jamk

Kulttuuriperintö-PAS-palvelun testiympäristöjen rakennus ja konfigurointi

Aleksi Vauhkonen

Opinnäytetyö, AMK
Marraskuu 2024
Tieto- ja viestintätekniikan tutkinto-ohjelma (AMK)

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Vauhkonen, Aleksi

Kulttuuriperintö-PAS-palvelun testiympäristöjen rakennus ja konfigurointi.

Jyväskylä: Jyväskylän ammattikorkeakoulu. Marraskuu 2024, 41 sivua.

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Pitkäaikaissäilytys (PAS) tarkoittaa digitaalisen tiedon säilyttämistä sellaisessa muodossa, että se pysyy ymmärrettävänä ja käytettävänä kymmenistä satoihin vuosiin. Aiemmin CSC:n pitkäaikaissäilytyspalveluiden testiympäristössä oli käytössä CentOS 7 -käyttöjärjestelmä. Käyttöjärjestelmä vanhentui kesällä 2024, minkä takia testiympäristöt tuli päivittää. Päivittämisen myötä olisi mahdollista testata uusia RHEL9-käyttöjärjestelmälle tehtyjä RPM-paketteja sekä lisätä testiympäristön turvallisuutta. Siirtymällä uuteen käyttöjärjestelmään taattaisiin pitkäaikaissäilytyspalvelun toimivuus tulevaisuudessakin. Päivittäminen hoidettiin luomalla uudet testiympäristöt cPouta-palveluun ja liittämällä ne osaksi CI/CD-putkea. Testiympäristöt luotiin käyttäen Heat Templateja, joilla oli mahdollista automatisoida Heat Stackien pystyttäminen cPoutaan. Testiympäristöt alustettiin testausta varten käyttämällä Ansible-skriptiä. Näin olisi mahdollista automatisoida ympäristöjen konfigurointi. Paikallisen testiympäristön virtuaalikoneet rekisteröitiin GitLabiin runnereiksi käyttäen Ansible-skriptiä. Heat Templatejen ja Ansible-skriptien avulla oli mahdollista automatisoida testiympäristöjen luonti, niiden konfigurointi sekä ympäristöjen liittäminen osaksi CI/CD-putkea. Testiympäristöjen luonti saatiin automatisoitua, niin pitkälle kuin projektissa oli tarkoitus. Projektin automatisointia olisi voinut lisätä automatisoimalla levykuvan luomisen. Projektissa kaikki vastaan tulleet ongelmat saatiin ratkaistua.

Avainsanat (asiasanat)

Pitkäaikaissäilytys, testaus, päivitys, Linux

jamk

Description

Vauhkonen, Aleksi

Kulttuuriperintö-PAS-palvelun testiympäristöjen rakennus ja konfigurointi.

Jyväskylä: JAMK University of Applied Sciences, November 2024, 41 pages.

Bachelor's Degree Programme in Information and Communications Technology. Bachelor´s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

Digital preservation service (DPS) means the preservation of digital information in such a form that ensures it remains understandable and usable for decades or even centuries. Previously, the test environment for CSC's digital preservation services used the CentOS 7 operating system. However, this operating system became obsolete in the summer of 2024, which is why the test environments had to be updated. The update made it possible to test new RPM packages designed for the RHEL9 operating system and improve the security of the test environment. By transitioning to the new operating system, the long-term functionality of the digital preservation service could be ensured. The update was handled by creating new test environments in the cPouta service and integrating them into the CI/CD pipeline. The test environments were built using Heat Templates, which allowed the automated deployment of Heat Stacks in cPouta. The test environments were initialized for testing using Ansible scripts, enabling the automation of environment configuration. Virtual machines in the local test environment were registered as GitLab runners using Ansible scripts. With the help of Heat Templates and Ansible scripts, it was possible to automate the creation of test environments, their configuration, and their integration into the CI/CD pipeline. The creation of the test environments was automated as far as intended in the project. Further automation could have been achieved by automating the creation of disk images. All issues encountered during the project were successfully resolved.

Keywords/tags (subjects)

long-term preservation, testing, updating, Linux

Sisältö

Kuviot

Termiluettelo

API	Application Programming Interface, ohjelmointirajapinta
Ansible	Ohjelmisto, jolla voidaan automatisoida ohjelmistojen asennus ja niiden käyttöönotto
CI/CD	Continuous Integration / Continuous Delivery, Jatkuva integraatio / Jatkuva julkaisu
cPouta	CSC:n tarjoama pilvipalvelu
Git	Yleisesti käytössä oleva versionhallintatyökalu
Heat Template	YAML-muodossa oleva malli, joka määrittelee ja hallinnoi infrastruktuurin resursseja OpenStack-ympäristössä
Kansallisgalleria	Kuvataiteen valtakunnallinen museo, joka huolehtii ja ylläpitää kokoelmaa taideteoksista, jotka kuuluvat Suomen kansallisomaisuuteen
PAS	Pitkäaikaissäilytys
QEMU-virtualisointi	Quick Emulator on avoimen lähdekoodin virtualisointiohjelmisto
RPM-paketti	RPM Package, monissa Linux-jakeluissa käytettävä ohjelmistopaketti
Runneri	Komponentti, joka suorittaa tehtäviä CI/CD-putkessa
Skripti	Ajon aikana tulkattava ohjelmakoodi tai joukko komentoja
SSH	Secure Shell, tietoliikenteen salausprotokolla
vCPU	Virtuaalinen prosessori
VM	Virtual Machine, virtuaalikone

1 Johdanto

1.1 Taustaa ja työn tilaaja

Ohjelmistot vanhenevat, ja niiden ylläpito vaatii jatkuvaa kehitystä ja testausta. Uuden käyttöjärjestelmän myötä vaaditaan uusi testiympäristö sekä käyttöjärjestelmälle sopivat testit. Uudella testiympäristöllä varmistetaan, että uudet ohjelmistoversiot toimivat luotettavasti ja virheettömästi uudella käyttöjärjestelmällä.

Uuden testiympäristön avulla on mahdollista havaita ohjelmistokehityksessä tulleet virheet ja ongelmat, mikä vähentää riskiä ja parantaa lopputuotteen laatua. Tämä työ on olennainen osa ohjelmistokehitysprosessia ja sen avulla varmistetaan, että uudet ohjelmistot ovat valmiita kohtaamaan nykypäivän vaatimukset ja tarpeet.

Opinnäytetyön tilaajana oli CSC. CSC eli Tieteen tietotekniikan keskus Oy on Suomen valtion ja korkeakoulujen omistama tietotekniikan osaamiskeskus. CSC on voittoa tavoittelematon erityistehtäväyhtiö. Suomen valtio omistaa CSC:n osakekannasta 70 % ja korkeakoulut 30 %. Suurin osa CSC:n palveluista on kohdistettu omistajien tarpeisiin, ja omistajat voivat ostaa CSC:n palveluita kilpailuttamatta. (Mikä CSC? n.d.)

CSC tarjoamat palvelualat ovat tieteellinen laskenta, datanhallinta sekä tutkimuksen ja koulutuksen digitalisaatio. Tieteellisessä laskennassa CSC tarjoaa mm. suurteholaskentaa ja kvanttilaskentaa tutkimuslaitoksille ja yrityksille, digitalisaatiossa esim. Funet-tietoverkon tutkijoiden ja opiskelijoiden käyttöön sekä datanhallinnassa mm. tutkijoille tutkimusdatan hallintaa ja pitkäaikaissaatavuuden kulttuuriperintöaineistolle. (Osaamisemme n.d.)

1.2 Tavoitteet ja kehityshaasteet

Työn tavoitteena on toteuttaa sekä paikallinen että hajautettu testiympäristö ja lisäksi asiakastestiympäristö Kulttuuriperintö-PAS-palvelulle. Testiympäristöt koostuvat useista virtuaalikoneista. Näille virtuaalikoneille asennetaan Red Hat Enterprise Linux 9 (RHEL9) -käyttöjärjestelmä.

Ympäristöjen pystytys ja testaus tulisi automatisoida mahdollisimman paljon, jolloin palvelun kehittämisestä tulisi sujuvampaa. Ohjelmia testatessa testiympäristöön voi tulla virheitä, jolloin testiympäristö joudutaan pystyttämään uudelleen. Ilman testiympäristöä ohjelmiston kehitys pysähtyy. Tämän takia testiympäristö tulisi olla helposti ja nopeasti uudelleen pystytettävissä. Tavoitteena olisi myös saada testiympäristöt liitettyä osaksi CI/CD-putkea.

Työn keskeisiä kehityshaasteita ovat:

KH1	Kuinka ympäristön pystytys cPouta-ympäristöön tehdään?
KH2	Miten ympäristön pystyttäminen saadaan automatisoitua?
KH3	Kuinka testiympäristö saadaan osaksi automatisoitua CI/CD-putkea?
KH4	Kuinka ohjelmistokomponentit saadaan testattua RHEL9-testiympäristössä?

KH1:n tavoitteena on saada pystytettyä virtuaaliset testiympäristöt cPouta-palveluun. Haasteena pystyttämisessä on saada uudet virtuaalikoneet konfiguroitua sekä alustettua testausta varten. Virtuaalikoneille tulisi asentaa RHEL9-käyttöjärjestelmä.

KH2:n haasteena on KH1 prosessin automatisointi, jotta ympäristö olisi mahdollista luoda mahdollisimman vähällä vaivalla. Tämä on suunniteltu tehtäväksi käyttäen Heat Stackia sekä Ansible-skriptiä. Heat Stackilla luotaisiin virtuaalikoneet cPoutaan ja Ansible-skriptillä alustettaisiin virtuaalikoneet testausta varten.

KH3:na oli saada uudet testiympäristöt osaksi CI/CD-putkea. Tämä suunniteltiin toteutettavaksi rekisteröimällä paikallisen testiympäristön virtuaalikoneet GitLabiin Runnereiksi.

KH4:n haasteena olisi saada RPM-paketit testattua automatisoidulla CI/CD-putkella. Tämä vaatisi sitä, että aikaisemmat kehityshaasteet onnistuisivat. RPM-paketin testaus hoidettaisiin GitLabissa käyttäen aiemmin rekisteröityjä Runnereita.

2 Teoria

2.1 Pitkäaikaissäilytys

Pitkäaikaissäilytyksellä tarkoitetaan digitaalisen tiedon säilyttämistä useiden kymmenien ja jopa satojen vuosien ajaksi. Pitkäaikaissäilytys edellyttää tiedon eheyden ylläpitämistä koko säilönnän ajan. Vaikka laitteet ja tiedostot muuttuisivatkin vuosien varrelle tulisi tiedon olla saatavilla ja käytettävissä. (Fairdata PAS-palvelu – Tutkimusaineistojen pitkäaikaissäilytys n.d.) CSC aloitti kulttuuriperintöaineistojen digitaalisen pitkäaikaissäilytyksen 2015 yhteistyössä mm. arkistojen ja museoiden kanssa. Tutkimusaineistojen digitaalinen pitkäaikaissäilytys käynnistyi 2019, kun ensimmäinen aineisto tallennettiin CSC:n ylläpitämään kansalliseen pitkäaikaissäilytyspalveluun. (Tutkimusainestojen pitkäaikaissäilytys käynnistyi 2019.) Pitkäaikaissäilytyksen puolella CSC tarjoaa Kulttuuriperintö-PAS-palvelua sekä Fairdata PAS-palvelua (Takaamme tutkimusaineistojen ja kulttuuriperintöaineistojen pitkäaikaissaatavuuden n.d.).

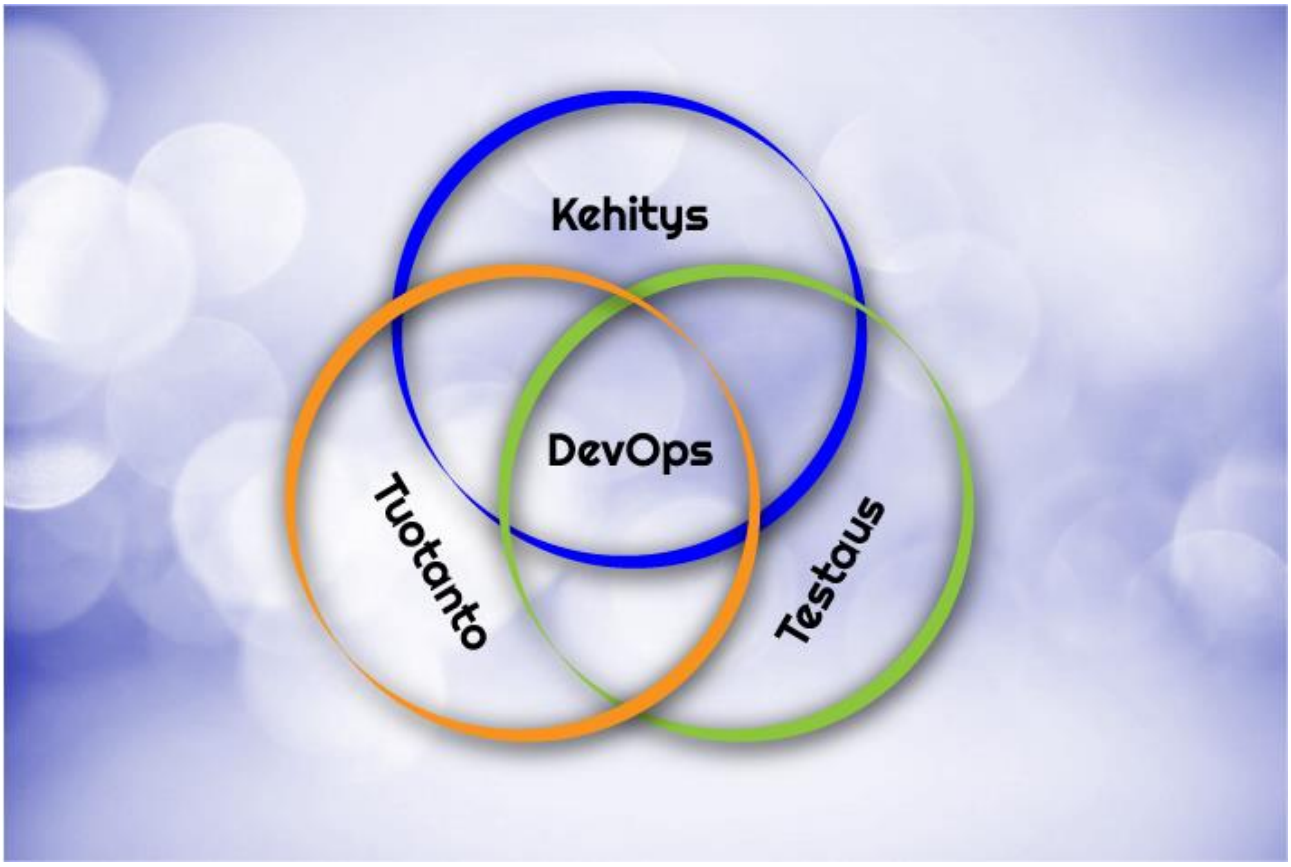
Pitkäaikaissäilytys voidaan jakaa kolmeen eri tasoon: bittitason, loogiseen ja semanttiseen säilyttämiseen. Bittitasolla varmistetaan aineiston muuttumattomuus ja eheys, mikä saavutetaan tiedostokopioiden vertailulla, virheiden korjaamisella ja maantieteellisellä hajautuksella. Loogisella talolla tavoitteena on pitää aineisto käytettävissä tulevaisuudessa, vaikka ohjelmistot ja tiedostomuodot muuttuisivatkin. Tämän takia pitkäaikaissäilytykseen otetaan tiedostomuotoja, joita on mahdollista muuntaa toiseen muotoon. Semanttisella tasolla pyritään pitämään aineisto ymmärrettävänä. Tähän vaaditaan aineiston syvällistä ymmärtämistä ja tuntemista. (Pitkäaikaissäilytys n.d.)

