Forprosjektrapport

Dette dokumentet beskriver funn i forprosjektet, og diskuterer de forskjellige løsningene som er foreslått. Videre beskrivelse av hva målene er med prosjektet, en rask innføring av dagens situasjon og hvilke virkninger de løsningene vi ser for oss vil ha på dagens situasjon.
Dokumentet er beregnet for oppdragsgiver og veileder, og er ment som en innføring i systemet og de underliggende applikasjonene.

Skrevet av: Tim Sæterøy;Ole Henrik Paulsen;Anders Struksnæs;Lars Haugan
Filnavn: Forprosjektrapport.docx
Status: Ferdig
Versjon: 1.0
Opprettet: 14.01.2013
Sist endret: 25.01.2013 19:56:00
Sider: 11

SpareBank 1	Forprosjektrapport			
Skrevet av: Tim Sæterøy;Ole Henrik Paulsen;Anders Struksnæs;Lars Haugan	Status: Ferdig	Versjon: 1.0	Opprettet: 14.01.2013	Sist endret: 25.01.2013 19:56:00

Revisjonshistorikk

Dato	Versjon	Beskrivelse	Forfatter
22.01.13	0.9	Utkast	*
25.01.13	1.0	Ferdig	*

*Alle punkter er dokumentert og godkjent av Ole Henrik Paulsen, Tim Sæterøy, Lars Haugan og Anders Struksnæs.

Innholdsfortegnelse

1. PRESENTASJON

PROSJEKTTITTEL Nettverksbasert applikasjonsovervåking

APPLIKASJONSTITTEL PySniff

OPPGAVE Utvikle en applikasjon for innhente, prosessere og presentere nettverksdata for å illustrere driftssituasjonen til applikasjoner.

1.1 MEDLEMMER I PROSJEKTET

Anders Struksnæs
Lars Haugan
Ole Henrik Paulsen
Tim Sæterøy

1.2 OPPDRAGSGIVER

SpareBank 1 Gruppen AS
KMIT & Innkjøp
Hammersborggata 2
0191 Oslo
origo@sparebank1.no

1.3 KONTAKTPERSON

SpareBank 1 Gruppen AS
Martin Jensen
Martin.Jensen@sparebank1.no
40218026
Avdelingsleder KMIT & Innkjøp, Drift Alliansen

1.4 PRESENTASJON AV OPPDRAGSGIVER

Origo og Drift Alliansen er avdelinger for IT brukerstøtte og drift i SpareBank 1 Gruppen. Avdelingene leverer tjenester til SpareBank 1 Gruppen AS med datterselskap og bankene i SpareBank 1 Alliansen. Som en del av oppgavene til Origo og Drift Alliansen jobbes det med monitorering og overvåking av IT-tjenestene innen SpareBank 1 Gruppen AS.

2. SAMMENDRAG

Formålet med prosjektet er å forenkle overvåking av applikasjoner, ved å detektere driftshendelser, minske falske positiver, og å kunne se dette i forhold til en normal driftssituasjon.

Løsningen skal utvikles i Python. Den vil bestå av flere deler som håndterer de forskjellige oppgavene. Data vil bli hentet inn fra sensorer som er plassert på nettverket, eller på maskiner som kjører andre applikasjoner. Deretter sendes data til core, som jobber mot den sentrale databasen. Databasen er delt i to logiske lag, hvor sanntidsdata lagres i det ene laget og data som er fra lengere tid tilbake lagres i det andre logiske laget. Dette gjøres for at data skal ta minst mulig plass, og igjen være raskt tilgjengelig.
Presentasjonen av data vil foregå på frontend, som er en webapplikasjon. Denne kommuniserer med core som gjør utregninger på data som er lagret i databasen. Frontend vil være optimalisert for å brukes på både overvåkingsskjermer og arbeidsmaskiner.

