alle filtre overholder det samme interface, f.eks. med hensyn til input og output, så det er nemt at indsætte et nyt filter i pipelinen, hvis der viser sig behov for det.
Der er én instans af pipeline for hvert stamdataregister, men arkitekturen for alle pipelines er den samme, nemlig de i dette afsnit følgende komponenter.

4.2.1.1 Modtagelse

Henter stamdataudtræk og læser datasættet ind fra fil til memory.

4.2.1.2 Parse og split

Opsplitter et datasæt i events, der er opbygget af entiteter med givne veldefinerede relationer imellem sig.

4.2.1.3 Validering

Foretager validering. Dette kan foregå på flere niveauer: Datasæt, event, entitet, og sågar på tværs af forskellige stamdataregistre om ønsket.

4.2.1.4 Berigelse

Beriger et event med relevant metadata, herunder information der muliggør præcis identifikation af hvor og hvornår et event kom fra (hvilket stamdataudtræk fra hvilken stamdatakilde hvornår).

4.2.1.5 Routing og sinks

Giver mulighed for at et event, der måtte være interessant for andre end stamdatadatabasen, kan sendes til andre interessenter.

4.2.1.6 Persistering

Gemmer events og information om modtagne datasæt i stamdatadatabasen.

4.2.2 Datamodel

Datamodel komponenten indeholder datadefinitionerne for alle de centrale begreber som Pipelinen opererer på.

4.2.2.1 Datasæt

Repræsenterer de stamdata som er tilstede i et stamdataudtræk, som kan være enten et fuldt udtræk, dvs. samtlige stamdata af en given type fra den pågældende stamdatakilde, eller et delta-udtræk, der kun indeholder de stamdata af den givne type, der har ændret sig siden sidste stamdataudtræk af den givne type. I tilfælde med delta-udtræk er det allerførste stamdataudtræk, det såkaldte etableringsudtræk, som hentes, dog nødvendigvis lig med et fuldt udtræk.

4.2.2.2 Entiteter og events

Et datasæt for et stamdataudtræk af typen X indeholder en til mange X events – f.eks. vil et stamdataudtræk for ydere indeholde yder events. Hver enkelt event er videre bygget op af entiteter, som er relateret til hinanden, hvilket udtrykkes i et skema – i eksemplet med ydere af de to entiteter yder og yderperson. Dette skema, der er i nøje overensstemmelse med stamdatadefinitionen hos stamdatakilden er helt centralt både i forhold til kvalitetssikring af de indlæste stamdata samt på sigt for anvendersystemerne, når de får mulighed for at hente de indlæste stamdata events ad gangen.

4.2.2.3 Metadata

Repræsenterer data om data. Disse kan være registreret på stamdataregister niveau, datasæt niveau eller entitet/event niveau. På entitet/event niveau ikke mindst data, der muliggør nøjagtig traceability fra en entitet/event til et stamdataudtræk. På datasæt niveau primært data, der fortæller tidspunkt og status om indlæsningen af et datasæt. På stamdataregister niveau primært status om registeret.

4.2.2.4 Datadestinationsskema

Repræsenterer i stamdataindlæseren det logiske databaseskema for stamdataregisteret hvor stamdata af en given type persisteres i stamdatadatabasen.

4.2.2.5 Datatemporalitet

Definerer versionering og tidsmæssig validitet af de indlæste stamdata. Et givet event/entitet kan opdateres flere gange (og eventuelt slettes) efter dets oprettelse, hvilket definerer de enkelte versioner af eventet/entiteten og deres rækkefølge, samt de tidsrum den givne version er gældende i – fra det tidspunkt versionen blev skabt til det tidspunkt næste version blev skabt. Dette er såkaldt teknisk historik, og det er vigtigt ikke at sammenblande denne med forretningsmæssig historik. Sidstnævnte er f.eks. i tilfældet ydere beskrevet via attributterne tilgangs- og afgangsdato, og det kan sagtens ske, at en yder opdateres med en tilgangs- eller afgangsdato i fremtiden, for at signalere til anvendere på forhånd at den pågældende yder (f.eks. en lægepraksis) åbner henholdsvis lukker på det givne fremtidige tidspunkt, så anvenderen kan agere bedst muligt derefter.

4.2.3 Platformsfunktioner

Platformsfunktioner komponenten indeholder de funktioner, som Pipelinen anvender, der er fælles for alle pipelines og i nogle tilfælde fælles med andre programmer/processer, der måtte blive afviklet på NSP platformen, som stamdataindlæseren afvikles på.

4.2.3.1 Fetch

Står for at hente stamdataudtræk. Dette kan foregå efter to forskellige mønstre (jf. afsnit 6), afhængig af stamdatakilden. Fetch sørger for, at alle pipelines med stamdataudtræk, der skal hentes efter samme mønster, får hentet stamdataudtrækket på samme måde. Fetch anvender Sikkerhedskomponenten Autentifikation.

4.2.3.2 Backup

Håndterer backup af modtagne datasæt, så dette bliver gjort på samme måde for alle pipelines og samtidig overholder NSP platformens backuppolitik.

4.2.3.3 Logning

Håndterer både statuslogning og fejllogning så dette bliver gjort på en uniform måde ikke bare på tværs af alle pipelines men også på tværs af hele NSP platformen stamdataindlæseren afvikles på. Dette gælder både SLA-, audit-, og applikations-log.

4.2.3.4 Fejlhåndtering

Håndterer fejlhåndtering så dette bliver gjort på en uniform måde på tværs af både alle pipelines og hele NSP platformen stamdataindlæseren afvikles på. Som en del af fejlhåndtering kaldes logning.

4.2.3.5 Destinationsforbindelse

Håndterer forbindelse til stamdatabasen så disse håndteres på samme måde for samtlige pipelines. Destinationsforbindelse anvender Sikkerhedskomponenten Autentifikation.

4.2.3.6 Transaktionsstyring

Håndterer transaktioner imod stamdatabasen så disse håndteres på samme måde for samtlige pipelines.

4.2.4 Sikkerhed

Komponenten sikkerhed varetager sikkerhedsaspekter for stamdataindlæseren.