Kulttuuriperintö-PAS-palvelu on palvelukokonaisuus, joka mahdollistaa kulttuuriperinnön säilyvyyden useille sukupolville. Sen avulla turvataan informaation säilyminen, vaikka laitteet, ohjelmistot ja tiedostomuodot vanhenisivatkin ajan myötä. PAS-palvelun suurimpia asiakkaita ovat Kansallisgalleria sekä Museovirasto. (Kulttuuriperintö-PAS kirjastoille, arkistoille ja museoille n.d.)

Fairdata PAS-palvelut ovat suunnattuja tutkimuksen ja tieteen avoimuuden edistämiseen. Fairdata-kokonaisuus muodostuu digitaalisten aineistojen hallintaa tukevista palveluista. Näitä palveluja ovat muun muassa datan säilytys, aineistojen hakupalvelu ja kuvailutyökalu sekä aineistojen pitkäaikaissäilytyspalvelu, joka sisältää hallinnan ja paketoinnin. (Fairdata PAS tutkimusorganisaatioille n.d.)

2.2 DevOps

DevOps tulee sanoista Development (ohjelmistokehitys) ja Operations (IT-toiminnot). DevOps on joukko käytäntöjä, työkaluja sekä kulttuurifilosofiaa. Näillä pyritään automatisoimaan ja integroimaan ohjelmistokehityksen prosesseja. DevOps kehittyi vuoden 2007 vaiheilla, kun aikaisemmat ohjelmistokehityksen mallit eivät toimineet riittävän hyvin ohjelmistokehittäjien mielestä. (DevOps n.d.) DevOps-mallin tarkoituksena on liittää kehitys ja julkaisu yhteen jatkuvaan malliin (ks. kuvio 1) (Winter n.d.).



Kuvio 1. Ohjelmistokehityksen DevOps-malli (Winter n.d.)

Tämän mallin avulla saadaan kaikki projektissa mukana olevat osat toimimaan keskenään. Koko ohjelmistokehitysprosessi on mahdollista tehdä joko yhdellä tiimillä tai useammalla. Yhdessä projektissa voi olla esimerkiksi erikseen ohjelmistokehittäjät, testaajat ja ohjelmiston julkaisijat. DevOpsin vaiheita ovat: suunnittelu, ohjelmointityö, tehdyn ohjelmiston kääntäminen, testaus, julkaisuversion tekeminen, julkaisuversion asennus, ohjelmiston toiminta tuotannoissa ja ohjelmiston seuranta. (DevOps n.d.)

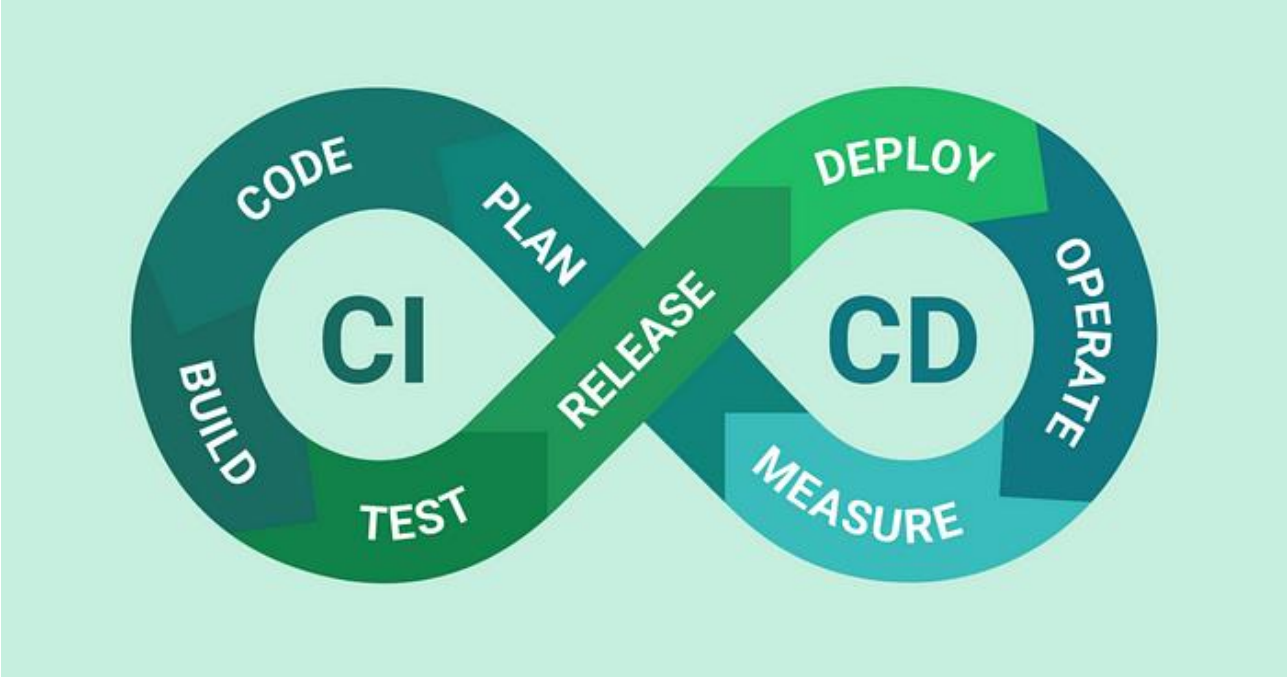
2.3 CI/CD

Yksi ohjelmistokehityksen malli, joka noudattaa DevOpsin periaatteita on CI/CD, joka tarkoittaa jatkuvaa integrointia (Continuous Integration) ja jatkuvaa julkaisua (Continuous Delivery/Continuous Deployment). CI/CD:n avulla voidaan tehdä DevOps-mallista jatkuvan kehityksen malli. CI/CD-putkella voidaan automatisoida ohjelmien ja palveluiden testaus. Tällä mallilla on mahdollista julkaista uusia muutoksia ja korjauksia nopealla tahdilla. (What is CI/CD? n.d.)

CI/CD-putken ensimmäisessä vaiheessa eli jatkuvassa integroinnissa on tavoitteena integroida uutta koodia kehitettävään palveluun/ohjelmistoon. Nykyään koodiin tulevat muutokset on mahdollista hallita version hallintatyökaluilla, kuten GitLabilla. CI/CD-mallissa koodin integrointi tehdään automaattisesti. Tämä tarkoittaa sitä, että uuden koodin testaus suoritetaan automaattisesti sitä mukaan, kun kehittäjät saavat sitä tehtyä ja kun uusi koodi on päässyt läpi asetetuista testeistä, lisätään se kehitettävään palveluun/ohjelmistoon. (Laureat 2020.)

Uuden koodin integroinnin jälkeen tulee CI/CD-putkessa jatkuvan julkaisun vaihe. Siinä on tarkoituksena julkaista uusi versio ohjelmistosta/palvelusta asiakkaiden käyttöön heti integroinnin jälkeen. Näin asiakkaat saavat uusimmat ominaisuudet käyttöön ja pystyvät antamaan palautetta uusista ominaisuuksista. Palautteen myötä voidaan taas aloittaa ohjelmiston/palvelun kehittäminen tai mahdollisten vikojen korjaaminen, joita ei testeissä saatu kiinni. (Laureat 2020.)

Saifullahin (2023) mukaan CI/CD-putken toimivuuden takaamiseksi vaaditaan oikeanlaiset työkalut sekä toimintatavat. Kuviossa 2 esitetään CI/CD-putken toimintavaiheet ja havainnollistetaan, että CI/CD-putkea on mahdollista jatkaa niin kauan, kun ohjelmassa tai palvelussa on kehittämistä.



Kuvio 2. CI/CD-putken malli (Saifullah 2023)

2.4 RPM-paketti

RPM tulee sanoista Red Hat Package Manager. RPM-paketteja käytetään asentamaan ohjelmia tai ohjelmien osia Linux-käyttöjärjestelmälle, joka käyttää RPM-paketin hallintaa. RPM-paketit tekevät ohjelmista helposti siirrettäviä. RPM:n avulla on mahdollista asentaa, päivittää ja poistaa ohjelmia. (Chapter 1. Introduction to RPM n.d.)

RPM-paketit sisältävät ohjelman GPG-allekirjoituksen, metadatan (Header) sekä hyötykuorman. Ohjelman GPG-allekirjoituksella voidaan varmistaa paketin eheys. Metadataa käyttämällä paketinhallintatyökalu saa selville paketin riippuvuudet, minne tiedostot asennetaan ja muut paketin tiedot. Hyötykuorma on ”cpio”-arkistomuodossa. Hyötykuormassa voi olla ohjelman lähdekoodi sekä ”spec”-tiedosto. Tällöin paketti pitää vielä kasata binääri-RPM-paketiksi. Vaihtoehtoisesti hyötykuormana voi olla suoraan valmiiksi kasatut binääritiedostot. (Chapter 1. Introduction to RPM n.d.)

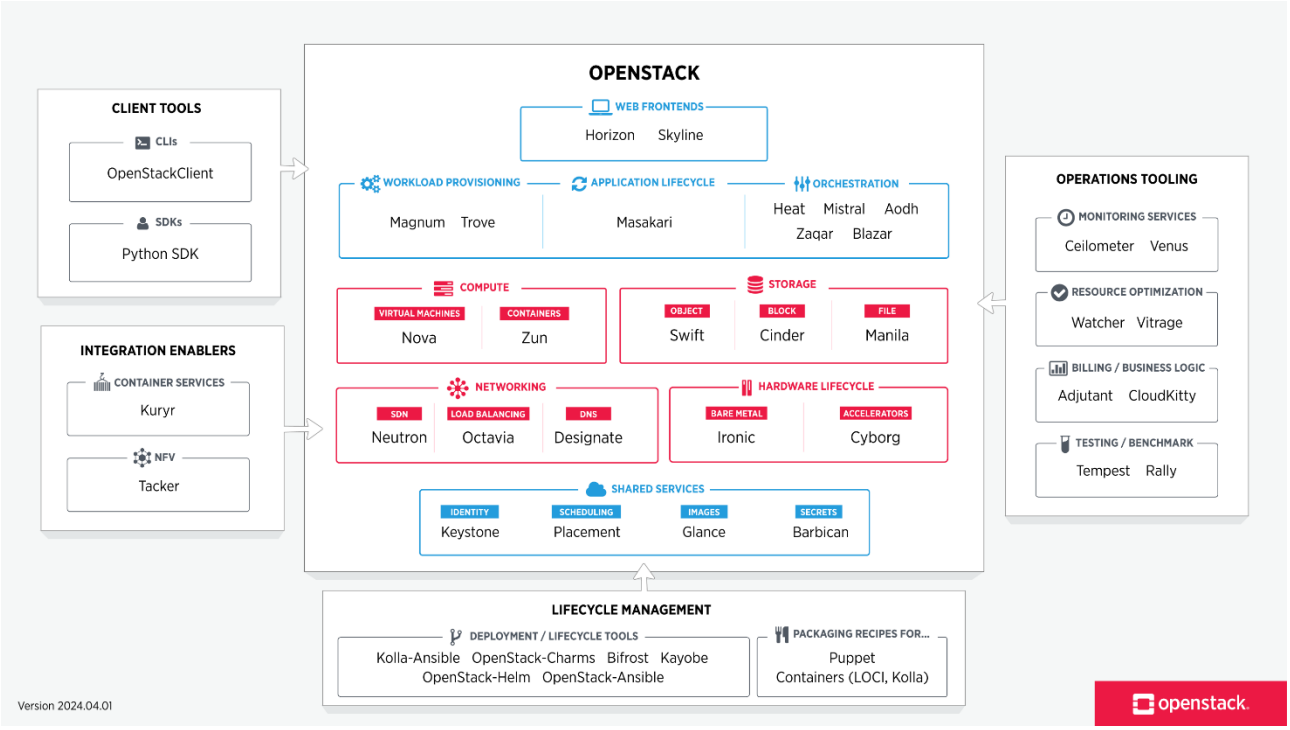
RPM-paketit asennetaan paketinhallintatyökalulla. Paketinhallintatyökaluja ovat muun muassa RPM (Red Hat Package Manager), YUM (Yellow Dog Updater) ja DNF. (Jevtic 2024.) Näitä työkaluja voi käyttää Linuxin terminaalin kautta. Paketinhallintatyökalulla on mahdollista asentaa RPM-paketteja suoraan tiedostosta taikka RPM-pakettivarastosta. RPM-pakettien asennus on myös

mahdollista graafisen käyttöliittymän kautta. Komentorivityökalu on kuitenkin suositumpi, sillä se tekee pakettien asennuksesta ja poistamisesta helpompaa. (Chinthaguntla 2020.)

Toinen suosittu ohjelmistopaketti on DEB-paketti. DEB-paketit on suunniteltu Debian-pohjaisille Linux-alustoille. DEB-paketti sisältää kaksi ”.tar”-tiedostoa. Toinen sisältää ohjelman datan ja toinen sen asennusohjeet. Loppukäyttäjälle DEB- ja RPM-paketit toimivat hyvin samalla tavalla. Kumpiakin on mahdollista hallita paketinhallintatyökalulla. Nämä kaksi pakettia eroavat tosistaan niiden rakenteen osalta sekä käytettävän käyttöjärjestelmällä mukaan. (DEB vs RPM n.d.)

2.5 OpenStack

OpenStack on avoimen lähdekoodin pilvikäyttöjärjestelmä, jolla on mahdollista hallinnoida useita koneita, tallennustiloja ja verkkoresursseja. Näiden resurssien hallinta tapahtuu verkkoselaimen tai rajapinnan (API) kautta. OpenStack koostuu pienemmistä komponenteista (ks. kuvio 3), joita on mahdollista ottaa käyttöön tarpeen mukaan. OpenStack on suunniteltu toimimaan monien kolmansien osapuolten palveluiden kanssa. (OpenStack n.d.)

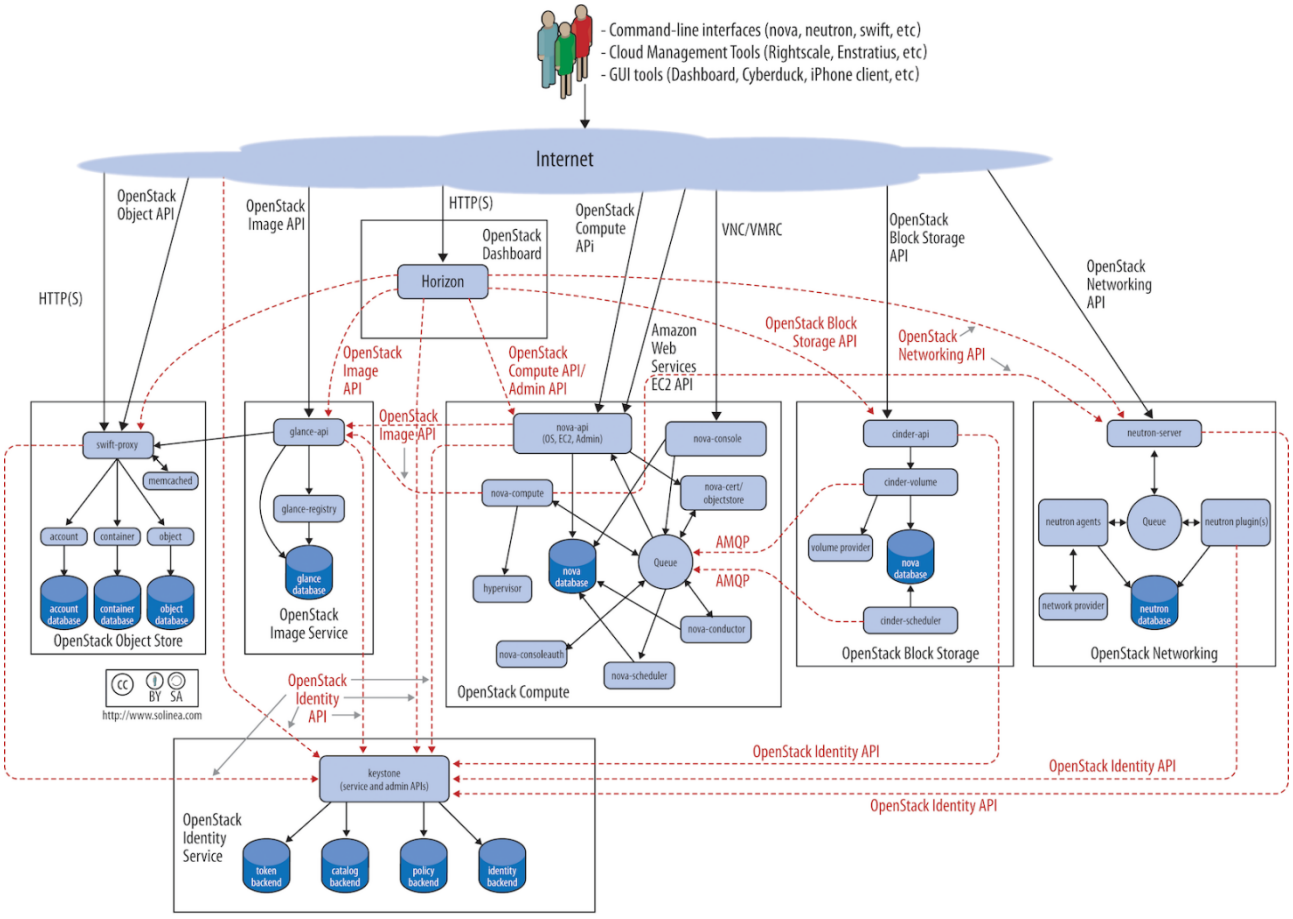


Kuvio 3. OpenStack-järjestelmän komponenttikartta (OpenStack n.d.)