3. DAGENS SITUASJON

SpareBank 1 har i dag flere avdelinger som er avhengig av god overvåking for å detektere og håndtere driftshendelser og avvik så raskt som mulig. Viktigheten av overvåkingen er å sørge for at de riktige tiltakene blir tatt så raskt som mulig for å sørge for at tjenester tilgjengelig og funksjonable innenfor avtalte mål beskrevet i SLA.
Dagens situasjon er preget av god overvåking, men det kreves kontinuerlig kjennskap til den pågående driftsituasjonen for å være i stand til å forstå utslag i overvåkingssystemene på en god måte. Det er også mye vedlikehold av overvåkingen ved nye leveranser av applikasjoner, noe som gjør sannsynligheten til falske utslag større og muligheten for at hendelser ikke blir detektert og håndtert korrekt.

4. MÅL OG RAMMEBETINGELSER

4.1 Mål

Målet med dette prosjektet er å forenkle feilsøking for applikasjoner. Det ønskes trending ved at det utarbeides en grense på hva som er normal og ikke normal i tidsperioder, for å sammenligne driftsstatus og eventuelle avvik. Ved hjelp av trending skal det enkelt være mulig å se hva som er reelle feilsituasjoner, uten opplæring eller erfaring med det aktuelle systemet.

Det ønskes at trendingen skal kunne luke ut false positives for systemer som ofte feilrapporterer feil eller ustabiliteter i dagens systemer. Mye feilrapportering kan føre til at faktiske feilsituasjoner ikke blir tatt hånd om like fort, ("ulv ulv").

Implementasjonen av systemet er i stor grad opp til gruppen, men veiledes av oppdragsgiver. Det viktigste for SpareBank 1 er at det vil fungere i praksis, kunne kjøre på deres systemer og at det er muligheter for videreutvikling (enten av gruppen eller ansatte hos oppdragsgiver).

4.2 Rammebetingelser

4.2.1 Prosjekt

4.2.1.1 Tid og rammebetingelser

- Statusrapport ble ferdigstilt 26.10.12
- Prosjektskisse ble ferdigstilt 07.12.12
- Forprosjektrapport skal leveres 25.01.13
- Prosjektet skal være ferdig innen 28.05.13.
- Presentasjon av prosjektet skal skje i uke 24 (juni).
- Produksjonssetting etter milepæler:
    - Milepæl 1: Uke 10
    - Milepæl 2: Uke 14
    - Milepæl 3: Uke 17
- Deadline for produksjonssetting hos SpareBank1 uke 21.

4.2.1.2 Egne mål og rammbetingelser

- Gjennom dette prosjektet vil gruppen få innblikk i utvikling og systemutvikling i arbeidslivet. Teori i praksis.
- Vi vil måtte lære oss et nytt programmeringsspråk, Python, samt sette oss inn i mange nye systemer og teknikker vi ikke har erfaring fra skolen:
    - Nettverksprogrammering og -analyse
    - Applikasjonsovervåking
    - Statistikk, aggregering av data
    - Håndtering av store mengder data i database
    - Produksjonssetting med pakker i Linux distroer (RPM).

- Erfaring med dokumentasjon av prosjekter i arbeidslivet.

4.2.2 Tekniske

4.2.2.1 Programvare

- Overvåkingssystemet skal kunne kjøre på et Linux-miljø.
- PySniff skal, som navnet tilsier, utvikles i Python.
  Python-versjonen må samsvare med versjon av Python i distribusjoner brukt i produksjonsmiljø (RHEL 5/6, CentOS 5/6).

4.2.2.2 Hardware / Nettverk

- Virtuell eller dedikert server for testing som samsvarer med produksjonsmiljø.
- Mulighet for å arbeide med reelle testdata under utvikling med tanke på størrelsen på dataene.
- Tilgang til innkommende og utgående trafikk på systemer som skal overvåkes.

4.2.2.3 Annet

- Testdata levert av oppdragsgiver; i første omgang vil utviklingen kunne ta utgangspunkt i genererte testdata i et privat utviklingsmiljø, men etterhvert vil det være nødvendig med reelle testdata. Reelle testdata er viktig med tanke på størrelsen på datamengden.
  - Nettverkstrafikk
  - Logger
- Datamaskiner og arbeidslokale ved arbeid på interne systemer.