4.2.4.1 Autentifikation

Håndterer autentifikation dels i forhold til forbindelse til stamdatadatabasen og dels i forhold til forbindelse til hvor stamdataudtrækket skal hentes.

4.2.4.2 Kryptering

Håndterer eventuel kryptering af stamdataudtrækkene.

4.2.5 Management

Management komponenten indeholder de funktionelle komponenter, som håndterer styringen af stamdataindlæseren i driftsmæssig sammenhæng.

4.2.5.1 Konfiguration

Indeholder konfigurationen for stamdataindlæseren. Herunder for hver enkelt type af stamdata hvilke valideringer er aktive, placering af backup for modtagne datasæt, og parametre, der styrer Fetch (hvor, hvordan, og hvor ofte hentes stamdataudtræk).

4.2.5.2 Status og Monitorering

Giver via en velfineret standardiseret overvågningsservice mulighed for at få alarmer ved fejlsituationer samt mulighed for at se aktuel status for de enkelte stamdataindlæsere. Herunder hvornår og hvor hurtigt den pågældende stamdataindlæsning sidst er blevet effektueret, og hvor mange fejl der var ved denne lejlighed (hvis nogen).

4.2.5.3 Rapportering

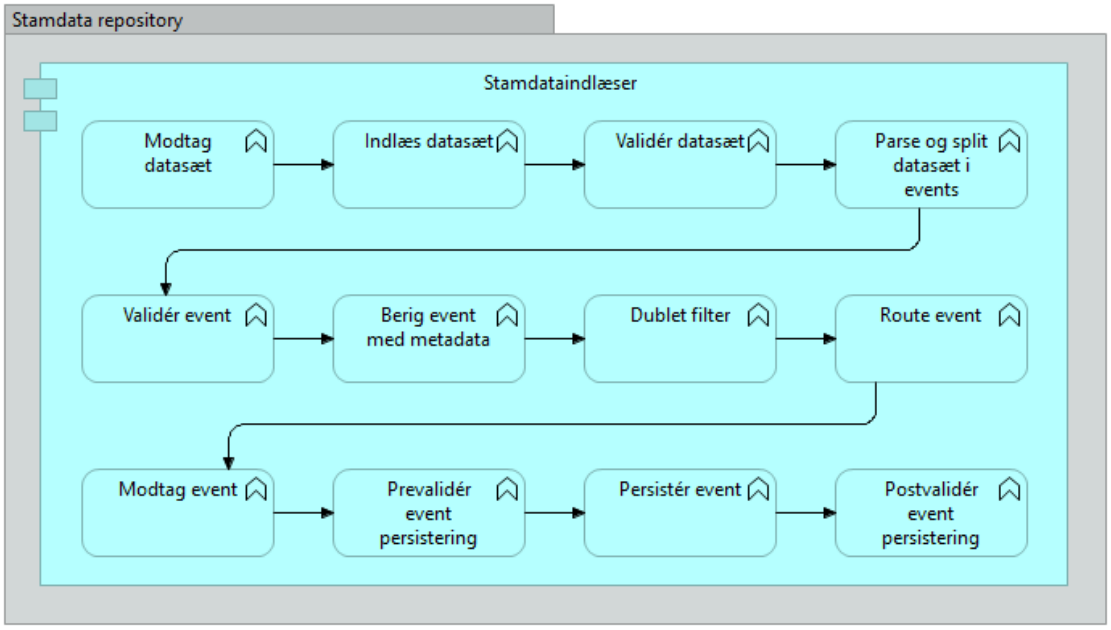
Giver mulighed for at generere rapporter til interessenter om driftssituationen og kvalitetsopfølgning for stamdataindlæseren på baggrund af monitorerings- og statuskomponenterne.

4.2.5.4 Fejlopfølgning

Giver mulighed for at identificere præcist hvilket event af hvilken stamdatatype, der har givet anledning til en given fejlsituation, så det bedst mulige udgangspunkt for opfølgning på og korrektion af fejlen er tilstede.

4.3 Indlæser proces

Vi zoomer nu yderligere ind på Pipeline applikationskomponenten i stamdataindlæseren fra Figur 3, og fokuserer på hvordan denne er bygget op af de enkelte indlæser proces trin. Pipeline med dens indlæser proces er som nævnt baseret på Pipes and Filter EIP'et [2], som er en anerkendt best practise på området, og i vores sammenhæng kan illustreres ved:



Figur 4: Detaljeret applikationsarkitektur for Pipeline i Stamdataindlæser

I resten af dette afsnit beskrives de enkelte komponenter (filtre) i pipelinen hver især i den rækkefølge de optræder i pipelinen. Hvor alternative arkitekturmønstre kunne optræde forklares det, hvorfor nærværende arkitektur er at foretrække frem for alternativer. Hver enkelt komponent foretager statuslogning via platformsfunktionen logning, så det altid er muligt at vide, hvor langt en given proces er kommet i pipelinen, såfremt logningsniveauet er konfigureret til det. Skulle der opstå en fejl under processen i en given komponent anvendes platformsfunktionen fejlhåndtering til at håndtere fejlen og herunder logges de exceptionelle omstændigheder ligeledes af komponenten via platformsfunktionen logning.
Pipelinen for indlæsning af stamdata på NSP er realiseret via Apache Camel [4], der er et meget bredt anvendt open source integrationsframework, som har indbygget support for Pipes and Filter, og derfor jf. vores applikationsprincipper er meget attraktivt at bygge vores applikationsarkitektur op omkring.
For yderligere, mere implementeringsnære, detaljer angående de enkelte filtre, der beskrives i de følgende underafsnit, se Guide til Udviklere [5].