OpenStackin kaltaisia palveluita on todella monia kuten VMware vCloud sekä OpenNebula (Sinhoreli 2023). OpenStack on järjestön omistuksessa eikä se ole kaupallinen tuote, joten sen käyttäminen on ilmaista. OpenStackille ei ole tukea kuten monelle muulle samanlaiselle palvelulle. Tämän takia yrityksellä, joka käyttää OpenStackia, täytyy olla osaamista sen konfiguroimiseen ja käyttöönottoon. Koska OpenStack on monipuolinen ohjelmistokokonaisuus, niin sen käyttöönotto voi olla hieman monimutkaisempaa kuin monen muun samanlaisen palvelun. (Sultan 2024.)

OpenStackia on mahdollista käyttää monissa eri palvelumalleissa. Yleisin tapa OpenStackin käyttöön on IaaS. Tämä johtuu siitä, että OpenStackin vahvuudet ja suosio keskittyvät erityisesti infrastruktuuripalveluiden tarjoamiseen. OpenStackin useiden komponenttien ansiosta palveluntarjoajan on mahdollista valita malli, joka sopii parhaiten kuhunkin käyttötarkoitukseen. (is openstack iaas or paas 2024.)

OpenStackilla on mahdollista luoda monenlaisia eri palvelukokonaisuuksia, sillä se on hyvin mukautuva ohjelmistokokonaisuus. Yksi esimerkki mukautuvuudesta on IaaS-palvelumallin mukainen arkkitehtuurimalli (ks. kuvio 4), jossa loppukäyttäjä voi käyttää OpenStackin resursseja nettikäyttöliittymän kautta. Nettikäyttöliittymänä käytetään ”Horizon”-komponenttia. Komponenttien välinen kommunikointi hoidetaan API-rajapintojen kautta. (Design 2018.)



Kuvio 4. OpenStackin yleisin arkkitehtuurimalli (Design 2018)

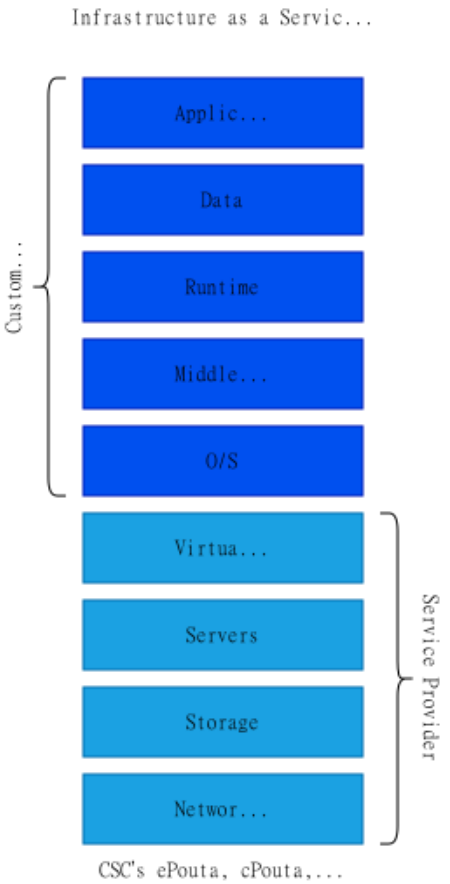
Tällä hetkellä CSC:n cPoudassa on käytössä OpenStackin versio Queens (Tahir & Gonzales n.d., 21). cPoudan nettikäyttöliittymänä toimii Horizon (OpenStack command line client tools for Pouta 2024). Nettikäyttöliittymän kautta on helppo saada yleiskatsaus resursseista sekä luoduista virtuaalikoneista. Virtuaalikoneiden luonti sekä niiden päivittäminen käy kuitenkin helpoiten Heat Templateilla sekä OpenStackin komentorivityökalulla.

OpenStackin Heat Templateilla voidaan automatisoida infrastruktuurin hallinta. Heat Templatejen avulla on mahdollista hallita infrastruktuurin resursseja koodina. Heat Templatet ovat YAML-muotoisia tiedostoja. Näihin tiedostoihin määritellään ympäristön tarvitsemat resurssit ja asetukset, kuten palvelimet, verkot ja tallennustilat. Heat Templateihin on mahdollista määritellä monimutkaisia ja isojakin ympäristöjä. Heat Templateilla ympäristöjen pystyttämisestä tulee helppoa, virheettömämpää sekä toistettavaa. (Orchestration Service: OpenStack Heat 2018.)

2.6 cPouta

cPouta on CSC:n tuottama pilvipalvelu, joka perustuu OpenStackin avoimeen lähdekoodiin. cPouta on suomalainen palvelu, joka tarjoaa käyttäjille mahdollisuuden pystyttää ja hallinnoida omia virtuaalikoneitaan. cPoutaan luodussa ympäristössä on mahdollista testata erilaisia sovelluksia ja ohjelmistoja sekä käyttää ympäristöä sovelluksien ja palveluiden kehittämiseen. Käyttö onnistuu sekä verkkoselaimen että ohjelmointirajapinnan (API) kautta. (cPouta n.d.)

cPouta toimii IaaS (Infrastructure as a Service) -mallilla (Tahir & Gonzales n.d., 19). Tämä tarkoittaa, että asiakkaan on mahdollista vuokrata IT-infrastruktuuria, kuten palvelimia, tallennustilaa ja verkkoresursseja pilvipalveluntarjoajalta (ks. kuvio 5). Vuokraamalla resurssit palveluntarjoajalta palvelun tuottajan ei tarvitse itse hankkia eikä ylläpitää kalliita laitteita. Tällä tavalla palvelun tuottajan on mahdollista vähentää kuluja. Käytettäessä IaaS-mallia on palvelua mahdollista skaalata sen tarpeiden mukaan. (Hashemi-Pour & Bigelow 2024.)



Kuvio 5. IaaS-palvelumalli (Tahir & Gonzales n.d., 14)

cPoudan virtuaalikoneille voidaan määritellä julkiset IP-osoitteet, jolloin virtuaalikoneet ovat saavutettavissa internetin välityksellä (Pouta 2024). Tämä mahdollistaa palveluiden integroinnin muihin julkisiin palveluihin.

Virtuaalikoneen luonti onnistuu cPoudassa verkkoselaimen tai API:n kautta. Virtuaalikonetta luodessa voidaan määrittää SSH-yhteyteen käytettävä SSH-avain sekä valita virtuaalikoneella käytettävät Security Group -säännöt. Virtuaalikoneelle voi valita joko cPoudasta valmiina olevan käyttöjärjestelmän tai tuoda cPoutaan oman käyttöjärjestelmän. Virtuaalikoneiden suojaaminen cPoutapalvelussa onnistuu Security Group -toiminnon avulla. Virtuaalikoneella voi olla yksi tai useampi Security Group -sääntö, joka määrittelee palomuurin asetukset. (Creating a virtual machine in Pouta 2024.)

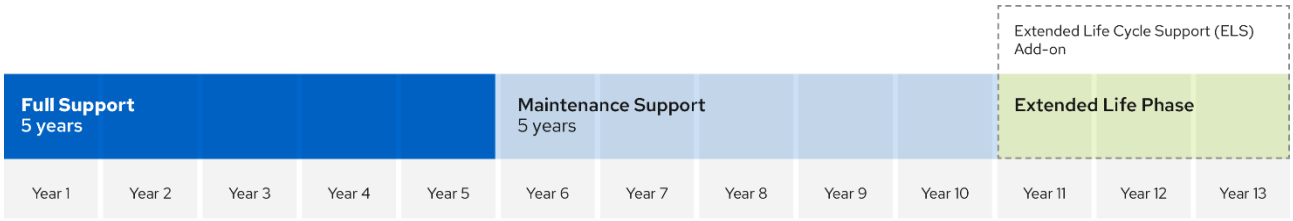
2.7 Käyttöjärjestelmä (RHEL9)

Testiympäristöjen käyttöjärjestelmänä käytettiin RHEL9 eli Red Hat Enterprise Linux 9 -käyttöjärjestelmää. RHEL9 on avoimen lähdekoodin Linux-käyttöjärjestelmä, joka julkaistiin toukokuussa 2022 (Cattelain 2022). RHEL9 on suunniteltu täyttämään hybridipilviympäristön tarpeet. Se on suunniteltu toimimaan virtuaalikoneilla pilvipalveluympäristössä. Tämä käyttöjärjestelmä on mahdollista asentaa myös fyysiselle laitteelle. (Red Hat Enterprise Linux 9 is now available n.d.)

Ennen testiympäristön päivittämistä uuteen käyttöjärjestelmään oli testiympäristössä käytössä CentOS Linux 7. CentOS 7 -käyttöjärjestelmä on rakennettu Red Hatin lähdekoodin pohjalta (What is CentOS Linux? n.d.). CentOS 7 -käyttöjärjestelmä julkaistiin vuonna 2014. CentOS 7 -käyttöjärjestelmällä ei ollut teknistä tukea kuten RHEL-käyttöjärjestelmällä. Mutta CentOS 7:n käyttäminen ei vaatinut maksullista tilausta kuten RHEL-käyttöjärjestelmä. (CentOS 7 n.d.).

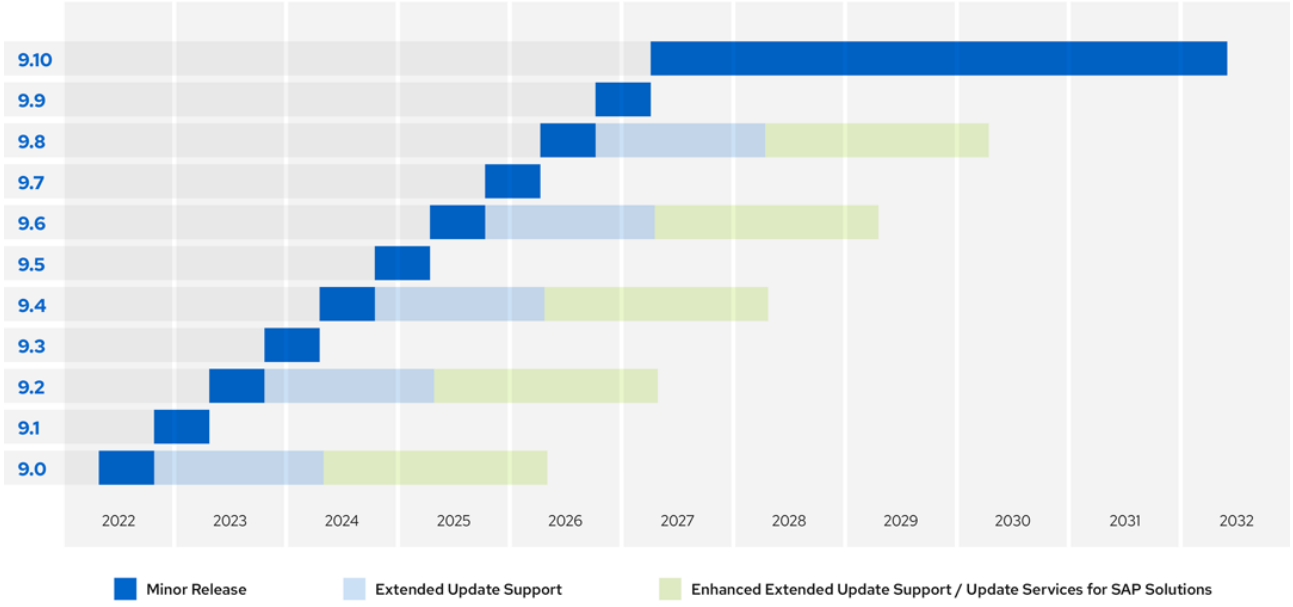
Vuonna 2020 ilmoitettiin, että CentOS Linuxin päivittäminen loppuisi ja Red Hat alkaisi keskittymään CentOS Streamin kehittämiseen. CentOS Stream on RHEL-alustojen tuotantoketjun alkupää. Eri CentOS-versioiden tuki loppui vuosien 2021 ja 2024 välillä. Viimeisin CentOS:n päivitys tuli kesäkuun lopussa 2024 CentOS 7 -käyttöjärjestelmälle. (What to know about CentOS Linux EOL 2024.)

RHEL9-käyttöjärjestelmä lisää huomattavasti turvallisuutta, sillä se saa säännöllisesti turvallisuus-päivityksiä. RHEL9-käyttöjärjestelmän tuki jatkuu melko pitkään (ks. kuvio 6). Käyttöjärjestelmän täysi tuki jatkuu vuoteen 2027 asti, jonka jälkeen käyttöjärjestelmä saa vielä huoltopäivityksiä vuoteen 2032 saakka. Käyttöjärjestelmän käyttäminen on mahdollista tämänkin jälkeen. Tekninen tuki jatkuu jo valmiiksi asennetuille käyttöjärjestelmille vuoteen 2035. Vuoden 2035 jälkeen ei ole saatavissa enää mitään tukea. (Red Hat Enterprise Linux Life Cycle n.d.)



Kuvio 6. RHEL9-käyttöjärjestelmän elinkaari (Red Hat Enterprise Linux Life Cycle n.d.)

