Opinnäytetyö (AMK)

Tietojenkäsittely

2022

Viivi Luokkala

Azure Pipelines -liukuhihna tiedostojen vientiin Databricks-ympäristöihin

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tietojenkäsittely

2022 | 32 sivua

Viivi Luokkala

Azure Pipelines -liukuhihna tiedostojen vientiin Databricks-ympäristöihin

Automaatio on nykyään keskeinen osa teknisiä järjestelmiä. Automaation tarve tulee edelleen kasvamaan, kun pilvipalveluiden käyttö lisääntyy ja ne mahdollistavat entistä suurempien datamäärien käsittelyn.

DevOps on käytänteitä ja periaatteita, joiden tavoitteena on nopeuttaa kehitystyötä. Automaatio voi mahdollistaa esimerkiksi nopean käyttöönoton, kun päivitys koodiin voi laukaista liukuhihnan, joka kokoaa ja ottaa koodin käyttöön.

Tämän opinnäytetyön tarkoituksena oli luoda automatisoitu liukuhihna Azure Pipelines -alustalle. Liukuhihna ajaa ohjelman, jolla viedään tiedostoja useaan Databricks-ympäristöön, jotka ovat pilvipalveluja. Aikaisemmin toimeksiantajalla tiedostot oli viety yksitellen käsin jokaiseen ympäristöön. Tämä prosessi oli aikaa vievä ja altis inhimillisille virheille.

Opinnäytetyön tuloksena saatiin toimiva liukuhihna. Prosessi saatiin pitkälti automatisoitua, ja se on nopea.

Asiasanat:

Automaatio, Databricks, DevOps, Microsoft Azure

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Degree programme: Business Information Technology

2022 | 32

Viivi Luokkala

Azure Pipelines for importing files to Databricks environments

Automatization is nowadays a central part of any technical system. The need for automatization will grow as the usage of cloud services grows. Cloud services are enabling the handling of ever larger amounts of data.

DevOps is a set of principles aimed to make development faster and more efficient. Automatization enables for example fast deployments as a code update can trigger a new build and deploy pipeline.

The goal of this thesis was to create and automate a pipeline in Azure Pipelines. The pipeline runs a script that imports files to Databricks environments hosted in the cloud. Previously this task was carried out by hand and file by file. The process was time-consuming and prone to mistakes.

The pipeline was deployed successfully. The file import process became mostly automated and faster.

Keywords:

Automatization, Databricks, DevOps, Microsoft Azure.

Sisältö

Kuvat

Taulukot

Käytetyt lyhenteet

ADO	Azure DevOps, pilvipalvelu
API	Ohjelmointirajapinta (Application Programming Interface)
DevOps	Kehitys ja toiminta (Development and Operations)

1 Johdanto

Opinnäytetyön aiheita ovat automaatio, DevOps ja Microsoftin Azure Pipelines –alusta. Automaatiojärjestelmien ja sovellusten kehitys ja ylläpito ovat yrityksissä tärkeitä toimintoja. Nykyään automaatio on osa lähes kaikkia teknisiä järjestelmiä ja monet niistä perustuvat erilaisiin pilvipalveluihin. (Koskinen, 2018.) Pilvipalvelut, joista Microsoft Azure on yksi, tarjoavat monia etuja paikallisiin palveluihin verrattuna, ja niiden käyttö tulee edelleen lisääntymään tulevaisuudessa (Modi, 2019).

Opinnäytetyön toimeksiantaja on yritys, jolla on toimisto Turussa. Yritys tekee muun muassa data-analyysia ja haluaisi siirtyä käyttämään enemmän pilvipalveluja, esimerkiksi Databricksia. Databricksin käytön helpottamiseksi yrityksessä on luotu, ja luodaan jatkuvasti edelleen, Python-kielellä erilaisia funktioita sekä tutoriaaleja niiden käyttöön. Python-tiedostot on viety ympäristöihin yksitellen käsin. Prosessi halutaan automatisoida sen nopeuttamiseksi ja estämään inhimillisiä virheitä, kuten yksittäisen ympäristön unohtaminen.

Opinnäytetyö on toiminnallinen, ja sen tavoitteena on automatisoida Databricks-ympäristöihin vietyjen Python-tiedostojen päivitys. Tähän käytetään Azure Pipelines –alustaa ja sille luodaan liukuhihna, joka suorittaa PowerShell-ohjelman. PowerShell-ohjelmassa varsinaisesti viedään Python-tiedostot ympäristöihin käyttäen Databricksin tarjoamia API-rajapintoja. Lisäksi ympäristöihin täytyy luoda kansioita ja asettaa eri käyttäjäryhmille niihin erilaisia käyttöoikeuksia.

Opinnäytetyö on jaettu viiteen lukuun seuraavalla tavalla. Työn 2. luvussa käsittelen yleisesti automaatiota ja DevOpsia. 3. luvussa käsittelen käytettyjä teknologioita, eli liukuhihnaan valittua kieltä PowerShelliä, Microsoft Azurea, sekä lyhyesti Databricksiä. 4. luvussa käsittelen ennen varsinaisen työn alkamista tehtyä suunnittelua ja lopulta implementaatiota jaettuna kahteen osioon. Ensimmäisessä käyn läpi kesällä 2021 kirjoitetun version ohjelmasta ja liukuhihnasta. Toisessa käyn läpi keväällä 2022 kirjoitetun toisen version, jossa

hyödynnetään Databricksin uudempia ominaisuuksia. Lopuksi vielä pohdin tehdyn liukuhihnan toimivuutta, sen jatkokehitystä, sekä mahdollisten uusien prosessien automaatiota.

2 DevOps ja automaatio

DevOps on lyhenne sanoista *Development* ja *Operations*, jotka tarkoittavat kehitystä ja toimintaa. DevOps on kokoelma toimintaperiaatteita ja käytänteitä, jotka tuovat yrityksen sisällä kehittäjät ja toiminnasta vastaavat tekemään yhteistyötä. Sen tavoitteena on tuoda nopeasti ja tehokkaasti, sekä ennakoitavalla tavalla, valmis ohjelmisto asiakkaalle käyttöön. DevOpsin mahdollistajia ovat erilaiset työkalut ja teknologiat, sekä automaatio. (Modi 2019.)

DevOps on saanut paljon huomiota, ja monet yritykset ovat muokanneet prosessejaan vastaamaan sen käytäntöjä saadakseen kilpailuetua sen lupaamista hyödyistä. DevOpsin avainkäytänteitä ovat jatkuva integraatio, jatkuva käyttöönotto, jatkuva toimitus ja jatkuva oppiminen. Jatkuva integraatio tarkoittaa, että säännöllisin väliajoin kehittäjien tuottama koodi tulisi koota ohjelmistoksi ja testata sen toimivuus. Näin voidaan huomata koodissa, tai ohjelmiston logiikassa olevat virheet, ennen kuin sen kehitykseen on tuhlattu paljoa aikaa. Lopputuloksena saadaan toimivaksi todettua koodia, jota voidaan jatkuvasti ottaa käyttöön ja toimittaa asiakkaille. Lisäksi DevOpsiin kuuluu jatkuva palaute asiakkailta ja ohjelmiston monitoroinnista. Palautetta kuuntelemalla kehittäjät voivat jatkuvasti oppia ja ohjelmistoa voidaan edelleen kehittää haluttuun suuntaan. (Modi 2019.)