5. LØSNINGER / ALTERNATIVER

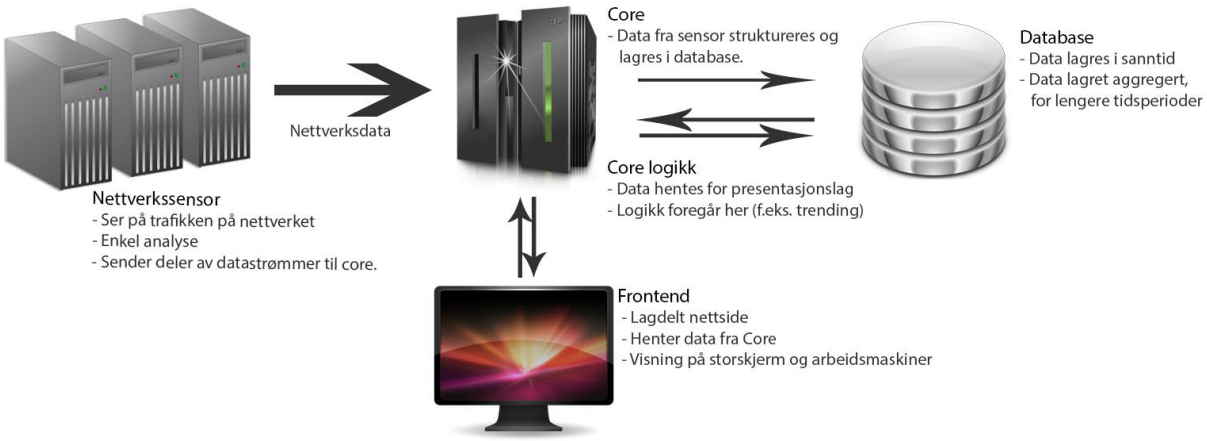
Vi beskriver her mulige løsninger ved utviklingen av et nettverksbasert applikasjonsovervåkingssystem. Ut i fra de gitte målene, er dette en beskrivelse om hvordan løsningen er tenkt utført for å kunne forenkle feilsøking og, ved hjelp av trending, lett kunne se feilsituasjoner.

Hovedforskjellen mellom andre allerede eksisterende overvåkingssystemer og denne applikasjonen er måten man innhenter data og prosesserer disse. I vanlige overvåkingssystemer blir applikasjoner overvåket ved for eksempel å se på loggfiler og andre data som ligger på serveren som applikasjonen kjører på. Tanken med denne applikasjonen er å gjøre det mulig å se på strømmer som kommer fra nettverket. I store applikasjoner blir gjerne viktig data, som brukerinformasjon, hentes over nettverket fra andre applikasjoner i samme nettverk. Det er derfor viktig å kunne detektere hvis det er noe som feiler, alle steder i applikasjonsstrukturen.

Hvis vi tar for oss nettbanken som et eksempel, er det kun det øverste laget man er i kontakt med når man som bruker er logget inn. I virkeligheten er det mange systemer som jobber bakover for å hente data som kontooversikt, meldinger fra banken og egentlig all annen informasjon som har noe med deg som bruker å gjøre. Ved benyttelse av et vanlig overvåkingssystem vil mye av overvåkingen basere seg på logger fra de serverne som er foran, og det kreves ofte mye leting for å finne årsaker til eventuelle feil. Når man da kan se på nettverkstrafikken vil man kunne hente ut det som er relevant å overvåke fra nettverket og ikke være avhengig av at applikasjonen gjør det samme for hver leveranse. Det er også viktig å nevne at det er mye av det som skjer på applikasjonene som man ikke vil kunne se med et nettverksbasert applikasjonsovervåkingssystem, og er ment som et supplement til vanlig applikasjonsovervåking.

For å lage en applikasjon som skal gjøre det mulig å se på nettverkstrafikk og lage fornuftig data av dette har vi i forprosjektet sett at å dele opp applikasjonen i adskilte programmer som jobber sammen vil være en god løsning, spesielt med tanke på skalerbarhet og distribusjon, men også med tanke på stabilitet.