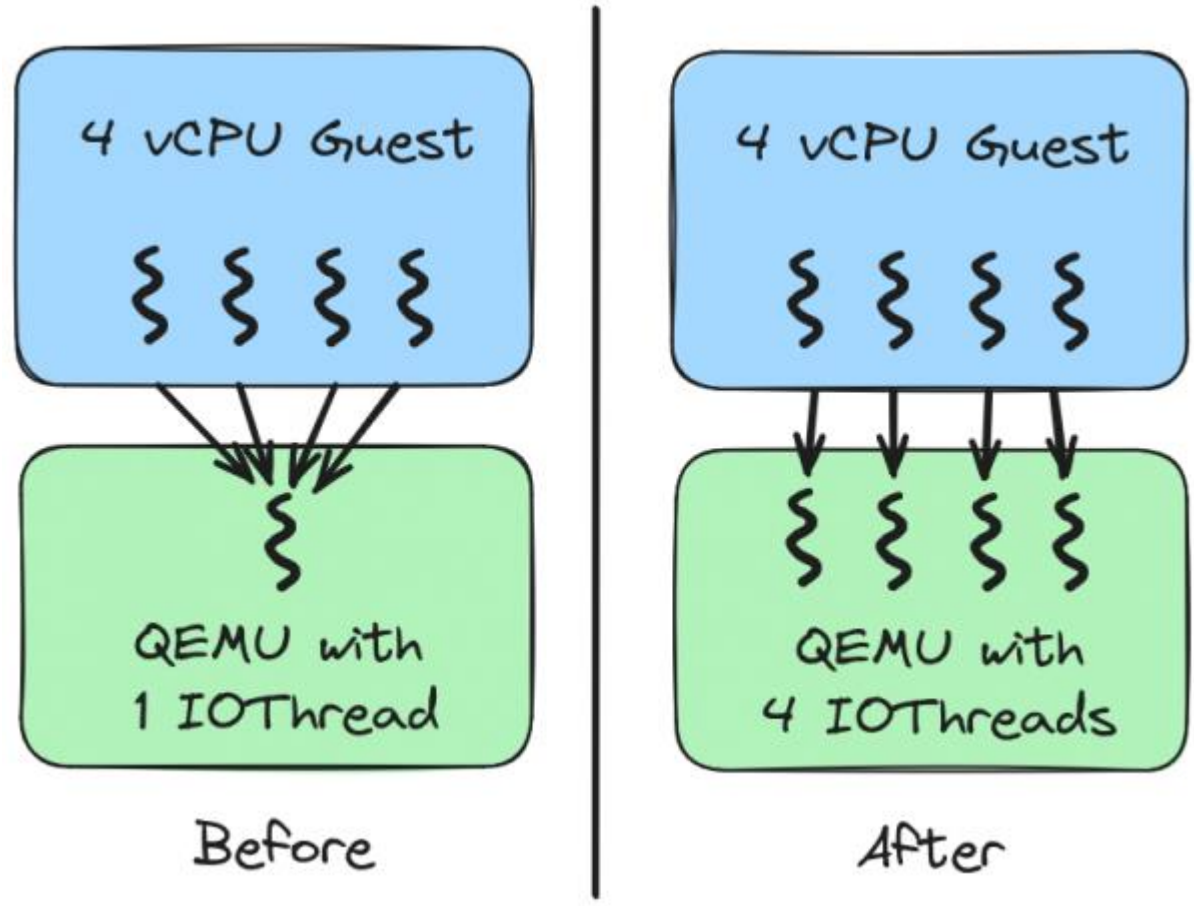
RHEL9-versoina käytettiin viimeisintä 9.4-versiota. Tämä versio ei kuitenkaan pysy tuettuna koko RHEL9:n elinkaarta (ks. kuvio 7) (Red Hat Enterprise Linux Life Cycle n.d.). Tämän vuoksi testiympäristö täytyy päivittää taas uuteen versioon, kun 9.4-version tuki loppuu.



Kuvio 7. Eri RHEL9-versioiden tuki (Red Hat Enterprise Linux Life Cycle n.d.)

RHEL9:llä on myös käytössä SELinux. SELinuxin avulla on mahdollista parantaa testiympäristön turvallisuutta. SELinuxin avulla voidaan rajoittaa prosessien, käyttäjien sekä ohjelmien käyttöoikeuksia. Tällä tavalla on mahdollista estää luvaton pääsy järjestelmän tietoihin. Prosesseja on myös mahdollista erotella toisistaan, jolloin prosessin haavoittuessa on mahdollista suojata koko muu järjestelmä. (Using SELinux n.d.)

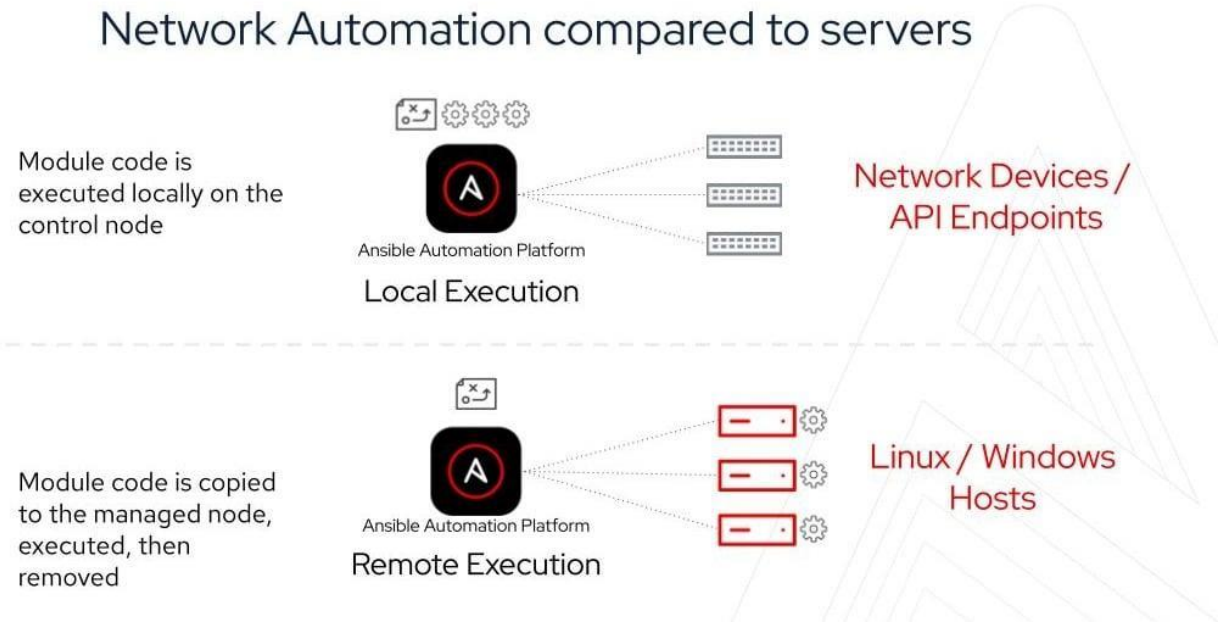
RHEL9-käyttöjärjestelmällä on parannettu suorituskyky aiempiin RHEL-käyttöjärjestelmiin verrattuna. Esimerkiksi QEMU-virtualisoinnin suorituskykyä on parannettu. Aikaisemmin QEMU-virtuaalikone hallinnoi I/O-pyyntöjä yhdellä säikeellä käytettäessä useita vCPU:ita. Tämä saattoi johtaa prosessien hitaampaan suorittamiseen. Nykyään RHEL9.4-virtuaalikoneissa on kuitenkin mahdollista hyödyntää vCPU:ita paremmin. RHEL9.4-käyttöjärjestelmäversion ansiosta QEMU-virtualisoinnissa on mahdollista käyttää useaa I/O-säiettä, kun käytetään useita vCPU:ita (ks. kuvio 8). (Sanjay & Hajnoczi 2024.)



Kuvio 8. Säikeiden käyttö käytettäessä useaa vCPU:ita (Sanjay & Hajnoczi 2024)

2.8 Ansible

Ansible on tehokas avoimen lähdekoodin automaatiotyökalu, joka on tehty Python-ohjelmointikielellä. Se on suunniteltu helppokäyttöiseksi ja luotettavaksi. Ansiblen vahvuuksia ovat sen helppokäyttöisyys sekä luotettavuus. Ansiblella on mahdollista mm. konfiguroida ympäristöjä, ottaa käyttöön ohjelmistoja ja suorittaa ympäristön päivityksiä. Ansible toimii SSH:n kautta. Ansiblella onnistuu useiden koneiden hallinta (ks. kuvio 9). (How Ansible works n.d.)



Kuvio 9. Usean koneen hallinta Ansiblen avulla (How Ansible works n.d.)

Ansible koostuu kahdesta pääkomponentista ”inventory” ja ”playbook”. Näillä kahdella tiedostalla voidaan määrittää, mitä tehtäviä ajetaan kullakin koneelle. Inventory määritellään INI-muodossa. Tässä tiedostossa määritellään kaikki hallittavat koneet, joita Ansible-skripti hallitsee (ks. kuvio 10). (How Ansible works n.d.)

```
[webservers]
www1.example.com
www2.example.com

[dbservers]
db0.example.com
db1.example.com
```

Kuvio 10. Esimerkki Ansible-skrptin inventory-tiedostosta (How Ansible works n.d.)

Playbookiin kirjoitetaan YAML-muotoiseen tiedostoon. Tässä tiedostossa määritellään kaikki tehtävät, joita Ansiblella suoritetaan. Kun inventory ja playbook on luotu, voidaan playbook ajaa komentoriviltä, jolloin playbookissa olevat tehtävät suoritetaan inventoryssa määritellyille koneille (How Ansible works n.d.) Ansible playbookiin on mahdollista tuoda toisia playbookeja, jolloin on mahdollista suorittaa useita playbookeja yhdellä playbookilla (ansible.builtin.import_playbook module – Import a playbook 2024).

Ansible playbookissa on mahdollista käyttää myös rooleja Ansiblen roolien avulla voi käyttää erilaisia valmiiksi määriteltyjä resursseja, kuten muuttujia, tiedostoja sekä tehtäviä. Roolit vaativat tietyn tiedostorakenteen toimiakseen. Rooleja on helppo siirtää ja jakaa muihin Ansible playbookeihin. (Roles 2024.) Rooleilla yleensä hoidetaan yksi kokonaisuus, kuten jonkun palvelun asennus ja konfigurointi. Tällöin saadaan organisoitua Ansible-koodia selkeisiin osiin. (Ansible Roles 2024.)

2.9 Docker

Docker on ohjelmistoalusta. Dockerilla voidaan paketoida ohjelmistoja kevyiksi ja helposti siirrettäviksi konteiksi. Kontit sisältävät kaiken mitä ohjelma tarvitsee toimiakseen. Tämän takia Docker-kontit toimivat riippumatta missä ympäristössä niitä ajetaan. Docker-kontit mahdollistavat myös palvelun skaalaamisen. Docker-kontteja on mahdollista luoda useita samalle koneelle. (What is Docker?. n.d.a.) Docker on mahdollista ajaa työasemilla, pilvipalveluissa tai hybridiympäristöissä. Tämän takia Docker sopi erinomaisesti käytettäväksi CI/CD-putkessa. (What is Docker? n.d.b.)

Docker-konttien merkittävä etu perinteiseen virtualisointiin verrattuna on niiden keveys ja tehokkuus. Virtuaalikoneet sisältävät kokonaisen käyttöjärjestelmän, kun kontit sen sijaan jakavat isän-

täkoneen resursseja. Saman sovelluksen ajaminen virtuaalikoneella voi kuluttaa noin 30 % enemmän resursseja konttiratkaisuun verrattuna. (Wallenius 2022.)

3 Ympäristöjen pystytys

3.1 Lähtökohdat

KH1 Haasteena oli testiympäristöjen pystyttäminen. Pystytettävät ympäristöt tulisivat olemaan hyvin samanlaisia kuin aiemmin CSC:llä pitkäaikaissäilytyksessä käytetyt testiympäristöt. Tämän vuoksi ympäristön pystyttämiseen oli suunniteltu käytettäväksi OpenStackin Heat Templateja. Lisäksi Heat Templaten avulla olisi mahdollista pystyttää koko testiympäristö kerralla. Ympäristöjen alustus oli aiemmin tehty Ansiblella, joten uusien ympäristöjen alustukseenkin päätettiin käyttää Ansiblea.

cPoudassa ei ollut valmista RHEL9-levykuvaa. Tämän luontiin ei ollut vielä suunniteltu tapaa, jolla levykuva luotaisiin. Päädyin kuitenkin käyttämään KVM-työkalua RHEL9-käyttöjärjestelmän asentamiseksi sekä levykuvan tekemiseksi. Työn alussa levykuvalle käytettävää muotoa ei ollut vielä päätetty. Ympäristöä pystyttäessä tulisi testata, mikä levykuvan muoto sopisi parhaiten cPoudassa käytettäväksi.

3.2 Käyttöjärjestelmän asennus ja levykuvan luonti

CSC:n testiympäristö on CSC:n cPouta-palvelussa. Edellisessä testiympäristössä käytettiin CentOS Linux 7 -käyttöjärjestelmää. CentOS Linux 7 -käyttöjärjestelmä vanhentui 30.6.2024, jonka jälkeen käyttöjärjestelmä ei saa enää päivityksiä (What to know about CentOS Linux EOL 2024). Käyttöjärjestelmän vanhentumisen takia testiympäristön käyttöjärjestelmä täytyi päivittää. Päivittämällä testiympäristö uudempaan versioon on mahdollista testata RHEL9-käyttöjärjestelmälle tehtyjä RPM-paketteja. Tietoturvallisuuskin on tärkeässä osassa RHERL9-siirtymää. Ilman RHEL9-siirtymää ei olisi mahdollista varmistaa ohjelmistojen eikä testiympäristön tietoturvallisuutta.

Uudeksi käyttöjärjestelmäksi testiympäristölle valittiin RHEL9-käyttöjärjestelmä. RHEL9-käyttöjärjestelmä on uusin Red Hatin julkaisema käyttöjärjestelmäversio. Koska kaikki CSC:n pitkäaikaissäilytyksen ohjelmistot oli kehitetty CentOS 7 -käyttöjärjestelmälle, ohjelmistojen migraatio

RHEL9-käyttöjärjestelmälle oli mahdollisimman vaivatonta, sillä näiden kahden käyttöjärjestelmän arkkitehtuuri on hyvin lähellä toisiaan. Vaikka pakettien migraatio aiheuttikin paljon työtä, oli siirtymä välttämätön. RHEL9-siirtymän avulla mahdollistetaan, että CSC:n pitkäaikaissäilytyksen ohjelmistoja voidaan käyttää tulevaisuudessakin.

Testiympäristön virtuaalikoneet pystytettiin cPoutaan. Virtuaalikoneille käytettiin RHEL9-käyttöjärjestelmää. RHEL9-käyttöjärjestelmän asennusmedia ladattiin Red Hatin omilta sivuilta. Asennusmedia oli ISO-tiedostona. Asennusmedian avulla oli mahdollista luoda uusi levykuva asennetusta käyttöjärjestelmästä. Uutta levykuvamuotoa varten kokeiltiin paria erilaista levykuvan muotoa. Näitä olivat raw- ja qcow2-muotoiset levykuvat.

Raw-muotoinen levykuva toimi hieman epävakaasti. Välillä raw-levykuvaa käyttäessä virtuaalikone ei käynnistynyt oikein ja jäi virhetilaan. Syytä tälle virheelle ei saatu selville. Päädyttiin siihen tulokseen, että qcow2-muotoinen levykuva toimisi cPoudassa parhaiten. Qcow2-muotoinen levykuva toimi joka kerralla oikein. Lisäksi Qcow2-muotoinen levykuva oli myös CSC:llä enemmän käytetty levykuvamuoto. Asennusmedia asennettiin käyttäen KVM (Kernel-based Virtual Machine) virtualisointia. Käyttöjärjestelmän asennuksen yhteydessä levy osioitiin ja luotiin virtuaalimuistiosio. Tämän jälkeen luotiin qcow2-muotoinen levykuva virtuaalikoneen levystä. Näin levykuva olisi valmiina käyttöä varten, eikä virtuaalikonetta luodessa tarvitsisi enää tehdä alustusta. Levykuvan alustuksen ja levykuvan oikeaan muotoon viemisen jälkeen se ladattiin cPouta-palveluun. Siten se olisi valmiina käyttöä varten, kun luodaan virtuaalikoneita.

3.3 Heat Stackin luonti ja ympäristön konfigurointi

Virtuaalikoneiden pystytys suoritettiin OpenStack-rajapinnan kautta. Virtuaalikoneiden luonti suoritettiin luomalla Heat Template -tiedosto. Tähän tiedostoon määriteltiin kaikki virtuaalikoneen luontiin tarvittavat resurssit ja konfiguraatiot, kuten käytettävä SSH-avain, virtuaalikoneen nimi, virtuaalikoneen IP-osoite sekä virtuaalikoneelle asennettava käyttöjärjestelmä. SSH-avaimena käytin aikaisemmekin cPouta-palveluun määrittelemääni SSH-avainta.

