Tuomas Mäkinen

INTEGRAATIOVÄYLÄN VALVONTAPROTOTYYPPI

Zabbixin automaattinen jakelu ja konfigurointi Ansiblella

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutusohjelma
Joulukuu 2019

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Joulukuu 2019	**Tekijä/tekijät** Tuomas Mäkinen
Koulutusohjelma Tieto- ja viestintätekniikka		
Työn nimi INTEGRAATIOVÄYLÄN VALVONTAPROTOTYYPPI. Zabbixin automaattinen jakelu ja konfigurointi Ansiblella.		
Työn ohjaaja Sakari Männistö		**Sivumäärä** 22+1
Työelämäohjaaja Juho Muuraiskangas		

Opinnäytetyössä määriteltiin, suunniteltiin ja toteutettiin prototyyppi integraatioalustan valvontajärjestelmästä. Tavoitteena oli tuottaa integraatioiden valvonnasta määrittelydokumentti, johon on määritelty valvontatarpeet, sekä luoda automaattinen jakelu Zabbix-valvontajärjestelmälle.

Opinnäytetyö alkaa kuvauksella integraatioista ja integraatioväylästä. Kuvauksen tarkoitus on saada lukijalle kuva siitä, mitä integraatiot ovat ja mistä komponenteista integraatioväylä koostuu. Kuvauksen jälkeen lukijalle esitellään työn toteuttamiseen liittyvät tärkeimmät työkalut ja tekniikat.

Määrittely- ja suunnitteluosuudessa on käyty läpi opinnäytetyöhön kuuluneen määrittelydokumentin kirjoitusprosessi. Käytännön osuus käsittelee Zabbix-valvontajärjestelmän automaattisen jakelun toteuttamista ja käyttöönottoa.

Molemmat opinnäytetyössä asetetut tavoitteet saatiin toteutettua. Tuotin Alfamelle määrittelydokumentin, johon on määritelty keskeisimmät valvontatarpeet. Prototyyppi on asennettu käyttöön ja sitä voidaan edelleen jatkokehittää.

Asiasanat
Ansible, DevOps, Docker, integraatio, ohjelmointi, provisiointi, Zabbix.

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Centria University of Applied Sciences	**Date** December 2019	**Author** Tuomas Mäkinen

Degree programme
Information Technology

Name of thesis
INTEGRATION PLATFORM MONITORING PROTOTYPE. Automatic provisioning and configuration of Zabbix with Ansible.

Instructor Sakari Männistö	**Pages** 22+1

Supervisor
Juho Muuraiskangas

This thesis describes the process of defining specifications, planning and implementing a prototype of a system that monitors an integration platform. The goal of this thesis was to create a specification document of the needs for monitoring integrations and to create an automatic provisioning and configuration pipeline for a monitoring system, Zabbix.

The thesis begins with an introduction to integrations and an integration platform. The purpose of this introduction is to familiarize the reader with the concept of integrations, and the components that construct an integration platform. After the introduction, the reader is presented with the technologies and tools that were essential in implementing the project.

Defining the specifications and project planning are explained in a dedicated chapter. The chapter also includes a description of the process that was used for creating the specification document.

The actual implementation and deployment of the provisioning pipeline is also covered in their own chapters.

Both of the goals for this thesis were achieved. A specification document for Alfame about the fundamental needs for the integration monitoring was produced. The monitoring system prototype is deployed for use, and it can be further developed.

Key words
Ansible, DevOps, Docker, integration, programming, provisioning, Zabbix.

KÄSITTEIDEN MÄÄRITTELY

API	Ohjelmointirajapinta
JSON	JavaScript Object Notation, data- ja tiedostoformaatti, johdettu JavaScriptistä mutta on siitä riippumaton
XML	Extensible Markup Language, ihmis- ja koneluettava merkintäkieli ja tiedostoformaatti
YAML/YML	YAML Ain't Markup Language, ihmisluettava datan serialisaatiokieli, käytetään usein konfiguraatiotiedostoissa
TLS/SSL	Transport Layer Security/Secure Sockets Layer, varmenteisiin perustuva salausprotokolla verkkoliikenteen suojaamiseen
HTTP	Hypertext Transfer Protocol, selainten ja www-palvelimien tiedonsiirtoprotokolla
HTTPS	Hypertext Transfer Protocol Secure, TLS/SSL-protokollalla suojattu versio HTTP:sta
FTP	File Transfer Protocol, verkkoprotokolla tiedostojen siirtoon palvelimien välillä
SSH	Secure Shell, salattu tietoliikenneprotokolla, jota käytetään palvelimien etäyhteyksiin
SFTP	SSH/Secure File Transfer Protocol, SSH:lla suojattu versio FTP-protokollasta
REST	Representational state transfer, www-sovelluksien ohjelmointirajapintojen arkkitehtuurimalli
RESTful	Ohjelmointirajapintoja, jotka käyttävät REST-arkkitehtuuria, kutsutaan RESTful-rajapinnoiksi
RAML	RESTful API Modeling Language, YAML-pohjainen kieli, jolla kuvataan RESTful-rajapintoja
JMS	Java Message Service, viestipohjainen rajapinta Java-sovellusten väliseen kommunikointiin
Web service	Verkkopalvelu, käytännössä www-ohjelmointirajapintaprotokolla, johon kuuluu kolme World Wide Web Consortiumin määrittelemää standardia: SOAP, WSDL ja UDDI

SOAP	Simple Object Access Protocol, XML-perustainen HTTP:n yli toimiva viestipohjainen tietoliikenneprotokolla
WSDL	Web Service Description Language, XML-pohjainen verkkopalvelujen kuvaamiseen tarkoitettu kieli
UDDI	Universal Description Discovery and Integration. XML-pohjainen standardi verkkopalvelujen kuvaamiseen, julkaisuun ja etsimiseen.
JDBC	Java Database Connectivity, Javan tietokantaohjelmointirajapinta, käytetään tietokantayhteyksiin
BPMN	Business Process Model and Notation, visuaalinen liiketoimintaprosessien kuvaustyökalu

TIIVISTELMÄ
ABSTRACT
KÄSITTEIDEN MÄÄRITTELY
SISÄLLYS

KUVAT

1 JOHDANTO

Tämä opinnäytetyö on toteutettu Alfame Systems Oy:lle syystalvella 2019. Työn tarkoituksena on kehittää Alfamen asiakkailleen toteuttamien integraatioiden valvontaa. Valvonnan kehittämisellä pyritään parantamaan näkyvyyttä asiakkaiden verkoissa toimiviin sovelluksiin ja helpottamaan ylläpitotyötä tuomalla siihen ennakoitavuutta. Valvonnan kehittämisen ohella pyritään kehittämään myös statistiikkadatan keräämistä.

Opinnäytetyön tavoitteeksi otettiin integraatioiden valvonnan tarpeiden kartoittamisen ja määrittelyn sekä automaattisen jakelun luomisen Zabbixille. Työssä haluttiin lähinnä evaluoida Zabbixin soveltuvuutta Alfamen tarpeisiin. Varsinaista integraatioiden valvontaa ei vielä toteutettu, vaan sitä voidaan tarkastella ja kokeilla myöhemmin.

Toisessa luvussa esittelen integraatiosovellusten periaatetta ja arkkitehtuuria. Kolmannessa luvussa käydään läpi osa opinnäytetyössä käytettyjä tekniikoita ja työkaluja. Neljännessä luvussa kerron valvontatarpeiden määrittelystä. Viides luku käsittelee jakeluputken rakentamista Ansiblella ja kuudes luku Zabbixin käyttöönottoa. Lopuksi esitän omia ajatuksiani projektin kulusta ja tavoitteiden saavuttamisesta.

2 INTEGRAATIOT JA INTEGRAATIOVÄYLÄ

Nykänen (2015) määrittelee integraatiot käytänteiksi, ratkaisuiksi ja prosesseiksi, joiden tuloksena järjestelmät, tietokannat, sovellukset ja laitteistot saadaan jakamaan tietonsa esteettömästi keskenään sekä muodostamaan yhtenäisen tietojärjestelmän. Teknisestä näkökulmasta integraatiot voi jaotella kolmeen eri tasoon: datatason, viestitason ja prosessitason integraatioihin. Datatason integraatiossa järjestelmät saatetaan käyttämään samaa tietokantaa tai muuta tietovarastoa. Viestitason integraatiossa tietoa siirretään ennalta määritetyillä sanomarakenteilla ja -sisällöillä. Prosessitason integraatio toteuttaa organisaation liiketoimintaprosessia osittain tai kokonaan, suorittaen tehtäviä ja analysoiden dataa ennalta määritetyn prosessikaavion mukaan. Nykänen (2015) jaottelee integraatiot myös tieto-, palvelu-, prosessi- ja käyttäjäpohjaisiin integraatioihin. Tietopohjainen perustuu tiedonvaihtoon eli tietokantoihin, viesteihin ja rajapintoihin. Palvelupohjaisessa integraatiossa järjestelmillä on yhteisiä toimintalogiikoita, metodeja sekä sovelluspalveluja. Prosessipohjainen integraatio perustuu ennalta määriteltyihin ja keskitetysti ylläpidettyihin liiketoimintaprosesseihin, joiden kautta siirretään tietoa ja kutsutaan aliprosesseja. Käyttäjäpohjaisessa integraatiossa pyritään luomaan käyttäjälle toimiva järjestelmäkokonaisuus. (Nykänen 2015.)

Integraatiot voivat olla muihin sovelluksiin tai järjestelmiin sisäänrakennettuja, tai ne voidaan toteuttaa erillisinä sovelluksina. Ensimmäiset erillisenä sovelluksena toteutettavat integraatiot, integraatiosovellukset, kehitettiin jo 1990-luvulla. Microsoft julkaisi BizTalkin ensimmäisen version vuoden 2000 lopussa, jolloin digitalisaation ensimmäinen vaihe oli monessa organisaatiossa ajankohtainen asia ja uusia tietojärjestelmiä otettiin käyttöön. Ryhänen (2019) jakaa integraatiot kolmeen ryhmään: point-to-point-, on-premise- ja pilvi-integraatioihin. Point-to-point -integraatiot toteutetaan yleensä suoraan järjestelmien ohjelmakoodiin. Tämän tyyppisiä integraatioita on vaikea ylläpitää tai valvoa, eikä niillä ole yhteneväistä arkkitehtuuria. On-premise -integraatioilla Ryhänen (2019) viittaa asiakkaan sisäverkkoon asennettaviin integraatiosovelluksiin, jotka usein toteutetaan alusta-arkkitehtuuria, integraatioista puhuttaessa väyläarkkitehtuuria, hyödyntäen. Väyläarkkitehtuurissa kaiken keskellä on integraatioväyläksi kutsuttu sovellus tai järjestelmä, jonka kautta integroitavien järjestelmien tieto kulkee. Väyläarkkitehtuurissa on useita etuja, kuten parempi ylläpidettävyys ja valvonta. Pilvi-integraatiot eivät teknisesti juuri eroa on-premise-integraatioista. Pilvi-integraatio voi olla tavallinen integraatioväylä asennettuna pilvessä olevaan virtuaalipalvelimeen tai perustua moderniin palvelupohjaiseen serverless-malliin. (Ryhänen 2019.)

Alfamen toteuttamat integraatiot ovat pääosin viesti- ja prosessitason on-premise-integraatioita, väylä-arkkitehtuuria noudattaen. Integraatiot toteutetaan lähes poikkeuksetta MuleSoftin Mule ESB (Mule) -integraatioalustalla. Mule on kevyt Java-pohjainen integraatioalusta, joka tukee useimpia tarvittavia tekniikoita suoraan. Tuettuja tekniikoita ovat muun muassa JMS, Web services, JDBC ja HTTP. (MuleSoft 2019.) Mule tukee myös REST APIen luontia omalla APIkit-työkalullaan, joka generoi APIn rungon RAML-tiedoston pohjalta. Mulen kaveriksi Alfamella on liitetty Java-pohjainen, avoimen lähdekoodin prosessimoottori, Activiti. Activitilla voidaan suorittaa BPMN 2.0 -prosessikuvaimia ja liittää niihin toiminnallisuutta esim. kutsumalla ulkoisia palveluita.

Integraatioväylä vaatii toimiakseen myös muita ohjelmistoja, kuten tietokannan ja palvelinohjelmiston. Alfamella suosituimmat tekniikat ovat MySQL:stä haarautunut MariaDB-tietokantasovellus sekä Nginx-palvelinohjelmisto. Integraatioväylän palvelinympäristöt vaihtelevat yksinkertaisesta hyvinkin monimutkaiseen kokonaisuuteen. Yksinkertaisimmillaan väylä voidaan asentaa yhdelle palvelimelle, jolloin se sisältää tietokannan, palvelinohjelmiston ja Mule-sovelluksen. Tietokantaakaan ei välttämättä tarvita, jos Mule-sovellus ei käytä Activitia. Toisessa ääripäässä voisi olla sitten taas kahdennettu palvelinympäristö, jossa kummassakin salissa on kaksi Mule-sovelluspalvelinta, yksi kuormanjakopalvelin sekä salien välille jaettu kolme klusteroitua tietokantapalvelinta.

3 TEKNOLOGIAT

Tässä luvussa esitellään työn tekemiseen käytetyt keskeiset työkalut ja teknologiat. Osa työkaluista, kuten Ansible ja Docker, ovat olleet Alfamella käytössä pitkään, ja ne olivat minullekin tuttuja ennen projektin aloitusta. Alla olevien tekniikoiden ohella käytin työkaluina lähinnä tekstieditoria, tarkemmin sanottuna JetBrainsin IntelliJ IDEA:a sekä komentorivityökalua.