Applikasjonen som skal utvikles i dette prosjektet deles inn i 4 logiske, og adskilte enheter; sensor, core, database og frontend. Vi ser her hvordan data hentes inn fra sensoren og blir prosessert og til slutt vist i frontend.



Bilde 1 - Enkel modell av applikasjonens ralasjon

5.1 Sensor

Sensoren er den delen av applikasjonen som henter ut data fra nettverket. I forprosjektets fase har vi sett to måter å løse denne delen av applikasjonen på, analyserende og ikke-analyserende sensor. Et viktig element i valget mellom de to forskjellige versjonene av denne vil være ressursbruken på maskin og nettverk. Produksjonsnettverket har en øvre kapasitet på ca. 30 Gbps, noe som legger grunnlaget for valg av type sensor.

5.1.1 Analyserende sensor

En analyserende sensor benytter serveren den står på til å gjøre beslutninger om den skal sende data videre til core eller ikke. Det kreves at det skrives moduler eller plugins til sensoren som hver har ansvaret for å håndtere data fra forskjellige type protokoller og applikasjoner. Dette vil gjøre det mulig å sortere ut all trafikk som ikke er relevant, og kun sende videre det som er av interesse. Denne type sensoren vil som den ikke-analyserende delen sende data videre til core på en strukturert måte for videre håndtering. Hvor mye analyse som skal gjøres, er vanskelig å si i denne delen av prosjektet, da det bør testes hvor mye data som kan analyseres før det oppstår tregheter i systemet.

Fordeler

+ Mindre data blir sendt videre
+ Kun data som er interessant blir sendt til database
+ Veldig skalerbart
+ Mulig å minske duplisering av data

Ulemper

- Ved ny funksjonalitet må en ny versjon av sensor produksjonssettes
- Mulig viktig data kan gå tapt ved feilkonfigurasjon eller uventede situasjoner
- Mer ressursbruk på applikasjonsservere
- Tar noe lenger tid å implementere

5.1.2 Ikke-analyserende sensor

Den ikke-analyserende sensoren er tenkt til å oversette nettverkstrafikken og sende denne videre til core for å bli satt inn i den sentrale databasen. Som navnet tilsier vil den ikke analysere dataene og velge ut det relevante, men i stedet sende all data videre. Denne type sensor vil være veldig risikabel i forhold til nettverkskapasitet inn til core, og kan også resultere i veldig store datamengder som ikke er nødvendig for analyse av driftsituasjonen.

Fordeler
+ Mindre resursbruk på applikasjonsserevere.
+ Mindre vedlikehold av sensor.
+ Ingen data går tapt.
+ Tar kortere tid enn analayserende sensor å utvikle.

Ulemper
- Veldig høy båndbreddebruk.
- Lite skalerbart
- Samme data blir sendt flere ganger hvis sensorer er plassert forskjellige steder i applikasjonskjeden. Fører til høyere disk-, minne- og nettverskbruk.

Sensoren må også sende data som enten analyseres eller ikke analyseres videre til enten core eller direkte til databasen. Det er derfor to alternativer knyttet til dette, nemlig bruk av en webservice mot core, eller kobling direkte knyttet til databasen.

5.1.3 Fra sensor til core med en webservice
En webservice er en måte å la to applikasjoner kommunisere over et nettverk. Dette gjør det mulig å sende data som igjen prosesseres av funksjoner på den mottakende enden. Denne metoden er rask og effektiv både å sette opp og å videreutvikle, men vil gjøre at det blir tregheter i applikasjonen i motsetning til om data blir sendt direkte til databasen.
Ved å benytte en webservice vil det være mulig å gjøre videre analyse av dataene som sendes fra sensor til core før disse settes inn i databasen. Det vil da være mulig å foreta en analyse av dataene på core, og avgjøre om pakken er av interesse eller ikke basert på denne.