Heat Template -tiedosto luotiin jokaiselle pystytettävälle virtuaalikoneelle. Paikallisen ja hajautetun testiympäristön virtuaalikoneet sekä resurssit määriteltiin yhteen Heat Stackiin. Tiimin kanssa päädyttiin tekemään asiakastestiympäristöä varten jokaiselle koneelle oma Heat Stack. Tällä taval-

la virtuaalikoneita olisi helppo hallita erikseen, esimerkiksi jos haluaisi uudelleen luoda vain yhden virtuaalikoneen. Joillekin koneille luotiin myös erillinen tallennustila. Näin koneita uudelleen luodessa tallennustila säilyisi entisellään, eikä sitä tarvitsisi luoda uudelleen. Security Group -säännöt näille virtuaalikoneille määriteltiin kuitenkin yhdessä tiedostossa. Tämä päätettiin tehdä yhtenä Heat Stackina, sillä Security Group -sääntöihin ei tarvitse juurikaan tehdä muutoksia, kun ne on kerran asetettu. Security Group -säännöillä määritellään ulkoisten organisaatioiden pääsy virtuaalikoneille.

Heat Template -tiedostojen luonnin jälkeen virtuaalikoneet ja tallennustilat voitiin luoda cPouta-palveluun OpenStackin Api-rajapintaa käyttäen. OpenStackin rajapintaa on mahdollista käyttää monella tavalla. Rajapintaa vai käyttää OpenStackin komentorivityökalulla, HTTP-pyynnöllä tai käyttämällä jonkun ohjelmistokielen moduulia. Tässä työssä käytettiin OpenStackin komentorivityökalua. Tämän avulla Heat Stackin luonti olisi mahdollista suorittaa automaattisesti Heat Templaten avulla.

Ennen virtuaalikoneiden asentamista cPouta-palveluun oli luotava tunnistautumistiedosto. Tiedoston luonti tapahtui Makefilen taskilla. Task on määritetty kysymään käyttäjätunnuksen sekä salasanan, jonka jälkeen se luo ”secure.yaml”-tiedoston. Tätä tiedostoa voi käyttää tunnistautumisessa cPouta-palveluun, kun käytetään OpenStackin Api-rajapintaa. OpenStackin komentorivityökalulla Heat Stackin luonti onnistui todella helposti, kun tunnistautumistiedosto sekä Heat Template oli määritelty. Virtuaalikoneen luontiin vaadittiin vain yksi ”create”-komento, johon määriteltiin luotava Heat Template sekä tunnistautumistiedosto.

”Create”-komento voi olla esim. seuraavanlainen: *”openstack --os-cloud kdkpas stack create -e environment.yaml -t template.yaml <STACK>*”. Tässä ”*openstack*” on työkalu, jota käytetään komennon ajamiseen. *”--os-cloud kdkpas*”-optiolla määritellään, mitä projektia halutaan hallita. Tässä tapauksessa hallitaan ”*kdkpas*”-projektia. Projektit on määritelty ”*cloud.yml*”-tiedostoon, jossa on viittaus kirjautumistiedostoon ”*secure.yaml*”, jota puolestaan käytetään tunnistautumiseen projektia käytettäessä. ”*stack*” määrittelee, mitä halutaan hallita. Tässä tapauksessa halutaan hallita koko stackia. ”*create*”:lla määritellään mitä stackille halutaan tehdä. ”Create”-komennolla voidaan luoda uusia stackeja.

Muita mahdollisia toimintoja on mm. *delete*, *update*, *list* ja *show*. Kaikille näille komennoille voi antaa erilaisia argumenttejä, kuten tässä tapauksessa ”*create*”-komennolle on määritelty ”*-e*” (environment) ja ”*-t*” (template) argumentit. Environment määrittelee environment-tiedoston, jossa on stackin luomiseen kaikki tarvittavat muuttujat. Template määrittelee Heat templaten, jossa on ohjeet stackin luomiseen yaml-muodossa. Lopuksi ”*<STACK>*” osuudessa määritellään luotavan stackin nimi. OpenStackin komentorivityökalulla on mahdollista tehdä kaikki samat asiat ja enemmänkin kuin OpenStackin nettikäyttöliittymällä (Horizon). Komentorivityökalu on tehokkaampi ja skaalautuvampi tapa hallita OpenStackin ympäristöä.

Kun virtuaalikoneet ja niiden tarvitsemat resurssit oli saatu asennettua cPoutaan, voitiin aloittaa testiympäristön alustaminen. Virtuaalikoneiden alustus suoritetaan Ansible-koodilla, joka asentaa koneille tarvittavat SSH-avaimet, RHEL9-lisenssin sekä testiympäristön tarvitsevat ohjelmistot. Testiympäristön alustamiselle oli olemassa valmis Ansible-koodi. Tämä ei kuitenkaan semmoisenaan käynyt, vaan Ansible-koodia piti päivittää RHEL9-käyttöjärjestelmälle yhteensopivaksi.

Myös testiympäristön siivousskripti tuli päivittää testikoneiden päivityksen myötä. Skriptien päivityksen suoritettiin omalla paikallisella virtuaalikoneella, jolla pystyttiin testaamaan asennus- ja siivousskriptejä. Siivousskriptiin tuli jonkin verran UID:iden (käyttäjätunniste) ja GID:iden (ryhmätunniste) muokkauksia sekä ohjelmien nimien muutoksia, jotka oli päivittynyt uuden käyttöjärjestelmän myötä. Testiympäristön alustusskriptiin tuli enemmän muutettavaa. Suurimpia muutoksia olivat mm. RHEL9-lisenssin asentaminen, ohjelmien versioiden ja nimiin tulleet muutokset sekä joidenkin pakettivarastojen asennuksien suorittamien eri tavalla. Koska testiympäristössä oli käytössä yksi ALMA9-käyttöjärjestelmällä oleva virtuaalikone, piti skriptiä päivitettäessä ottaa huomioon se, että alustus- ja siivousskriptit toimisivat kummallakin käyttöjärjestelmällä.

Myös testiympäristön päivityksen alkuvaiheessa piti ottaa huomioon, että skriptit toimisivat myös CentOS 7 -käyttöjärjestelmällä, sillä ennen täyttä RHEL9-siirtymää oli myös vanha testiympäristö pidettävä toimivana. Tämä aiheutti skriptin kanssa hieman hankaluuksia, kun sama skripti piti saada toimimaan kolmella eri käyttöjärjestelmällä. Myöhemmin, kun CentOS 7 -käyttöjärjestelmästä siirryttiin pois, voitiin skripteistä siivota vanhalle CentOS 7 -käyttöjärjestelmälle tarkoitettu koodi. Skriptien valmistuttua testiympäristön alustus sekä siivoaminen onnistuivat yhdellä komennolla.

3.4 Runnerit

Paikallisen testiympäristön virtuaalikoneet sekä niiden sisällä olevat Docker-kontit rekisteröidään runnereiksi GitLabiin. Runnereiden avulla on mahdollista suorittaa CI/CD-putken eri vaiheita. Tässä tapauksessa Docker-kontteja käytetään kasaamaan RPM-paketteja ja virtuaalikoneilla suoritetaan CI/CD-putken automaattiset testit. Runnereiden rekisteröiminen ja konttien asentaminen hoidetaan Ansible-skriptillä.

GitLab osaa hallita automaattisesti runnereille annettuja töitä. Kaikki yksikkötestit voidaan ajaa kaikilla RHEL9-runnereilla. Hajautettuihin testeihin on kuitenkin määritelty käyttöön vain yksi runneri. Käyttämällä yhtä runneria hajautetussa testissä varmistetaan, että vain yksi testi pystyy käyttämään hajautetun testiympäristön resursseja kerrallaan.

3.5 Paikallinen testiympäristö

Paikallisen testiympäristön tavoitteena olisi pystyä testaamaan yksittäisiä ohjelmistokomponentteja ja niiden toimivuutta. Ohjelmistokomponentteja testattaessa ladataan luotu RPM-paketti ja asennetaan se ja paketin riippuvuudet testikoneelle. Ohjelman asennuksen jälkeen suoritetaan paketille ennalta määritetyt testit.

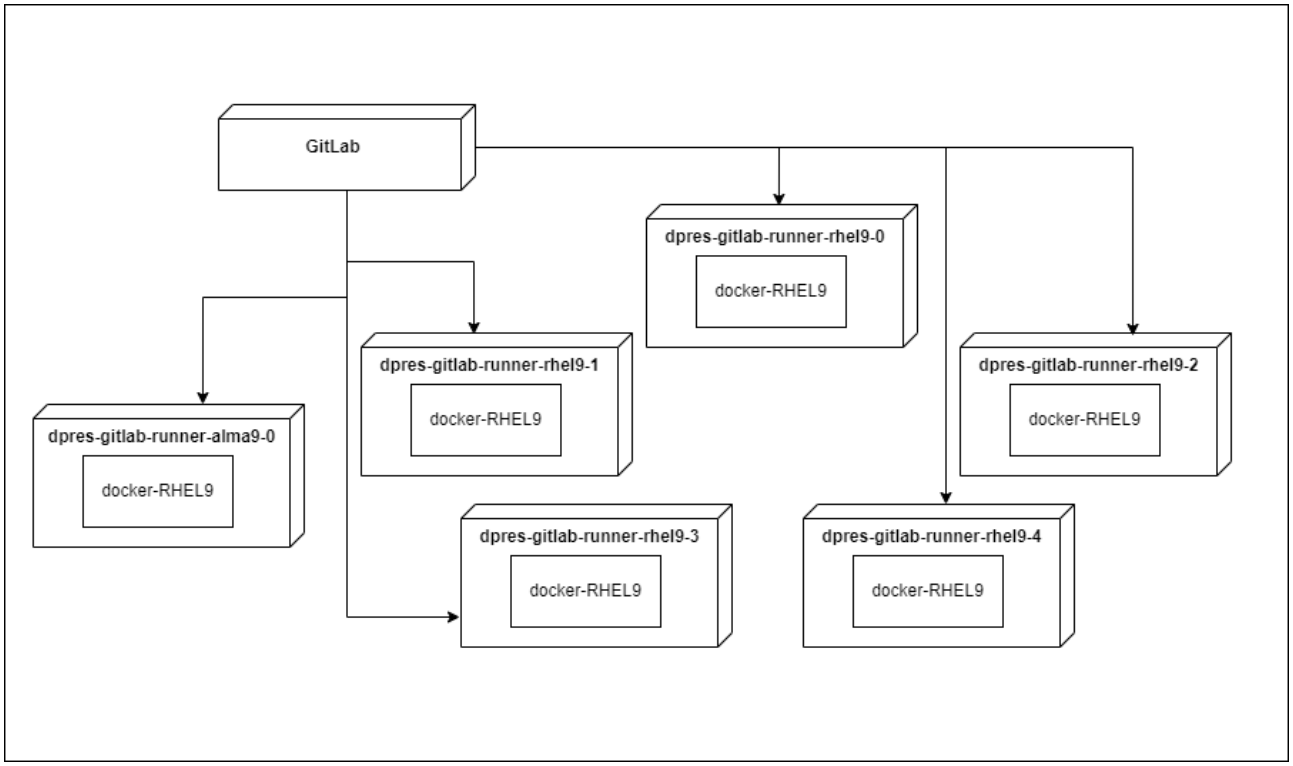
Jokaiselle paikallisen testiympäristön virtuaalikoneelle asennetaan Docker-kontit, jotka hoitavat RPM-pakettien kasauksen. RHEL9-alustalle tarkoitetut paketit kasataan käyttämällä Alma9-pohjaista Docker-konttia. CentOS 7 -alustalle tarkoitetut paketit kasataan CentOS 7 Docker-kontilla. Rhel9-siirtymän jälkeen CentOS 7 Docker-kontti tulee tarpeettomaksi, sillä kun RHEL9-siirtymä on saatu kokonaan valmiiksi, niin CentOS 7 RPM-paketin testaus lopetetaan CSC:llä.

Paikallisen testiympäristö pystytettiin cPoutaan käyttämällä OpenStackin komentorivityökalua. Virtuaalikoneiden pystyttämisen jälkeen koneet alustettiin testikäyttöä varten Ansible-skriptillä. Alustuksen jälkeen koneille ajettiin vielä yksi Ansible-skripti. Tämä skripti asensi jokaiselle koneelle Docker-kontin, jolla RPM-paketit voidaan pakata. Skripti myös rekisteröi koneet ja Docker-kontit GitLabin runnereiksi. Näin virtuaalikoneita ja Docker-kontteja on mahdollista käyttää GitLabin CI/CD-putkessa.

Paikallisessa testiympäristössä testataan vain yhtä ohjelmistokomponenttia. Kun ohjelmistokomponenttiin tehdään muutos GitLabissa, aloittaa se automatisoidun CI/CD-putken. Ensimmäisessä vaiheessa ohjelmisto ja sen muutokset kasataan RPM-paketiksi käyttäen runnereilla olevaa Docker-konttia. Kun RPM-paketti on onnistuneesti luotu, siirrettään se CSC:n RPM-pakettivarastoon.

Tämän jälkeen voidaan aloittaa testausvaihe. Testausvaiheessa valitaan vapaana oleva runneri, jolla testi suoritetaan. GitLab osaa hoitaa itse vapaana olevan runnerin valitsemisen. Ennen testien aloittamista runneri tyhjennetään ja sen jälkeen alustetaan testikäyttöä varten. Tyhjennys ja koneen alustus tehdään Ansible-skripteillä. Nämä kaksi skriptiä suoritetaan automaattisesti aina ennen testien ajamista. Näin varmistetaan, että testiympäristö pysyy samanlaisena jokaista testiä varten eikä testiin tule aikaisemmin ajetusta testeistä tulleita muutoksia, jotka voivat johtaa vääriin tuloksiin. Runnerille ladataan uusin RPM-paketti, johon muutokset on tehty. Tämän jälkeen paketti asennetaan koneelle. Jokaiselle paketille on määritelty omat testit. Testit on tehty käyttämällä pytest-moduulia. Monissa paketeissa on kahdet paikalliset testit. Osa testeistä ajetaan venvin sisällä, joka on Pythonin virtuaaliympäristö. Loput CI/CD-putken testeistä suoritetaan suoraan testikoneella. Testien tulokset raportoidaan GitLabiin. Jos nämä testit menevät läpi, voidaan jatkaa seuraavaan testausvaiheeseen. Jos testit eivät kuitenkaan mene läpi, niin CI/CD-putki pysäytetään ja virtuaalikone vapautetaan seuraavaa työtä varten.

Paikallinen testiympäristö koostuu viidestä RHEL9-virtuaalikoneesta sekä yhdestä Alma9 Linux -testikoneesta (ks. kuvio 11). Jokaisella runnerilla on Docker-kontti RPM-pakettien luontia varten. Alma9 Linux -testikoneella testataan vain osa paketeista, ja sen takia ympäristöön ei tarvita kuin yksi Alma9-kone.