3.1 Ansible

Ansible on yksinkertainen ja helppokäyttöinen IT-automaatiotyökalu. Ansiblella voidaan automatisoida mm. pilviprovisiointi, konfiguraationhallinta, sovellusten jakelu ja käyttöönotto sekä palveluiden välinen orkestraatio. Ansible on suunniteltu alusta lähtien monitasoisia jakeluita varten, ja sen toimintaperiaate onkin mallintaa koko IT-infrastruktuuri kuvailemalla, kuinka eri järjestelmät vaikuttavat toisiinsa, eikä vain hallita yhtä järjestelmää kerrallaan. Ansible ei käytä agentteja eikä vaadi erillistä turvallisuusinfrastruktuuria, ja se käyttää konfiguraatioihinsa yksinkertaista YAML-merkintäkieltä, jolla automaatiotehtävät ja infrastruktuuri kuvataan. (Ansible 2019.)

3.1.1 Arkkitehtuuri

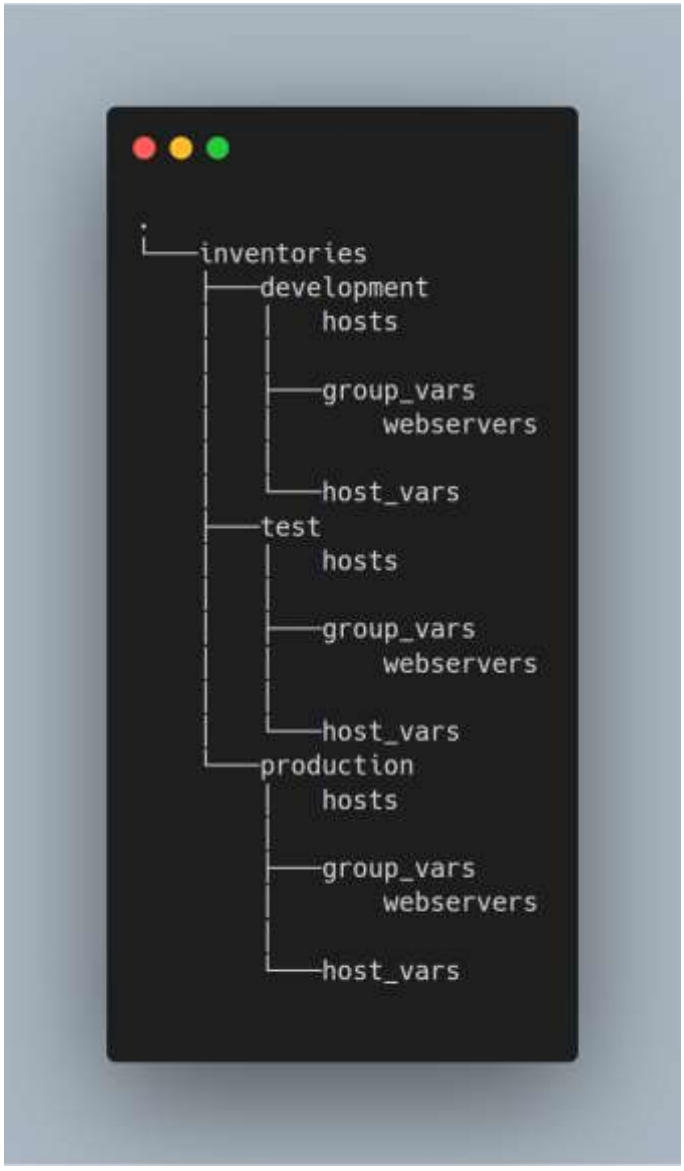
Ansible ottaa SFTP-yhteyden kohdepalvelimeen ja lataa sinne pieniä apuohjelmia, joita kutsutaan Ansiblen moduuleiksi. Moduulit on yleensä kirjoitettu mallintamaan kohdejärjestelmän haluttua tilaa. Näitä apuohjelmia sitten suoritetaan, yleensä SSH-yhteyden kautta, ja ne poistetaan, kun ohjelman suoritus päättyy. Moduulikirjasto voi sijaita millä tahansa laitteella, ja palvelimia, palveluprosesseja tai tietokantoja ei tarvita. Tyypilliset tarvittavat työkalut Ansiblen käyttöön ovat komentorivityökalu ja tekstinkäsittelyohjelma. (Ansible 2019.)

3.1.2 Inventory

Inventory-tiedosto on tavallinen tekstitiedosto, johon kuvataan, mitä palvelimia halutaan hallita. Palvelimet jaotellaan ryhmiin, jotka yleensä kuvaavat. minkä tyyppisistä palvelimista on kyse ts. mitä ohjelmistoja näille palvelimille halutaan asentaa. Ansible voidaan myös integroida muihin palveluihin, joilla palvelininfrastruktuuri on kuvattu, esim. EC2, Rackspace tai OpenStack. Inventory-tiedostossa ryhmän nimi kirjoitetaan ensin hakasulkujen sisälle, ja ryhmän nimen alle listataan palvelimen tai palvelimien isäntänimi tai IP-osoite. (Ansible 2019.) Ansiblen inventory-tiedosto voi näyttää esimerkiksi tältä (KUVA 1).

KUVA 1. Esimerkki Ansiblen inventory-tiedostosta.

Palvelimiin ja palvelinryhmiin voidaan liittää erilaisia muuttujia tekstitiedostoissa. Nämä tiedostot ovat yleensä ”group_vars”- tai ”host_vars”-nimisissä kansiossa, ryhmän tai palvelimen mukaan nimettyinä. (Ansible 2019.) Palvelinympäristöt, kuten vaikka testi ja tuotanto, on myös hyvä erottaa omiin kansioihinsa, joiden alla inventory-tiedosto ja muuttujat sijaitsevat. Inventory-tiedoston tiedostonimi voi olla esimerkiksi ”hosts”. Kuvassa 2 on esimerkki kansiorakenteesta.



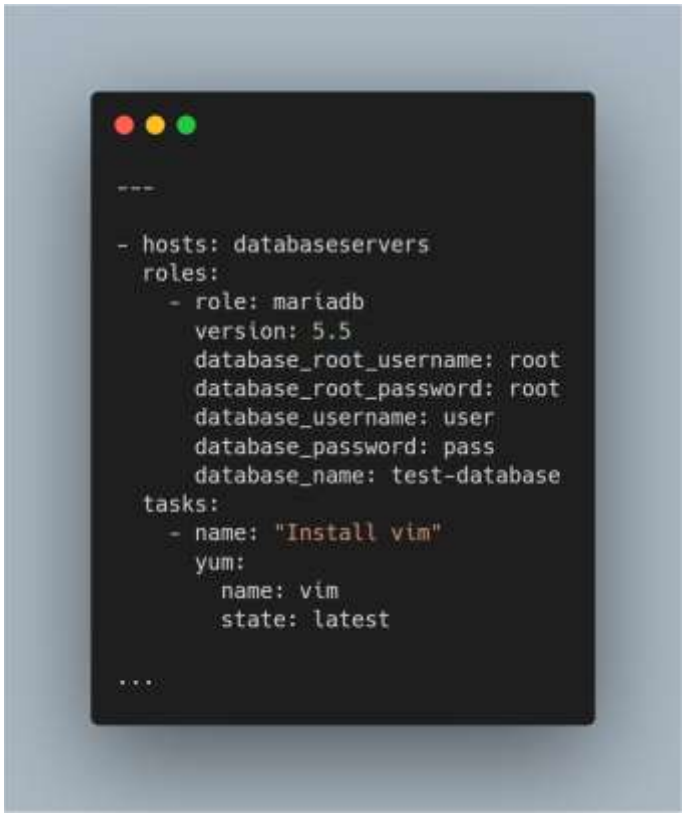
KUVA 2. Esimerkki Ansiblen kansiorakenteesta.

3.1.3 Playbook

Siinä missä inventory-tiedostoissa kuvattiin palvelinryhmiä, niihin kuuluvia palvelimia ja näiden kahden erilaisia muuttujia, Ansiblen playbook-tiedostoissa kuvataan, mitä näiden palvelimien halutaan tekevän. Playbookit usein ovatkin Ansiblen kiinnostavin osa-alue. Playbookien kieliasu on haluttu tehdä mahdollisimman yksinkertaiseksi, jotta niitä ymmärtäisi vielä vuosia myöhemmin. Turhaa ja hankaloittavaa syntaksia ja ominaisuuksia on karsittu minimiinsä. (Ansible 2019.)

Ansiblen playbookien toiminta rakentuu toimien (engl. tasks) ja roolien (engl. roles) varaan. Toimet voivat olla yksinkertaisia tiedostonkopioimistehtäviä tai Shell-skriptin ajoja, mutta myös vaikeammat tehtävät onnistuvat, esimerkiksi käyttöjärjestelmän paketinhallinnasta jonkin ohjelman asentaminen tai tietokannan luominen tietokantasovellukseen. Pääosassa toimissa ovat Ansiblen moduulit, jotka ovat sisäänrakennettuja Ansiblen jakeluversioon. Moduuleja voi myös kehittää itse. Roolit ovat kokoelmia toimista, jotka muodostavat yleensä jonkinlaisen loogisen kokonaisuuden. Esimerkkinä roolista ja sen toiminnasta voisi olla rooli, joka ensin asentaa MariaDB-tietokantasovelluksen, vaihtaa sen pääkäyttäjän salasanan, luo uuden käyttäjän tietokantasovellukseen, luo uuden tietokannan ja myöntää aiemmin luodulle käyttäjälle luku- ja kirjoitusoikeuden uuteen tietokantaan. Roolille voi antaa parametrejä, esimerkkitapauksessa esim. tietokantasovelluksen versionumero, pääkäyttäjän käyttäjänimi ja salasana, uuden, luotavan käyttäjän käyttäjänimi ja salasana, tietokannan nimi jne.

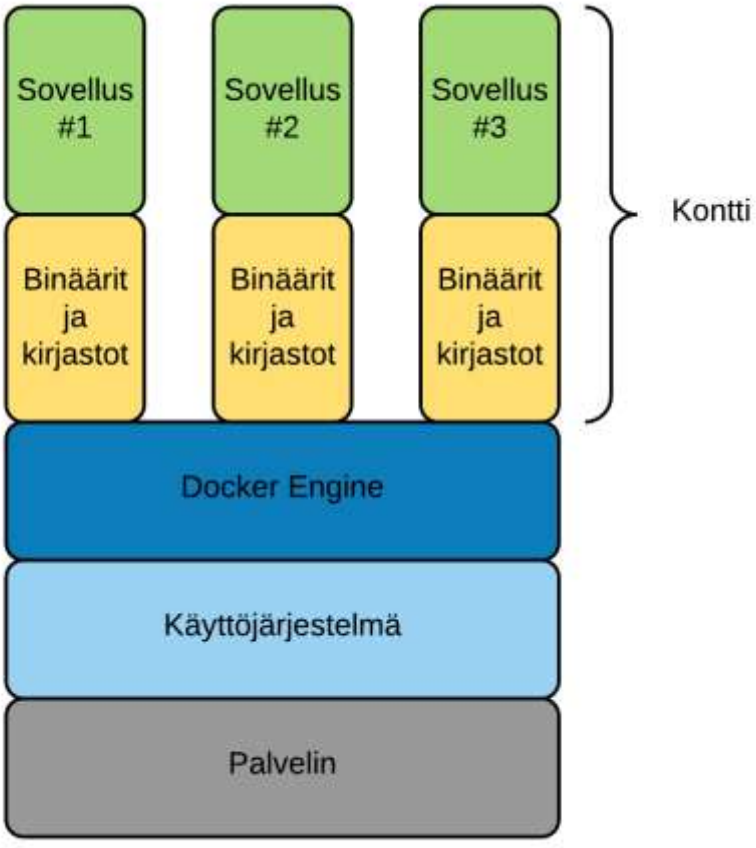
Playbookissa on ylimpänä tasona hosts-muuttuja, jossa määritellään, mille palvelimelle tai palvelinryhmälle kyseisessä lohkossa kuvata toimintaa. Hosts-muuttujan alle määritellään, mitä toimia ja rooleja kyseiseen palvelimeen halutaan suorittaa (KUVA 3). Kyseiseen lohkoon voidaan myös esim. tuoda jostain toisesta palvelinryhmästä muuttuja tai määritellä yhteyden tyyppi, kuten lokaali yhteys SSH-yhteyden sijaan, jos halutaan hallita sitä palvelinta, jossa playbook suoritetaan.



KUVA 3. Esimerkki Ansiblen playbook-tiedostosta.