5.1.4 Fra sensor direkte til database
Et alternativ til å benytte core for videre analyse er å desentralisere analyseringen av data før dette settes inn i det første laget av databasen (beskrevet i neste punkt). Tanken med denne logikken rundt datastrømmen, er at data skal gå fra sensoren og direkte inn i databasen, uten å gå via core. Dette vil resultere i mye mer effektiv innhenting av data, og optimalt mindre trafikk på nettverket, i form av overhead, enn ved å sende data til en webservice. Dette betyr at sensoren kan koble seg rett mot databasen, og gjøre direkte innsetting i denne.
Dette er nok en mindre ønskelig situasjon enn benyttelse av en webservice, da man mister et mulig ledd med analyse, som kan gjøre at man får mer av den informasjonen man faktisk er interessert i.

5.2 Core
Core er den sentrale enheten i applikasjonen med den viktigste oppgaven å kommunisere med databasen(e). Mye av den sentrale logikken vil også være plassert her, blant annet for utregning av trender, ved at det utarbeides en grense på hva som er normal og ikke normal i tidsperioder. Det er denne delen av applikasjonen som er viktigst for å utføre alle oppgavene fra alle de andre delene av applikasjonen.

Core er den sentrale delen av applikasjonen og er den delen som knytter alle de andre delene sammen. Derav navnet core eller "kjernen". Core er en blanding av Business Logic Layer (BLL) og Model (Databasemodell) som har som ansvar å utføre logikk og databasetransaksjoner for andre enheter i systemet. Nærmere bestemt betyr dette at core vil fungere som den delen som henter data som skal vises på frontend, innsetting av data som kommer fra sensor, og alle logiske operasjoner på de dataene som kommer fra sensor. Dette betyr en kodebase som er sentralisert og ikke er kopiert rundt i applikasjonen.
Core vil fra starten av ha essensiell funksjonalitet for vanlig drift, men vil utvides med plugins for å utbedre funksjonaliteten til selve applikasjonen. Vanlig funksjonalitet som vil befinne seg i plugins er funksjoner for å håndtere de forskjellige nettverksprotokoller og håndtere disse på riktig måte. Et eksempel på dette er når

man ser på nettsider som går på http trafikk vil det da gjøres spesielle instruksjoner som henter ut adressen til nettsiden, hvor lang tid det har tatt å hente nettsiden og hva slags feilmelding man eventuelt får.

Core har ansvaret for to hovedoperasjoner, der den ene er å jobbe mot databasen(e) og den andre å gjøre analyse og operasjoner på dataene.

En av oppgavene til core vil være å sørge for at strukturen på dataene som kommer fra sensoren er riktig i forhold til databasen, samtidig som man kan verifisere at dataene som kommer inn er relevante å lagre. Dette kan for eksempel være et ønske om å kunne filtrere ut deler av dataene allerede før de kommer inn i databasen ved administrasjonsmuligheter fra frontend.
Core vil også være bindeleddet mellom frontend og dataene som ligger lagret i databasen, og vil derfor gi frontend tilgang på funksjoner som kan benyttes for å hente ut data. Et eksempel på dette vil være å få en ferdig utregnet trend av dataene for den siste time, og gir da muligheten for å se om situasjonen er tilsvarende tidligere perioder eller forskjellig.
Når det kommer til dataene som ligger lagret i databasens første lag skal disse også aggregeres. Aggregering av data gjør at kun de dataene som er nødvendig å ta vare på i lengere tid er lagret, mens data som er mindre viktig blir kastet. Håndteringen av dette vil være skrevet i plugins som gir ekstra funksjonalitet ved utvidelser. Et eksempel på utvidelser i form av plugins vil være ved håndtering av både http trafikk, men også trafikk som benyttes for spørringer mot database (sql).

5.3 Database

I dette systemet vil data bli lagret i en sentralisert database. Dette gjøres for å ha muligheten til å hente ut data som det skal gjøres logiske operasjoner på. Det vil være en kontinuerlig strøm til databasen, og det setter ekstra krav til hvordan databasen struktureres, og ikke minst hva slags databaseteknologi man skal benytte.
Databasen vil bli delt opp i to logiske deler, layer 1 og layer 2, som vil inneholde henholdsvis sanntidsdata i det første laget og mer aggregerte data i lag to. Det er i det første laget i databasen som sensor vil sende en kontinuerlig strøm til. Dataene, fra sensor, som er lagret i det første laget skal kun lagres for en kort periode, slik at man etter denne perioden skal aggregere dataene.
Lag to i databasen er delen som vil inneholde data for tidsperioder lengere tilbake i tid. Her er noe av det viktige at uønsket data ikke skal lagres, slik at man trenger så lite plass i databasen som mulig.