DevOps on riippumaton työkaluista, tiimit voivat itse valita toimivimmat ja yhteensopivimmat työkalut. Seuraavassa luvussa käyn läpi työkalut, joita käytän tässä työssä. Seroter (2014) kuitenkin tunnistaa useamman tarvittavan työkalu-kategorian, jotka tulisi täyttää. Ne ovat yhteistyö, suunnittelu, ongelmien seuraus (issue tracking), monitorointi, konfiguraation hallinta, versionhallinta, kehitysympäristö, jatkuva integraatio, ja käyttöönotto. Yhteistyön edistämiseen sopii työkaluksi esimerkiksi Slack. Suunnitteluun tarvitaan työkalu, josta jokainen tiiminjäsen voi nähdä, mitä muut tekevät, sekä tehtävien riippuvuudet toisistaan. Tähän sopii esimerkiksi Trello, tai muu samantapainen Kanban-taulu. Ongelmien seuraus -työkalulla tarkoitetaan työkalua, johon voidaan kirjata

löytyneet ohjelmavirheet ja seurata niiden korjausta. Monitorointi on tarpeellista, että voidaan huomata mahdolliset ongelmat tuotetun ohjelmiston toiminnassa. Esimerkiksi Graphite on siihen sopiva työkalu. Konfiguraation hallinnalla tarkoitetaan esimerkiksi konfiguraatiota, jolla luodaan pilvipalveluun infrastruktuuria, jonka halutaan pysyvän samanlaisena käyttöönotosta riippumatta. Infrastruktuurin perusteella siihen on monia työkaluja, esimerkiksi Puppet ja PowerShell. Versionhallintaan työkaluksi sopii esimerkiksi Git. Kehitysympäristön olisi hyvä olla samanlainen jokaiselle tiiminjäsenellä, jota vältytään tilanteilta, jossa koodi ei toimikaan samalla tavalla ajettaessa eri tiiminjäsenen ympäristöissä estäen yhteistyön tekemisen sen kehityksessä. Esimerkiksi Vagrant on hyvä työkalu rakennettaessa virtuaaliympäristöjä. Jatkuva integraatio ja käyttöönotto liittyvät toisiinsa sekä versionhallintaan, ja saatavilla on työkaluja, jotka hoitavat molemmat tai vain toisen tehtävän. Esimerkiksi TravisCI on työkalu jatkuvan integraation hoitoon ja Octopus käyttöönottoon. (Seroter 2014.)

Automaation tavoitteena on helpottaa ihmisten työtä. Automaatio voi suorittaa tehtävänsä jopa paremmin ja luotettavammin kuin ihminen. Lisäksi jotkut sovellusten hallintaan liittyvät tehtävät ovat liian monimutkaisia ja vaativat nopeutta, johon ihminen ei pysty. Nykyään automaatio on osa monia teknisiä järjestelmiä ja se toteutetaan tietotekniikalla. Jotta automaatio olisi helppokäyttöinen ja kustannustehokas, tulee sen suunnittelijan tuntea kohteena oleva prosessi ja sen halutut toiminnot hyvin. (Koskinen 2017.)

Automaatio on myös osa DevOpsia sen mahdollistajana. Automatisoitu koodin koonti ja testaus mahdollistaa sujuvan jatkuvan integraation. Koonti suoritetaan esimerkiksi aina, kun koodiin tehdyt muutokset tallennetaan repositorioon. (Modi 2019.)

DevOpsin hyödyiksi nähdään nopeutuneet julkaisut ja sitä kautta myös nopeammin saatava palaute asiakkailta. Lisäksi automatisoidun julkaisemisen nopeus ja helppous vapauttaa kehittäjien aikaa, jota voidaan käyttää paremmin esimerkiksi uusien ominaisuuksien kehittämiseen. Toisena hyötynä nähdään kommunikaation lisääntyminen eri tiiminjäsenten välillä, sekä yhteistyö yli

tiimirajojen. Eri asioiden parissa työskentelevät ihmiset voivat oppia toisiltaan, ja muuten piiloon jääneet taidot voivat nousta esiin parantaen yrityksen kilpailukykyä. (Riungu-Kalliosaari et al. 2016.)

Kritiikkiä DevOps on saanut määritelmänsä epäselvyydestä. Se tekee yrityksille ja tiimeille hankalaksi muuttaa toimintaansa DevOpsin suuntaan, kun standardisoituja käytäntöjä ei ole. Lisäksi DevOps voi näyttää erilaiselta eri yrityksissä, kun kukin ottaa käyttöönsä vain toimintaansa sopivat periaatteet, mikä toisaalta osoittaa DevOpsin mukautuvuuden kunkin tarpeisiin, mutta toisaalta tekee hankalaksi toisista yrityksistä ja tiimeistä mallin ottamisen omaan muutokseen. Muutos entisiin toimintamalleihin voi myös olla iso, ja sen vieminen kunnolla läpi ja toimivaksi riippuu paljon yrityksen sisäisestä kulttuurista ja jopa yksittäisten työntekijöiden suhtautumisesta uuteen malliin. Yrityksen sisällä eri prosessit ja ympäristöt voivat olla niin erilaisia keskenään, että kaiken automatisointi on erittäin hankalaa. Myös avoin kommunikaatio saa kritiikkiä siitä, että se voi olla turvallisuusriski, joten DevOps ei sovellu kaikille aloille, eikä välttämättä kaikille tiimeille jonkin yrityksen sisällä. (Riungu-Kalliosaari et al. 2016.)

3 Teknologiat

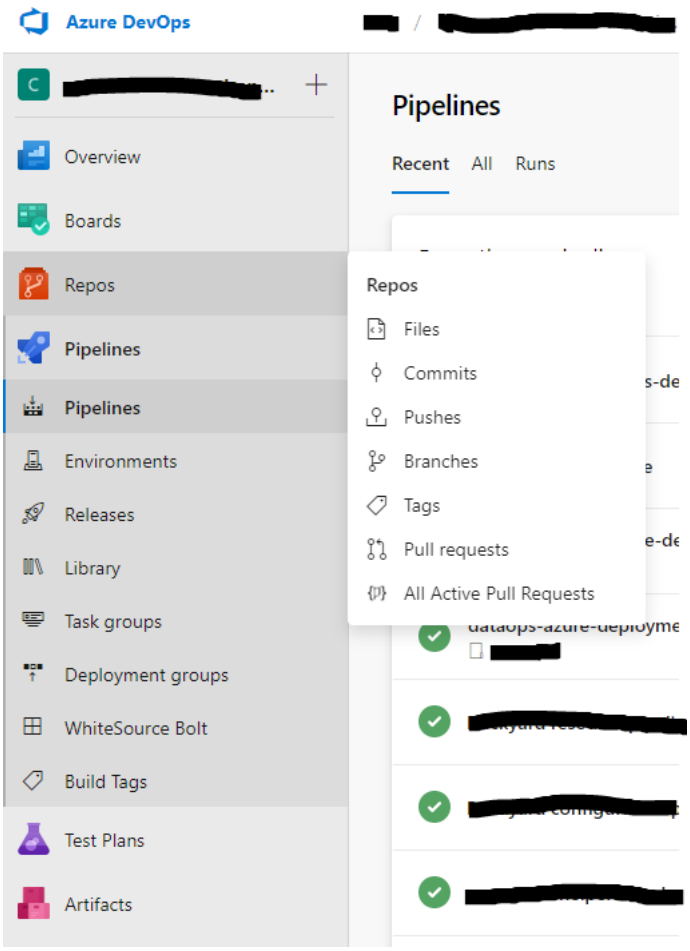
3.1 Azure-pilvialusta

Pilvipalvelut ovat kasvava osa lähes jokaisen yrityksen IT-strategiaa. Azure on Microsoftin pilvialusta, joka tarjoaa monia palveluja, jotka helpottavat DevOpsin periaatteiden edistämistä. Azure tukee monia eri työkaluja, ohjelmointikieliä ja kehyksiä. Sen ideana on olla yhteensopiva monien teknologioiden kanssa, jotta käyttäjä voi valita mieleisensä. (Modi 2019.)