Kuvio 11. Paikallisen testiympäristön topologia

3.6 Hajautettu testiympäristö

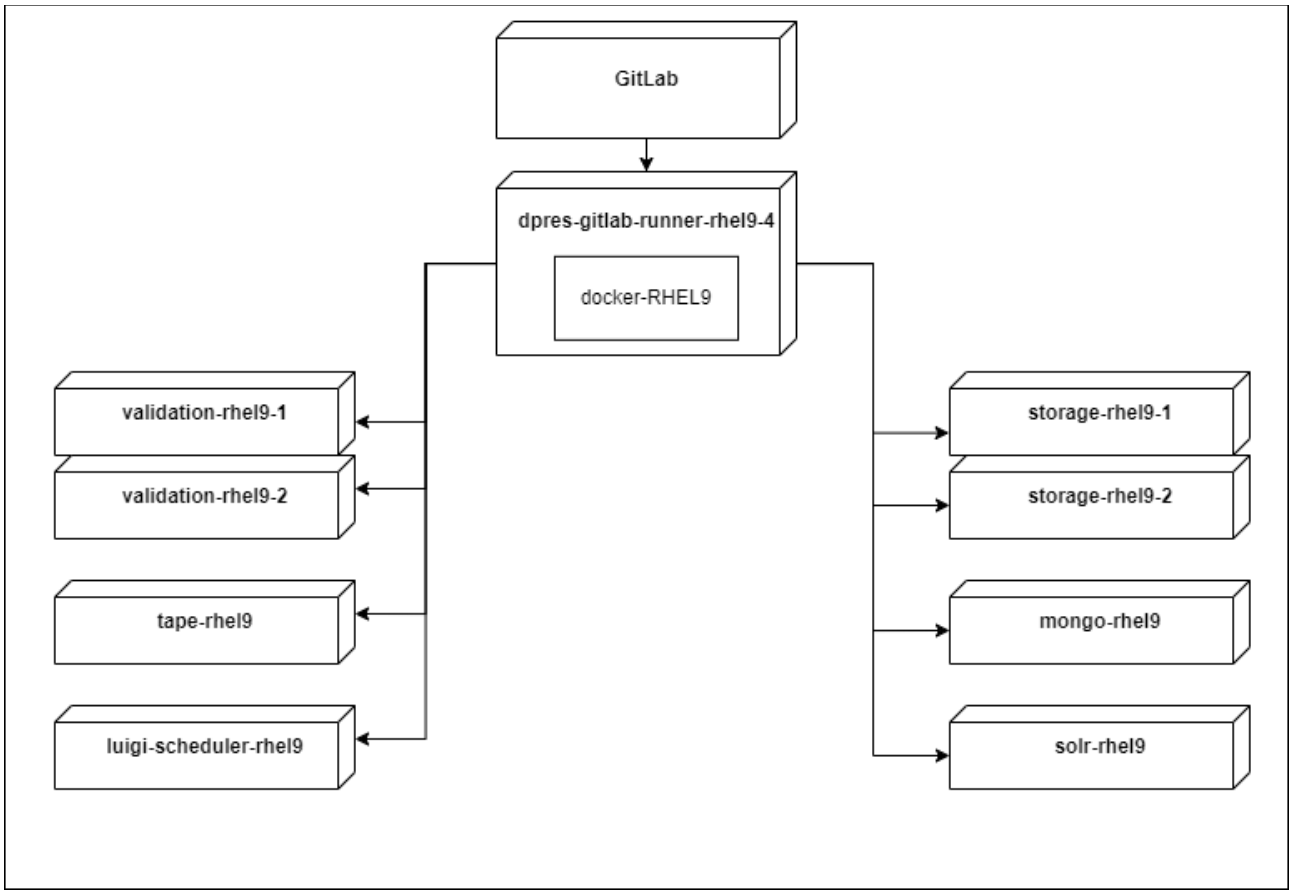
Hajautetussa testiympäristössä ajetaan integraatiotestit sekä End-to-End-testit (E2E). Integraatiotestillä on tarkoituksena testata pakettien toimivuus keskenään. E2E-testien tarkoituksena on testata ohjelmiston kokonaisuutta. Näillä testeillä varmistetaan, että kaikki ohjelmistokomponentit toimivat keskenään oikein. Hajautettu testi on toinen testausvaihe automatisoidussa CI/CD-putkessa.

Hajautetussa testiympäristössä käytetään ”dpres-gitlab-runner-rhel9-4”-runneria. Hajautetussa testiympäristössä on käytössä vain yksi runneri. Näin varmistetaan, ettei toinen testi sotke testituloksia käyttämällä testeissä käyttäviä resursseja samanaikaisesti. Tämä hieman hidastaa hajautettujen testien suorittamista, mutta cPoudan resurssien takia ei ole mahdollista pystyttää kuin yksi hajautettu testiympäristö.

Hajautetun testiympäristön pystyttäminen tapahtuu samalla tavalla kuin paikallisen testiympäristönkin. Hajautettu testiympäristö on määritelty yhdellä Heat Templatella, jolloin koko hajautetun testiympäristön pystyttäminen käy kätevästi yhdellä ”create”-komenolla. Kun kaikki virtuaaliko-

neet on luotu cPoutaan, voidaan ne alustaa. Ansiblen inventorissa määritellään jokainen kone eri ryhmiin: ”validation”, ”tape”, ”luigi-scheduler”, ”storage”, ”mongo” ja ”solr”. Näiden ryhmien avulla pidetään huoli, että jokaiselle koneelle asennetaan oikeat ohjelmistot ja resurssit hajautettuja testejä varten. Ansible-skriptiä ajettaessa vain tietyt taskit suoritetaan tietyille ryhmille. Näin jokainen kone saa oikeat asetukset ja ohjelmistot.

Hajautettu testiympäristö koostuu yhdestä runnerista ja kahdeksasta muusta virtuaalikoneesta (ks. kuvio 12). Aiemmassa vaiheessa kasattiin RPM-paketti, johon on tehty muutoksia. Tässä vaiheessa ei siis ole tarvetta tehdä enää RPM-paketin kasausta. Testiympäristön puhdistus ja alustus suoritetaan kuitenkin runnerilla. Tämän jälkeen voidaan aloittaa hajautettujen testien suorittaminen.

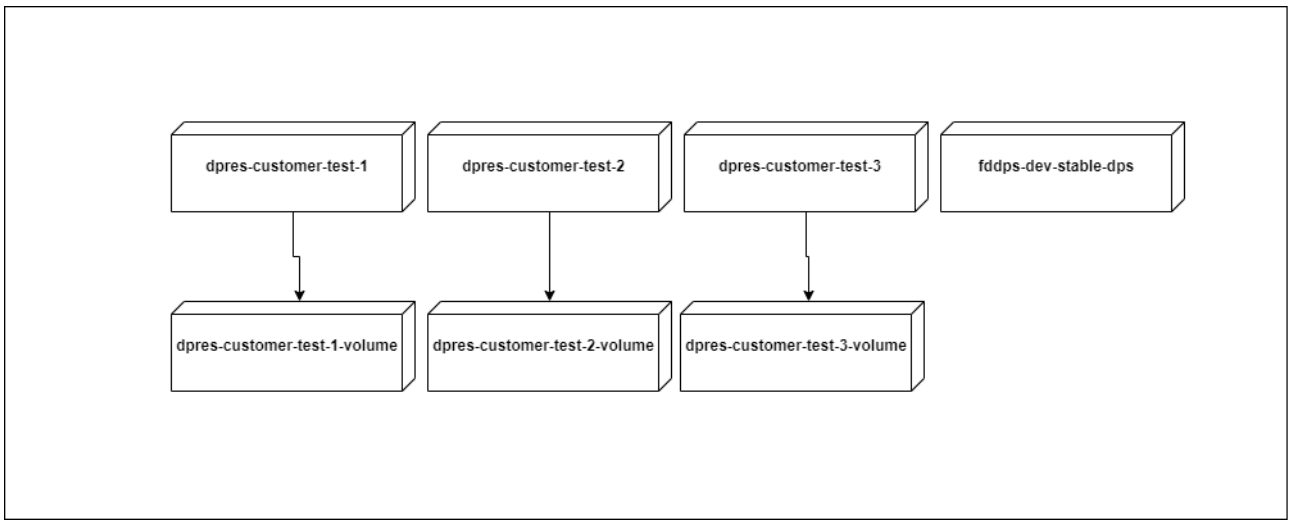


Kuvio 12. Hajautetun testiympäristön topologia

3.7 Asiakastestiympäristö

Asiakastestit ovat viimeinen testausvaihe. Näissä testeissä testataan koko ohjelmiston toimivuus ja varmistetaan, että se vastaa asiakkaan tarpeisiin. Näitä testejä ei ole automatisoitu, vaan testaus tapahtuu asiakkaan toimesta.

Asiakastestiympäristö koostuu muutamasta eri virtuaalikoneesta (ks. kuvio 13). Tässä työssä päädyttiin käyttämään neljää eri virtuaalikonetta asiakastestiympäristöön. Neljä virtuaalikonetta on asiakastesteille riittävä määrä. Jokaiselle asiakkaalle oman virtuaalikoneen luominen olisi turhaa, ja veisi aika paljon cPoudan resursseja. Tämän takia asiakkaat jakavat testikoneita. Neljä testikonetta riittää tällä hetkellä nykyiselle asiakasmäärälle käyttämättä liikaa cPoudan resursseja.



Kuvio 13. Asiakastestiympäristön topologia

Kolmelle koneelle on tehty erillinen tallennustila. Tämä helpottaa virtuaalikoneiden uudelleen pystyttämistä muutos- tai virhetilanteiden sattuessa. Erillisen tallennustilan ansiosta voidaan vanhat tiedot säilyttää, vaikka virtuaalikone haluttaisiinkin pystyttää uudelleen. Asiakkailla on pääsy tähän testiympäristöön. Asiakkaiden pääsyä eri testikoneille hallitaan Security Group -sääntöjen avulla. Testiympäristössä asiakkaiden on mahdollista testata, että ohjelmistot vastaavat heidän tarpeitaan ja odotuksiaan.

4 Ohjelmistokehityksen automaatio

4.1 Ympäristön pystytyksen automatisointi

KH2 haasteena oli saada ympäristön pystytys automatisoitua. Automatisointiin oli suunniteltu käytettäväksi Heat Stackeja sekä Ansiblea. Nämä valittiin sen takia, sillä ne olivat olleet käytössä jo aikaisemmassa testiympäristössä. Silloin ei tarvitsisi muuttaa prosessia kovin paljoa entisestään.

Automaatio on tärkeä osa modernia ohjelmistokehitysprosessia. Ohjelmistokehityksen vaiheiden automatisoinnilla voidaan parantaa kehitystyön tehokkuutta ja vähentää virheitä. Automaatiolla on mahdollista automatisoida toistuvia töitä. Esimerkiksi ohjelmiston julkaisu, sen testaus sekä mahdollisen testiympäristön pystyttäminen voidaan automatisoida. Eri vaiheiden ja prosessien automatisointi jättää enemmän aikaa ohjelmiston kehittämiselle. Automaation ansiosta on mahdollista poistaa ihmisen tekemät virheet testausprosessissa. Automaatio helpottaa projektin skaalausta sekä yhteistoimintaa. Automatisoidulla testiympäristön alustuksella ja siivouksella usean kehittäjän on mahdollista toimia samaan aikaan, kun testiympäristö voidaan palauttaa alkuasemaan, jolla on voi testata uutta ohjelmistoa.

Tässä projektissa testiympäristöt pystytettiin Heat Stackien avulla. Heat Stackeja on helppo hallita Heat Templatejen avulla. Yhdellä Heat Templatella pystytään hallitsemaan koko Heat Stackia. Siksi tässä työssä jokaiselle testiympäristölle luotiin yksi Heat Template. Poikkeuksena oli asiakastestiympäristö, jossa käytettiin useampaa Heat Templatea ympäristön pystyttämiseen. Heat Templatella ja OpenStackin komentorivityökalulla on mahdollista automatisoida testiympäristön pystyttäminen cPoutaan. Jos testiympäristön pystyttämiseen käytetään vain yhtä Heat Templatea, kuten paikallisen testiympäristön sekä hajautetun testiympäristön kanssa käytettiin, niin ympäristö on mahdollista pystyttää automaattisesti yhdellä OpenStackin komennolla. Heat Stackin muokkaaminen ja päivittäminen onnistuu todella helposti. Tarvittavat muokkaukset tehdään Heat Templateen, jonka jälkeen yhdellä komennolla saadaan koko Heat Stack päivitettyä.

Virtuaalikoneiden pystyttämisen jälkeen virtuaalikoneet piti vielä alustaa testaamista varten. Tämäkin vaihe automatisoitiin. Testiympäristön alustuksen automatisointi hoidettiin Ansible-skriptillä. Uusia testiympäristöjä varten luotiin uudet inventorit. Ansible-playbook oli osittain valmis, sillä sitä oli käytetty edellisen testiympäristön asennuksessa. Playbook piti muuntaa uudelle

RHEL9-käyttöjärjestelmälle sopivaksi. Muutoksien jälkeen playbook voitiin ajaa yhdellä Ansiblen komennolla. Tämä suoritti kaikki playbookin tehtävät ja alusti testiympäristön käyttöä varten.

Ansiblellä tehdyt kaksi muutakin skriptiä piti päivittää, jotta saataisiin CI/CD-putken luonti ja käyttö automatisoitua. Yhtä skriptiä käytettiin luomaan Docker-kontit cPoudassa oleville virtuaalikoneille ja rekisteröimään koneet sekä Docker-kontit runnereiksi. Toinen päivitettävä Ansible-skripti oli testikoneen tyhjennysskripti. Tyhjennysskriptillä saatiin kone alkutilaan, jonka jälkeen voitiin ajaa testikoneen alustusskripti ja näin saataisiin joka kerta testeille sama lähtötilanne.

4.2 CI/CD-putken hyödyntäminen

KH3 haaste oli saada luodut testiympäristöt toimimaan osana CI/CD-putkea. Ympäristöt suunniteltiin toimimaan CI/CD-putken kanssa tekemällä paikallisen testiympäristön virtuaalikoneista sekä niillä olevista Docker-konteista runnereita. Runnereiden avulla GitLab pystyisi käyttämään virtuaalikoneita sekä Docker-kontteja osana CI/CD-putkea. Runnereiden rekisteröintiskripti oli olemassa, ja sitä oli käytetty aikaisemmassa testiympäristössä. Skripti olisi kuitenkin päivitettävä toimiaan uusille testiympäristöille.

Testaus automatisoidaan käyttämällä GitLabin CI/CD-toimintoa. GitLabin CI/CD-putkella on mahdollista automatisoida ohjelmistojen testaus. CI/CD-putki määritellään ”.gitlab-ci.yml”-tiedostossa. Jokaiselle RPM-paketille on määritelty paketin kasaus sekä testit. Testien määrä voi vaihdella riippuen RPM-paketista.

Heti kun ohjelmistokomponenttiin tehty muutos on siirretty GitLabiin, niin GitLab aloittaa automatisoidun CI/CD-putken suorittamisen. Jokaisella RPM-paketille on määritelty ensimmäiseksi vaiheeksi RPM-paketin kasaus. Tämä hoidetaan jollain runnereiksi asennetulla Docker-kontilla. GitLab osaa automaattisesti valita vapaana olevan runnerin ja käyttää sitä määriteltyyn tehtävään. Paketin kasauksen jälkeen siirrytään testausvaiheeseen. Automaattisesti tehdään paikalliset testit ja Integraatiotestit sekä hajautetut testit (E2E-testit). Viimeinen testausvaihe suoritetaan asiakkaan toimesta. CD/CD-putkessa voi olla eri määrä testejä paketista riippuen. Jokaiselle testille valitaan automaattisesti runneri, jonka jälkeen testiympäristö valmistellaan testausta varten puhdistamalla ja alustamalla se. Näin saadaan paketti testattua joka kerta samanlaisella testikoneella.

5 RPM-pakettien testaus

5.1 Testausputki

KH4 haasteena oli ohjelmistokomponenttien (rpm-pakettien) automaattinen testaus. Testaus päätettiin hoitaa osana CI/CD-putkea, niin kuin oli aiemminkin tehty. Silloin ei tarvitsisi muuttaa testausprosessia. Haasteeksi kuitenkin muodostui pakettien sekä niiden testien muuttaminen RHEL-9 käyttöjärjestelmälle sopivaksi. Pakettien migraatio ei kuulunut tähän työhön, mutta se tuli tehtyä työn ohella. Jokainen paketti ja niiden testit piti migroida erikseen, jolloin tämä aiheutti aika paljon työtä. Testien migraatiossa piti ottaa huomioon eri moduulien versioiden muutokset.

RPM-paketti testataan kolmessa eri vaiheessa. Kaiksi ensimmäistä vaihetta hoidetaan automaattisesti GitLabin CI/CD-putkessa. Kolmas testivaihe hoidetaan asiakkaan toimesta. Näillä kolmella testillä saadaan testattua koko ohjelmistokokonaisuus kattavasti. Kaksi ensimmäistä vaihetta käyttää cPoudassa olevia virtuaalikoneita, jotka ovat rekisteröity runnereiksi GitLabiin.

Jokaiselle CSC:n pitkäaikaissäilytyksessä käytetylle paketille suoritetaan nämä kolme testivaihetta. Aina kun ohjelmiston komponenttiin tulee muutoksia, tulee testit suorittaa uudestaan. Myös itse testejä tulee päivittää aika ajoin. Kun ohjelmistolle tulee uusia vaatimuksia, tulee testit päivittää vastaamaan niitä. Näin voidaan tulevaisuudessakin varmistaa testien kattavuus.