For lag en, er det en vurdering om man skal benytte en "in memory"-database eller en relasjonsdatabase. Vi kommer her nærmere inn på forskjellene mellom disse.

5.3.1 "In memory"-database

"In memory"-database er i hovedsak en database som ikke ligger lagret på harddisken på serveren som den kjører på, men i minne. Dette gjør at behovet for kraftig hardware i form av dyre harddisker ikke lenger er nødvendig, og med dagens priser vil det derfor være mulig at det lønner seg å benytte en maskin med store mengder ram i stedet.

Fordeler
+ Rimeligere drift over lengere perioder.
+ Raskere aksesseringstider

Ulemper
- Kan risikere datatap ved strømbrudd
- Avhengig av to forskjellige databasetyper for lag en og lag to

5.3.2 Relasjonsdatabase

Som motsetningen til å kjøre en "In memory"-database er alternativet en vanlig relasjonsdatabase hvor data lagres på fil og ikke i minnet. Lag to vil benytte en relasjonsdatabase for lagring av persistente data lengere tilbake i tid, og i den sammenhengen er det en fordel å benytte samme database for lagringen av både sanntidsdata og aggregerte data.

Fordeler
+ Ingen fare for tap av data
+ Benytte samme database til lag en og lag to.

Ulemper
- Tregere aksesseringstider enn "In memory"-database
- Mye diskaksess som sliter på hardware

5.4 Frontend

Det å kunne presentere data er viktig for å kunne vise om noe avviker fra normalen, eller ikke er som det skal. For å gjøre hele applikasjonen så lett å vedlikeholde som mulig er det derfor viktig å kunne distribuere dataene på en fornuftig måte slik at alle kan få presentert det på samme måte, men uten å ha spesielle hardware- eller softwarekrav for at man skal kunne kjøre den. Dette gjelder spesielt ved bruk som et generelt overvåkingsverktøy hvor man ønsker å ha applikasjonen kjørende på en stor overvåkingsskjerm, men også muligheten til å presentere dette på mindre skjermer som på en arbeidsmaskin.

Dette løses ved å tilgjengeliggjøre data på en nettside. Nettsiden vil være et viktig ledd for å kunne presentere de dataene som blir analysert, og vise om dette tilsvarer normale tilstander, eller om det er en uønsket driftssituasjon.
Planen for frontend er å kunne benytte seg av funksjonene i core for å presentere de viktige analysene som gjøres av dataene. Dette vil være for eksempel kall mot webservices eller liknende. Det som etter hvert ønskes av frontend er å få til en grafisk representasjon av trenden som er i driftsbildet, ved for eksempel å vise hvor mange som er logget inn i nettbanken, og da samtidig vise hvor mange som normalt er innlogget. Et annet ønske når det gjelder gjennomføringen av denne siden er både å kunne mellomlagre de analyserte dataene, men også å kunne få siden til å oppdatere seg automatisk, og gjerne ved bruk av javascript for å gjøre selve opplevelsen av applikasjonen bedre.

6. ANALYSE AV VIRKNINGER

Det er presentert flere forskjellige løsninger innenfor de forskjellige delene av applikasjonen, men strukturering og dataflyten i selve applikasjonen er veldig klar. Det er derimot flere alternativer til hvordan delene av applikasjonen PySniff kan utvikles. Alternativene diskutert i denne rapporten viser til forskjellige elementer som er viktige for gjennomføringen og hvordan den ferdige applikasjonen vil håndtere data.