4.3.1 Modtag datasæt

Stamdataudtrækket modtages som en fil med et datasæt via platformsfunktionen Fetch. Det sikres, at stamdataudtrækket er komplet, så man ikke risikerer, at sende et ikke komplet datasæt videre til næste komponent (Indlæs sæt), og det er på dette tidspunkt at stamdataindlæseren overtager ansvaret for stamdataudtrækket.
En foretrukken og enkel måde at sikre integriteten af stamdataudtrækket på er, at stamdataudtrækket skrives til en temporær fil i den samme filfolder, som Fetch er konfigureret til at hente fra, og først når denne temporære fil er komplet om-navngives filen til den endelige form for

navn, som Fetch kigger efter. Såfremt der er kontrolfiler udover selve filen med datasættet, f.eks. en checksum fil eller en tekstfil med antallet af events og/eller entiteter i datasættet, henter Fetch også disse.

4.3.2 Indlæs datasæt

Datasættet, og eventuelle kontrolfiler, indlæses fra fil til memory i stamdataindlæseren. Anvender, hvis relevant, sikkerhedskomponenten Kryptering.

4.3.3 Validér datasæt

Datasættet valideres i henhold til det forventede, som fremgår af den skriftlige dataaftale med stamdatakilden:

- Er datasættet encoded som aftalt.
- Overholder datasættet det aftalte veldefinerede format.
- Er de nødvendige med stamdatakilden aftalte metadata for datasættet tilgængelige.
- Har datasættet en størrelse som afviger væsentligt fra det forventede/normale.
- Indeholder datasættet det korrekte antal events og/eller entiteter i forhold til det angivne i selve datasættet, eller fra en separat kontrolfil, såfremt denne information er tilgængelig.

Hvilke af disse valideringer, der er aktive for en given pipeline er en del af konfigurationen i Management komponenten.

4.3.4 Parse og split datasæt i events

Datasættet parses til de enkelte events, og splittes derved op. Eventet følger et givet skema (jf. afsnit 4.2.2.2), hvis indhold og regler er dikteret af den tilsvarende stamdatadefinition hos stamdatakilden. Alle de efterfølgende filtre i pipelinen opererer således på eventniveau. Da dette er det sidste filter, der opererer på datasæt niveau, er det også her, der foretages backup af det modtagne datasæt via platformsfunktionen Backup.
Der er flere fordele ved at splitte datasættet op og operere med events i forhold til at lade være med dette og alene operere med de fulde datasæt:

- **Hurtigere tilgængelighed af stamdata**: Med datasæt alene er der ingen mellemvej imellem fuld succes og total fiasko for stamdataindlæsningen. Enten indlæses alt eller intet – en enkelt fejl holder indlæsningen af hele stamdataudtrækket tilbage, og intet af det pågældende datasæt bliver således tilgængeligt for anvendersystemerne. Med events er det de enkelte events der hver især indlæses korrekt eller fejler. Dvs. selvom, der er et par events fra et datasæt, der fejler, så indlæses alle de øvrige, og de er således tilgængelige for anvendersystemerne med det samme. Dette er helt i overensstemmelse med et af de centrale mål og tilhørende gevinster listet i afsnit 1.2.
- **Performance ved fejlhåndtering**: Efter korrektion af en fejl, det være sig runtime fejl (som nedgået forbindelse til stamdatadatabasen) eller datafejl, skal man med datasæt forsøge at indlæse hele datasættet igen. Med events kan man nøjes med at indlæse de fejlede events

igen, som er en (ofte meget væsentligt) mindre opgave, og derfor en mindre performance-belastning.

- **Nemmere parallelisering**: Opsplitningen i events giver et simpelt og naturligt udgangspunkt for at parallelisere processen, i det de enkelte events kan håndteres parallelt.
- **Performance pga. databasetransaktionsstørrelse**: Datasæt kan blive forholdsvis store, hvilket ville give anledning til potentielt meget store databasetransaktioner, når datasættet skulle persisteres i stamdatadatabasen. Dette er ikke en performancefordel for de fleste databaser, hvor det er en udfordring at holde store datamængder i rollback-segmenterne (og i tilfælde af fejl at rulle den store transaktion tilbage). Dette undgås ved de langt mindre events, der tilsvarende giver meget mindre databasetransaktioner. Der er så til gengæld naturligvis langt flere af disse små transaktioner, men dette håndteres væsentligt bedre af de fleste databaser (se i øvrigt også senere i afsnit 4.3.9).
- **Højere fleksibilitet i forhold til routing og sinks**: Uden opsplitning i events kan andre interessenter end stamdatadatabasen reelt kun blive informeret om, at der er en opdatering for den pågældende type af stamdata (f.eks. CPR). Med events kan de øvrige interessenter i stedet blive informeret om, at der er en opdatering af en instans af den pågældende type af stamdata med en given unik identifikation (f.eks. et specifikt CPR-nummer). Dette øger fleksibiliteten og forenkler situationen for de øvrige interessenter.

Hvis der er tale om et fuldt stamdataudtræk, hvor slettede events fremgår (implicit) ved ikke at være til stede i stamdataudtrækket, skal filteret sammenholde det fulde udtræk med det aktuelle billede af stamdateregisteret i stamdatadatabasen. På baggrund af denne sammenligning genereres slette-events for de events, der på denne måde er identificeret som slettede. Disse genererede slette-events skal håndteres på samme måde som de øvrige events i stamdataudtrækket og derfor sendes videre i pipelinen på samme vis. Denne generering skal foregå på dette tidspunkt i pipelinen, hvor det fulde overblik over events haves og så eventuelle andre modtagere af eventene end stamdatadatabasen også modtager slette-eventene.

4.3.5 Validér events

Det enkelte event valideres imod eventets skema, der er defineret i overensstemmelse med den skriftlige dataaftale med stamdatakilden:

- Er alle obligatoriske attributter på de enkelte entiteter i eventet tilstede.
- Er grænser for attributter overholdt, f.eks. en maksimal længde af en tekstattribut, et lovligt interval for en numerisk værdi eller et datotidspunkt, eller et veldefineret udfaldsrum (lookup og herunder enumeration).
- Er konditionelle regler for attributter internt i eventet overholdt, f.eks. hvis en givet attribut har en speciel værdi, så skal en anden attribut være valoriseret (eller ikke-valoriseret).