5.2 Testauksen osat

Staattinen koodin analyysi hoidetaan kaikille CSC:n sisäisille ohjelmistoille. Tämä suoritetaan paikallisessa testiympäristössä ensimmäisessä testausvaiheessa. Koodin analyysi tehdään lintereiden avulla. Paketin ei tarvitse päästä läpi staattisesta koodianalyysista, mutta jokainen paketti pyritään tekemään siten, että ne pääsisivät analyysin läpi.

Ensimmäisessä vaiheessa suoritetaan myös paketin yksikkötestaus (unit test). Tämän testausvaiheen läpäisy vaaditaan jokaiselta itse kehitetyltä Python-koodilta. Yksikkötestauksella testataan ohjelmistokomponentin/metodin toimivuus erilaisten testitapausten avulla. Jokaisella paketilla on vähän erilaiset testit. Kuitenkin jokaiselle paketille testataan ainakin yksi onnistunut testitapaus ja yksi epäonnistunut testitapaus sekä vähintään 80 %:n koodin kattavuus. Jokaiselle paketille on

määritelty Makefilen tehtävä ”test”, joka aloittaa yksikkötestien suorittamisen. Yksikkötestejä on mahdollista ajaa myös omalla virtuaalikoneella, jos virtuaalikoneeseen on asennettu tarvittavat riippuvuudet.

Toisessa testausvaiheessa suoritetaan integraatiotestaus sekä E2E-testaus. Nämä testit suoritetaan hajautetussa testiympäristössä. Integraatiotestauksessa testataan komponenttien toimivuus keskenään. Riippuen paketista testit voivat vaihdella vain kahden välisestä testauksesta useamman ohjelmistokomponentin yhteentoimivuuden testaukseen. Lähes jokaiselle RPM-paketille suoritetaan integraatiotestaus.

Integraatiotestien jälkeen suoritetaan E2E-testaus, jossa testataan koko ohjelmistokokonaisuus. Tämä testi on kaikkein kattavin, ja se testaa kaikkien pakettien väliset riippuvuudet sekä niiden yhteistoiminnan. Testit suoritetaan syöttämällä palveluun SIP-paketti (Submission Information Package), minkä jälkeen testit palauttavat tuloksena AIP-paketin (Archival Information Package) tai raportin testistä. Testaamiseen käytetään sekä päteviä SIP-paketteja että epäpäteviä SIP-paketteja, jotta palvelu saadaan testattua mahdollisimman kattavasti. Nykyisin E2E-testit suoritetaan vain parille RPM-paketille. Tätä testausta ei tarvitse tehdä jokaiselle paketille, sillä testauksessa testataan kerralla koko palvelun toimivuus.

Viimeisessä testausvaiheessa suoritetaan toiminnallinen testaus sekä hyväksymistestaus. Nämä testit suoritetaan asiakkaan toimesta asiakastestiympäristössä. Asiakkaat voivat raportoida testien tulokset kehittäjille, joiden perusteella palvelua on mahdollista kehittää.

6 Ylläpito

Ohjelmistokehitysprosessissa kaikki on pyritty tekemän mahdollisimman automaattiseksi, jotta kehittäjillä olisi aikaa parannella ohjelmistoa ja korjata ohjelmistosta löytyviä virheitä. Testiympäristöä on helppo hallita Heat Stackien avulla. Testiympäristön Heat Stackit on valmiiksi luotuna, jolloin ympäristöjen uudelleen pystyttäminen käy parilla komennolla. Kaikki Heat Templatet ovat samassa tietovarastossa. Tämä tekee kaikkien cPoudassa olevien virtuaalikoneiden ja niiden resurssien hallinnasta helppoa. Asiakastestiympäristön muuttaminen on tehty mahdollisimman yksinkertaiseksi tekemällä asiakastestiympäristön virtuaalikoneiden Heat Templateista samanlaisia. Vain ”enivironment.yaml”-tiedostoissa on eroja. Tässä tiedostossa on määritelty Heat Stackin

muuttujat, kuten Stackin nimi, osoite, Security Groupit ja virtuaalikoneen käyttämät resurssit. Uuden asiakastestikoneen luominen onnistuu kopioimalla valmis Heat Stack ja muuttamalla ”environment.yaml”-tiedostossa olevat muuttujat vastaamaan uutta virtuaalikonetta.

Runnereiden hallintaan tehdyllä Ansible-skriptillä runnereiden poisto sekä lisääminen on tehty helpoksi. Ansible-skriptille ei tarvitse kuin kertoa, mitkä virtuaalikoneet halutaan rekisteröidä runneriksi, ja skripti hoitaa loput automaattisesti. Runnerit on määritelty Ansible-skriptin inventorissa. Runnereiden poistaminen onnistuu myös GitLabin web-käyttöliitymän kautta.

Testit ovat pakettikohtaisia. Pienien korjausten ja parannusten myötä testejä ei ole tarvetta muuttaa, sillä testit ovat tehty kattamaan monenlaisia testitapauksia. Testejä tarvitsee päivittää vain, jos RPM-paketteihin tai niiden käyttämiin moduuleihin tulee suuria muutoksia tai tarvitaan tehdä uusia testitapauksia esimerkiksi uusien tiedostomuotojen hyväksymiseksi. Jos testeihin kuitenkin tarvitsee tehdä muutoksia, tulisi ne tehdä paketin päivityksen yhteydessä.

RPM-paketteja ylläpidetään ja päivitetään jatkuvasti. Kun pakettien käyttämät julkiset moduulit ja työkalut päivittyvät, tulee CSC:n omiakin RPM-paketteja päivittää. Paketteja päivitetään myös asiakkaan tarpeen mukaan. Jos asiakas tarvitsee ohjelmistoon uutta ominaisuutta tai jotain parannettavaa palveluun, ohjelmistot päivitetään vastaamaan asiakkaan tarpeita. Paketit tulisi pitää ajan tasalla turvallisuussyistä, ja päivittää vastaamaan asiakkaiden odotuksia. Hyvät työkalut ja toimintatavat helpottavat tätä prosessia. Hyvän CI/CD-putken ansiosta voidaan automatisoida ohjelmistojen kehitystä. Myös hyvät versionhallintakäytännöt on tärkeitä ohjelmistokehityksessä. Versionhallinnalla mahdollistetaan usean kehittäjän yhteistyö sekä uuden koodin tarkistaminen, jotta se vastaa tavoitteita.

7 Tulokset

7.1 KH1 Kuinka ympäristön pystytys cPouta-ympäristöön tehdään?

Ympäristön pystyttäminen onnistui suunnitellusti. Ympäristö pystytettiin käyttäen OpenStackin Heat Templateja. Testiympäristöjen luonti olisi myös onnistunut cPoudan verkkoportaalin kautta. Tämä olisi kuitenkin vaatinut jonkin verran enemmän työtä, ja myöhemmin testiympäristöjen hallinta olisi työläämpää. cPoudan-verkkoportaali sopii hyvin resurssien yleiskatsaukseen ja myös yk-

sittäisen virtuaalikoneen luontiin. Usean koneen luonti onnistuu kätevämmin käyttämällä Heat Templatea.

Heat Templatejen luonti onnistui melko helposti, sillä aikaisempi testiympäristö oli hyvin samanlainen uuteen testiympäristöön verrattuna, jolloin aikaisemman testiympäristön Heat Templatea pystyi käyttämään mallina. Jonkin verran aikaa meni kuitenkin oikeiden Security Groupien sekä virtuaalikoneiden osoitteiden varmistamisessa. Security Groupit piti katso tarkkaan, ettei vahingossa antanut pääsyä ulkopuolisille testiympäristöön. Virtuaalikoneiden osoitteet piti tarkistaa siksi, ettei muilla virtuaalikoneilla ollut jo sitä osoitetta käytössä ja vältyttäisiin osoitteiden päällekkäisyyksiltä.

7.2 KH2 Miten ympäristön pystyttäminen saadaan automatisoitua?

Automatisointi hoidettiin Heat Templateilla sekä Ansible-skripteillä. Heat Templateilla on mahdollista määritellä useita virtuaalikoneita ja resursseja. Paikalliselle sekä hajautetulle testiympäristölle kummallekin luotiin yksi Heat Template. Yhdellä Heat Templatella pystyi hallitsemaan koko testiympäristöä. Asiakastestiympäristö luotiin kuitenkin käyttäen erillisiä Heat Stackejä. Asiakastestiympäristöön päädyttiin luomaan erilliset Heat Stackit, jotta resurssien erillinen hallinta olisi helpompaa.

Kokonaisuudessaan cPoudassa olevien testiympäristöjen alustaminen ja siivoaminen hoidettiin kolmella eri Ansible-skriptillä. Hajautetun testiympäristön koneille oli oma alustusskripti. Tämä skripti hoiti kaikki hajautetun testiympäristön eri koneiden alustuksen sekä eri palveluiden asentamisen koneille. Toinen alustusskripti tehtiin kaikille muille virtuaalikoneille. Tällä skriptillä voitiin alustaa sekä asiakastestiympäristön että paikallisen testiympäristön virtuaalikoneet testikäyttöön. Kolmas skripti oli tarkoitettu testikoneiden siivoamiseen. Siivoamisskripti on tarkoitettu ajettavaksi kaikilla testikoneilla. Siivoamisskriptin avulla testikoneet saadaan aloitustilaan. Heat Templatejen sekä Ansible-skriptien avulla ympäristöjen pystytys saatiin automatisoitua.

Ansible-skriptit saatiin toimimaan halutulla tavalla. Ansible-skriptit saatiin toimimaan kaikilla kolmella käyttöjärjestelmällä CentOS 7, RHEL9 sekä ALMA9. RHEL9-siirtymän jälkeen kuitenkin CentOS 7 -käyttöjärjestelmälle tehty koodi poistettiin, jottei käyttämätön koodi jäisi skripteihin turhaan. RHEL9-siirtymän alkuvaiheessa huomattiin, että jotkut asennettavat ohjelmat saattoivat

ottaa käyttöön UID:n tai GID:n, jotka oli tarkoitettu käytettäväksi toiselle käyttäjälle/ryhmälle. Tämä ongelma korjattiin lisäämällä siivousskriptiin poistettavaksi ne ohjelmat ja niiden luomat mahdolliset käyttäjät ja ryhmät, jotka estivät testejä varten tarvittavien käyttäjien ja ryhmien luonnin. Alustusskriptiin siirrettiin käyttäjien ja ryhmien määrittely ennen minkään ohjelman asennusta. Korjauksen jälkeen skripti toimi niin kuin pitikin.

7.3 KH3 Kuinka testiympäristö saadaan osaksi automatisoitua CI/CD-putkea?

Ympäristön liittäminen osaksi CD/CD-putkea tehtiin käyttäen Ansible-skriptiä. Ansible-skripti oli jo valmiina, sillä sitä oli käytetty aikaisemman testiympäristön kanssa. Ainoastaan tähän skriptiin piti päivittää Docker-kontin asennus, jotta se toimisi RHEL9:llä sekä määritellä uudet testikoneet, jotka rekisteröitäisiin runneriksi. Myös runnereiden tageja päätettiin muuttaa, jotta ne kuvastaisivat paremmin runneria.

Docker-runnereita käytettiin RPM-pakettien kasaamiseen. Docker-kontit toimivat cPoudassa olevilla virtuaalikoneilla juuri kuten pitkin, eikä niiden kanssa ollut mitään ongelmia. Virtuaalikoneita rekisteröidessä alussa tuli pientä ongelmaa, sillä virtuaalikoneen konfiguroinnissa oli tullut pieni virhe. Virtuaalikoneelle oli konfiguroitu väärä pakettivarasto, joka ei RHEL9:llä toiminut. Virtuaalikone saatiin rekisteröityä GitLabiin Ansible-skriptillä, mutta CI/CD-putki pysähtyi ja antoi virheilmoituksen, kun RPM-pakettia koetettiin testata. Tämä virhe korjattiin vaihtamalla virtuaalikoneen konfigurointiskriptiin oikea pakettivarasto, joka toimi RHEL9-käyttöjärjestelmällä. Virheen korjauksen jälkeen virtuaalikoneen runneriksi rekisteröiminen sekä CI/CD-putken ajaminen toimivat suunnitellusti.

7.4 KH4 Kuinka ohjelmistokomponentit saadaan testattua RHEL9-testiympäristössä?

RPM-pakettien testaus toteutettiin automatisoidulla GitLabin CI/CD-putkella. CI/CD-putki käyttää rekisteröityjä runnereita eri tehtävien suorittamiseen. Ympäristön ja runnereiden konfiguroinnin jälkeen testaus onnistui niin kuin oli suunniteltu. RPM-paketit oli kehitetty CentOS 7 -alustalle, tämän takia paketit eivät suoraan toimineet RHEL9-testiympäristössä. Jokainen paketti täytyi muuttaa RHEL9-alustalle sopivaksi. Pakettien migroinnin jälkeen osa testeistä toimi sellaisinaan, mutta joidenkin pakettien testit piti päivittää toimimaan RHEL9-käyttöjärjestelmällä.

Pakettien migroinnin jälkeen pystyi pakettien testauksen kokeilemaan CI/CD-putkessa. Kaikki CSC:n pitkäaikaissäilytyksen paketit testattiin RHEL9-testiympäristössä. Kaikkiaan ympäristöt toimivat niin kuin pitikin. Asiakastestiympäristössä jätettiin paketit testaamatta, sillä asiakastestivaihe kuului asiakkaiden hoidettavaksi. Uskon kuitenkin, että ympäristö toimii toivotulla tavalla, sillä se oli konfiguroitu hyvin samalla tavalla kuin paikallinen testiympäristö.

8 Pohdinta

8.1 Työn onnistuminen

Työ onnistui kokonaisuudessaan melko mutkattomasti, sillä käytettävät työkalut oli jo määritelty ennen työn aloittamista sekä Ansible-skriptit olivat osittain valmiit, sillä niitä oli käytetty edellisessä testiympäristössä. Skriptit piti vain migroida uuteen testiympäristöön sopivaksi. Skriptien migraatio tehtiin paikallisella Alma9-virtuaalikoneella. Skriptien toimivuuden testaaminen Alma9-alustalle onnistui omalla paikallisella virtuaalikoneella. Rhel9-alustalle skriptien testaaminen suoritettiin yhdellä paikallisen testiympäristön koneella. Testaamisen olisi voinut tehdä täysin paikallisesti, mutta tämä olisi vaatinut uuden virtuaalikoneen luomista.

Työn aikana tavoitteet muuttuivat hieman alkuperäisestä. Aluksi tavoitteena oli luoda vain yksi testiympäristö, mutta projektin edetessä työn kuva muuttui. Uudeksi tehtäväksi muodostui luoda kaikki kolme testiympäristöä. Sain kuitenkin onnistuneesti mukautettua työni vastaamaan uusia vaatimuksia. Projektityöosuuden sain valmiiksi suunnitellussa aikataulussa, vaikka tavoitteet muuttuivatkin matkan varrella. Sen sijaan opinnäytetyön kirjoittamisen aikatauluttaminen olisi voinut olla parempaa, sillä kirjoittamisen aloittaminen viivästyi alkuperäisistä suunnitelmista ja jäi osittain viime hetkeen. Kaikesta huolimatta sain työn valmiiksi kokonaisuudessaan ja pystyin vastaamaan sille asetettuihin tavoitteisiin. Työ oli myös ensimmäinen suurempi kehitystyö, jonka olen toteuttanut, ja olen tyytyväinen siihen, miten lopputulos vastasi odotuksia.

8.2 Oman oppimisen arviointi

Projektin aikana pääsin perehtymään useisiin uusiin työkaluihin ja teknologioihin, jotka olivat ennestään vieraita. Erityisesti Ansible ja OpenStackin Heat Templatet nousivat keskeiseen rooliin

testiympäristöjen pystyttämisessä. Näiden työkalujen oppiminen sujui melko nopeasti, sillä molemmat hyödyntävät selkeää ja helposti ymmärrettävää YAML-muotoista syntaksia.