Analyse av alternativer i sensor
Sensoren kan som beskrevet løses på flere måter, der de viktige elementene er om, og i hvilken grad, data skal analyseres på sensor.
Med dette som utgangspunkt er det teoretisk sett bedre utnyttelse av resurser både på nettverket og på serveren som core skal kjøre på. Dette medfører større ressursbruk på andre servere, men i praksis kan dette føre til lavere kostnader ved drift av overvåkingssystemet. På den andre siden avhenger en analyserende sensor av mer vedlikehold, og dette er et punkt som ønskes å gjøre så minimalt som mulig. Derfor krever en sensor som skal gjøre deler av analysen oppdateringer for å fungere optimalt med denne løsningen.
En ikke-analyserende sensor vil i denne sammenhengen kreve veldig lite vedlikehold, men veldig mye høyere ressursbruk på core og nettverk. Dette er mindre ønskelig, men vil også fungere i mindre miljøer. Dette er veldig lite skalerbart i forhold til å produksjonssette dette i et produksjonsmiljø med veldig mye datatrafikk.

Et annet punkt som også er direkte relatert til om sensoren er analyserende eller ikke, er hvor sensoren skal sende dataene. Hvis sensoren i større eller mindre grad gjør analyse av dataene, er det mulig å få sensoren til å gå utenom core og direkte til database. Fordelen med å kutte ut dette ene leddet (som er core), vil være at dataene kommer raskere inn i databasen, og et mindre mellomledd som må utvikles.
Det vil være mange problemer med å designe sensor til å sette data direkte inn i database uten å gjøre analyse på dataene først. Et godt eksempel på dette er hvis det lastes ned store oppdateringer til operativsystemet. Dette er noe man ikke ønsker å lagre i databasen, og må derfor filtreres bort. Den optimale løsningen som vi kan se her vil være å utvikle en analyserende sensor som sender data videre til core, før det kan analyseres (i større eller mindre grad) før det settes inn i databasen. Dette gjør det også lettere å skrive programvare som er rettet mot databasen, da dette kun trengs å gjøres en gang.

Sensoren er en potensielt stor del av oppgaven, men det er ikke meningen at denne skal utgjøre en stor del. Det er derfor, ved problemer eller høyt tidsforbruk på denne delen, mulig å benytte http-access logger (som er en logg over alle kall som kommer til en webserver) for å illustrere en type data (http). Dette vil fortsatt kunne illustrere funksjonaliteten i resten av oppgaven.

Analyse av virkninger i core
Mye av funksjonaliteten til core ligger i analyseringen av dataene. Dette skjer først etter at dataene har blitt sendt til databasen. Mye av analysen som vil skje på core vil gjøres av plugins, og vil være veldig avhengig av struktur på database og strukturering av analysedelen. Dette vil derfor være en kontinuerlig prosess for å analysere de virkningene pluginene har for funksjonaliteten i core. En av de viktige oppgavene er å aggregere data til det andre databaselaget. Dette vil nødvendigvis også være en løpende prosess på linje med analyse av virkningene pluginen har på funksjonaliteten i core.

Analyse av virkningen i database
Det er flere alternativer som i hovedsak dreier seg om hva slags database som skal benyttes, da spesielt i det første laget. Her står valget mellom "in memory" database, eller en vanlig relasjonsdatabase. Dette valget avhenger av hva som er nødvendig i forhold til datamengdene som skal lagres i den første timen før dataene aggregeres videre. Ved benyttelse av "in memory" database vil det være veldig mye raskere å kunne sette inn dataene i databasen. Denne typen database vil også gjøre det mulig å lagre mer data i minne, slik at slitasjen på fysiske disker blir minst mulig, da det vil være kontinuerlige skrive/lese-prosesser mot den.

Totalt sett utgjør dette de forskjellige elementene og alternativene som er diskutert i dette dokumentet, og konkluderer forprosjektet.
Det er mange konkrete detaljer som mangler, men som vil bli viktig å kunne formulere underveis i prosjektet. Disse løsningene beskriver et godt grunnlag for funksjonaliteten i applikasjonen PySniff, og sammen med kravspesifikasjonen viser disse dokumentene hvordan applikasjonen skal