3.2 Docker

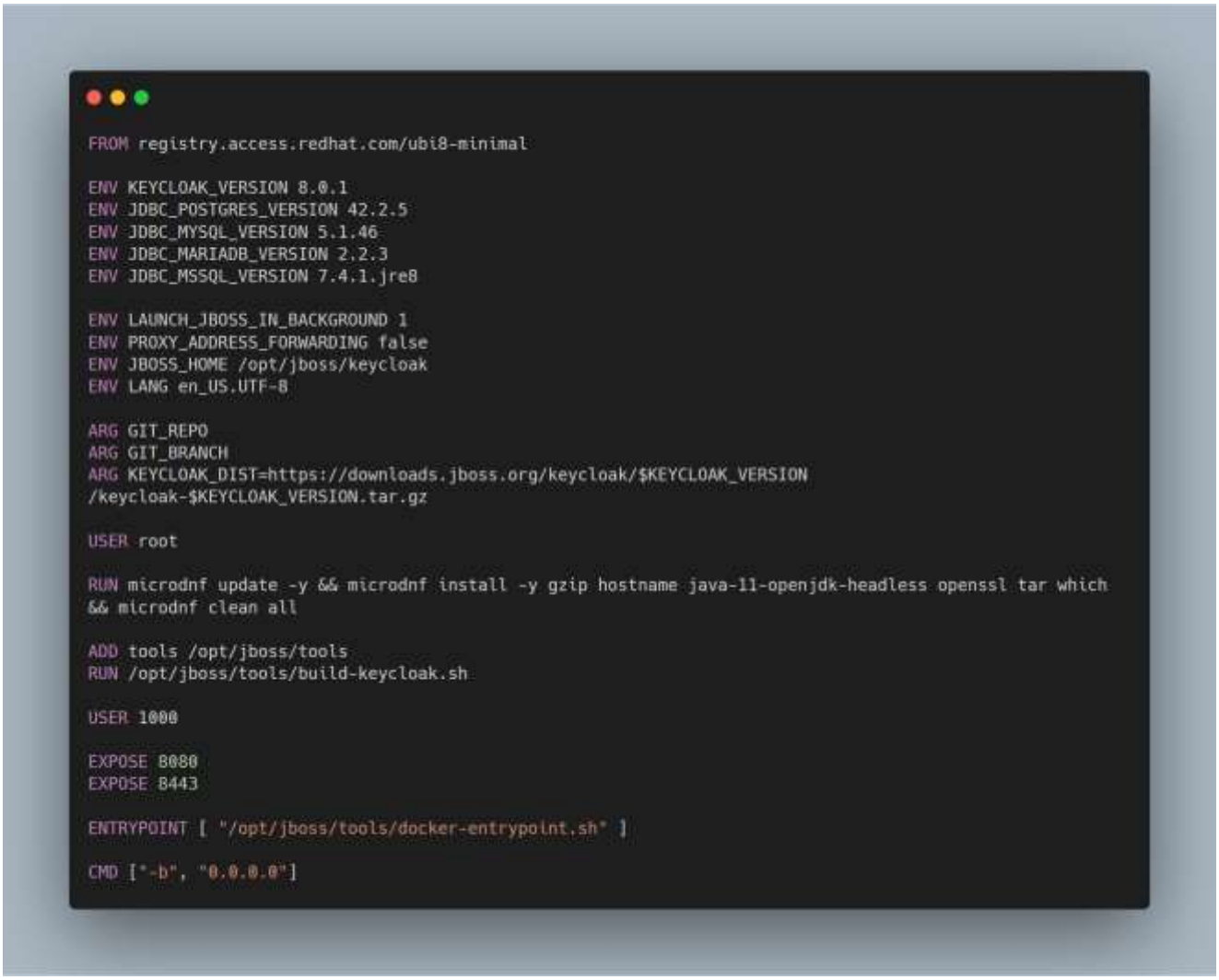
Docker on kokoelma työkaluja, joilla sovelluksia voidaan paketoida niin sanottuihin kontteihin. Toisin kuin virtuaalikone, jossa virtualisoidaan tietokoneen laitteisto, Docker-kontissa virtualisoidaan vain käyttöjärjestelmän ns. ”käyttäjätila”, muistialue, jossa sovellukset ajetaan. Samalla palvelimella ajossa olevat kontit jakavat käyttöjärjestelmän ytimen. Konttia luotaessa ainoastaan sovelluksen vaatimat binäärit ja kirjastot luodaan tyhjästä. Tällä tavoin samalla isäntäpalvelimella voidaan ajaa toisistaan riippumattomia sovelluksia turvallisesti ja kevyesti. (Kasireddy 2016.) Docker-alustan ja -konttien arkkitehtuuri on kuvattuna kuvassa 4.



KUVA 4. Docker-alusta ja kontit käyttöjärjestelmän päällä (mukaillen Kasireddy 2016)

3.2.1 Docker image

Docker-kontit luodaan Docker image -tiedostojen perusteella. Imaget taas on kuvattu Dockerfile-tiedostoilla, joka on käytännössä lista ohjeita imagen koostamiseen. Usein otetaan pohjaksi jokin valmis käyttöjärjestelmän Docker image, johon voidaan asentaa lisää paketteja, antaa ympäristömuuttujia, kopioida tiedostoja, avata portteja sekä kertoa, mitä komentoa valmiin kontin tulisi ajaa käynnistyessään (KUVA 5). Kun Dockerfile-tiedosto kootaan imageksi, se sisältää vain lukuoikeudellisen tiedostojärjestelmän. Kun imagen pohjalta käynnistetään kontti, sen päälle asennetaan luku- ja kirjoitusoikeudellinen tiedostojärjestelmä. Docker luo kontille myös verkkoliitännän, antaa sille IP-osoitteen ja alkaa suorittamaan imagelle määriteltyä komentoa. (Kasireddy 2016.)

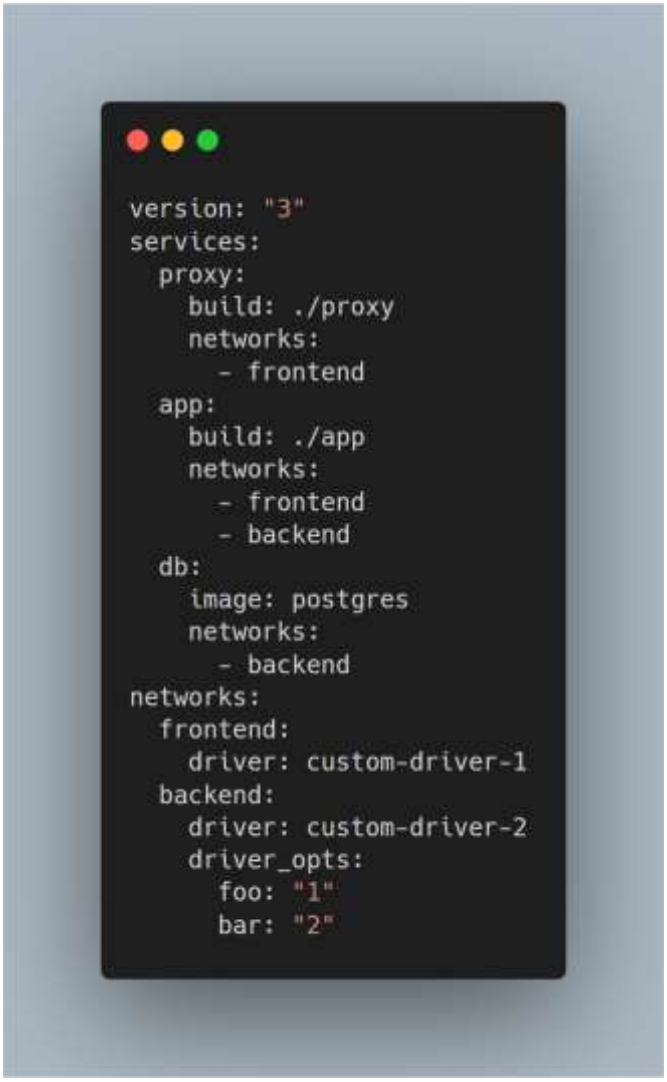


```
FROM registry.access.redhat.com/ubi8-minimal

ENV KEYCLOAK_VERSION 8.0.1
ENV JDBC_POSTGRES_VERSION 42.2.5
ENV JDBC_MYSQL_VERSION 5.1.46
ENV JDBC_MARIADB_VERSION 2.2.3
ENV JDBC_MSSQL_VERSION 7.4.1.jre8

ENV LAUNCH_JBOSS_IN_BACKGROUND 1
ENV PROXY_ADDRESS_FORWARDING false
ENV JBOSS_HOME /opt/jboss/keycloak
ENV LANG en_US.UTF-8

ARG GIT_REPO
ARG GIT_BRANCH
ARG KEYCLOAK_DIST=https://downloads.jboss.org/keycloak/$KEYCLOAK_VERSION
/keycloak-$KEYCLOAK_VERSION.tar.gz

USER root

RUN microdnf update -y && microdnf install -y gzip hostname java-11-openjdk-headless openssl tar which
&& microdnf clean all

ADD tools /opt/jboss/tools
RUN /opt/jboss/tools/build-keycloak.sh

USER 1000

EXPOSE 8080
EXPOSE 8443

ENTRYPOINT [ "/opt/jboss/tools/docker-entrypoint.sh" ]

CMD ["-b", "0.0.0.0"]
```

KUVA 5. Esimerkki Dockerfile-tiedostosta.

3.2.2 Docker Compose

Docker Compose on työkalu, jolla kuvataan ja ajetaan Docker-sovelluksia, jotka koostuvat useammasta kontista eli palvelusta. Palvelut kuvataan YAML-tiedostossa, jolloin kaikki sovellukseen liittyvät palvelut käynnistyvät yhdellä komennolla. Compose-tiedostossa voidaan kuvata, miten eri palveluiden halutaan linkittyvän toisiinsa, niille voidaan luoda omia yksityisverkkoja, määritellä uudelleenkäynnistymisehtoja sekä paljon muuta palveluiden kannalta tärkeää. Compose-tiedostossa voidaan myös määritellä konttien sisälle reitittyvien ympäristömuuttujien arvoja. Kuvassa 6 on esimerkki Compose-tiedostosta. (Docker 2019a; Docker 2019b.)



KUVA 6. Esimerkki Docker Compose -tiedostosta (Mukaillen Docker 2019a).

3.3 Zabbix

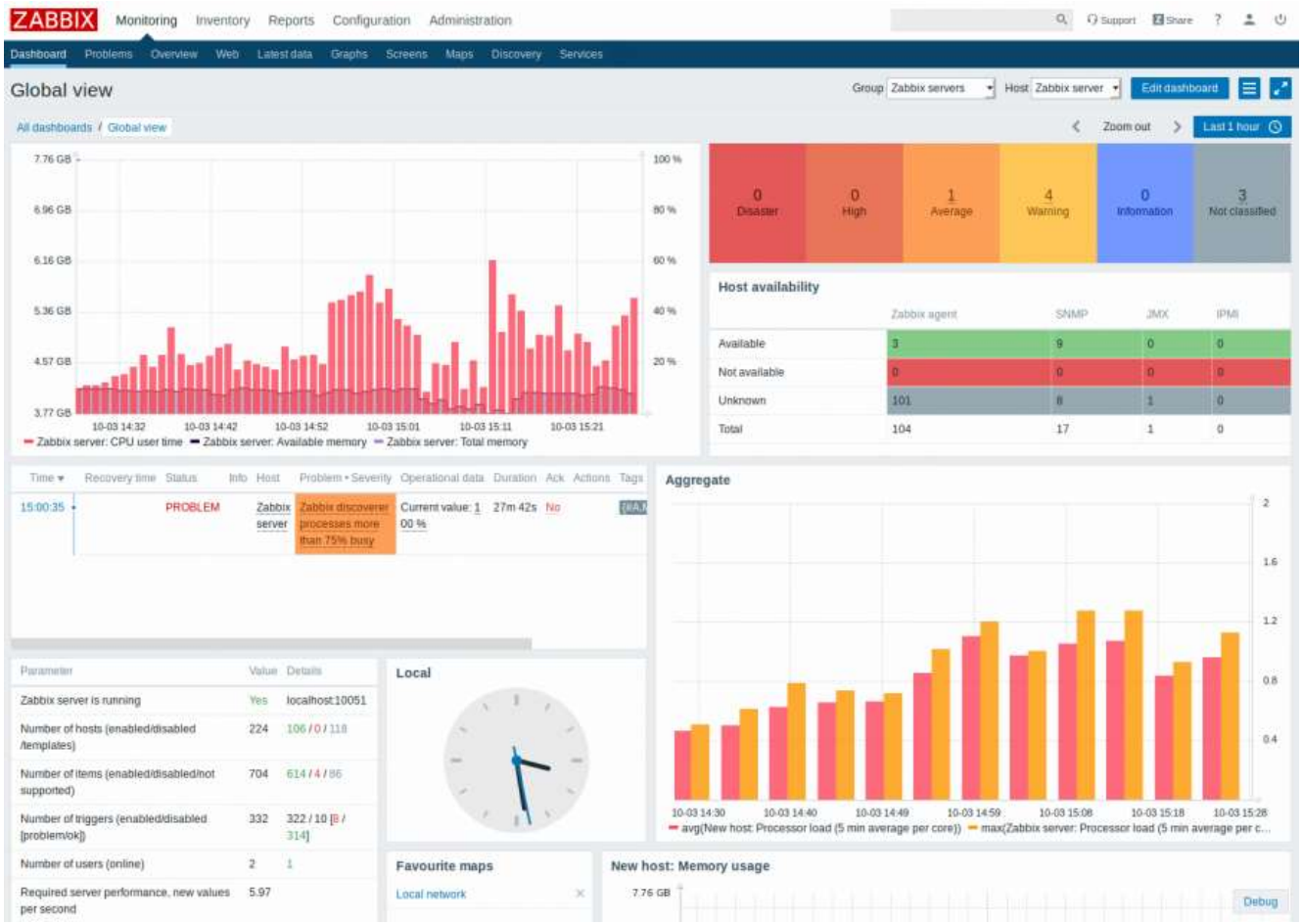
Zabbix on yritysluokan avoimen lähdekoodin valvontaratkaisu. Pääperiaatteena Zabbixissa on verkon ja siihen kytkettyjen laitteiden valvonta niistä kerätyn tiedon perusteella. Zabbix sisältää myös valmiudet kerätyn datan visualisointiin ja raporttien muodostamiseen. Datan ja raporttien pohjalta voidaan tehdä esimerkiksi kapasiteettisuunnittelua. Dataan, raportteihin ja statistiikkatietoon pääsee käsiksi selainpohjaisen käyttöliittymän, Zabbix webin, kautta. (Zabbix 2019.) Muita Zabbixin komponentteja ovat mm. Zabbix server ja Zabbix agent. Zabbix server vaatii toimiakseen myös tietokannan, johon kerätty data kirjoitetaan.