Testauspuoli oli minulle täysin uusi alue, sillä en ollut aiemmin opiskellut tai työskennellyt sen parissa. Tämän projektin myötä opin ymmärtämään, miten tärkeä rooli testauksella on ohjelmistokehityksen laadunvarmistuksessa. Opin myös, miten tehokas testausprosessi voi säästää merkittävästi aikaa ja resursseja. Ohjelmistokehitykseen kuuluvien testausprosessien oppiminen mahdollistaa tulevaisuudessa osallistumisen entistä vaativimpiinkin ohjelmistokehitysprojekteihin.

Työskentely tässä projektissa tarjosi minulle myös arvokasta kokemusta ensimmäisestä oman alani työpaikasta. Sain hyvän käsityksen isomman projektin hallinnasta ja siihen liittyvistä vaiheista lisäksi pääsin käyttämään GitLabia työelämässä testauksen ja ohjelmistokehityksen tukena. Aiemmin olin käyttänyt GitLabia opinnoissa pienempien projektien hallintaan ilman CI/CD-putkea, joten tämän työkalun laajempi hyödyntäminen oli uutta. Lisäksi työpaikalla saamani tuki auttoi minua ymmärtämään pitkäaikaissäilytysprosessin kokonaisuutta ja kehityksen eri vaiheita.

8.3 Eettisyys ja luotettavuus

Opinnäytetyön toteutuksessa on noudatettu Jyväskylän ammattikorkeakoulun eettisiä periaatteita, jotka korostavat vastuullisuutta, rehellisyyttä ja tutkimuksen luotettavuutta. Työn jokaisessa vaiheessa on kiinnitetty huomiota hyvän tieteellisen käytännön mukaisiin toimintatapoihin. Työprosessin läpinäkyvyys ja rehellisyys ovat olleet keskeisessä roolissa, ja kaikki työssä tehdyt ratkaisut on dokumentoitu huolellisesti eettisten ohjeiden mukaisesti. (Eettiset periaatteet 2024.)

Työssä käytetyt lähteet ovat huolella valittuja ja perustuvat ajantasaisiin, luotettaviin tietolähteisiin. Tiedonkeruussa ja sen käsittelyssä on pyritty varmistamaan tiedon oikeellisuus ja muuttumattomuus. Viittaustekniikassa on noudatettu tieteellistä tarkkuutta, ja työssä on pyritty välttämään tulkintavirheitä sekä varmistamaan, että kaikki tiedot ja johtopäätökset perustuvat tosiasioihin. Tämä lähestymistapa varmistaa, että työn tulokset ovat luotettavia ja kestävät kriittisen tarkastelun.

8.4 Jatkokehitysmahdollisuudet

Jos ympäristön päivitysprosessin haluaisi automatisoida, voisi cPoutaan levykuvan tekemisen sekä viemisen automatisoida skriptillä. Päivityksen automatisointi veisi kuitenkin työaikaa, eikä päivitystä tarvitsi tehdä kuin muutaman kerran RHEL9-testiympäristön elinkaaren aikana. Tämän prosessin automatisointi ei siis ole kovinkaan tarpeellista.

Testiympäristön suojausta olisi voinut parantaa SELinuxin avulla. Tämä jäi kuitenkin tekemättä. Kaikissa testiympäristön virtuaalikoneissa SELinux jätettiin ”Permissive”-tilaan. Tällöin kaikki SELinuxin sääntöjen rikkomiset raportoidaan logiin, mutta sääntörikkomuksia ei kuitenkaan estetä. SELinuxia ei laitettu kokonaan päälle, sillä kaikki paketit eivät olleet vielä migroitu toimimaan SELinuxin kanssa. SELinuxin käyttöönotto oli kuitenkin työn alla, ja uskoisin sen tulevan käyttöön lähitulevaisuudessa. Kun ohjelmistoille saadaan asetettua oikeat SELinuxin säännöt, voidaan SELinux laittaa ”Enforcing”-tilaan. Tässä tilassa kaikki SELinuxin säännön rikkomiset estetään.

Testiympäristössä käytettiin viimeisintä RHEL9-versiota. Tämä versio tulee jossain vaiheessa vanhentumaan, jolloin testiympäristön käyttöjärjestelmä tulisi päivittää uuteen versioon. Tällöin tulisi luoda uusi levykuva uudesta RHEL9-versiosta ja ladattava se cPoutaan. Tämän jälkeen testiympäristöt voidaan luoda uudestaan käyttäen uutta levykuvaa. Luominen onnistuu helposti Heat Templatejen avulla, sillä kaikki Heat Templatet on luotu valmiiksi. Niihin tulee vain vaihtaa käytettävä käyttöjärjestelmä, ja testiympäristön luonti onnistuu yhdellä OpenStackin ”update”-komennolla. Uskoisin, että testiympäristön alustava sekä sen puhdistava Ansible-skripti toimii uudessa versiossa samalla tavalla kuin nykyisessä RHEL9-versiossakin.

Jos Ansible-skripteihin ei tarvitse tehdä muutoksia, niin alustavan Ansible-skriptin ajettu testiympäristö on taas valmiina käyttöä varten. Testiympäristön liittäminen CI/CD-putkeen vaatii kuitenkin entisten runnereiden poistamisen GitLabista ja uusien runnereiden luomisen sinne. Tämäkin automatisointi ja runnereiden poistaminen onnistuu joko GitLabin nettiliittymän kautta tai käyttämällä runnereiden poistamiseen tehtyä skriptiä. Runnereiden luonti onnistuu yhdellä Ansible-skriptillä. Jos virtuaalikoneiden nimiä ei ole muutettu niiden luontivaiheessa, niin runnereiden rekisteröintiin käytettävä Ansible-skripti toimii sellaisenaan. Jos kuitenkin virtuaalikoneiden nimiä on muutettu, pitää ne päivittää Ansiblen inventoriin, ennen skriptin ajamista. Näin saadaan helposti testiympäristö päivitettyä, ja voidaan keskittyä ohjelmistokehitykseen.

Lähteet

Ansible Roles. 2024. Artikkeli GeeksforGeeksin verkkosivustolla. Julkaistu 9.7.2024. Viitattu 17.11.2024. https://www.geeksforgeeks.org/ansible-roles/.

ansible.builtin.import_playbook module – Import a playbook. 2024. Dokumentti Ansible Community Documentation -sivustolla. Julkaistu 12.11.2024. Viitattu 17.11.2024. https://docs.ansible.com/ansible/latest/collections/ansible/builtin/import_playbook_module.html .

Cattelain, G. 2022. Red Hat Enterprise Linux 9.1 is now available. Artikkeli Red Hat Blog -sivustolla. Julkaistu 16.11.2022. Viitattu 15.7.2024. https://www.redhat.com/en/blog/rhel-91-now-available.

Chapter 1. Introduction to RPM. N.d. Dokumentti Red Hat Documentation -verkkosivustolla. Viitattu 17.11.2024. https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/packaging_and_distributing_software/introduction-to-rpm_packaging-and-distributing-software#introduction-to-rpm_packaging-and-distributing-software.

CentOS 7. N.d. Artikkeli TuxCaren verkkosivustolla. Viitattu 16.11.2024. https://tuxcare.com/resources/learning/centos-7/.

Chinthaguntla, K. 2020. Linux package management with YUM and RPM. Artikkeli Red Hat Blog -sivustolla. Julkaistu 22.4.2020. Viitattu 31.10.2024. https://www.redhat.com/en/blog/how-manage-packages.

cPouta. N.d. Artikkeli CSC:n verkkosivustolla. Viitattu 18.9.2024. https://research.csc.fi/-/cpouta.

Creating a virtual machine in Pouta. 2024. Dokumentti CSC:n verkkosivustolla. Julkaistu 7.5.2024. Viitattu 18.9.2024. https://docs.csc.fi/cloud/pouta/launch-vm-from-web-gui/.

DEB vs RPM. N.d. Artikkeli DataFlairin verkkosivustolla. Viitattu 17.11.2024. https://data-flair.training/blogs/deb-vs-rpm/.

Design. 2018. Artikkeli OpenStackin verkkosivustolla. Julkaistu 29.11.2018. Viitattu 17.11.2024. https://docs.openstack.org/arch-design/design.html.

DevOps. N.d. Atlassianin verkkosivusto. Viitattu 16.11.2024. https://www.atlassian.com/devops.

Eettiset periaatteet. 2024. Jyväskylä: Jyväskylän ammattikorkeakoulu. Julkaistu 10.6.2024. Viitattu 25.11.2024. https://www.jamk.fi/fi/media/41520.

Fairdata PAS tutkimusorganisaatioille. N.d. Artikkeli Digitalpreservationin verkkosivustolla. Viitattu 21.11.2023. https://digitalpreservation.fi/services/fairdata.

Fairdata PAS-palvelu – Tutkimusaineistojen pitkäaikaissäilytys. N.d. Artikkeli Fairdatan verkkosivustolla. Viitattu 31.10.2024. https://www.fairdata.fi/fairdata-pas/.

Hashemi-Pour, C. & Bigelow, S. J. 2024. Artikkeli TechTargetin verkkosivustolla. Julkaistu 7.2024. Viitattu 14.11.2024. https://www.techtarget.com/searchcloudcomputing/definition/Infrastructure-as-a-Service-IaaS.

How Ansible works. N.d. Ansiblen verkkosivusto. Viitattu 7.11.2024. https://www.ansible.com/how-ansible-works/.

is openstack iaas or paas. 2024. Artikkeli Alibaba Cloud -verkkosivustolla. Julkaistu 20.5.2024. Viitattu 16.11.2024. https://www.alibabacloud.com/tech-news/a/openstack/4oac1rpjb90-is-openstack-iaas-or-paas.

Jevtic, G. 2024. How to Install RPM File on Different Linux Distributions. Artikkeli phoenixNAPin verkkosivustolla. Julkaistu 8.2.2024. Viitattu 31.10.2024. https://phoenixnap.com/kb/how-to-install-rpm-file-centos-linux.

Kulttuuriperintö-PAS kirjastoille, arkistoille ja museoille. N.d. Artikkeli Digitalpreservationin verkkosivustolla. Viitattu 11.11.2024. https://digitalpreservation.fi/services/cultural_heritage.

Laureat, G. 2020. CI/CD isossa yrityksessä. Opinnäytetyö, AMK. Haaga-Helian ammattikorkeakoulu, tietojenkäsittelyn koulutusohjelma. Viitattu 7.10.2024. https://www.theseus.fi/bitstream/handle/10024/352208/opinn%C3%A4ytety%C3%B6_final.pdf.

Mikä CSC?. N.d. Artikkeli CSC:n verkkosivustolla. Viitattu 17.8.2024. https://csc.fi/tietoa-meista/mika-csc/.

OpenStack. N.d. OpenStackin verkkosivusto. Viitattu 9.10.2024. https://www.openstack.org/software.

OpenStack command line client tools for Pouta. 2024. Dokumentti CSC:n verkkosivustolla. Julkaistu 18.6.2024. Viitattu 30.10.2024. https://docs.csc.fi/cloud/pouta/command-line-tools/.

Orchestration Service: OpenStack Heat. 2018. Dokumentti University of Cambridgen Research Computing Services -sivustolla. Viitattu 7.11.2024. https://docs.hpc.cam.ac.uk/cloud/userguide/07-orchestration.html.

Osaamisemme. N.d. Artikkeli CSC:n verkkosivustolla. Viitattu 17.8.2024. https://csc.fi/osaamisemme/.

Pitkäaikaissäilytys. N.d. Artikkeli Kansalliskirjaston verkkosivuilla. Viitattu 5.11.2024. https://www.kansalliskirjasto.fi/fi/vapaakappalepalvelut/verkkoaineistot/pitkaaikaissailytys.

Pouta. 2024. Dokumentti CSC:n verkkosivustolla. Julkaistu 12.3.2024. Viitattu 18.9.2024. https://docs.csc.fi/cloud/pouta/.

Red Hat Enterprise Linux 9 is now available. N.d. Artikkeli Red Hat Developer -verkkosivustolla. Viitattu 7.11.2024. https://developers.redhat.com/products/rhel/rhel9-beta.

Red Hat Enterprise Linux Life Cycle. N.d. Artikkeli Red Hatin verkkosivustolla. Viitattu 7.11.2024. https://access.redhat.com/support/policy/updates/errata.

Roles. 2024. Dokumentti Ansible Community Documentation -sivustolla. Julkaistu 12.11.2024. Viitattu 17.11.2024. https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_reuse_roles.html.

Saifullah, H. 2023. CI/CD (In-depth Analysis). Artikkeli Medium-verkkosivustolla. Julkaistu 24.3.2023. Viitattu 2.10.2024. https://medium.com/@saifkhan666645/ci-cd-in-depth-analysis-8a402565807f.

Sanjay, R. & Hajnoczi, S. 2024. Virtualized database I/O performance improvements in RHEL 9.4. Artikkeli Red Hat Developer -verkkosivustolla. Julkaistu 10.9.2024. Viitattu 7.11.2024. https://developers.redhat.com/articles/2024/09/10/virtualized-database-io-performance-improvements-rhel-94.

Sinhoreli, M. 2023. OpenStack Alternatives. Artikkeli ShapeBluen verkkosivustolla. Julkaistu 5.10.2023. Viitattu 17.11.2024. https://www.shapeblue.com/openstack-alternatives/.

Sultan, S. 2024. OpenStack vs VMware: Guide to Choosing the Right Cloud Solution. Artikkeli Cloud7:n verkkosivustolla. Julkaistu 2.7.2024. Viitattu 17.11.2024. https://cloud7.io/blog/openstack-vs-vmware/.

Tahir, J. & Gonzales, A. N.d. Pouta Cloud Course. Viitattu 30.10.2024. https://pouta-course.a3s.fi/pouta-course-slides.pdf.

Takaamme tutkimusaineistojen ja kulttuuriperintöaineistojen pitkäaikaissaatavuuden. N.d. Artikkeli CSC:n verkkosivustolla. Viitattu 17.8.2024. https://csc.fi/osaamisemme/pitkaaikaissaatavuus/.

Tutkimusainestojen pitkäaikaissäilytys käynnistyi. 2019. Uutinen Digitalpreservationin verkkosivustolla. Julkaistu 18.12.2019. Viitattu 23.11.2024. https://digitalpreservation.fi/2023-tutkimusainestojen-pitkaaikaissailytys-kaynnistyi.

Using SELinux. N.d. Dokumentti Red Hat Documentation -verkkosivustolla. Viitattu 16.11.2024. https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html-single/using_selinux/index#proc_providing-feedback-on-red-hat-documentation_using-selinux.

Wallenius, N. 2022. Konttiteknologia – mitä kontit ovat ja mitä hyötyä niistä on?. Artikkeli Wallenius Consulting -sivustolla. Julkaistu 23.2.2022. Viitattu 17.11.2024. https://niklaswallenius.fi/konttiteknologia-mita-hyotya/.

What is CentOS Linux?. N.d. Artikkeli arkistoidulla CentOS-sivustolla. Viitattu 7.11.2024. https://web.archive.org/web/20190912091748/https://wiki.centos.org/FAQ/General#head-4b2dd1ea6dcc1243d6e3886dc3e5d1ebb252c194.

What is CI/CD?. N.d. Artikkeli GitlLabin verkkosivustolla. Viitattu 23.9.2024. https://about.gitlab.com/topics/ci-cd/.

What is Docker?. N.d.a. Artikkeli Amazon Web Services -verkkosivustolla. Viitattu 7.11.2024. https://aws.amazon.com/docker/.

What is Docker?. N.d.b. Dokumentti Dockerin verkkosivustolla. Viitattu 7.11.2024. https://docs.docker.com/get-started/docker-overview/.

What to know about CentOS Linux EOL. 2024. Artikkeli Red Hatin verkkosivustolla. Julkaistu 1.7.2024. Viitattu 30.10.2024. https://www.redhat.com/en/topics/linux/centos-linux-eol.

Winter, A. N.d. Mikä on DevOps?. Artikkeli Altorosin verkkosivustolla. Viitattu 16.11.2024. https://altoros.fi/mika-on-devops/.