Azure DevOps on pilvipohjainen alusta, joka tarjoaa erialisia palveluita sovellusten kehityksen, lähdekoodin, ja käyttöönoton hallintaan. Kuvassa 3.1 on DevOpsin käyttöliittymän sivupaneeli, jossa näkyy eri palveluita. Sen tarkoitus on helpottaa yhteistyötä tiiminjäsenten kesken tarjoamalla yksi käyttöliittymä sovelluksen koko elinkaaren – suunnittelusta toteutukseen ja käyttöönotosta monitorointiin ja päivittämiseen – hallintaan. (Modi 2019.)

Azure Pipelines on Azure DevOpsin palvelu, jota käytetään automatisoimaan koodin koonti ja testaus. Se siis toteuttaa jatkuvan integraation ja jatkuvan toimituksen periaatteita. Azure Pipelines tukee myös eri versionhallinta-järjestelmiä, joihin viety koodimuutos voi automaattisesti laukaista uuden liukuhihna-ajon. (Microsoft 2022f.)

Azure Pipelines käyttää YAML-tiedostoja liukuhihnojen määrittelyyn (Microsoft 2022c). YAML-lyhenne tulee sanoista Yet Another Markup Language, eli jälleen yksi merkintäkieli. Se on suunniteltu optimaalisesti konfiguraatio-tiedostojen kirjoittamiseen. (Ingerson & Evans & Ben-Kiki 2001.)



Kuva 3.1: ADO:n käyttöliittymää.

Toimeksiantaja käyttää Azure DevOpsia muun muassa lähdekoodin versiohallintaan, sekä Azure Pipelinesia automaatioon. Niitä käytetään myös tässä projektissa, jotta kaikki tehty olisi yhteensopivaa, sekä jotta joku toinen yrityksen työntekijä voisi mahdollisesti jatkaa tämän kehitystä eteenpäin.

3.2 Azure Databricks –pilvipalvelu

Databricks on pilvipalvelu, jossa voi ajaa Python-, SQL- ja R-koodia. Sitä käytetään data-analysointiin eli tarkemmin sanottuna esimerkiksi datan hakemiseen tietokannasta, datan muuttamiseen, visualisointiin ja koneoppimiseen.

Databricks tarjoaa interaktiivisen työtilan, jota sekä datainsinöörit, että data-analyytikot voivat käyttää. Se siis tuo yrityksen sisällä eri tekijöitä yhteen ja toteuttaa siten DevOpsin ihanteita. Työtilassa voidaan luoda uusia muistioita (notebook), jotka koostuvat soluista, joissa voi olla esimerkiksi vain tekstiä tai koodinpätkiä, joita voidaan ajaa erikseen. Muistioiden ja niitä sisältävien kansioiden käyttöoikeuksia voidaan hallita niin, että esimerkiksi tietyn käyttäjäryhmän jäsenet eivät pysty ajamaan tai muokkaamaan sitä. Databricksiä voidaan käyttää työtilan lisäksi myös ohjelmallisesti REST API -rajapinnan kautta. Databricks tarjoaa pilviresurssien laskentatehoa koodin ajamiseen. (Microsoft 2022d).

Databricksiä voidaan käyttää Microsoft Azuren lisäksi myös Amazon Web Servicen (AWS), Google Cloudin pilvialustoilla (Databricks 2022a). Toimeksiantaja käyttää Azure Databricksiä, jotta siihen voidaan helposti yhdistää myös muita Azuren palveluita, kuten tietokantoja.

Tässä projektissa Databricksitä itsessään käytetään vain API-rajapintaa, jossa on komennot lähes kaikille työtilan käyttöliittymän toiminnoille.

3.3 PowerShell-ohjelmointikieli

PowerShell on olio-ohjelmointi kieli, joka pohjautuu .NET:iin. Siitä on julkaistu erilaisia versioita, esimerkiksi Windows PowerShell, joka on tullut esiasennettuna Windows-käyttöjärjestelmien mukana vuodesta 2006. (das Neves & Peters 2018).

Tässä projektissa käytetään Windows PowerShell 5.1 -versiota, joka on esiasennettuna Windows 10 -käyttöjärjestelmässä.

Myös Azure tukee laajasti PowerShellin käyttöä ja monille Azuren resursseille on kehitetty omat PowerShell moduulinsa (Modi 2019). Tämän tuen vuoksi PowerShell on toimeksiantajalla käytössä ja se valittiin myös tähän kieleksi, vaikka tässä projektissa ei mainittuja moduuleita käytetäkään. Tulevaisuudessa sitä saatetaan kehittää edelleen, ja silloin voidaan haluta käyttää Azuren

moduuleja, esimerkiksi Databricksin kanssa tarvittavien Azuren resurssien käyttöön.

4 Suunnittelu ja implementaatio

4.1 Suunnitelma

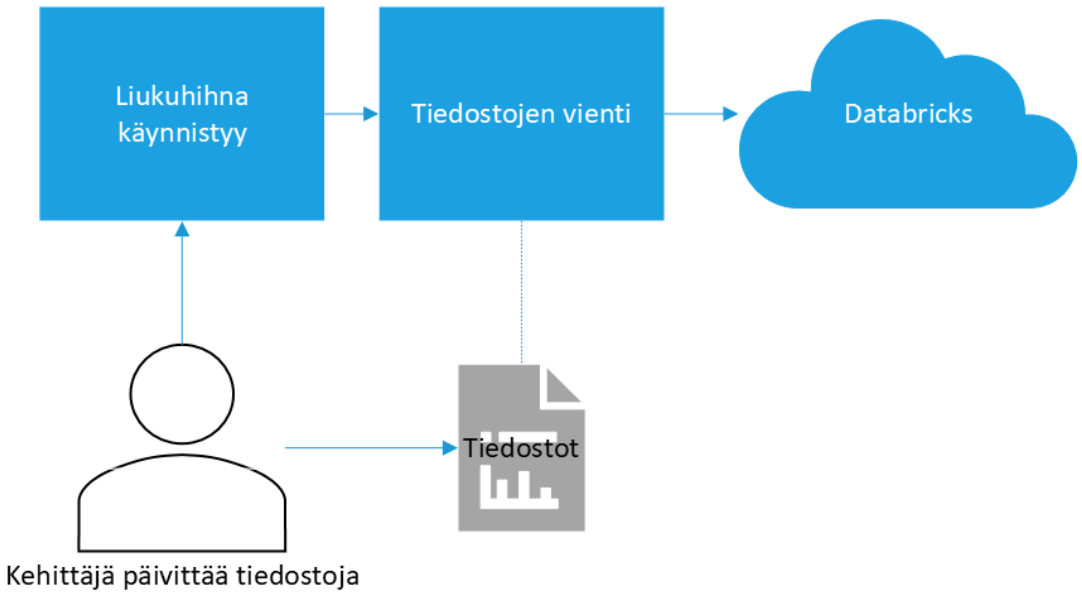
Nykyisessä ratkaisussa Databricks-ympäristöön viedään sen luonnin yhteydessä joukko Python-tiedostoja. Nämä tiedostot pitävät sisällään ympäristön käytössä tarpeellisia funktioita, jotka esimerkiksi helpottavat muiden palveluiden käyttöä Databricksin kanssa, sekä käyttäjille suunnattuja tutoriaaleja. Ympäristön luonti on suurimmaksi osaksi automatisoitu prosessi, mutta sen lisäksi ei ole käytössä muita automatisoituja prosesseja. Toimeksiantaja kehittää jatkuvasti Databricksiin lisättyjä palveluita, jolloin jo käytössä oleviin ympäristöihin joudutaan viemään Python-tiedostojen uusitut versiot. Tämä on tähän mennessä tehty manuaalisesti yksi ympäristö kerrallaan. Lisäksi Databricksin käyttöliittymä ei anna tuoda kuin yhden tiedoston kerrallaan. Niinpä tiedostojen vienti ympäristöihin on erittäin hidas prosessi ja siinä tapahtuu helposti inhimillisiä virheitä, kuten yksittäisen ympäristön, tai tiedoston unohtuminen. Esimerkiksi tilanteessa, jossa Python-tiedostoissa on havaittu isoja virheitä, jotka päivitys korjaa, tarkoittaa prosessin hitaus pitkää katkosta analyytikoiden työntekoon.