Disse valideringer er i høj grad med til at sikre en høj kvalitet af de indlæste stamdata. Event skemaet og dets regler, samt hvilke af reglerne er aktive, er igen en del af konfigurationen i Management komponenten, men som udgangspunkt er alle dem, der er nødvendige for at sikre samme datakvalitet som i stamdatakildesystemet, aktive jf. princip 3.

4.3.6 Berig event med metadata

Det enkelte event beriges med metadata. Det drejer sig ikke mindst om informationer, som identificerer hvilket stamdataudtræk eventet stammer fra, samt tidspunktet for stamdataudtrækket. Dette giver en høj grad af traceability for events i forhold til stamdataudtræk, hvilket er fordelagtig i forbindelse med fejlsøgning og datakvalitetskontrol. Derudover kan man også berige med enkelte centrale attributter fra eventet selv, som f.eks. et unikt id for eventet. Derved bliver disse informationer lettere tilgængelige, da de nu ikke kun er en del af eventets payload, men også en del af dets metadata. Dette filter afslutter det, man vil kunne kalde "producerdelen" af pipelinen.

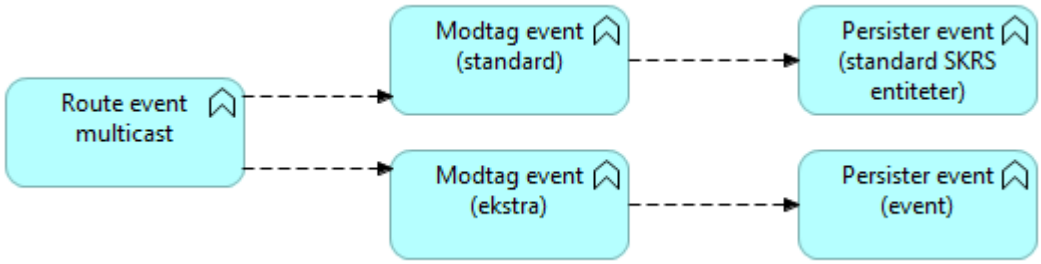
4.3.7 Dublet filter

De events, som er identiske med de gældende/eksisterende i stamdatadatabasen, frafiltreres. Dette foretages af performancehensyn, så stamdatadatabasen ikke fyldes op med ens events, og af hensyn til anvendersystemerne, så de ikke, via SKRS, kommer til at modtage en masse opdateringer, der reelt ikke opdaterer noget. Bemærk at dette ikke kun er relevant i forhold til fulde udtræk, hvor det altid er relevant, fordi mange events selvsagt kan være ens fra udtræk til udtræk, da stamdatakilden i dette tilfælde ikke skeler til hvilke events, der har ændret sig siden sidste udtræk. Det kan også være relevant for delta-udtræk, fordi i flere tilfælde modtages kun en delmængde af de data, som stamdatakilden har, og stamdatakilden har ikke nødvendigvis styr på, hvilke aftagere af data en givet opdatering er relevant for. Derfor kan det sagtens ske, at stamdatakilden markerer et event som opdateret i forhold til en aftager af data, selvom der reelt ikke er nogen ændring i den del af data, som den pågældende aftager modtager, og derfor kan dette event komme med i det næste delta-udtræk til aftageren.

4.3.8 Route event

Vi har nu det fulde event beriget med metadata, og dette sendes videre i pipelinen. Dette filter er derfor også det naturlige sted at give muligheden for også at route det fuldt berigede event til andre interessenter end bare videre i pipelinen imod stamdatadatabasen. Hvis der ikke er andre interessenter, sendes eventet blot trivielt videre i pipelinen til "Modtag event" filteret. Men hvis der er flere interessenter sendes eventet i stedet til en streamingplatform, hvorfra interessenterne hver især konsumerer eventet. Den ene af disse konsumere er i dette tilfælde så "Modtag event" filteret i vores stamdataindlæser pipeline, og de øvrige interessenter har hver deres anden konsumer – på denne måde bliver muligheden for en event sink inkluderet i pipelinen.
Muligheden for event sinks og streaming er således tænkt ind i referencearkitekturen, der derfor er klargjort til, at man kan ibrugtage Apache Kafka [6], der er den streamingplatform, som NSP allerede benytter i andre sammenhænge. Anvendelsen af Apache Kafka er dog ikke implementeret, da der fortsat udestår afklaringer af sikkerhedsmæssig karakter om tilgangen til Apache Kafka på NSP udefra. I stedet er der i dette filter i pipelinen blevet indbygget muligheden for at sende eventet til andre interessenter via Apache Camels indbyggede "multicast" funktion til

dette. Dette betyder, at udover at sende eventene til "standardmodtageren", der efterfølgende i pipelinen persisterer eventene splittet op i entiteter, som er den måde SKRS er vant til, at data gemmes på, så kan man sende eventene til en anden modtager, der gemmer eventene som de logiske events de er i stamdatadatabasen sammen med det skema, der definerer strukturen af eventene. Dette er illustreret i følgende figur:



Figur 5: Route event til flere konsumere via multicast

Den øverste vej i figuren er standardvejen, hvor eventet ender med at blive gemt som entiteter i stamdataregisteret i stamdatadatabasen, som SKRS er vant til. Den nederste vej er den ekstra, hvor eventet gemmes i stamdataregisteret i stamdatadatabasen som det logiske event med tilhørende eventstrukturdefinition. Bemærk i øvrigt at øvrige mellemliggende filtre i pipelinen, som "prevalider event persistering" er udeladt her for overskuelighedens skyld.
De logiske events sammen med skemaet, der definerer strukturen af eventene er netop hvad man ville skrive til Apache Kafka, og således opnås en repræsentation gemt i stamdatadatabasen, der er magen til den, man vil få, hvis man anvendte Apache Kafka. Dette betyder, at man via en særlig konfiguration af SKRS kan hente logiske events ad gangen, i stedet for entiteter ad gangen. På denne måde gøres det muligt for anvendersystemerne af SKRS at hente de logiske konsistente events ad gangen, så de slipper for selv at skulle holde et besværligt konsistensregnskab på tværs af de forskellige entiteter, når de henter stamdata via SKRS. Hvis et anvendersystem gør brug af denne mulighed, betyder det endvidere, at de er langt mere parate til at skifte til at konsumere de logiske events fra Apache Kafka, når den tid kommer, da det på det tidspunkt så kun er den underliggende teknologi til at hente de logiske events, der så skal ændres (webservice til Apache Kafka).