3.3.1 Zabbix server

Zabbix server on Zabbixin keskusprosessi ja tärkein yksittäinen komponentti. Zabbix server joko tekee datankeruukutsut agenteille tai vaihtoehtoisesti asettaa agentille listan valvontakohteita, joihin se odottaa agentilta vastausta. Tästä erosta kerrottu tarkemmin luvussa 3.3.3. Server myös tallentaa kerätyn valvontadatan tietokantaan, tarkistaa ylittyvätkö tai alittuvatko määritellyt turvarajat, lähettää hälytykset jne. Server voidaan myös konfiguroida tarkistamaan www-sivujen ja muiden verkkopalveluiden saatavuutta. Zabbix server voidaan asentaa useimpiin Unixin kaltaisiin käyttöjärjestelmiin, ja se on testattu mm. Linuxilla, Solariksella, Free- ja OpenBSD:lla sekä macOS:lla. Zabbix serverin kanssa yhteensopivat tietokantaohjelmistot ovat MySQL, Oracle, PostgreSQL, TimescaleDB sekä IBM DB2.

3.3.2 Zabbix web

Zabbixin selainpohjainen käyttöliittymä on toteutettu PHP:llä. Sen kautta voidaan hallita kaikkea Zabbixin toimintaan liittyvää. Kirjautumisen jälkeen aloitussivuna toimii käyttäjäkohtainen, muokattava kojelauta (engl. dashboard), johon voi lisätä erilaisia pienoisohjelmia (engl. widget), esim. kellon, serverin muodostaman graafin tai yhteenvedon palvelimien saatavuudesta (KUVA 7). (Zabbix 2019.)



KUVA 7. Esimerkki Zabbixin käyttöliittymän kojelaudasta. (Zabbix 2019)

Käyttöliittymän kautta voidaan tehdä uusia valvontamalleja (engl. template), joissa määritellään. miten ja mitä valvontadataa halutaan kerätä, lisätä uusia valvontakohteita (engl. host) sekä kiinnittää malleja haluttuihin kohteisiin. Kun kohteelle on määritelty malli, pyrkii Zabbix server keräämään mallissa määritetyn datan kohteesta, mallissa määritellyllä tavalla. Näitä tapoja ovat muun muassa suora HTTP-kutsu tai yhteys Zabbix agentin kautta.

Käyttöliittymän kautta hoidetaan myös Zabbix serverin konfigurointi. Serveriin voidaan konfiguroida esimerkiksi miten hälytykset toimitetaan, tuettuina ovat mm. sähköposti, tekstiviestit, web hookit ja käyttäjän määrittelemät skriptit. Käyttäjät, käyttäjäryhmät ja tunnistautuminen voidaan myös konfiguroida käyttöliittymän kautta. Tuettuja tunnistautumistapoja ovat HTTP ja LDAP.

3.3.3 Zabbix agent

Zabbix agent asennetaan valvottavalle kohdepalvelimelle, jossa sitä suoritetaan taustaprosessina. Agent kerää tietoa paikallisesta järjestelmästä ja lähettää ne takaisin Zabbix serverille. Agent voi toimia passiivisesti tai aktiivisesti. Passiivisessa tilassa agent odottaa Zabbix serverin ottavan siihen yhteyden ja antavan sille listan kerättävästä datasta, jolloin agentti kerää datan ja lähettää sen takaisin serverille. Aktiivisessa tilassa Zabbix server asettaa listan agentin luettavaksi, ja agentti ottaa yhteyden serveriin, lukee listan, kerää datan ja lähettää sen takaisin. Aktiivinen tila mahdollistaa Zabbixin toiminnan myös silloin, kun valvottava kohde on palomuurin takana tai yksityisverkossa, jolloin siihen ei saa suoraa yhteyttä. Zabbix agent tukee samoja käyttöjärjestelmiä kuin Zabbix server, sekä niiden lisäksi Windowsia. (Zabbix 2019.)

4 TYÖN MÄÄRITTELY JA SUUNNITTELU

Työn ensimmäinen vaihe oli määritellä asiakkaan, tässä tapauksessa opinnäytetyön työelämäohjaajan kanssa, mitkä toiminnallisuudet ja ominaisuudet valvontatyökalun tulisi sisältää. Ensimmäinen ajatuksemme oli ohjelmoida itse alusta asti uusi Java-komponentti, joka voitaisiin asentaa Mule ESB -alustan kylkeen ja joka keräisi tietoa alustan toiminnasta ja syöttäisi sitä johonkin keskitettyyn valvontajärjestelmään, jota toteutettiin toisen työntekijän opinnäytetyönä. Määrittely kuitenkin pyrittiin tekemään mahdollisimman riippumattomaksi toteutustavasta. Vaikka päädyimmekin kokeilemaan Zabbixia, joka sisältää niin valvonta-agentin kuin keskitetyn valvontajärjestelmänkin, kirjoittamaani määrittelydokumenttia voidaan silti hyödyntää. Tarpeet eivät muutu, vaikka toteutustapa ei olekaan se, joka kirjoittamisen aikana oli suunnitteilla.

Määrittelymetodin pääperiaate on lähteä liikkeelle liiketoimintamallista tai -prosessista, jota halutaan lähteä parantamaan. Prosessista otetaan yksi vaihe, josta määritetään ominaisuudet, jotka puretaan vaatimuksiksi. Nämä vaatimukset puretaan edelleen käyttötapauksiin, joista voidaan määritellä tarvittavat rajapinnat. Rajapinnat voidaan purkaa toiminnallisuuksiin.

Käytin määrittelyyn Alfamen sisäistä määritysdokumenttipohjaa. Dokumenttipohjaan on valmiiksi mietitty ja alustettu määrittelyn eri vaiheet ja tasot. Dokumentti alkaa johdannosta, jossa esitellään asiakkaan järjestelmät, ongelmat ja tarpeet. Tarvekuvaus on erityisen tärkeä osa, joka on syytä tehdä huolellisesti ja perusteellisesti. Nämä tarpeet toimivat pohjana koko määrittelytyölle. Dokumentissa määritellään myös asiakkaan liiketoimintaprosessit, joita määritys koskettaa. Järjestelmän käsitteet on kuvattu samoin kuin tässä opinnäytetyössä, minkä lisäksi niistä koostetaan käsitemalli, josta selviää, missä suhteessa eri käsitteet ovat toisiinsa. Käsitemalli mallinnetaan usein UML-luokkakaaviona.

Kirjoitin ja parantelin määrittelydokumentin sisältöä asiakkaan kanssa käytyjen palaverien pohjalta. Dokumenttiin kirjattiin mm. järjestelmän käyttäjät, rajat, toiminnot ja vaatimukset. Käyttäjien kuvauksiin kirjattiin käyttäjän tehtävä, osaamistaso ja määrä nyt sekä tulevaisuudessa. Tunnistimme käyttäjäksi kehittäjän, ylläpitäjän ja agentin. Kehittäjä on Alfamella työskentelevä henkilö, joka ottaa valvontasovelluksen käyttöön. Ylläpitäjä on valvontasovelluksen käyttäjä. Agentti tarkoittaa kohdepalvelimelle asennettavaa tiedonkeruuohjelmaa. Se ei ole varsinainen ihmiskäyttäjä, mutta on helpompaa kuvata järjestelmän toiminnallisuutta, kun se kuvataan käyttäjänä. Näin saadaan tehtyä käyttötapauskuvaukset sen suorittamista toiminnoista.

Järjestelmän rajoista tehtiin havainnollistava komponenttikuva. Tarpeena oli kuvata sellainen ympäristö, jossa asiakkaan järjestelmät sijaitsevat esim. yksityisverkossa. Tällöin ei voida muodostaa yhteyttä keskitetystä valvontajärjestelmästä agenttiin, vaan yhteyksien pitää tulla vain yksityisverkosta ulospäin, agentilta valvontajärjestelmään.

Toiminnot kirjattiin toimintoryhmittäin. Toimintoryhmästä tehtiin käyttötapauskaavio, ja jokaiselle toimintoryhmään kuuluvalle toiminnolle kirjattiin siihen liittyvät käyttötapaukset. Toiminnon yhteyteen kirjattiin kuvaus ja prioriteetti. Käyttötapauksista kirjattiin kuvaus, käyttäjät, alkuehto, tapahtumien normaali kulku, vaihtoehtoiset tapahtumien kulut, erikoisvaatimukset sekä loppuehto. Käyttötapauksiksi kirjattiin erilaisten integraatioalustan toiminallisuuksiin liittyvät informaation keruut sekä niiden lähettäminen. Hiljaisuudenvalvonta oli yksi tärkeimmistä tunnistetuista toiminnoista, joka valvontaan täytyy sisällyttää.

Dokumenttiin kuuluvat myös kuvaukset yleisistä toiminnallisista ja ei-toiminnallisista vaatimuksista, järjestelmän rajoitteista, rajapinnoista ja tietovarastoista. Toiminnallisiin ominaisuuksiin kirjattiin mm. virheenkäsittelyyn liittyviä asioita. Ei-toiminnalliset vaatimukset koostuivat lähinnä formaatti-, asennus- ja ylläpitovaatimuksista.

Määrittelyn valmistumisen ja varsinaisen toteutustyön aloittamisen välillä kului muutama kuukausi, joiden aikana Alfamella oli muodostunut idea, että Zabbix voisi olla hyvä, valmis työkalu tähän valvontatarpeeseen. Zabbixiin perehdyttyäni totesin sen kaikin puolin sopivaksi työkaluksi valvontaan, koska se täyttää kaikki määritellyt tarpeet. Sovimme asiakkaan kanssa, että kokeillaan Zabbixin soveltuvuutta asentamalla se pilvipalvelimelle ja valvomalla sillä muutamaa Alfamen omaa palvelinta.

Päätavoitteena tässä kokeilussa oli rakentaa Zabbixille jakeluputki Ansiblella, jossa pyrittäisiin automatisoimaan mahdollisimman suuri osa konfiguraatiosta. Alfamen integraatiotiimissä on ollut tavoitteena saada kaikkien sovellusten jakelu niin automaattiseksi, että mitään konfiguraatioita ei tehtäisi käsin, jotta mahdollisessa katastrofitilanteessa järjestelmän uudelleenpystytys olisi mahdollisimman kivutonta ja nopeaa. Kun jakeluputki olisi valmiina, tulisi mahdollinen Zabbixin käyttöönotto Alfamen asiakkaille helpommaksi. Tarkoitus oli tehdä ratkaisusta siis mahdollisimman geneerinen ja uudelleenkäytettävä.

5 JAKELUPUTKEN TOTEUTUS

Ensimmäisenä täytyi miettiä arkkitehtuuria työn toteutukselle. Päädyin ratkaisuun, jossa Zabbix server, Zabbix web ja Zabbix Java gateway provisioidaan Docker-kontteina palvelimelle. Tietokanta asennettaisiin normaalina ohjelmana, kuten myös valvottavien palvelimien Zabbix agentit. Tietokannan provisiointi Docker-konttina olisi ollut mahdollista, mutta sitten olisi täytynyt tehdä levyjako palvelimelta Docker-konttiin tietokannan käyttämän datan säilömiseksi palvelimen levyllä. Levyjako olisi ollut mahdollinen mutta tarpeettoman hankala ratkaisu. Myös tietokannan ylläpito helpottuu kun se on normaali palvelinohjelma. Tietokantaohjelmistoksi valitsin MariaDB:n, version 5.5. Kyseinen tietokanta on Alfamella useimmiten käytetty, joten tukea oli saatavilla sitä tarvittaessa.

Aloitin toteuttamisen Ansible-jakelusta. Ensimmäisenä kirjoitin playbookin, jolla jaellaan Zabbix server, Zabbix web, Zabbix Java gateway sekä tietokanta palvelimelle. Seuraavaksi kirjoitin Zabbix agentin jakeleva playbook, jossa käytin valmista Ansible-roolia. Viimeisenä kirjoitin playbookin, jolla voidaan jaella valvontamalleja Zabbix serveriin, käyttäen sen sisäistä APIa.

5.1 Zabbix serverin jakelu

Zabbix serverin jakeleva playbook alkaa tietokannan jakelulla. Tietokannan jakelussa käytin valmista roolia. Roolille määritellään muuttujina tietokannan perus- ja root-käyttäjän käyttäjänimi ja salasana sekä tietokannan nimi. Rooli asentaa käyttöjärjestelmästä riippuvat tarvittavat paketit, initialisoi MariaDB:n tarvittavilla konfiguraatioilla, muuttaa root-käyttäjän salasanan määriteltyyn salasanaan, luo määritellyn nimisen tietokannan sekä luo määritellyn nimisen peruskäyttäjän tietokantaan.

Seuraavaksi playbookissa asennetaan HTTPS-yhteyttä varten tarvittavat sertifikaatit. Zabbix webin Docker-asennus vaatii SSL-sertifikaatin, SSL-avaimen sekä Diffie-Helman (DH) Parameters -tiedoston HTTPS-yhteyttä varten. Loin tätä varten roolin, joka kopioi SSL-sertifikaatin ja avaimen kohdekansioon, jos ne löytyvät määritellystä paikallisesta kansiosta, tai generoi ne, jos niitä ei löydy. Rooli myös luo DH Parameters -tiedoston käyttämällä openssl-komentoa, jos tiedostoa ei löydy kohdekansiosta.

Seuraava vaihe on Docker-konttien provisiointi. Käytin provisiointiin valmista Docker Compose -roolia. Roolia käytetään siten, että siihen määritellään tarvittavat palvelut, verkot ja muut lähes samalla syntaksilla kuin oikeaan Docker Compose -tiedostoon, mutta samalla voidaan käyttää muuttujia Ansiblen inventorystä. Rooli generoi määritellyistä tiedoista Docker Compose -tiedoston, kopioi sen kohdepalvelimelle ja käynnistää kontit.