Jotta tiedostojen päivittäminen olisi helpompaa ja nopeampaa, halutaan prosessi automatisoida. Ideaalitoteutuksessa automatisoidun liukuhihnan laukaisisi Python-tiedostojen päivitys repositorion master-haaraan, eli se toteuttaisi jatkuvan käyttöönoton ja toimituksen periaatteita. Päivitysten nopeudella toivotaan olevan positiivisia vaikutuksia; esimerkiksi se, että kehittäjien aika ei mene ympäristöjen manuaaliseen päivitykseen, vaan he voisivat keskittyä vain kehitystyöhön ja tuoda entistä enemmän ja parempia palveluita analyytikoiden käyttöön.

Python-tiedostoille tulee luoda kansioita, tai käyttää ympäristössä jo valmiina olevia kansioita. Tiedostoja on sekä funktioita, että tutoriaaleja, jotka jaetaan erillisiin kansioihin. Lisäksi tutoriaaleille on vielä alakansioita niiden aiheen mukaan.

Toimeksiantajalla on noin kymmenen Databricks-ympäristöä niin sanotusti tuotannossa, eli analyytikoiden käytössä. Ne voidaan jakaa kahteen eri tyyppiin niitä käyttävien tiimien perusteella, joilla on hieman erilaiset käyttötarkoitukset ympäristöilleen. Kaikkia Python-tiedostoja ei haluta tuoda tietyn tyyppisiin ympäristöihin, sillä osa tiimeistä ei käytä tiettyjä muita palveluita. Lisäksi osa tiedostoista on sellaisia, että kaikkien käyttäjien ei tarvitse pystyä ajamaan niitä, sillä ne on tarkoitettu ympäristön hallinnointiin, eli vain ylläpitäjien käyttöön. Tämä tarkoittaa, että eri tiedostoihin tarvitsee asettaa erilaisia käyttöoikeuksia eri käyttäjäryhmille.

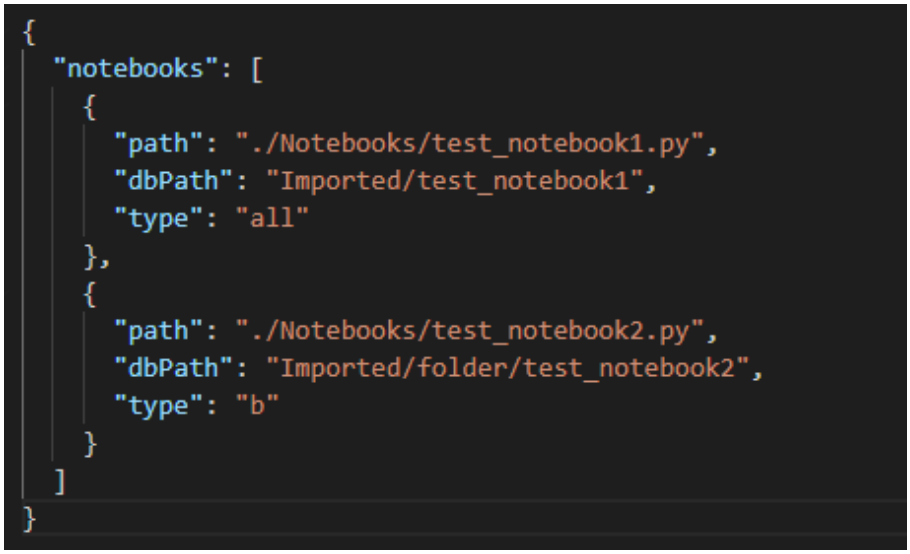
4.2 Ensimmäinen versio



Kuva 4.1 Prosessikaavio liukuhihnan ensimmäisestä versiosta.

Tavoitteenani oli kehittää Azure Pipelines-alustalla ajettava liukuhihna, joka vie tiedostoja Databricks-ympäristöihin. Kuvassa 4.1 on kuvattuna, millainen prosessi siitä tehtiin. Kun kehittäjä päivittää tiedostoja ja tekee pushin repositorion master-haaraan, liukuhihna käynnistyy automaattisesti ja vie repositoriossa olevat tiedostojen uusimmat versiot Databricks-ympäristöihin.

Seuraavaksi selostan tarkemmin, miten kehitin tiedostot vievän ohjelman. Aiemmin oli kehitetty liukuhihna Databricks-ympäristön luontiin ja sen osana viedään myös tiedostoja, joten otin sen mallipohjaksi. Sain siitä mallin, miten määritellä vietävät tiedostot JSON-tiedostoon. Kansiossa *Notebooks* on kaikki Python-tiedostot, jotka halutaan viedä Databricks-ympäristöihin. Tiedostossa *notebooks-list.json* on lista ”notebooks”, jossa on määriteltynä kaikista Python tiedostoista: 1) polku paikallisessa kansiossa, 2) haluttu polku Databricks—ympäristössä, 3) minkä tyyppisiin ympäristöihin se halutaan viedä (kuva 4.2).



Kuva 4.2: Esimerkki *notebooks-list.json* tiedostosta.

Tiedostot viedään Databricksiin tekemällä POST-pyyntö ympäristön API-endpointiin osoitteeseen ”<ympäristön osoite>/api/2.0/workspace/import” (Databricks 2022b). Pyynnön bodyssa on Python-tiedosto muutettuna Base64-muotoon, sekä tiedoston haluttu polku ympäristössä. Kukin tiedosto muutetaan Base64-muotoon koodin ajon aikana, eli tiedostot ovat kansiossa helposti luettavassa ja muokattavassa muodossa. Foreach-silmukassa käydään läpi jokainen *notebooks-list.json* -tiedostossa määritelty tiedosto, ja viedään jokainen Databricksiin.

