

Cerebrum Komponentarkitektur

Et forslag

Alexander Rødseth
for Cerebrum/USIT

2013

Temaer

Overblikk

Begreper

Frittstående komponenter

Komponentregisteret

Godkjente programmeringsspråk

Godkjente protokoller

Continuous Integration og deployment

Sammendrag

Innføring av ny arkitektur

Overblikk

Cerebrum trenger en arkitektur som:

- Kan oppgraderes uten å måtte ta ned viktige tjenester.
- Ikke er person-avhengig, men som kan videreutvikles av nye ansatte i fremtiden.
- Helst ikke er språk-avhengig, fordi programmeringsspråk, standarder, teknikk og metodikk utvikler seg over årenes løp.
- Er robust, slik at hvis en del av systemet går ned så kan resten fortsette å kjøre. Ideelt sett så bør det også kjøres redundante tjenester av alt, slik at dersom den nye versjonen av en tjeneste går ned, så tar den gamle over.
- Er lett testbar. Tester bør spille en sentral rolle, både for utvikling og av driftshensyn.
- Er modulær, slik at mindre deler av systemet lettere kan testes og forbedres.
- Er modulær, slik at mindre deler av systemet kan ha andre tilganger enn resten. Det kan være interessant å tilby noen tjenester eksternt, mens andre kun må tilbys internt.
- Ikke oppleves som fastlåst, men fleksibel.
- Har en gjennomtenkt rutine for deployment, som ikke belaster drift unødig.
- Som formaliserer leveransen fra utviklere til drift.
- Som kun forholder seg til filer som er sjekket inn i repositories.

Dette dokumentet er et forslag til en mulig arkitektur som kan øke fleksibiliteten, modulariteten og (først og fremst) testbarheten til programvaren i Cerebrum.

Begreper

For dette dokumentet

repo	En versjonskontrollert filbrønn
git-repo	En versjonskontrollert filbrønn som bruker versjonskontrollsystemet git
repoet	Et spesifikt repo
repositories	Flere versjonskontrollerte filbrønner
API	Grensesnitt for å kommunisere med applikasjonen / komponenten. I all hovedsak en liste med konstanter, strukturer, funksjonskall og hva de kan utføre.
directory	En mappe
releasen	En komponent som er ferdig utviklet nok til å slippes / settes i produksjon
slippe	Å gjøre klar til å sette ut i produksjon
komponent	En applikasjon eller del av en applikasjon som er i stand til å kommunisere over nettverk og utføre et begrenset sett med oppgaver.
tjeneste-byggkloss	Det samme som en komponent
tjenesten	Tjenesten som en komponent er ment å tilby
docker container	En fil som inneholder alt man trenger for å kjøre en komponent / tjeneste-byggkloss på en portabel måte. Sammenlignbart med .war filer i «Java-verden» fordi deployment kan foregå ved å kopiere over en fil.
deployment	Å sette ferdig utviklet programvare ut i live (mer eller mindre).
kommer opp	Dersom en tjeneste kommer opp, så er det det samme som at den begynner å kjøre og er tilgjengelig.
går ned	Dersom en tjeneste går ned, så slutter den å kjøre og å være tilgjengelig.
release tags	En måte å peke til visse commits i git, slik at man kan forholde seg til et gitt sett med kildefiler og endringer.
commits	«innsendte» endringer i git
deploybar	Programkode som er mulig å sette ut i live.
redeploy	Å sette ferdig utviklet programvare ut i live en gang til (mer eller mindre).