Ensimmäiseksi kuvasin palveluiden käyttämän yksityisverkon. Käytin verkossa sillattua tilaa, joka tarkoittaa sitä, että yksi verkon osoitteista on varattu Dockeria ajavalle palvelimelle. Tätä kautta päästään kiinni palvelimella ajettaviin muihin palveluihin, aivan samalla tavalla kuin ulkoiseen palvelimeen. Tämä vaadittiin tietokantayhteyttä varten. Määrittelin verkolle aliverkon peitteen 172.18.0.0/24, jonka mukaan verkko-osoitteet määrittyvät. Dockeria ajavan isäntäpalvelimen osoitteeksi tulee tällöin 172.18.0.1.

Seuraavaksi kuvasin Zabbix server -palvelun. Imagena käytän Zabbixin valmista server-imagea, versionumerolla 4.4-latest, joka tarkoittaa uusinta 4.4-versiosta tehtyä imagea. Näin saadaan jakelujen mukana uusimmat päivitykset nostamatta versionumeroa hallitsematta. Asetin palvelun käyttämään aiemmin kuvattua yksityisverkkoa ja asetin osoitteeksi 172.18.0.2. Kontin ympäristömuuttujiin asetin tietokannan osoitteen ja portin, jotka ovat tässä tapauksessa isäntäpalvelimen osoite ja MariaDB:n vakioportti 3306. Ympäristömuuttujiin tuli myös tietokannan nimi, käyttäjä ja salasana, jotka oli määritelty tietokantaroolin yhteydessä. Nämä arvot kirjoitin Ansiblen inventoryyn, jotta niitä pystyisi hallinnoimaan yhdestä paikasta. Määritin palvelulle porttiohjauksen isäntäpalvelimen portista 10051 kontin porttiin 10051. 10051 on Zabbix serverin vakioportti yhteyksille Zabbix agentiin. Portin voi määritellä isäntäpalvelimen puolella haluamakseen, mutta kyseinen portti oli vapaana enkä lähtenyt sitä muuttamaan.

Zabbix web -palvelun kuvaus on melko samanlainen kuin server-palvelun määrittely. Imageksi otin taas Zabbixin valmiin web-imagen versiolla 4.4-latest. Ympäristömuuttujat ovat samat kuin server-palvelulla, mutta niiden lisäksi määritellään myös Zabbix serverin osoite ja portti sekä Zabbix serverin nimi. Tämä nimi näkyy käyttöliittymässä. Liitin palvelun yksityisverkkoon ja asetin osoitteeksi 172.18.0.3. Porttiohjaukset tein porteille 80 ja 443, jotka ovat HTTP- ja HTTPS-portit. Määritin palvelulle levyjaon isäntäkoneen aiemmin luodut sertifikaatit sisältävästä kansiosta kontin sisällä olevaan sertifikaattikansioon, joka oli kerrottu Zabbix web -imagen ohjeessa.

Zabbix Java gateway -palvelun kuvaus oli yksinkertainen. Määritin palvelulle imagen, joka oli taas valmis Zabbixin hallinnoima image, versiolla 4.4-latest. Liitin palvelun yksityisverkkoon, osoitteella

172.18.0.4. Tein porttiohjauksen vakioportille 10052. Ympäristömuuttujiin asetin kaksi muuttujaa. Ensimmäisellä määritellään, kuinka monta kyselyprosessia (engl. poller) käynnistetään ja toisella määritellään näille kyselyille aikakatkaisuaika. Tämän lisäksi Zabbix server -palvelun ympäristömuuttujiin täytyi asettaa Java gatewayn osoite ja portti.

Kaikille palveluille määrittelin uudelleenkäynnistyssäännön. Uudelleenkäynnistyssääntö rekisteröidään Docker-prosessiin, joka tällöin tietää käynnistää kontit uudestaan virheen tai palvelimen uudelleenkäynnistyksen jälkeen. Asetin myös Zabbix web ja Zabbix Java gateway -palvelut riippuvaisiksi Zabbix server -palvelusta, mikä tarkoittaa, että server-palvelun täytyy käynnistyä ennen kuin muita palveluita aletaan käynnistämään.

5.2 Zabbix agentin jakelu

Zabbix agentin jakeleva playbook sisältää vain yhden roolin. Valitsin valmiin, avoimen lähdekoodin roolin oman kirjoittamisen sijasta, koska se täytti kaikki tarpeeni eikä ollut tarvetta keksiä pyörää uudestaan. Rooli tukee myös useita käyttöjärjestelmiä. Rooli asentaa ensin Zabbix agentin kohdepalvelimeen, konfiguroi sen sekä rekisteröi agentin uutena kohteena Zabbix serveriin käyttäen sen APIa. Roolia käytettäessä määritellään muuttujina mm. versionumero, Zabbix serverin osoite, mitä porttia agentin tulisi käyttää sekä kohdepalvelimen nimi. Uuden kohteen rekisteröintiä varten muuttujiin laitetaan Zabbix serverin APIn osoite sekä admin-käyttäjän käyttäjänimi ja salasana. Suurin osa näistä tiedoista oli jo valmiiksi Ansiblen inventoryssä. Roolille voi antaa myös listan niistä palvelinryhmistä, joihin sen halutaan kuuluvan, sekä listan niistä valvontamalleista, jotka siihen halutaan liitettävän. Nämä automaattiset konfiguraatiot nopeuttavat Zabbixin käyttöönottoa huomattavasti.

5.3 Valvontamallien jakelu

Valvontamallien (template) jakelu on myös yksinkertainen, se sisältää vain yhden kirjoittamani roolin. Rooli ottaa vastaan listan olioita, joissa on määritelty valvontamallin nimi, malliryhmä sekä polku paikalliseen tiedostoon, jossa valvontamalli on määritelty. Rooli, kuten Zabbixkin, tukee XML- ja JSON-muotoisia malleja. Roolissa käytetään Ansibleen sisäänrakennettua zabbix_template -nimistä moduulia, jolla voidaan lisätä Zabbixiin valvontamalleja Zabbix serverin APIn kautta. Muuttujana tuotu lista käydään läpi ja kutsutaan moduulia, jolle annetaan parametreina APIn osoite, käyttäjätunnus ja salasana,

sekä valvontamallin nimi, ryhmä ja sisältö. Moduuli kutsuu Zabbixin APIa ja lisää valvontamallin tiedot Zabbix serveriin.

6 ZABBIXIN KÄYTTÖÖNOTTO

Pilvipalvelun tarjoajaksi valitsin Amazon Web Servicesin (AWS). AWS:ssa on palvelu nimeltä Elastic Compute Cloud (EC2), josta voi tilata pilvipalvelimia helposti valmiiden levykuvien pohjalta. Valitsin levykuvaksi Ubuntu 18.04 -käyttöjärjestelmään perustuvan levykuvan. Palvelimen suorituskyvyn, kapasiteetin ja levytilan saa joko itse konfiguroida tai valita valmiista erikokoisista vaihtoehdoista. Valitsin t2.medium -kokoisen instanssin, jossa on kaksi virtuaalisuoritinta sekä neljä gigatavua muistia. Tallennustilan koko on 30 gigatavua. Näiden valintojen pohjalta EC2 käynnistää automaattisesti virtuaalipalvelimen, jolla on oma kiinteä ja julkinen IP-osoite. Palvelin käynnistyy muutamissa minuuteissa.

Kun palvelin oli tilattu ja IP-osoite selvillä, lisäsin palvelimen tiedot Ansiblen inventoryyn. Lisäsin myös EC2:in generoiman SSH-avaimen omaan SSH-konfiguraatiotiedostooni ja testasin SSH-yhteyden toimivuuden palvelimeen. Ansible tarvitsee toimiakseen vain SSH-yhteyden. Kun yhteys toimi, pystyin aloittamaan palveluiden jakelun. Ensimmäisen playbookin ajon jälkeen Zabbix server oli toiminnassa. Jakelin palvelimelle myös Zabbix agentin, ja kohde rekisteröityi Zabbix serveriin. Lisäsin Zabbix server -kohteeseen mallit yleisen Linux-palvelimen sekä Zabbix serverin valvontaan. Linux-mallissa on normaalit levytilan, muistin sekä verkkoliikenteen valvonnat. Linux-malli sisältää myös muutaman turvallisuuskohteen valvonnan, mm. asennettujen pakettien listauksen sekä kaikki käyttäjät sisältävän passwd-tiedoston valvonnan. Passwd-tiedostoa valvomalla voidaan hälyttää, jos järjestelmään lisätään käyttäjiä. Zabbix server -valvontamallissa, niin kuin nimestä voi päätellä, monitoroidaan Zabbix serverin toimintaa, mm. sen suorituskykyä.

Valitsin myös kaksi Alfamen omassa käytössä olevaa pilvipalvelinta, joiden tiedot kirjoitin Ansiblen inventoryyn. Jakelin Zabbix agentit samalla tavoin kuin Zabbix-palvelimeen. Niihin otin käyttöön vain Linux-valvontamallin. Näin saatiin evaluoitua Zabbixin toimintaa useamman palvelimen kanssa.

7 YHTEENVETO

Työn tavoitteena oli tuottaa integraatioiden valvonnan tarvemäärittely sekä rakentaa automaattinen jakeluputki Zabbixille. Tuotin yhdessä asiakkaan kanssa määrittelydokumentin valvontatarpeista, rakensin Ansiblella jakeluputken Zabbixille sekä jakelin Zabbixista demoversion pilvipalvelimelle. Zabbix oli teknologiana itselleni entuudestaan tuntematon. Valvontasovellukset ylipäätään olivat itselleni hieman vieras konsepti, joten jouduin tekemään melko paljon taustatutkimusta ennen varsinaisen toteutusosuuden aloitusta. Docker-kokemustakaan minulla ei ennen projektin aloitusta ollut lähes yhtään. Alfamella Docker on ollut käytössä jo hetken, joten työkavereilta sain jonkin verran apua siihen liittyvissä haasteissa. Ansiblea olen töissäni käyttänyt melko paljon, joten se ei tuottanut varsinaisia ongelmia. Jonkin verran täytyi tietysti perehtyä uusiin ominaisuuksiin, kuten sen sisäänrakennettuihin Zabbix-moduuleihin.

Mielestäni onnistuin työn tavoitteissa hyvin. Valvontatarpeiden määrittely oli arvokasta ja tarvittua työtä. Kun tarpeet saadaan selville, voidaan alkaa kehittämään tarpeisiin vastaavia täsmäratkaisuja. Myös Zabbix-kokeilu on mielestäni onnistunut. Olen erittäin vahvasti sitä mieltä, että Zabbix sopii hyvin Alfamen valvontatarpeisiin. Se on erittäin varmatoiminen ja suhteellisen yksinkertainen. Varsinkaan jälkimmäistä adjektiivia ei monesta valvontasovelluksesta voi hyvällä omatunnolla sanoa.

Opinnäytetyötä oli varsin mukava tehdä. Pääsin kokeilemaan rahkeitani arkkitehtinä, tekemään määrittelyä asiakkaan kanssa ja suunnittelemaan arkkitehtuuria. Näistä kokemuksista tulee olemaan työssäni suunnattomasti apua päästessäni toimimaan arkkitehtinä oikeissa asiakasprojekteissa. Pääsin myös syventämään osaamistani Dockerin käyttämisessä, joka on kasvava teknologia paitsi Alfamella, myös koko alalla. Zabbixiin tutustuminen ja sen käyttöönotto olivat mielenkiintoisia haasteita. Activiti-osaamisen lisääminen oli myös erittäin positiivista työssäni suoriutumisen kannalta.

LÄHTEET

Ansible. 2019. How Ansible Works. Saatavissa: https://www.ansible.com/overview/how-ansible-works. Viitattu 8.12.2019.

Docker. 2019a. Networking in Compose. Saatavissa: https://docs.docker.com/compose/networking/. Viitattu 15.12.2019.

Docker. 2019b. Overview of Docker Compose. Saatavissa: https://docs.docker.com/compose/. Viitattu 15.12.2019.

Kasireddy, P. 2016. A Beginner-Friendly Introduction to Containers, VMs and Docker. Julkaistu 4.3.2016. Saatavissa https://www.freecodecamp.org/news/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b/. Viitattu 15.12.2019.

MuleSoft. 2019. What is Mule ESB?. Saatavissa: https://www.mulesoft.com/resources/esb/what-mule-esb. Viitattu 15.12.2019.

Nykänen, P. 2015. Tietojärjestelmien integraatiosta ja integraation suunnittelusta. Julkaistu 24.2.2015. Saatavissa: http://www.uta.fi/sis/tie/tjsum/index/TJSUM_Luento5_2015_PirkkoNyk%C3%A4nen.pdf. Viitattu 10.11.2019.

Ryhänen, J. 2019. Johdanto integraatioihin ja iPaaS:iin. Julkaistu 14.8.2019. Saatavissa: https://www.devisioona.fi/2019/08/14/johdanto-integraatioihin-ja-ipaasiin/. Viitattu 10.11.2019.

Zabbix. 2019. Zabbix Documentation 4.4. Saatavissa: https://www.zabbix.com/documentation/4.4/start. Viitattu 15.12.2019.