Jotta tiedosto saadaan vietyä ympäristöön, pitää siellä olla valmiina sen halutussa polussa mainitut kansiot. Tiedostossa *directories-list.json* on lista ”directories”, jossa on määriteltynä jokaisesta tarvittavasta kansiosta 1) polku

Databricksissä ja 2) minkä tyyppisiin ympäristöihin se halutaan. Taas foreach-silmukassa käydään läpi jokainen kansio ja luodaan se ympäristöön POST-pyynnöllä osoitteeseen ”<ympäristön osoite>/api/2.0/workspace/mkdir” (Databricks 2022b). Jos ympäristössä on jo samanniminen kansio, ei sitä poisteta, vaan se pysyy muuttumattomana. Kansioiden luonnin jälkeen haetaan GET-pyynnöllä osoitteeseen ”<ympäristön osoite>/api/2.0/workspace/list” (Databricks 2022b) lista ympäristön kansioista. Vastauksessa on mukana kunkin kansion id-tunniste, joka tarvitaan seuraavaan pyyntöön.

Seuraavaksi asetetaan kansioihin käyttäjille pääsyoikeuksia. Se tehdään PATCH-pyynnöllä osoitteeseen ”<ympäristön osoite>/api/2.0/preview/permissions/directories/<kansion id>” (Databricks 2022b) Ylläpitäjille annetaan ”Can Manage” -oikeus kaikkiin kansioihin ja tavallisille käyttäjille annetaan ”Can Run” -oikeus. Tavallisten käyttäjien ei haluta pystyvän muokkaamaan tuotuja tiedostoja, jotta niiden turvallisuus ja toimivuus voidaan taata. Eri käyttöoikeudet on selitetty taulukossa 5.1.

Taulukko 4.1. Kansion käyttöoikeudet (Microsoft, 2022e).

Käyttöoikeus	Selitys
Can Read	Voi nähdä ja kopioida tiedoston.
Can Run	Voi ajaa tiedoston koodin.
Can Edit	Voi muokata tiedostoa.
Can Manage	Voi luoda uusia tiedostoja kansioon, poistaa, siirtää ja uudelleen nimetä tiedostoja, sekä muuttaa käyttöoikeuksia.

Foreach-silmukassa käydään läpi taas jokainen *directories-list.json* -tiedoston kansio ja haetaan kansion id aiemmin saadusta listasta suodattamalla listasta kansiot, joiden nimi ei ole sama kuin käsiteltävän kansion. Lähetetään PATCH-pyyntö, jonka bodyyn laitetaan lista käyttäjäryhmistä ja halutusta käyttöoikeudesta.

Kirjoitettiin ulompi silmukka, joka käsittelee jokaisen ympäristön erikseen, jonka sisään tuli edellä kuvatut vaiheet. Tiedostossa *workspaces-list.json* on lista Databricks-ympäristöistä sekä 1) ympäristön nimi, 2) ympäristön URL-osoite ja 3) ympäristön tyyppi. Ympäristön nimeä käytetään tulosteessa, jotta on helppo tietää minkä ympäristön käsittelyssä mahdollinen ongelma tapahtuu. Silmukan alussa tulostetaan viesti ”Working on workspace: <ympäristön nimi>”. URL-osoitetta käytetään osana API-pyyntöjen URLia.

Muutettiin kansioiden luomista ja tiedostojen vientiä niin, että ensin tarkistetaan, onko käsiteltävän ympäristön ja kansion, tai tiedoston, tyyppi samat. Jos ei ole, siirrytään silmukoissa seuraavaan kansioon, tai tiedostoon. Tätä varten tehtiin erillinen funktio *Compare-Type*, joka vertaa sille annettuja *workspace-type* ja *object-type* muuttujia keskenään. Funktio palauttaa joko arvon tosi, tai epätosi, riippuen siitä ovatko muuttujat samanarvoiset. Kansio tai tiedosto voi olla myös tyypiltään ”all”, joka tarkoittaa, että se halutaan kaiken tyyppisiin ympäristöihin. Jos *object-type* on ”all”, palauttaa *Compare-Type* arvon tosi.

Databricksin API-pyynnöt on aina todennettava käyttämällä ympäristössä luotua personal access tokenia. Token annetaan API-pyynnön headerissa. Koodiin tokenit annetaan parametreinä liukuhihnan määrittelevästä YAML-tiedostosta. Jokaisen ympäristön token annetaan omana parametrinään, ja ne on nimetty ”<ympäristön nimi>Token”. Koodissa tokeneista luodaan sanakirja, jossa avaimena on ympäristön nimi ja arvona vastaava token. Kunkin ympäristön token saadaan sitten haettua sanakirjasta ympäristön nimellä (kuva 4.3). Kovakoodattujen sanakirjan avaimien täytyy olla samoja, kuin *workspaces-list.json* -tiedostossa määritellyt ympäristöjen nimet.

```
param(
    [string] $test1Token,
    [string] $test2Token,
    [string] $test3Token,
    [string] $workspaceListPath = '../Templates/workspaces-list.json'
)

$tokensDict = @{
    'test1' = $test1Token
    'test2' = $test2Token
    'test3' = $test3Token
}

$workspaceList = Get-Content $workspaceListPath | ConvertFrom-Json

foreach ($workspace in $workspaceList.workspaces)
{
    $token = $tokensDict[$workspace.name]
}
```

Kuva 4.3: Esimerkki tokenin hakemisesta sanakirjasta.

YAML-tiedostossa määritellään liukuhihna. Sille voidaan määritellä erilaisia osia, joissa voidaan ajaa kooditiedostoja, tai vaikka vain yksi komento. Tässä määritellään liukuhihnalle vain yksi tehtävä (job), ajaa *import-notebooks.ps* -tiedosto, ja antaa sille parametreinä eri ympäristöihin tokenit. Liukuhihna hakee tokenit sille Azure Pipelinesiin käyttöliittymässä annetuista muuttujista (variables). Kun liukuhihna-tiedosto tallennetaan git-repositorioon, osaa Azure Pipelines ajaa sen automaattisesti, jos sen nimenä on *azure-pipeline.yml*. Tallennetaan tiedosto tuolla nimellä, ja viedään se Azure DevOpsissa tallennettuun git-repositorioon. Sitten voidaan siirtyä Azure Pipelines -sivulle, ja löytää sieltä luotu liukuhihna. Avaamalla liukuhihna, voidaan sitä muokata. Variables-listaan lisätään uusina arvoina ympäristöissä luodut personal access tokenit. Kussakin ympäristössä luodaan token käyttäjälle, jolla on ”Can Manage” –oikeudet ympäristön juurihakemistoon, jotta tokenia voidaan käyttää kansioiden luomiseen juurihakemistoon. Kopioidaan token ja tallennetaan se Variables-listaan nimellä ”<ympäristön nimi>-ACCESS-TOKEN”. Asetetaan arvo salaiseksi, jotta sitä ei pysty näkemään Azue Pipelinesin käyttöliittymässä, sillä tokenin ei haluta päätyvän vääriin käsiin. Arvo saadaan haettua *azure-*

pipeline.yml-tiedostossa syntaksilla ”${{<muuttujan nimi>}}”. Muuttujan nimi tiedostossa täytyy olla sama kuin Azure Pipelinesissa asetettu. Asetetaan kukin token liukuhihnan tehtävän ympäristömuuttujaksi, jotka sitten annetaan parametreinä.