4.3.9 Modtag event

Eventet modtages fra "Route event" filteret enten direkte via pipelinen eller via Apache Kafka alt efter om streaming anvendes eller ej. I førstnævnte tilfælde er det således en fortsættelse af den samme pipeline, i sidstnævnte startes en ny pipeline, hvor dette første filter konsumerer fra Apache Kafka. Dette filter markerer starten på det, man vil kunne kalde "konsumer-delen" af pipelinen. I forbindelse med modtag event er der mulighed for at gruppere events sammen i en batch af konfigurerbar længde, så det er denne batch der samlet set ender med at blive skrevet og committet til stamdatadatabasen to filtre længere henne i pipelinen. Denne mulighed er introduceret af performancehensyn til stamdatadatabasen.

4.3.10 Prevalidér event persistering

Eventet inklusive metadata valideres. Da eventet uden metadata allerede tidligere er blevet valideret i "producer-delen" af pipelinen foretages udelukkende valideringer på tværs af payload og metadata. F.eks. kunne en validering være at sammenligne en given datotidsattribut i eventet med tidspunktet for stamdataudtrækket. I mange tilfælde er der ikke behov for denne type validering, og dette filter udfører således i disse tilfælde intet arbejde. Men filteret er medtaget i arkitekturen af fleksibilitetshensyn til de tilfælde, hvor denne type af validering er nødvendig, og muligheden for at foretage den således er tilvejebragt.
Hvilke af disse valideringer, der er aktive for en given pipeline er en del af konfigurationen i Management komponenten.
I de tilfælde hvor der er mere end én konsumer af det pågældende event, og vi altså har en sink i "Route event" filteret, skal denne type valideringer ligge i "konsumer-delen" af pipelinen, fordi disse valideringer kan variere fra konsumer til konsumer. Derfor er dette filter lagt på dette sted i pipelinen.

4.3.11 Persistér event

Eventet persisteres i stamdataregisteret i stamdatadatabasen via Datadestinationsskemaet under anvendelse af platformsfunktionen Destinationsforbindelse.

4.3.12 Postvalidér event persistering

Eventet kan afslutningsvis valideres en sidste gang efter persistering i stamdatadatabasen under anvendelse af platformsfunktionen Destinationsforbindelse:

- Kan eventet læses op igen fra stamdatadatabasen og er det identisk med det man sendte ned til databasen.
- Kontroller referenceintegritet i forhold til andre stamdataregistre end det den givne pipeline opererer på. Det er at foretrække, at kontrollere referenceintegritet på denne måde fremfor via fremmednøgler i databasen, fordi de enkelte stamdataregistre principielt er uafhængige af hinanden og derfor opdateres uafhængigt af hinanden.

Som for "Prevalidér event persistering" er der i mange tilfælde ikke behov for denne type valideringer, og dette filter udfører således oftest intet arbejde. Men filteret er medtaget i arkitekturen af fleksibilitetshensyn til de tilfælde, hvor denne type af validering ønskes, og muligheden for at foretage den således er tilvejebragt.
Hvilke af disse valideringer, der er aktive for en given pipeline er en del af konfigurationen i Management komponenten. I de fleste tilfælde er førstnævnte validering ikke aktiv af performancehensyn, men i situationer med driftsustabilitet kan den være nyttig.
Afslutningsvis foretages Commit af databasetransaktionen til stamdatadatabasen via platformsfunktionen Transaktionsstyring. På dette tidspunkt er eventet tilgængeligt for anvendersystemerne via stamdataregisterservicen.

5. Dataperspektiv

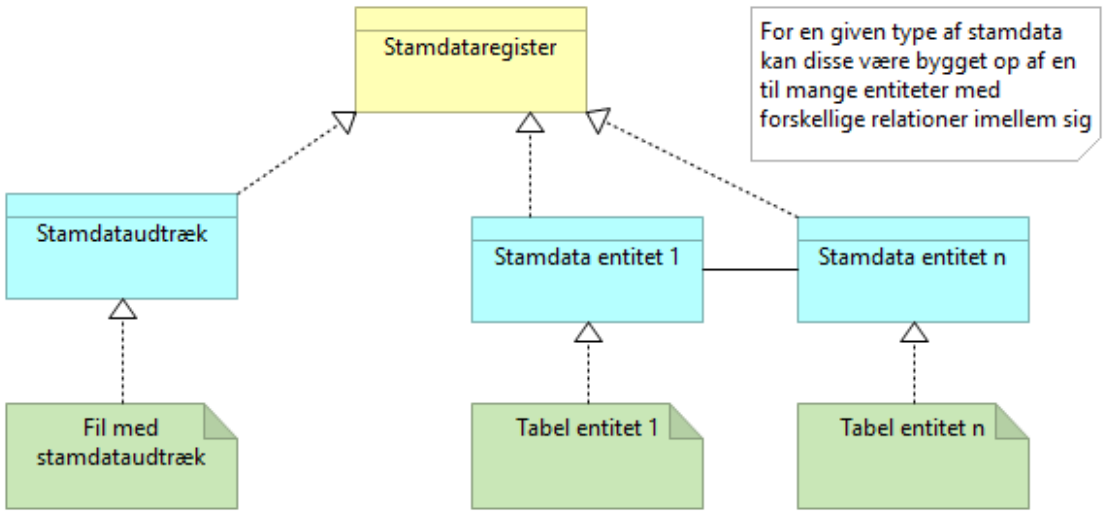
Vi zoomer nu ind på stamdataregisteret fra Figur 1 og nogle af komponenterne fra Datamodel komponenten i Figur 3.