Määritys (1)

Integraatioalustan valvonta-agentti

Allekirjoitukset

Vastuullinen	*Organisaatio*	*Päiväys*	*Kommentit*
Tuomas Mäkinen	Alfame Systems Oy	12.1.2020	Siistiminen, julkaisu
Juho Muuraiskangas	Alfame Systems Oy	04.12.2018	Dokumenttipohjan muokkaus

Versiohistoria

Versio	*Päiväys*	*Muutetut kappaleet*	*Vastuuhen-kilö*	*Muutoksen Kuvaus*
1.0.0	12.1.2020	1, 6	Mäkinen	Siistiminen ja korjaukset.
0.1.11	16.4.2019	1, 3, 7	Mäkinen	Johdantoon tehty korjauksia. Lisätty linkki JSON-objektin skeemaan. Käsitemallin kuvauksia selkeytetty. Yleisiä toiminnallisia vaatimuksia selkeytetty.
0.1.10	15.4.2019	6	Mäkinen	Käyttötapauksia selkeytetty.
0.1.9	3.4.2019	1, 2	Mäkinen	Oikoluku ja korjaukset johdantoon. Ylläpitoprosessi muokattu kolmeen osaan.
0.1.8	2.4.2019	3, 7	Mäkinen	Käsitemallin päivitys. Ei-toiminnallisten vaatimusten päivitys. JSON-skeema.
0.1.7	29.3.2019	2, 3, 4, 6, 7	Mäkinen	Ylläpitoprosessi kuvattu kahdessa osassa. Käsitemallia yksinkertaistettu. Lisätty asiakas-käyttäjä. Toiminnot ja vaatimukset kirjoitettu uusiksi kuvien kera. Toiminnallisia vaatimuksia päivitetty.
0.1.6	28.3.2019	1, 2, 4, 5, 6, 7	Mäkinen	Lisätty asiakkaan näkökulma tarpeisiin. Lisätty uusi prosessikuvaus ylläpitoprosessille. Uusi agentti-käyttäjä. Muokattu järjestelmän rajat -kuvaa. Refaktoroitu käyttötapauksia. Kaksi uutta toiminnallista vaatimusta.
0.1.5	25.3.2019	6, 7	Mäkinen	Toiminnot ja vaatimukset, uudet use caset. Lisätty agentin jakeluun liittyvä

				toiminnallinen vaatimus.
0.1.4	29.1.2019	2	Mäkinen	Lisätty ylläpitoprosessi kuvattuna aktiviteettikaaviona
0.1.3	28.1.2019	3, 5, 6, 7	Mäkinen	Uudet linkit kuviin, täydennyksiä ja fraasimuutoksia
0.1.2	10.12.2018	2, 3, 4, 5, 6	Mäkinen	Liiketoimintaprosessi, järjestelmän rajat, toiminnot ja vaatimukset
0.1.1	7.12.2018	2	Mäkinen	Käsitteet ja käsitemalli
0.1.0	5.12.2018	1	Mäkinen	Johdanto, tarkoitus, esittely & ongelmat ja tarpeet

Sisällys

Määritys (7)

1 Johdanto

Valvonta-agentti on applikaatio, joka asennetaan asiakkaan integraatioalustalle. Agentti lähettää tasaisin väliajoin Alfamelle statistiikkatietoa integraatioalustan ja palvelimen tilasta. Tämä tieto visualisoidaan, ja siitä voidaan seurata integraatioalustan toimintaa lähes reaaliajassa. Valvonta-agentti toteutetaan Tuomas Mäkisen opinnäytetyönä.

1.1 Dokumentin tarkoitus

Dokumentissa määritellään opinnäytetyön runko ja toiminnallisuus. Mahdollista jatkokehitystä ei määritellä.

1.2 Järjestelmän esittely

Mule ESB (Mule) on MuleSoftin kehittämä integraatioviitekehys. Yksinkertaistettuna Mule-applikaatiot ovat Java-applikaatioita, joissa käytetään MuleSoftin rakentamia liitäntöjä eri tiedonsiirto- ja viestintäteknologioihin. Alfamella käytetään Mulen rinnalla Activiti-prosessikuvainmoottoria. Activitilla voidaan automatisoida bpmn-prosessikuvaimia ja liittää niihin toiminnallisuutta kutsumalla Mule-applikaatiota prosessin eri vaiheista. Activiti tuodaan projekteihin osana Alfame ESB -viitekehystä, joka sisältää Activitin lisäksi muun muassa Alfamen kehittämiä liitäntöjä Mule-applikaatioita varten, sekä Alfamen oman prosessihallintatyökalun, prosessimanagerin. Alfame ESB on Alfamen sisäinen sovellustyökalupakki, joka on käytössä käytännössä jokaisessa integraatioprojektissa.

Näitä integraatioalustoja ja -sovelluksia, sekä palvelimia joille ne on asennettu, halutaan seurata ja tehdä ennakoivaa ylläpitoa. Agentti kerää tarvittavat tiedot alustalta ja lähettää ne Alfamen sovellusrajapintaan. Nämä tiedot visualisoidaan Alfamen valvontasovelluksessa. Valvonta-agentti toteutetaan Alfame ESB -viitekehyksen sisälle, josta se on helppo ottaa käyttöön, vaikka kaikille asiakkaille. Valvontasovellusta on tarkoitus tarjota myös asiakkaille, rajattuna sellaiseksi, että he näkisivät vain heidän alustaltaan kerätyn statistiikkatiedon.

1.3 Ongelmat ja tarpeet

Ylläpitonäkökulmasta tarpeena on ylläpitoprosessin parantaminen ja vienti ennakoivampaan suuntaan. Asiakkaiden alustat ovat usein fyysisesti eristetty omaan sisäverkkoonsa ja ainakin suojattu palomuurilla, joten tähän asti ylläpitotilanteessa on jouduttu ensin avaamaan VPN-yhteys, ja sitten selvittämään mitä valvontamenetelmiä asiakkaalla on käytössä. Valvontaa ja ennakointia ei ole tästä hankaluudesta johtuen voitu tehdä juuri ollenkaan, vaan ylläpito on toiminut virheilmoitusten pohjalta. Tarpeena on siis päästä näkemään kaikkien asiakkaiden

alustojen tila yhdestä keskitetystä paikasta, jolloin tarvittavia toimia voidaan tehdä ennakoivasti ja ennen kuin ongelmia syntyy.

Statistiikan näkökulmasta tarpeena olisi saada tehtyä asiakkaallekin näkyväksi alustan tila. Liikennemäärät ja läpimenoajat ovat asiakasta kiinnostavaa statistiikkaa, mutta tällä hetkellä niitä ei pystytä tuomaan esille. Tiedot ovat kyllä periaatteessa olemassa, sovellusten logeissa ym. paikoissa, ne pitäisi vain saada kerättyä ja muutettua visuaaliseksi. Esimerkiksi sairaalan tietoväylältä voisi olla todella kiinnostavaa ja jopa hyödyllistä saada statistiikkaa siitä, mihin aikoihin ja paljonko esimerkiksi radiologiassa tehdään kuvauksia. Meillä on integraatioalustoissa todella paljon mahdollisuuksia kerätä tätä tietoa, mutta sitä ei ole vielä toistaiseksi tehty.

Kun tieto saadaan kerättyä, siitä on helppo tehdä visuaalista informaatiota jota on helppo ymmärtää, vaikkapa prosessin käyttäytymiskäyriä (SPC). Myös virhetilanteet voitaisiin visualisoida, vaikka vain vilkkuvana punaisena valona. Prosessi-informaatiota voitaisi myös analysoida mahdollisesti tekoälyllä, joka voisi ennustaa tulevia virheitä.

Valvonta-agentti hakee alustalta ylläpitoon ja statistiikan muodostamiseen tarvittavaa tietoa. Tärkeimpinä tietoina

- Sanomaliikenteen tiedot
  - Alkuajat
  - Läpimenoajat
  - Määrä
- Palvelimen tiedot
  - Vapaa levytila / koko levytila
  - Vapaa muisti / koko muisti
  - Prosessorin käyttöaste
- Sovellustiedot
  - Javan virtuaalikoneen muisti

1.4 Viittaukset

Alfamen sovellusrajapinta (API Gateway) määritelty Minttu Rinta-Piirron Integraatioalustan valvonta API ja UI määritys -dokumentissa. https://drive.google.com/open?id=1DTU0__5sNOqWHYHpY-ZdNWWJj2p7682XnT1B3mbcR1_Q

Määritys (10)

Valvonta-agentin lähettämän JSON-objektin alustava skeema on määritelty erillisessä dokumentissa.
https://drive.google.com/open?id=1uVsXnu9_bGpeAXnBfqcgAYt2AkUmhyfM

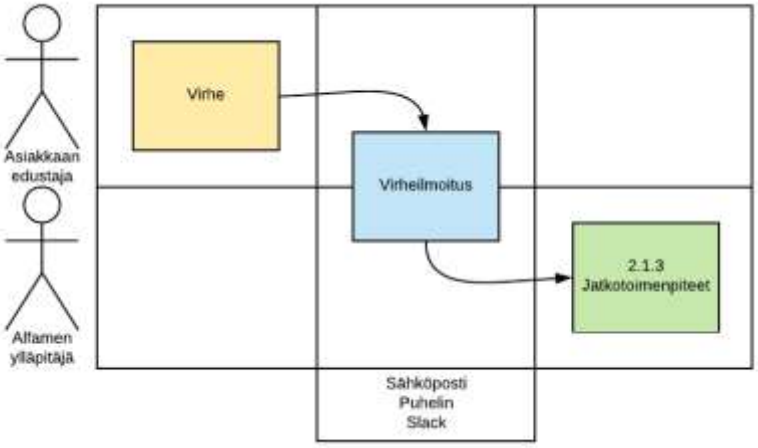
2 Liiketoimintaprosessit

Tavoitteena on saada ylläpitoprosessi sellaiseksi, että ei tarvitse odottaa asiakkaan yhteydenottoa. Ylläpitoa voidaan tehdä ennakoivasti. Voidaan tarkistaa nopeasti asiakkaiden integraatioalustan tila, palveluaste, levytila jne., yhdestä paikasta.

2.1 Prosessimallit

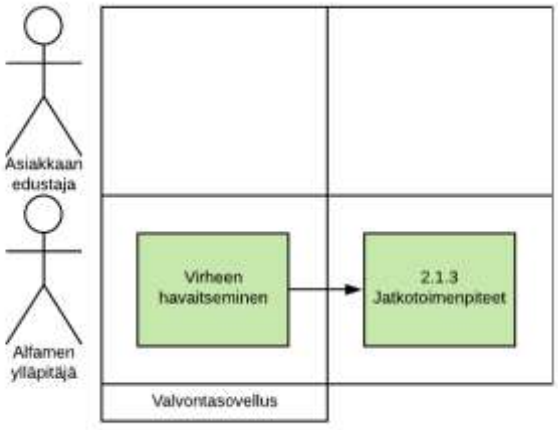
2.1.1 Asiakkaan virheilmoitus

Asiakas huomaa virheen tai vikatilanteen integraatioalustassa. Virheestä tehdään virheilmoitus Alfamelle, ja ylläpitäjä aloittaa virheen perusteella jatkotoimet.



2.1.2 Virheen havaitseminen

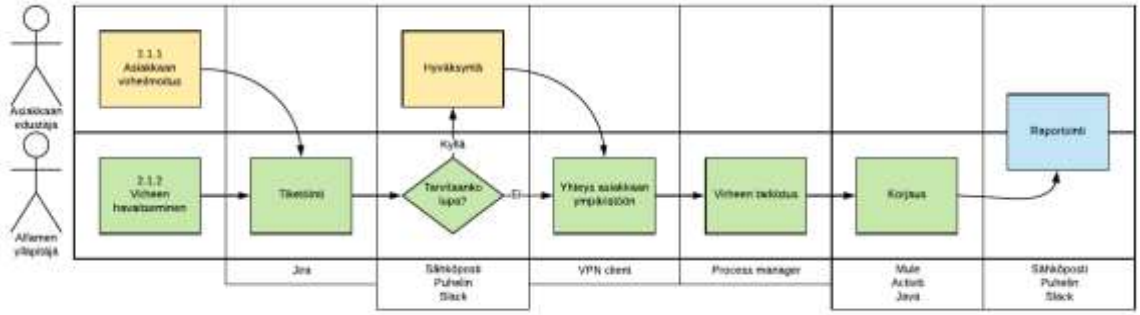
Alfamen ylläpitäjä huomaa virheen valvontasovelluksen avulla ja aloittaa jatkotoimet.



2.1.3 Jatkotoimenpiteet

Määritys (12)

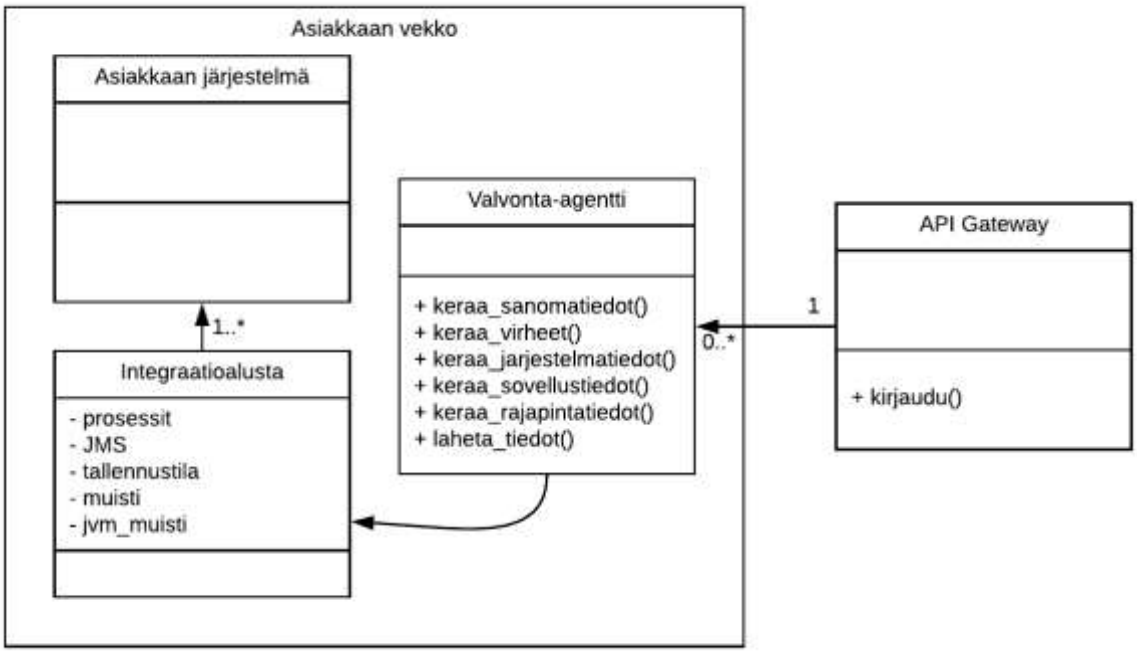
Virheen ratkaisun toimenpiteet.