Liukuhihna halutaan ajaa aina, kun Python-tiedostoihin on tehty muutoksia. Se onnistuu automatisoimalla ajo asettamalla sille laukaisin (trigger) *azure-pipeline.yml*-tiedostossa. Laukaisijaksi asetetaan ”main” eli master-haara (kuva 4.4), eli aina kun haaraan tehdään push, liukuhihna ajetaan. Python-tiedostoja kehitetään omissa haaroissaan ja ne tuodaan masteriin vasta kun ne ovat valmiit vietäväksi ympäristöihin.

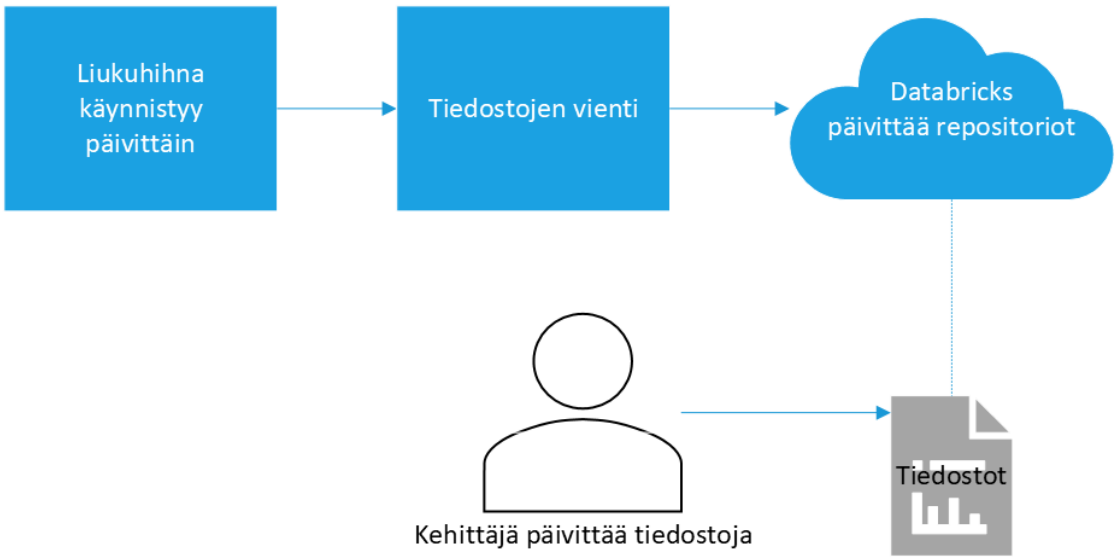
```
# Run pipeline when pushed to branch 'main'
trigger:
- main
```

Kuva 4.4 Liukuhihnan laukaisin.

4.3 Toinen versio

Jatkuvan kehityksen ja DevOpsin hengessä päätettiin tehdä toinen versio liukuhihnasta ja ohjelmasta. Ensimmäisessä versiossa tokenit haettiin sanakirjasta ympäristön nimellä. Tämä ratkaisu vaati, että, jos haluttiin lisätä uusi ympäristö käsiteltäväksi, piti tehdä muutoksia *azure-pipelines.yml-, import-notebooks.ps*-, ja *workspaces-list.json*- tiedostoihin. Lisäksi piti olla tarkkana, että eri tiedostoissa ympäristön nimi tuli kirjoitettua samalla tavalla. Ainakin pariin kertaan tässä tapahtui virheitä. Siispä haluttiin tehdä koodiin muokkauksia, jotta uusien ympäristöjen lisääminen olisi suoraviivaisempaa ja vähemmän altis kirjoitusvirheille. Sen lisäksi Databricks julkaisi elokuussa 2021, ensimmäisen version kehityksen aikana, uuden ominaisuuden (Databricks 2021): git repositoriot. Git-repositorio on versionhallintaan käytetty jaettu koodivarasto. Kehittäjät voivat ladata itselleen paikallisen version projektin koodista ja pilvipalvelussa pidetään keskinen etä-repositorio, johon kehittäjät tekevät tarvittaessa päivityksiä. Etä-repositoriosta muut voivat sitten kätevästi

hakea myös itselleen päivitetyt versiot. (Modi 2019.) Päätettiin tiedostojen viennin sijaan tallentaa tiedostot etä-repositorioon ja liukuhihnalla sitten hakea repositorio Databricksiin. Tämän ratkaisun etuna on sen nopeus: tiedostojen vienti yksitellen vei useamman minuutin ja pull vain pari sekuntia ympäristöä kohden. Heikkouksena on, ettei yksittäisille kansioille voi asettaa erillisiä käyttöoikeuksia, vaan ainoastaan koko repositorion tasolla, eikä erityyppisiin ympäristöihin voida viedä erilaisia tiedostoja, ellei niitä erottele eri repositorioihinsa. Päädyttiin jakamaan tiedostot kahteen eri repositorioon, joihin käyttäjät saavat erilaiset käyttöoikeudet. Kaikkiin ympäristöihin viedään molemmat repositoriot, vaikka joitain tiedostoja ei toisen tyyppisissä ympäristöissä tarvitakaan.



Kuva 4.5 Prosessikaavio liukuhihnan toisesta versiosta.

Kuvassa 4.5 on kuvattuna prosessi ohjelman toisen version kanssa. Tällä kertaa liukuhihna on ajastettu käynnistymään päivittäin ja se päivittää Databricks-ympäristöjen repositoriot hakemalla uusimmat versiot etä-repositoriosta. Kehittäjä päivittää tiedostoja etä-repositorioon, mutta päivitys ei enään laukaise liukuhihnan ajoa.

YML-tiedostoon tuli muutoksia. YAML-kielen syntaksi oli aikaisemmin ollut minulle vieras, jolloin kaikkia sen ominaisuuksia en osannut hyödyntää. Projektin aikana luettiin siitä enemmän ja otettiin enemmän käyttöön toisessa versiossa. Luotiin YML-tiedostossa silmukka, joka luo jokaiselle ympäristölle oman askeleensa (step) liukuhihnalla (kuva 4.6). Lista ympäristöistä on nyt YML-tiedostossa, jotta sen silmukka voi käydä niistä jokaisen läpi. Jokaisesta ympäristöstä on listattuna 1) nimi, 2) URL-osoite ja 3) muuttujan, joka on ympäristön access token, nimi. Silmukassa sitten käydään jokainen ympäristö läpi, ajetaan ohjelma ja annetaan sille parametreinä vain käsiteltävän ympäristön URL ja token. Sillä jokaisessa askeleessa käsitellään erillisiä ympäristöjä, asetetaan askeleen ehdoksi (condition) funktio *succeededOrFailed*, jolloin askel suoritetaan riippumatta siitä, tapahtuiko edellisellä askeleella virheitä (Microsoft 2022b). Vaikka edellisen ympäristön käsittely ei olisi onnistunut, ei sillä ole mitään vaikutusta seuraavaan ympäristöön.

```
jobs:
- job: Pull_From_Remote_Repo
  # Specify agent pool.
  pool:
    name: specific-agent-pool
  steps:
  - ${{ each workspace in parameters.workspaces }}:
    - task: PowerShell@2
      # Run step even if previous step failed.
      condition: succeededOrFailed()
      displayName: ${{ workspace.workspaceName }}
      env:
        TOKEN: ${{ workspace.tokenName }}
      inputs:
        filePath: './Scripts/repo-pull.ps1'
        Arguments:
          -token "${env:TOKEN}"
          -workspaceName ${{ workspace.workspaceName }}
          -workspaceUrl ${{ workspace.url }}
        pwsh: True
```

Kuva 4.6 Silmukka YML-tiedostossa.