5.1 Principper

Følgende detaljerede dataarkitekturprincipper, afledt af de overordnede principper i afsnit 1.4, er anvendt:

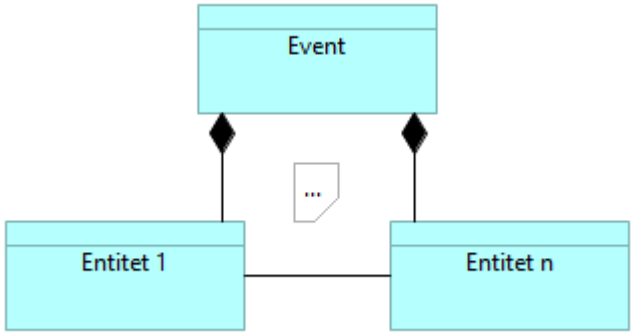
Id.	Princip	Reference
D1	Dataarkitekturen for indlæsning af stamdata på NSP skal basere sig på identifikation af logiske events, der er bygget op af atomare entiteter i overensstemmelse med datamodellen fra stamdatakilden	2, 4, F1
D2	Indlæsningen af stamdata på NSP skal validere de modtagne data i forhold til datamodellen fra stamdatakilden	3, 4, F2
D3	Indlæsningen af stamdata på NSP må ikke foretage datavalideringer, der ikke er belæg for i datamodellen fra stamdatakilden	3, 4, F2
D4	Indlæsningen af stamdata på NSP skal berige de indlæste events og entiteter med metadata, der entydigt angiver i hvilket datasæt det givne event/entitet stammer fra	4
D5	Indlæsningen af stamdata på NSP skal stille information til rådighed for de fælles driftsovervågningsservices, der eksisterer på NSP	4

5.2 Dataarkitektur



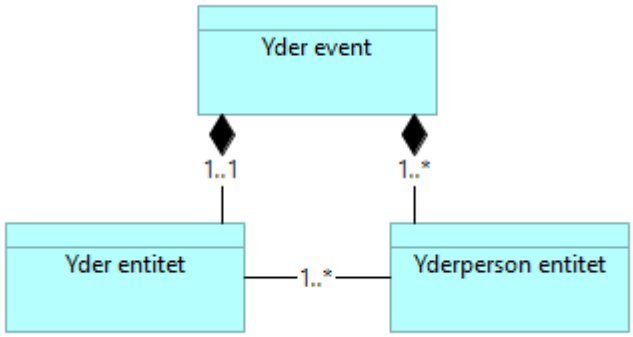
Figur 6: Overordnet dataarkitektur

Stamdataudtrækket leveres som et samlet datasæt i et veldefineret format. Dette datasæt består af mange forskellige logiske events, der hver især er opbygget af en til mange logiske entiteter, som er internt relateret til hinanden, og som tilsammen udgør eventet – f.eks. er yder events opbygget af de to entiteter yder og yderperson. Typisk vil det fysiske stamdataudtræk blive leveret i en fil, og de logiske entiteter vil blive repræsenteret en til en ved fysiske tabeller i stamdataregisteret i stamdatadatabasen.
Sammenhængen mellem events og entiteter kan illustreres generisk på følgende måde:



Figur 7: Events og disses sammenhæng med entiteter

Et event kan bestå af mange forskellige entiteter (i figuren angivet ved … imellem composerelationerne), der hver især kan være relateret til hinanden med forskellige kardinaliteter. F.eks. består et yder event af en yder entitet og en til mange yderpersoner entiteter, der hører til yderen, som illustreret ved:



Figur 8: Yder events og disses sammenhæng med yder entiteterne

Metadata gemmes på to niveauer:

1. Dels på event/entitetsniveau, hvor der beriges med information om hvilket unikt identificeret stamdataudtræk det givne event/entitet stammer fra, og placeringen af det givne event/entitet i det pågældende stamdataudtræk, da dette i høj grad forenkler fejlsøgnings og datakvalitetskontrolsituationer
2. Dels på stamdataregisterniveau, hvor der registreres information om de datasæt, der er blevet indlæst i stamdataregisteret, disses status (inklusiv eventuelle fejl), samt, baseret på disse enkelte datasæts status, stamdataregisterets akkumulerede status, da dette er vigtig information i forhold til driftssupporten af det pågældende register

6. Teknologiperspektiv

Vi zoomer nu ind på forskellige dele af teknologiarkitekturen, der ligger under forretnings- og applikationsarkitekturen i henholdsvis Figur 1 og Figur 3, og herunder Modtag sæt applikationsfunktionen fra Figur 4.

6.1 Principper

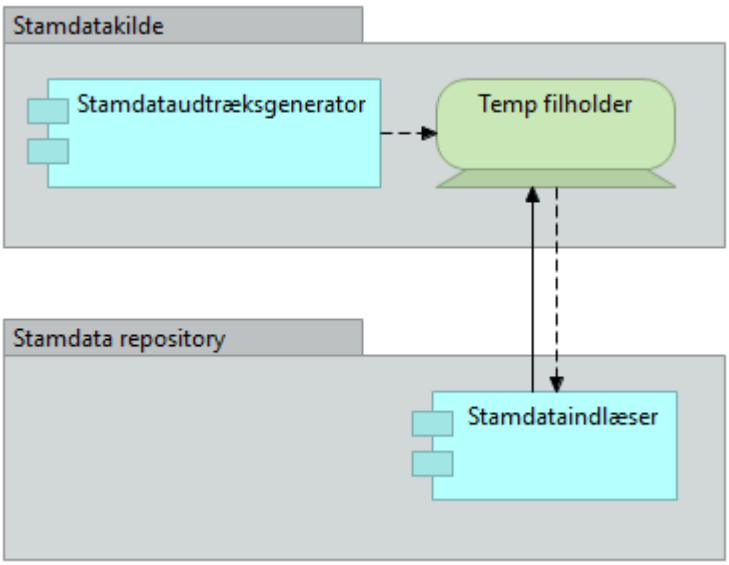
Følgende detaljerede teknologiarkitekturprincipper, afledt af de overordnede principper i afsnit 1.4 og applikationsarkitekturprincipperne i afsnit 4.1, er anvendt:

Id.	Princip	Reference
T1	Teknologiarkitekturen for indlæsning af stamdata på NSP skal basere sig på de eksisterende velafprøvede driftsmodnede teknologier på NSP	6, F4, A4
T2	Teknologiarkitekturen for indlæsning af stamdata på NSP skal basere sig på best practices, som anvender open source teknologier	7, F5, A5
T3	Teknologiarkitekturen for indlæsning af stamdata på NSP skal overholde husreglerne på NSP	8, A6
T4	Ved indlæsning af stamdata på NSP er det udelukkende stamdataindlæseren, der må hente stamdataudtræk og skrive i stamdataregisteret	6, 9, F6, T1

6.2 Teknologiarkitektur

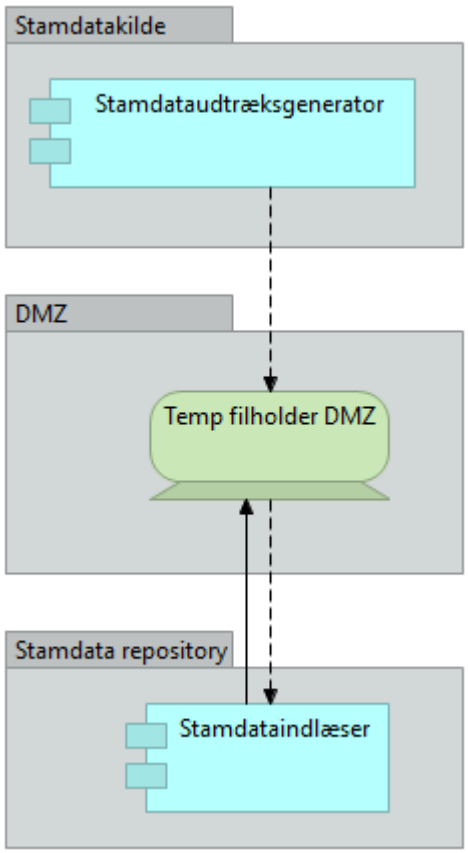
6.2.1 Modtag sæt

Når stamdataudtrækket genereres hos stamdatakilden, så kan udtrækket enten placeres hos stamdatakilden selv, hvorfra Modtag sæt proces trinnet i stamdataindlæserens pipeline via platformsfunktionen Fetch henter det, som illustreret i følgende figur:



Figur 9: Overordnet teknologiarkitektur for Modtag sæt ved direkte forbindelse imellem stamdataindlæseren og stamdatakilden

Eller stamdatakilden kan aflevere stamdataudtrækket til stamdata repository. I sidstnævnte tilfælde er man dog af sikkerhedshensyn, jf. princip T4, ikke interesseret i, at stamdatakilden skal have en direkte forbindelse ind i stamdata repository. I stedet får stamdatakilden kun adgang til at aflevere stamdataudtrækket til en demilitariseret zone (DMZ) med lidt lavere sikkerhedskrav på forsiden af repositoriet, hvorfra stamdataindlæseren via Fetch henter det, som illustreret i følgende figur:



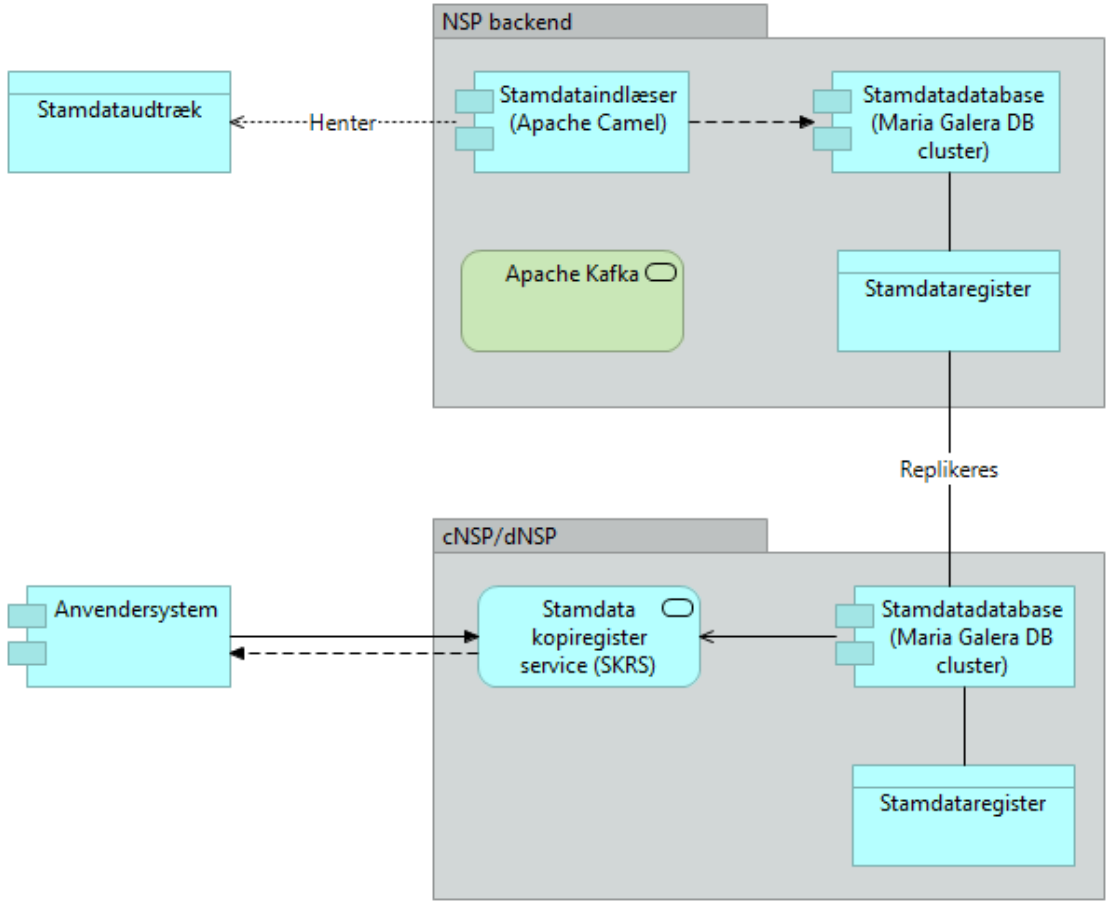
Figur 10: Overordnet teknologiarkitektur for Modtag sæt ved anvendelse af DMZ