3 Järjestelmän käsitteet

Käsite	Kuvaus	Tyyppi
Järjestelmä	Asiakkaan tietojärjestelmä, voi olla esimerkiksi laskutusohjelmisto tai toiminnanohjausjärjestelmä	LK
Integraatio-alusta	Asiakkaalle toimitettu integraatioalusta, Mule + Activiti	LK
Valvonta-agentti	Integraatioalustan rinnalle asennettu sovellus, joka kerää tietoa alustan tilasta ja toiminnasta	LK
API Gateway	Alfamen sovellusrajapinta	LK
Valvontasovellus	Sovellus, jossa visualisoidaan agentin keräämä informaatio	LK

3.1 Käsitemalli



4 Järjestelmän käyttäjät

4.1 Kehittäjä

Tehtävä

Alfame Systems Oy:ssä työskentelevä (integraatio)kehittäjä. Kehittäjä ottaa valvonta-agentin käyttöön integraatioalustan konfiguraatiossa.

Tekninen osaaminen

Valvonta-agentin käyttöönotto vaatii työtehtävissäkin tarvittavaa teknistä osaamista.

Käyttäjien määrä nyt ja tulevaisuudessa

5-40

Muuta

4.2 Ylläpitäjä

Tehtävä

Alfame Systems Oy:ssä työskentelevä ylläpitoa tekevä henkilö. Ei käytä varsinaista valvonta-agenttia, vaan valvontasovellusta, jonka data tuodaan valvonta-agentilla.

Tekninen osaaminen

Valvontasovelluksen käyttäminen ei vaadi erityistä teknistä osaamista.

Käyttäjien määrä nyt ja tulevaisuudessa

5-40

Muuta

4.3 Agentti

Tehtävä

Looginen käyttäjä. Integraatioalustalle asennettu instanssi agenttisovelluksesta.

Tekninen osaaminen

N/A

Käyttäjien määrä nyt ja tulevaisuudessa

1-1000

Muuta

N/A

4.4 Asiakas

Tehtävä

Asiakkaan edustaja. Ei käytä varsinaista valvonta-agenttia, vaan valvontasovellusta, jonka data tuodaan valvonta-agentilla.

Tekninen osaaminen

Valvontasovelluksen käyttäminen ei vaadi erityistä teknistä osaamista.

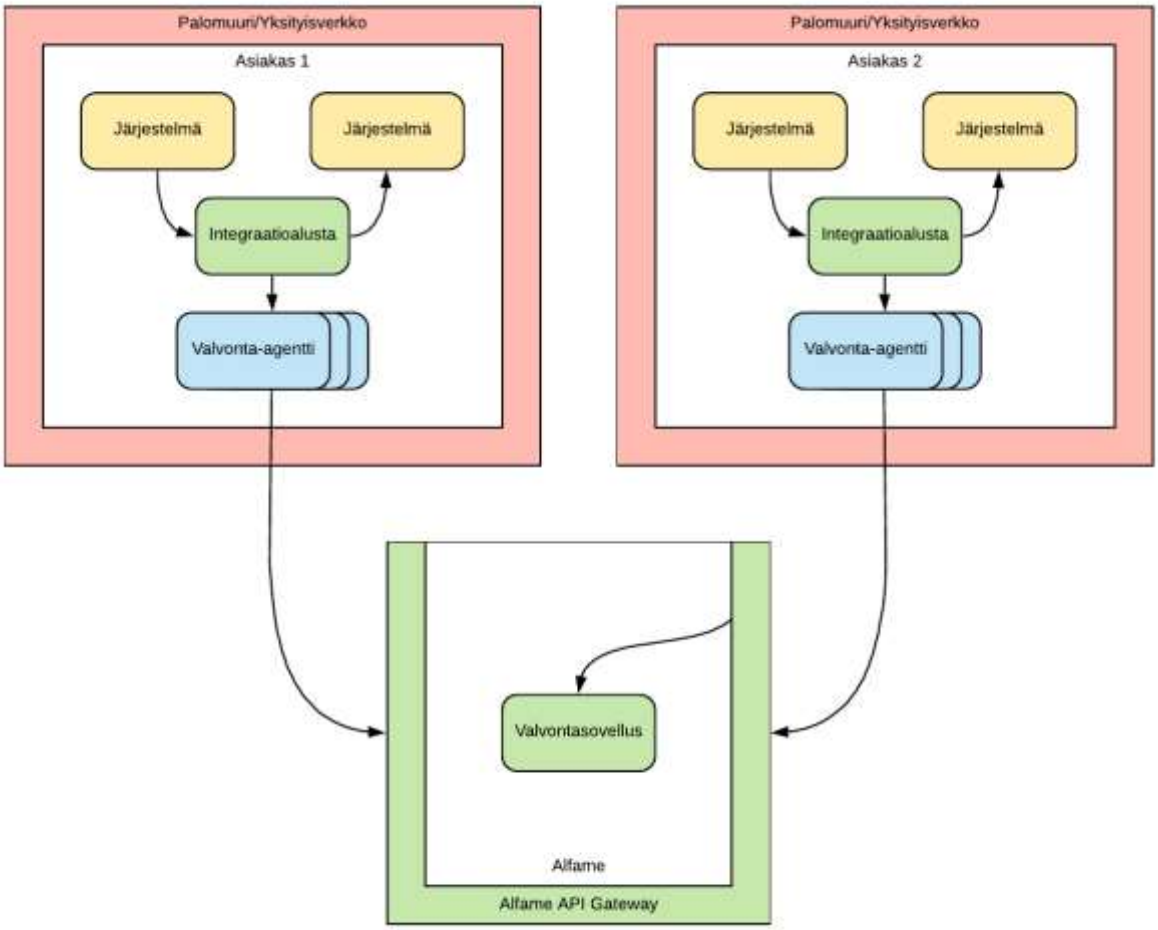
Käyttäjien määrä nyt ja tulevaisuudessa

20-100

Muuta

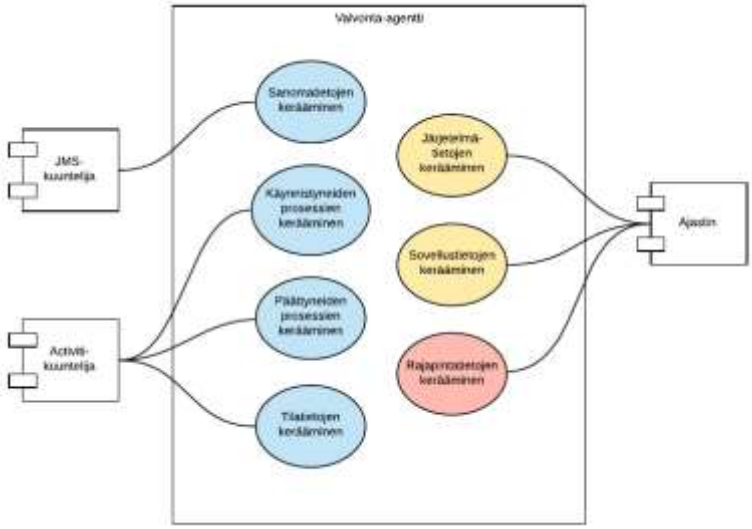
N/A

5 Järjestelmän rajat



6 Järjestelmän toiminnot ja vaatimukset

6.1 Tietojen kerääminen



6.1.1 JMS-kuuntelija

Kuvaus

JMS-kuuntelija -komponenttiin liittyvät toiminnot. Käyttötapauksia on yksi, sanomatietojen kerääminen. JMS-kuuntelija havaitsee uuden JMS-viestin ja tallentaa siitä tiedon JSON-objektiin.

Prioriteetti: 3

Vakaus: Toimiva | Vakaa | Kriittinen

Versio: 1.0.0

Vastuuhenkilö:

Tila: **Keskeneräinen** | Valmis | Katselmoitu | Hyväksytty | Siirretty

6.1.1.1 *Sanomatietojen kerääminen*

Kuvaus

JMS-jonoon luodaan uusi viesti. Agentin JMS-kuuntelijan message listenerille tulee tieto uudesta viestistä. JMS-kuuntelija kirjoittaa uuden viestin tunnisteen ja aikaleiman lähettämistä varten JSON-objektiin.

Tila: Keskeneräinen | Valmis | Katselmoitu | Hyväksytty | **Siirretty**

Käyttäjät: Agentti

Alkuehto

Agentti on asennettu alustalle jossa käytetään JMS:ä ja JMS-kuuntelija on käytössä.

Normaali tapahtumien kulku

1. JMS-jonoon luodaan uusi viesti.
2. Agentin JMS-kuuntelijan message listener saa tiedon uudesta JMS-viestistä.
3. Agentti kirjoittaa JMS-viestin tunnisteen ja aikaleiman JSON-objektiin.

Vaihtoehtoiset tapahtumien kulut

N/A

Erikoisvaatimukset

JMS-viestin sisältöä ei viedä ulos.

Loppuehto

Uuden JMS-viestin tiedot on tallennettu lähettämistä varten.

6.1.2 Activiti-kuuntelija

Kuvaus

Activiti-kuuntelija -komponenttiin liittyvät toiminnot. Kolme käyttötapausta, käynnistyneiden ja päättyneiden prosessien kerääminen sekä virhetietojen kerääminen. Activiti-kuuntelija kuuntelee Activitin tapahtumia, päättelee uudet viestit ja virhetilanteet tapahtumista ja tallentaa niistä tiedot JSON-objektiin.

Prioriteetti: 1

Vakaus: Toimiva | Vakaa | Kriittinen

Versio: 1.0.0

Vastuuhenkilö:

Tila: **Keskeneräinen** | Valmis | Katselmoitu | Hyväksytty | Siirretty

6.1.2.1 *Käynnistyneiden prosessien kerääminen*

Kuvaus

Activitissa käynnistyy uusi prosessi. Agentin Activiti-kuuntelijan event listenerille tulee tieto uudesta prosessista. Activiti-kuuntelija kirjoittaa uuden prosessin tilan, tunnisteen ja aikaleiman lähettämistä varten JSON-objektiin.

Tila: **Keskeneräinen** | Valmis | Katselmoitu | Hyväksytty | Siirretty

Käyttäjät: Agentti

Alkuehto

Agentti on asennettu alustalle jossa käytetään Activitia ja Activiti-kuuntelija on käytössä.

Normaali tapahtumien kulku

1. Activitissa tapahtuu jokin tilamuutos.
2. Agentin Activiti-kuuntelijan event listener saa tiedon tilamuutoksesta.
3. Jos tilamuutos on uusi käynnistynyt prosessi, kirjoitetaan kyseinen tila, prosessin tunniste ja aikaleima JSON-objektiin.

Vaihtoehtoiset tapahtumien kulut

3.1 Ei tehdä mitään jos tilamuutos on päättynyt prosessi, muuttujan muutos tai jokin muu jota ei kuunnella.

Erikoisvaatimukset

Prosessissa kuljetetun viestin tai tiedoston sisältöä ei viedä ulos.

Loppuehto

Käynnistyneen prosessin tiedot on kirjoitettu JSON-objektiin.

6.1.2.2 *Päättyneiden prosessien kerääminen*

Kuvaus

Activitissa päättyy prosessi. Agentin Activiti-kuuntelijan event listenerille tulee tieto päättyneestä prosessista. Activiti-kuuntelija kirjoittaa päättyneen prosessin tilan, tunnisteen ja aikaleiman lähettämistä varten JSON-objektiin.

Tila: **Keskeneräinen** | Valmis | Katselmoitu | Hyväksytty | Siirretty

Käyttäjät: Agentti

Alkuehto

Agentti on asennettu alustalle jossa käytetään Activitia ja Activiti-kuuntelija on käytössä.

Normaali tapahtumien kulku

1. Activitissa tapahtuu jokin tilamuutos.
2. Agentin Activiti-kuuntelijan event listener saa tiedon tilamuutoksesta.
3. Jos tilamuutos on päättynyt prosessi, kirjoitetaan kyseinen tila, prosessin tunniste ja aikaleima JSON-objektiin.

Vaihtoehtoiset tapahtumien kulut

3.1 Ei tehdä mitään jos tilamuutos on uusi prosessi, muuttujan muutos tai jokin muu jota ei kuunnella.

Erikoisvaatimukset

Prosessissa kuljetetun viestin tai tiedoston sisältöä ei viedä ulos.

Loppuehto

Päättyneen prosessin tiedot on kirjoitettu JSON-objektiin.

6.1.2.3 *Tilatietojen kerääminen*

Kuvaus

Activiti-prosessi päättyy virheeseen. Agentin Activiti-kuuntelijan event listenerille tulee tieto virheestä. Activiti-kuuntelija kirjoittaa prosessin tilan, virheeseen päättyneen prosessin tilan, tunnisteen ja aikaleiman lähettämistä varten JSON-objektiin.

Tila: **Keskeneräinen** | Valmis | Katselmoitu | Hyväksytty | Siirretty

Käyttäjät: Agentti

Alkuehto

Agentti on asennettu alustalle jossa käytetään Activitia ja Activiti-kuuntelija on käytössä.

Normaali tapahtumien kulku

1. Activitissa tapahtuu jokin tilamuutos.
2. Agentin Activiti-kuuntelijan event listener saa tiedon tilamuutoksesta.
3. Jos tilamuutos on virheeseen viittaava tila kuten “ACTIVITY_ERROR_RECEIVED”, “JOB_EXECUTION_FAILURE” tai “TASK_CREATED”, kirjoitetaan kyseinen tila, prosessin tunniste ja aikaleima JSON-objektiin.