Lisäksi askeleen nimeksi (display name) asetetaan ympäristön nimi. Näin liukuhihnaa ajettaessa tulee jokaiselle ympäristölle oma alasivunsa liukuhihnan tulosteeseen ja siitä on helpompi huomata, minkä ympäristön käsittelyssä on mahdollisesti tullut ongelmia. Ensimmäisessä versiossa kaikki ympäristöt käsiteltiin samassa askeleessa ja tulostettiin ohjelman etenemisestä tietoa samalle sivulle. Sivu oli pitkä ja sitä joutui vierittämään paljon löytääkseen kohdan, jossa käsiteltiin jotain tiettyä ympäristöä.

Koodiin tuli isoja muutoksia ja tiedosto sai uuden nimen *repo-pull.ps*. Uloin silmukka siirtyi *azure-pipeline.yml* –tiedostoon, parametri-lista lyheni ja tokenit sisältävä sanakirja poistui. Ohjelma tarkistaa, että ympäristössä on *repositories-list.json*-tiedostossa määritellyt kansiot *Repos/Notebooks* ja *Repos/Maintanence*. Foreach-silmukassa käydään kansiot läpi ja, jos niitä ei ole, luodaan kansiot POST-pyynnöllä osoitteeseen ”<ympäristön osoite>/api/2.0/workspace/mkdir” (Databricks 2022b). Käyttäjille annetaan käyttöoikeudet kansioihin toisella POST-pyynnöllä osoitteeseen ” <ympäristön osoite >/api/2.0/preview/permissions/directories/<kansion id>” (Databricks 2022b). Kansioon *Notebooks* käyttäjät saavat käyttöoikeuden ”Can Run”. Kansioon *Maintanence* käyttäjät eivät saa mitään käyttöoikeuksia, sillä siihen lisättävä repositorio pitää sisällään vain palveluita, joita vain ympäristön ylläpitäjien tarvitsee käyttää ja mahdollisesti muokata. Aikaisemmassa versiossa käyttäjät olivat saaneet näihin tiedostoihin ”Can Run” –oikeuden, mutta ylläpitäjät luottivat siihen, etteivät käyttäjät ajaisi niitä. Palveluiden ajaminen turhan päiten ei aiheuttaisi ongelmia, mutta tulevaisuudessa kansiossa voi olla myös palveluita, jotka aiheuttaisivat. Ilman mitään käyttöoikeuksia tavalliset käyttäjät eivät edes näe kansiota Databricksin käyttöliittymässä.

Tiedostossa *repositories-list.json* on määriteltynä lista ”repositories” ja jokaisesta repositoriosta 1) haluttu polku ympäristössä ja 2) URL-osoite remote-repostiorioon. Taas foreach-silmukassa käydään läpi jokainen repositorio. Tarkistetaan, onko repositorio jo ympäristössä tekemällä GET-pyyntö osoitteeseen ”<ympäristön osoite>/api/2.0/repos” (Databricks, 2022), josta

saadaan paluuarvona lista kaikista ympäristön repositorioista. Käydään foreach-silmukassa lista läpi ja, jos käsiteltävän repositorion polku on sama kuin luotavan, otetaan sen id-numero talteen. Silmukka on esitetty kuvassa 4.7, jossa ”response” on pyynnön paluuarvo ja ”repository” on *repositories-list.json*-tiedostossa määritelty repositorio. Jos repositoriota ei löydy listalta, luodaan se POST-pyynnöllä samaan osoitteeseen. Tämän pyynnön paluuarvona saadaan luodun repositorion tiedot ja niistä otetaan talteen id-numero.

```
# Find repo with correct path.
    foreach ($repo in $response.repos)
    {
        if ($repo.path -match $repository.path)
        {
            $repoId = $repo.id
            Write-Host "Found repository $($repository.path)."
        }
    }
```

Kuva 4.7 Repositorion id:n etsiminen.

Databricksissä jo oleva repositorio saadaan päivitettyä tekemällä pull etä-repositoriosta. Se onnistuu tekemällä PATCH-pyyntö osoitteeseen ”<ympäristön osoite>/api/2.0/repos/<repositorion id>” (Databricks 2022b). Osoitteessa on mukana repositorion id-numero, joka oli aikaisemmin otettu talteen. Pyynnön bodyyn laitetaan tekstinä sen repositorion haaran (branch) nimi, josta pull halutaan, tässä tapauksessa käytetään ”main” eli master-haaraa.

Etä-repositoriot ovat tallennettuna Azure DevOpsiin. Niiden tuomiseksi Databricksiin ohjelmallisesti, tarvitaan Azure DevOpsissa luoda myös personal access token. ADO:n tokeneille voi asettaa erilaisia oikeuksia eri ADO:n palveluihin. Tähän tarkoitukseen riittää Read-oikeus repositorioihin. Token tallenetaan Databricks-ympäristön *Git Integration* -asetuksiin, sille käyttäjälle, jonka Databrciks tokenia käytetään API-kutsuissa. Ensimmäisessä versiossa oli luotu Databricks tokenit minun käyttäjänimelläni, mutta se ei ollut sopiva ratkaisu, sillä muut eivät voi nähdä milloin tokenit ovat vanhenemassa ja tarvitsevat päivitystä. Niinpä luotiin Azureen ”Service Account” eli tekninen

käyttäjä, jonka salasana, on useamman henkilön tiedossa. Tekninen käyttäjä lisättiin kaikkiin Databricks ympäristöihin käyttäjäksi ja se sai ”Can Manage” -oikeudet Repos-kansioon. Luotiin uudelleen tokenit teknisen käyttäjän käyttäjänimellä ja tallennettiin ne Azure Pipelinesissa liukuhihnan muuttujiksi.

Näillä muutoksilla liukuhihnan toinen versio päätettiin ottaa käyttöön, eli niin sanotusti viedä tuotantoon. Tätä varten liukuhihna asetettiin pyörimään tietyssä agent-poolissa, mikä onnistuu kirjoittamalla poolin nimi YML-tiedostoon (kuva 4.6). Joissain ympäristöissä on rajoitettu mistä IP-osoitteista niihin pystyy ottamaan yhteyttä ja lähettämään pyyntöjä, ja vain tämän tietyn agent-poolin osoite ei ole estetty.

```yaml
# Schedule for pipeline runs.
# The time zone for cron schedules is UTC.
schedules:
- cron: "0 1 * * *"
  displayName: Daily 1AM run
  branches:
    include:
    - main
  # Run pipeline even if no code changes.
  always: true

# CI trigger must be disabled to run on a schedule.
trigger: none
```

Kuva 4.8 Liukuhihnan ajastus.

Lisäksi liukuhihna päätettiin ajastaa pyörimään joka yö (kuva 4.8). Ajastuksen asettaminen onnistuu määrittelemällä YML-tiedostossa aikataulu ja ottamalla pois laukaisin, eli liukuhihnan automaattinen ajo aina kun sen repositorioon tehdään päivitys. Lisäksi asetetaan arvo ”always” todeksi, jolloin liukuhihna ajetaan aina aikataulun mukaisesti, vaikka mitään päivityksiä ei koodiin olisi tehtykään. (Microsoft, 2022a)

Liukuhihnan repositorioon ei pitäisi tulla päivityksiä ainakaan samassa tahdissa kuin Python-tiedostojen repositorioihin, joten liukuhihnan ajaminen vain silloin,

jos siinä on muutoksia olisi liian harvoin. Päivittäinen ajaminen katsottiin sopivaksi, vaikka päivityksiä tiedostoihin ei päivittäin tehdäkään. Liukuhihnan ajaminen turhaan ei aiheuta mitään haittaa, ja silloin kun päivityksiä on, halutaan ne kaikkien käyttäjien saataville seuraavaan päivään mennessä. Lisäksi liukuhihna voidaan ajaa myös manuaalisesti Azure Pipelinesin käyttöliittymästä, jos esimerkiksi kriittisen virheen korjaaminen vaatii nopeampaa toimintaa.