På denne måde sikres det, at der aldrig er mulighed for at aflevere/skrive data direkte ind i det højt sikrede stamdata repository. Det er altid stamdataindlæseren, der inde fra det højt sikrede stamdata repository henter/læser data. Hvilken løsning man vælger kan variere fra stamdatakilde til stamdatakilde, så et stamdata repository kan hente nogle typer af stamdata via den ene måde og andre typer via den anden måde. Bemærk i øvrigt at alt efter sikkerhedskravene hos stamdatakilden kan den temporære filholder i Figur 9 godt ligge i stamdatakildens eget DMZ.
Den foretrukne protokol til overførslen af stamdataudtrækket er SFTP, og som nævnt i afsnit 4.3.1 er den foretrukne måde at sikre integriteten af stamdataudtrækket på, at dette skrives til en temporær fil, der først om-navngives til den endelige form for navn, som stamdataindlæseren er konfigureret til at kigge efter, når skrivningen af den temporære fil er komplet.
Hvis der ønskes yderligere høj sikkerhed omkring overførslen af stamdataudtrækket i form af end to end kryptering skal stamdataindlæseren være konfigureret med den korresponderende nøgle til den krypteringsnøgle som stamdatakilden har krypteret stamdataudtrækket med.

6.2.2 NSP teknologiarkitektur

Generelt har NSP en distribueret arkitektur, hvor der er en backend instans, mange ens dNSP (distribueret NSP) instanser, en for hver dansk region, samt en cNSP (central NSP) instans, magen til dNSP, primært til kommunal anvendelse, således at anvendersystemer af services på NSP

altid kan kalde den instans, der "er nærmest ved dem selv". Stamdataindlæserne afvikles på NSP backend, og som følge af den distribuerede arkitektur er stamdatadatabasen på NSP, som stamdataregistrene for de enkelte typer af stamdata tilhører, et Maria Galera DB cluster, der er replikeret ud på samtlige NSP instanser, således at de anvendende systemer kan kalde den SKRS instans, der er mest opportun for dem, for at hente deres relevante stamdata. Dette er illustreret i følgende figur:



Figur 11: Udvalgt NSP teknologiarkitektur

På figuren er det ligeledes illustreret at stamdataindlæserne er baseret på Apache Camel, og Apache Kafka, der allerede eksisterer til andre formål på NSP, er ligeledes indikeret for potentiel fremtidig anvendelse i forbindelse med indlæsning af stamdata.

6.2.3 Stamdataregister

De enkelte stamdataregistre i Maria Galera DB clusteret er modelleret via databasetabeller, som repræsenterer de enkelte entiteter for de givne typer af stamdata, som illustreret i højre halvdel af Figur 6 i afsnit 5.2. En enkelt række i databasetabellen for en given entitet svarer til en given version af entiteten. Ud over de attributter som datamodellen fra stamdatakilden definerer en entitet har, har databasetabellen for den givne entitet nogle yderligere attributter, der er defineret af SKRS. Dette drejer sig om følgende:

Navn	Type	Forklaring
Id	Varchar	Unik id for rækken – kan eventuelt være sammensat af (beregnet ud fra) flere forskellige attributter for entiteten
ModifiedDate	Datetime	Tidspunkt for sidste ændring af rækken
ValidFrom	Datetime	Tidspunkt fra hvilket rækken er gyldig
ValidTo	Datetime	Tidspunkt til hvilket rækken er gyldig

De to sidste attributter i tabellen er måden, hvorpå teknisk historik håndteres i forhold til SKRS, ikke at forveksle med forretningsmæssig historik jf. afsnit 4.2.2.5. I de, heldigvis, sjældne tilfælde hvor stamdataudtrækket selv indeholder forretningsmæssige attributter kaldet ”ValidFrom” og ”ValidTo” skal disse attributter gives et klart pre-fix i databasetabellen, der bevirker, at man kan skelne dem fra de to SKRS attributter. Derudover skal databasetabellen for entiteten indeholde attributten DataSetUUID, der skal indeholde den unikke nøgle, der definerer hvilket datasæt rækken stammer fra.

6.2.4 Opsummering af teknologiarkitektur

Flowet i indlæsning af stamdata på NSP med fokus på teknologiarkitektur kan jf. de tidligere afsnit opsummeres ved:

1. Stamdataudtrækket hentes som det første indlæser proces trin i Apache Camel pipelinen på NSP, hvilket fortrinsvis foregår via SFTP – dog kan andre protokoller undtagelsesvis forekomme i særlige tilfælde
2. Stamdataudtrækket videreprocesseres i Apache Camel pipelinen på NSP – dette gives der ingen dispensationer for
3. Stamdataudtrækket gemmes som det sidste indlæser proces trin i Apache Camel pipelinen i den underliggende Maria Galera DB cluster stamdatadatabase på NSP – dette gives der pt. ingen dispensationer for, men på sigt kan dette ændres til Apache Kafka på NSP.

Der gives ikke dispensationer for anvendelsen af Apache Camel implementeringen af Pipes and Filter EIP’et. Der gives som udgangspunkt heller ikke dispensationer i forhold til manglende overholdelse af husreglerne på NSP, bortset fra i helt særlige og velbegrundede tilfælde.

Henvisning

1. Arkitekturprincipper for Sundhedsområdet. https://sundhedsdatastyrelsen.dk/-/media/sds/filer/rammer-og-retningslinjer/referenceaktitektur-og-it-standarder/arkitekturprincipper_version-2,-d-,0.pdf?la=da
2. Gregor Hohpe & Bobby Woolf: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions
3. Archimate®: https://pubs.opengroup.org/architecture/archimate3-doc/
4. Apache Camel ™: https://camel.apache.org/
5. Guide til udviklere: https://www.nspop.dk/display/NSI/Guide+til+udviklere
6. Apache Kafka®: https://kafka.apache.org/