Vaihtoehtoiset tapahtumien kulut

3.1 Ei tehdä mitään jos tilamuutos päättynyt prosessi, muuttujan muutos tai jokin muu jota ei kuunnella.

Erikoisvaatimukset

Virheviestiä eikä prosessissa kuljetetun viestin tai tiedoston sisältöä ei viedä ulos.

Loppuehto

Activiti-virheen tiedot on kirjoitettu JSON-objektiin.

6.1.3 Ajastin

Kuvaus

Ajastin-komponenttiin liittyvät toiminnot. Käyttötapauksia kolme, järjestelmä-, sovellus- ja rajapintatietojen kerääminen. Ajastin käynnistää tiedon keruun näiltä kolmelta tasolta, ja tiedot tallennetaan JSON-objektiin.

Prioriteetti: 2

Vakaus: Toimiva | Vakaa | Kriittinen

Versio: 1.0.0

Vastuuhenkilö:

Tila: **Keskeneräinen** | Valmis | Katselmoitu | Hyväksytty | Siirretty

6.1.3.1 *Järjestelmätietojen kerääminen*

Kuvaus

Agentti aloittaa ajastetun järjestelmätason tiedon keruun ja kirjoittaa tiedot JSON-objektiin.

Tila: **Keskeneräinen** | Valmis | Katselmoitu | Hyväksytty | Siirretty

Käyttäjät: Agentti

Alkuehto

Agentti on asennettu alustalle ja järjestelmätietojen kerääjä on käytössä.

Normaali tapahtumien kulku

1. Agentti aloittaa ajastetun tiedonkeruun.
2. Agentti hakee palvelimelta levytilan, prosessorin ja muistin tiedot.
3. Agentti kirjoittaa kerätyt tiedot ja aikaleiman JSON-objektiin.

Vaihtoehtoiset tapahtumien kulut

2.1 Agentti ei saa tietoja haettua, kirjoitetaan JSON-objektiin virheviesti ja aikaleima.

Erikoisvaatimukset

N/A

Loppuehto

Järjestelmästä kerätyt tiedot on kirjoitettu JSON-objektiin.

6.1.3.2 *Sovellustietojen kerääminen*

Kuvaus

Agentti aloittaa ajastetun sovellustason tiedon keruun ja kirjoittaa tiedot JSON-objektiin.

Tila: **Keskeneräinen** | Valmis | Katselmoitu | Hyväksytty | Siirretty

Käyttäjät: Agentti

Alkuehto

Agentti on asennettu alustalle ja sovellustietojen kerääjä on käytössä.

Normaali tapahtumien kulku

1. Agentti aloittaa ajastetun tiedonkeruun.
2. Agentti hakee sovellusalustalta JVM:n muistin tiedot.
3. Agentti kirjoittaa kerätyt tiedot ja aikaleiman JSON-objektiin.

Vaihtoehtoiset tapahtumien kulut

2.1 Agentti ei saa tietoja haettua, kirjoitetaan JSON-objektiin virheviesti ja aikaleima.

Erikoisvaatimukset

N/A

Loppuehto

Sovellustasolta kerätyt tiedot on kirjoitettu JSON-objektiin.

6.1.3.3 *Rajapintatietojen kerääminen*

Kuvaus

Agentti aloittaa ajastetun rajapintatason tiedon keruun ja kirjoittaa tiedot JSON-objektiin.

Tila: Keskeneräinen | Valmis | Katselmoitu | Hyväksytty | **Siirretty**

Käyttäjät: Agentti

Alkuehto

Agentti on asennettu alustalle ja rajapintatietojen kerääjä on käytössä.

Normaali tapahtumien kulku

1. Agentti aloittaa ajastetun tiedonkeruun.
2. Agentti kutsuu sovelluksen käyttämiä rajapintoja.
3. Agentti kirjoittaa rajapintojen vastaukset ja aikaleiman JSON-objektiin.

Vaihtoehtoiset tapahtumien kulut

N/A

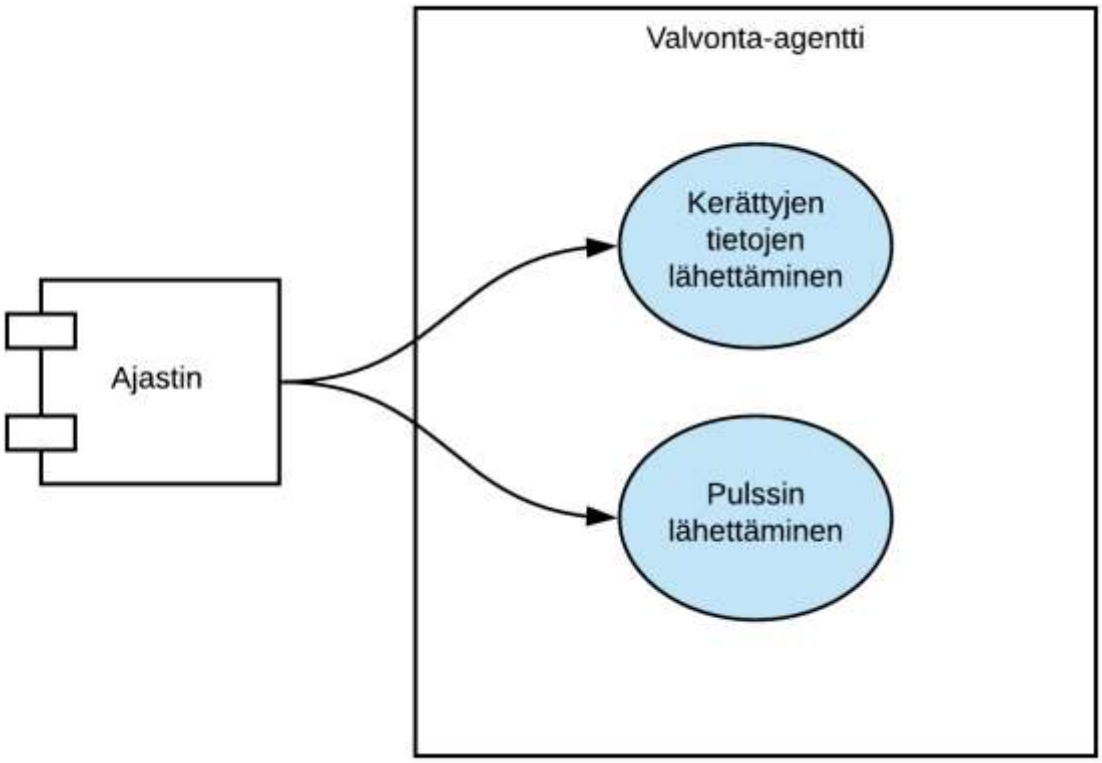
Erikoisvaatimukset

N/A

Loppuehto

Rajapintatasolta kerätyt tiedot on kirjoitettu JSON-objektiin.

6.2 Tietojen lähettäminen



6.2.1 Ajastin

Kuvaus

Ajastin-komponenttiin liittyvät toiminnot. Käyttötapauksia on kaksi, kerättyjen tietojen lähettäminen ja pulssin lähettäminen. Kerättyjen tietojen lähettäminen tapahtuu vain silloin kun tietoja on saatu kerättyä. Pulssi lähetetään aina. Molemmat tapahtuvat kiinteällä, määriteltävällä aikavälillä.

Prioriteetti: 1

Vakaus: Toimiva | Vakaa | Kriittinen

Versio: 1.0.0

Vastuuhenkilö:

Tila: **Keskeneräinen** | Valmis | Katselmoitu | Hyväksytty | Siirretty

6.2.1.1 *Kerättyjen tietojen lähettäminen*

Kuvaus

Agentti lähettää Alfamen API Gatewayhin JSON-objektin johon on kirjoitettu integraatioalustalta kerätyt statistiikkatiedot.

Tila: **Keskeneräinen** | Valmis | Katselmoitu | Hyväksytty | Siirretty

Käyttäjät: Agentti

Alkuehto

Agentti on asennettu integraatioalustalle ja integraatioalustalta on saatu kerättyä JSON-objektiin tietoja alustan tilasta.

Normaali tapahtumien kulku

1. Agentti aloittaa ajastetun lähetyksen.
2. Agentti lähettää JSON-objektin Alfamen API Gatewayhin.

Vaihtoehtoiset tapahtumien kulut

2.1 Jos JSON-objektia ei ole tai se on tyhjä, tehdään 6.2.1.2 Pulssin lähettäminen

Erikoisvaatimukset

N/A

Loppuehto

Alfamen API Gatewayhin saapuu agentin lähettämä JSON-objekti sisältäen kerätyt statistiikkatiedot.

6.2.1.2 *Pulssin lähettäminen*

Kuvaus

Agentti lähettää Alfamen API Gatewayhin pulssitiedon, eli tiedon siitä että agentti on vielä käynnissä integraatioalustalla.

Tila: **Keskeneräinen** | Valmis | Katselmoitu | Hyväksytty | Siirretty

Käyttäjät: Agentti

Alkuehto

Agentti on asennettu integraatioalustalle ja statistiikkaa sisältävää JSON-objektia ei ole, tai se on tyhjä.

Normaali tapahtumien kulku

1. Agentti aloittaa ajastetun tiedon lähetyksen.
2. Agentti lähettää JSON-objektin Alfamen API Gatewayhin.

Vaihtoehtoiset tapahtumien kulut

N/A

Erikoisvaatimukset

N/A

Loppuehto

Alfamen API Gatewayhin saapuu agentin lähettämä JSON-objekti.

7 Yleiset toiminnalliset vaatimukset

7.1 Agentin runko ja komponentit

Agentin rungossa määritellään lähetysintervalli ja kaikki tiedonhaku toteutetaan erillisinä komponentteina. Näihin komponentteihin voi määritellä eri hakukohteita ja kyselytapoja. Näitä komponentteja sisällytetään agenttiin kun se otetaan käyttöön integraatioprojektiin. Agentin runko vastaa kerättyjen tietojen ja pulssin lähetyksestä.

Hyväksymiskriteeri

Kuinka validoidaan

Tila

Keskeneräinen | Valmis | Katselmoitu | Hyväksytty | Siirretty

7.2 Yhteydet

Sisäänpäin suuntautuvat yhteydet avaavat hyökkäyspinta-alaa asiakkaan järjestelmiin, josta johtuen ne on usein suojattu palomuurilla ja yksityisverkossa. Agentti halutaan sellaiseksi, että yhteydet suuntautuvat ainoastaan ulospäin.

Hyväksymiskriteeri

Kuinka validoidaan

Tila

Keskeneräinen | Valmis | Katselmoitu | Hyväksytty | Siirretty

7.3 Autentikointi

Toteutetaan autentikointi siten, että käynnistyttyään agentti yrittää lähettää tiedot normaalisti API gatewayhin. Jos saadaan vastaukseksi HTTP 401 (unauthorized), yritetään uudestaan esimerkiksi 15 minuutin päästä. Lähetyksen jälkeen valvontasovellukseen tulee näkyville uuden yhteyttä ottaneen agentin tiedot ja se voidaan sallia lähettämään tietojaan. Tällöin kun agentti lähettää uudet tiedot ja saa vastaukseksi esimerkiksi HTTP 201 (created), alkaa se lähettää tietojaan normaalilla aikavälillä.

Hyväksymiskriteeri

Kuinka validoidaan

Tila

Keskeneräinen | Valmis | Katselmoitu | Hyväksytty | Siirretty

7.4 Tiedon keräyksen virheiden käsittely

Määritys (29)

Jos jokin valvonta-agentin tiedonkeräysyritys päättyy virheeseen/poikkeukseen, siitä muodostetaan virheviesti JSON-objektiin, sen tietueen alle jonka keräämisessä virhe tapahtui.

Hyväksymiskriteeri

Kuinka validoidaan

Tila

Keskeneräinen | Valmis | Katselmoitu | Hyväksytty | Siirretty

7.5 Yhteysongelmien virheiden käsittely

Jos agentti ei saa yhteyttä Alfamen API Gatewayhin, muodostetut tiedot tallennetaan välimuistiin ja lähetetään kun yhteys palaa. Konfiguroitava aika jonka jälkeen aletaan pudottamaan vanhinta kerättyä tietoa pois.

Hyväksymiskriteeri

Kuinka validoidaan

Tila

Keskeneräinen | Valmis | Katselmoitu | Hyväksytty | Siirretty

8 Yleiset ei-toiminnalliset vaatimukset

8.1 Formaattivaatimukset

Agentin muodostama tieto pitää olla JSON-skeeman mukaista.

8.2 Asennusvaatimukset

Valvonta-agentti toteutetaan Alfame ESB -viitekehykseen, josta kehittäjät ottavat sen käyttöön integraatioprojekteihin. Käyttöönoton yhteydessä tehtävät konfiguraatiot pyritään pitämään minimaalisena.

8.3 Dokumentointi- ja ohjeistusvaatimukset

Käyttöönotto dokumentoidaan wikiin ja/tai README.md-tiedostoon.

8.4 Ylläpitovaatimukset

Valvonta-agentin ylläpitovaatimukset pyritään saamaan minimaalisiksi.

8.5 Siirrettävyysvaatimukset

Valvonta-agentti toteutetaan niin, että se toimii kaikilla niillä alustoilla johon Alfamen Mule ESB -projekteja asennetaan.

8.6 Tietoturvavaatimukset

Valvonta-agentilla lähetetään vain statistiikkadataa, liikennemääriä ynnä muuta. Integraatioalustan kuljettamia viestejä ei ole tarkoitus tuoda ulos alustalta.

9 Sanasto

Lyhenne	Selitys
CSV	Comma Separated Value File Format
JMS	Java Message Service
JSON	JavaScript Object Notation
SSL	Secure Sockets Layer