Frittstående komponenter

Krav til en komponent / «tjeneste-byggekluss»:

- Må ha en README-fil som inneholder (på det minste):
 - Navn på byggeklossen.
 - En kort beskrivelse.
 - Hvordan testene kan kjøres.
 - Avhengigheter (hva som må installeres for å kjøre byggeklossen).
 - Årstall (og evt måned) for når komponenten sist ble endret.
 - Kontaktperson med e-post. («Navn <epost>»)
- Må ha tester som kan kjøres (f.eks.: «make test», «./test.sh», «go test» el.l)
- Må ha et eget git-repo (som hostes ved UiO/USIT, ikke bare lokalt)
- Alle kildefiler må ligge i en mappe som heter «src», eller rett i repoet hvis det bare er én fil
- Komponentene må ha avgrenset funksjonalitet og må kunne kommunisere med andre komponenter via en av de godkjente protokollene (se liste)
- Komponentene må ha API-dokumentasjon for hvilke funksjonskall den tilbyr (over nettverk). Denne må ligge i en mappe som heter «doc» og være så standardisert som mulig (javadoc for java, godoc for Go osv)
- Kan skrives i ett av de godkjente programmeringsspråkene (se liste)

Komponentregisteret

Komponentregisteret er en liste over komponenter, med følgende felter:

- Navn på tjeneste-byggeklossen («navn»)
- URL til git repo, f.eks:
 - `ssh+git://components.usit.uio.no/sensitive_user_server.git`
 - `git://github.com/cerebrum/open_service_client_system.git`
- Kontaktperson (navn <epost>)
- Dersom kontaktpersonen ikke svarer ved henvendelse, så skjer følgende:
 - Kollegaer, gruppeleder eller seksjonssjef for personen kontaktes og man spør om en alternativ kontaktperson
 - Dersom ingen alternativ kontaktperson finnes, så blir komponenten merket som DEPRECATED, gjerne med store bokstaver.

Denne listen kan hostes på <http://components.usit.uio.no/>, der også andre seksjoner kan tilby sine tjeneste-byggeklosser. Git-repo-adressene kan evt. redirectes til git.uio.no.

Listen over godkjente programmeringsspråk og godkjente protokoller legges også på denne siden.

Oppsummering av komponentregisteret:

- En liste over komponenter / tjeneste-byggeklosser, klar til bruk
- En liste over godkjente programmeringsspråk
- En liste over godkjente protokoller

Det vil være naturlig å også inkludere:

- En lenke til en jenkins installasjon, der man kan se byggestatus per komponent
- En lenke til en oversikt over alle komponentene, sortert etter seksjon og gruppe (basert på kontaktpersonen sin e-post, f.eks.)

Godkjente programmeringsspråk

Forslag til godkjente språk ved start

- Python (automatisk PEP8-formatering ved push, typesjekking med dekoratorer oppfordres)
- Go
- C
- Common Lisp (fordi det allerede brukes i gruppa)

Godkjenningsprosedyre

Om prosedyren:

- Skal være lav terskel for å innføre ett nytt språk
- Hovedproblemet med nye programmeringsspråk er bussfaktoren
- Så lenge tester og API-dokumentasjonen er velfungerende og på plass, og funksjonsområdet er begrenset, så skal man kunne bytte ut komponenter i fremtiden uten å kunne programmeringsspråket til fingerspissene
- Fordelen med ganske stor frihet ved valg av programmeringsspråk er:
 - Det er veldig motiverende
 - Det kan tiltrekke seg talentfulle programmerere (ref artikkel på internett, note to self: finn kilden)

Forslag til godkjenningsprosedyre:

- Presentere programmeringsspråket for de andre i løpet av 30 til 60 min
- Godkjennes ved akklamasjon

Godkjente protokoller

Forslag til godkjente protokoller ved start

Kun **JSON-RPC** og **Protocol buffers**

- JSON-RPC er som XML-RPC, bare enklere og mindre utskjelt
- Protocol buffers brukes internt av Google for nesten alle interne RPC protokoller og filformater, ref. <http://code.google.com/p/protobuf/>

Andre alternativer

- Strukturerte, tekst-baserte protokoller:
 - JSON-RPC
 - REST
 - SOAP
- Raskere protokoller, med tanke på binære data eller store datamengder:
 - Thrift (brukes av Apache)
 - BERT (brukes internt hos Github)
 - BSON (brukes av MongoDB)

Godkjenningsprosedyre

Om prosedyren:

- Skal være relativt høy terskel for å innføre en ny protokoll
- Man vil at alle komponentene skal kunne snakke med hverandre, derfor ønsker man å begrense måtene de kan kommunisere seg i mellom

Forslag til godkjenningsprosedyre:

- Holde et 30-60 min langt foredrag for de andre der fordelene med protokollen trekkes frem
- Man bør ta høyde for at protokoller som innføres fordi de er bedre også vil innbære å bytte ut gamle protokoller over tid. Denne kostnaden må taes med i beregningen.
- Det må beskrives hvorfor innføring av protokoll vil være en fordel.
- Godkjenning over e-post, der alle eiere av komponenter (ref komponentregisteret) stemmer over innføringen av protokollen. Snakk med Robert om hvordan avstemningen helst bør foregå. Han kan bestemme hvordan avstemningen helst bør være, hvis i tvil.
- Man bør kunne vise til en prototype der protokollen tas i bruk.

Continuous Integration og deployment

Deployment fra et utviklerperspektiv, to metoder

Med git / «manuelt»:

- Kjør testen, se at alt virker
- Sett merkelapp på releasen ved å gi den et versjonsnummer med «git tag»:
 - `git tag -a 275 -m "Tagging a release"`
- Push git-repoet.
- Ha en avtale med drift på plass om hvordan de skal hente fra dette repoet og kjøre komponenten.
- Kontakt drift dersom den nye tjenesten ikke kommer opp og det ikke er noen tilbakemelding om feil fra drift. Utvikleren skal selv kunne se når den nye tjenesten kommer opp.

Med docker:

- Kjør testen, se at alt virker.
- Bruk «docker commit» og «docker push». Lag en **docker container**.
- Legg docker container filen i riktig directory på serveren.
- Se at tjenesten kommer opp

Deployment fra et driftsperspektiv

Med git / «manuelt»:

- Vent på at en utvikler vil inngå en avtale om å følge med på en komponent
- Følg jevnlig med på visse git repositories for å se om det har dukket opp noen nye release tags.
- Dersom det er nye releases, kjør testene på en test-maskin
- Dersom testene passerer, gå til produksjonsmiljøet, stopp tjenesten, sjekk ut koden og start tjenesten igjen
- Dersom det feiler, stopp tjenesten (om nødvendig), sjekk ut forrige release-tag og start tjenesten igjen.
- Dersom det feiler, innkall utvikleren til et hastemøte der han får ansvaret for å fikse problemet.
- Dersom tjenesten ikke virker, send en e-post til utvikleren med feil-logg og problembeskrivelse.
- Dersom det virker, ikke gi noe tilbakemelding til utvikleren. Utvikleren kontakter drift dersom de ikke hører noe og skal selv kunne se det når den nye tjenesten kommer opp.

Med docker:

- Normalt må ingen ting gjøres. Utvikleren kopierer over en fil til riktig directory på serveren og docker containeren testes og deployes automatisk. Dette kan settes opp med et script som går i loop, cronjobber eller med **jenkins** og vil da være drift sin oppgave å få på plass.
- Utvikleren kan selv rulle tilbake ved å enten logge seg på serveren og endre på docker containeren eller å kopiere over en ny docker container.

Fordeler

Ved å innføre en komponent-basert arkitektur, som skissert ovenfor, så vil man kunne oppnå følgende:

- Ryddige forhold for drift.
- Lykkelige utviklere, med git. Det er også mindre terskel å sjekke inn eksperimentell kode.
- Lykkeligere utviklere med flere alternative programmeringsspråk og godt definerte grensesnitt over nettverk.
- Mindre komponenter som kan byttes ut etter hvert som årene går, uten å måtte bytte ut store systemer i en fei, med all risiko, bekymring og arbeidsmengde det medfører. Dette punktet er «stort».
- Godt isolerte komponenter, i det de sendes over fra utviklere til drift
- God kontroll på release-versjoner og tilbakerulling, ved bruk av tags i git og ved bruk av docker.
- God testing, siden testing er en forutsetning og et krav for å kunne ha frittstående komponenter man kan stole på og som skal kunne endres i fremtiden, da gjerne av helt andre ansatte. Automatisert testing.
- Automatisert deployment, om ønskelig (ikke så farlig som det høres ut, fordi man har gode tester på plass, alle testene passerer og man har gode muligheter for tilbakerulling). Dersom man i tillegg innfører god kontroll av innsending av kode ved hjelp av et verktøy som Gerrit, minskes risikoen ytterligere, og man kan virkelig vurdere å innføre automatisk deployment, med trygghet.
- Bussfaktoren: Med godt isolerte, dokumenterte og testede komponenter er det lettere å la en annen ansatt ta over jobben. Man kan til og med overføre komponenter mellom ansatte, basert på hvilke prosjekter de deltar i.
- Bedre tilgangskontroll på tjenester. Det er mulig å kunne tilby tilgang for eksterne konsulenter for en enkelt komponent, men ikke for hele cerebrum, av sikkerhetsårsaker.