5 Johtopäätökset

Työssäni käsittelin automaatiota ja DevOpsia, sekä Microsoft Azure -palveluja, Databricksiä ja PowerShell-kieltä. Opinnäytetyön tavoitteena oli luoda liukuhihna Azure Pipelines -alustalle, joka vie tiedostoja Databricks-ympäristöihin. Tavoite saavutettiin, kun luotu liukuhihna tuli toimeksiantajalle käyttöön ja se ajetaan ajastetusti päivittäin. Ohjelma päivittää ympäristöjen repositoriot, jolloin ympäristöihin saadaan Python-tiedostojen uusimmat versiot.

Opinnäytetyön kirjoittamisen aikana opin Azure Pipelines -alustasta paljon lisää, vaikka sitä olin käyttänyt töissä aiemminkin. Uudet opitut asiat auttavat varmasti jatkossa hyödyntämään alustaa entistä paremmin. Opin uutta myös Databricksistä, sen API:sta ja uusista repositorio-ominaisuuksista. Tulevaisuudessa yrityksessä halutaan hyödyntää repositorioita enemmänkin, jolloin oppimastani on hyötyä. Opin DevOpsista ja automaatiosta, sekä siitä, miten niiden käytäntöjä sovelletaan suunnittelussa ja tekemisessä. Tiedostojen viennistä saatiin automatisoitu ja nopea prosessi, ja kehittäjien aikaa säästyy tärkeämpään tekemiseen kuin ympäristöjen päivittämiseen käsin. Jatkuva kehitys, käyttöönotto ja toimitus ovat osa toimeksiantaja toimintakulttuuria, ja luotu liukuhihna tukee sitä. DevOps:iin siirtymistä ovat vaikeuttaneet joidenkin työntekijöiden haluttomuus muuttaa totuttuja toimintatapojaan. Mutta toimeksiantaja pyrkii laajentamaan DevOps:n käyttöä, sillä se on tuonut nopeutta ja tehokkuutta.

Vaikka liukuhihna onkin toimiva, jäi siihen vielä parannettavaa ja jatkokehitettävää. Uusien ympäristöjen lisääminen käsiteltävien joukkoon on edelleen hankalaa ja vaatii muutamia käsin tehtäviä toimintoja. Ympäristöön on lisättävä tekninen käyttäjä. Access tokenit on luotava sekä Azure DevOpsissa että Databricksissä, ja tallennetava oikeisiin paikkoihin. Liukuhihnan muuttajaan tallenetun access tokenin nimen täytyy olla tismalleen sama kuin YML-tiedostossa, eli pienikin kirjoitusvirhe estäisi ohjelman toiminnan kyseisen ympäristön osalta. Nykyisessä versiossa kaikki repositoriot viedään kaikkiin ympäristöihin, vaikka osassa ympäristöjä niitä ei välttämättä tarvitakaan. Olisi

suhteellisen yksinkertaista lisätä takaisin ympäristön tietoihin sen tyyppi ja viedä ympäristöihin ainoastaan samaa tyyppiä olevat repositoriot. Toimeksiantajan tavoitteena tosin on yhdenmukaistaa eri tiimien Databricks-ympäristöjä, joten tämä ratkaisu sopii siihen.

Jatkan työskentelyä toimeksiantajan palkkalistoilla, ja liukuhihnaa kehitetään edelleen eteenpäin. Suunnitteilla on lisätä siihen toiminnallisuutta päivittämään myös ympäristöjen muita osia. Oppimaani ja jo tehtyä voidaan soveltaa ja käyttää myös muiden liukuhihnojen kehittämisessä.

Lähteet

das Neves, D; Peters, J. H. (2018). Learn powershell core 6. 0 : Automate and control administrative tasks using devops principles. Packt Publishing, Limited.

Databricks (2021). Databricks Repos GA. [Databricks:n päivitykset.] https://docs.databricks.com/release-notes/product/2021/august.html#databricks-repos-ga. Luettu 16.4.2022.

Databricks (2022a). Learn. https://databricks.com/learn. Luettu 16.5.2022.

Databricks (2022b). Rest API 2.0. [Databricks:n dokumentaatio.] https://docs.databricks.com/dev-tools/api/2.0/index.html. Luettu 18.4.2022.

Ingerson, Brian & Evans, Clark & Ben-Kiki, Oren (2001). Yet Another Markup Language (YAML) 1.0. https://yaml.org/spec/history/2001-12-10.html. Luettu 26.4.2022.

Koskinen, Kari (2017). Automaatio – mitä se on? Automaatio ennen, nyt ja tulevaisuudessa -artikkelisarja. https://www.automaatioseura.fi/site/assets/files/1380/automaatio_ennen_nyt_ja_tulevaisuudessa_av_artikkelisarja_2018.pdf. Luettu 20.4.2022.

Microsoft (2022a). Configure schedules for pipelines. [Azure Pipelines dokumentaatio] https://docs.microsoft.com/en-us/azure/devops/pipelines/process/scheduled-triggers?view=azure-devops&tabs=yaml. Luettu 26.4.2022.

Microsoft (2022b). Specify conditions. [Azure Pipelines dokumentaatio.] https://docs.microsoft.com/en-us/azure/devops/pipelines/process/conditions?view=azure-devops&tabs=yaml. Luettu 26.4.2022.

Microsoft (2022c). Use Azure Pipelines. [Azure DevOps:n dokumentaatio.] https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/pipelines-get-started?view=azure-devops#define-pipelines-using-yaml-syntax. Luettu 4.4.2022.

Microsoft (2022d). What is Azure Databricks? [Azure Databricksin dokumentaatio.] https://docs.microsoft.com/en-us/azure/databricks/scenarios/what-is-azure-databricks. Luettu 16.5.2022.

Microsoft (2022e). What is Azure Pipelines? [Azure DevOpsin dokumentaatio.] https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops. Luettu 4.4.2022.

Microsoft (2022f). Workspace object access control. [Azure Databricksin dokumentaatio.] https://docs.microsoft.com/en-us/azure/databricks/security/access-control/workspace-acl. Luettu 10.4.2022.

Modi, Ritesh (2019). Azure for Architects: Implementing Cloud Design, DevOps, Containers, IoT, and Serverless Solutions on Your Public Cloud, 2nd Edition. Birmingham: Packt Publishing, Limited.

Riungu-Kalliosaari, L & Mäkinen, S & Lwakatare, L & Tiihonen, J & Männistö, T (2016). DevOps Adoption Benefits and Challenges in Practice: A Case Study. New York: Springer Publishing. https://www.cs.helsinki.fi/u/jutiihon/publications/RiunguKalliosaari2016DevopsAdoptionBenefits.pdf. Luettu 16.5.2022.

Seroter, Richard (2014). Exploring the ENTIRE DevOps Toolchain for (Cloud) Teams. https://www.infoq.com/articles/devops-toolchain/. Luettu 16.5.2022.