Kort fortalt: mer veldefinerte roller og pakker. Mindre stress og bedre nattesøvn?



Innføring av ny arkitektur

Trinn 1 – kodenavn: «the basics»

1. Prototyp for komponentregisteret opprettes internt hos Cerebrum. [Alexander]
2. Et git repo opprettes på git.uio.no eller internt hos Cerebrum for en svært enkel ping/pong/hello world protobuffer-tjeneste. [Alexander]
3. En release tagges. [Alexander]
4. Drift sjekker ut fra git repoet. [Tobias eller Estephan?]
5. Drift starter manuelt tjenesten. [Tobias eller Estephan?]

Trinn 2 – kodenavn: «deployment bliss»

1. Tjenesten fra trinn 1 pakkes inn som en docker container. [Alexander]
2. Drift oppretter et directory på serveren der docker containere kan kopieres inn. [Tobias eller Estephan?]
3. Drift setter opp et script som har en liste over godkjente filnavn. Scriptet ser kontinuerlig etter nye docker container filer og deployer etter hvert på forhåndsbestemte porter. Scriptet må kjøre tjenesten sine tester før deployment. [Tobias eller Estephan?]
4. Docker containeren til tjenesten fra trinn 1 kopieres over til serveren. [Alexander]
5. Dette skjer så automatisk: tjenesten testes og kjøres.
6. Utvikleren sjekker at tjenesten nå kjører. [Alexander]

Trinn 3 – kodenavn: «testing heaven»

1. Drift setter opp Jenkins til både å teste og deploye tjenestene. Scriptet fra trinn 2 fjernes. [Tobias eller Estephan?]
2. Utvikleren kan se status på testing og deployment via Jenkins sitt web-grensesnitt.
3. Evt. verktøy for å kunne se dekningsgrad av testingen vurderes. [Joakim]

Trinn 4 – kodenavn «all together now» (workshop-dag)

1. Innledes med et møte som forklarer med hva, hvorfor og hvordan.
2. Alle i Cerebrum tester den nye arbeidsflyten og lager en helt enkel tjeneste hver.
3. Drift ser hvordan det fungerer når flere personer tester og deployer.
4. Alle kan følge med i Jenkins og se hvordan det går.
5. Alle bytter eierskap på tjenesten sin og endrer koden til å være avhengig av en annen person sin tjeneste, som en øvelse.
6. Redeployment, alle ser hvordan det går med sine komponenter.

Trinn 5 – kodenavn «policy for the future»

1. All ny utvikling og alle nye prosjekter i Cerebrum skal følge den nye modellen.
2. En fast ukedag, gjerne fredag, settes av til å prioritere å overføre gammel kode til det nye komponent-baserte systemet (men ikke slik at det går ut over noe som haster i særlig grad).

Prosjekter for fremtiden:

- Sette hele uker for alle utviklere med det formål å øke hastigheten på uttrekkingen av kode fra Cerebrum, BOFHD og eValg som kan bli frittstående, testede, API-dokumenterte og lett deploybare komponenter.
- Innføre code-review med Gerrit på en måte som passer inn med alt det overnevnte.
- Innføre den samme komponentbaserte arkitekturen for andre seksjoner ved USIT som måtte være interesserte. Utveksling av erfaring og opplæring. Mye å tjene på å bruke hverandres komponenter, på tvers av seksjoner.
- Innføre utviklingsmetodikk som passer inn med alt det overnevnte.