Tampereen yliopisto

Ville Penttinen

INFRASTRUKTUURI KOODINA: PULUMI-ALUSTAN ARVIOINTI

Diplomityö
Informaatioteknologian ja viestinnän tiedekunta
Tarkastajat: Kari Systä
David Hästbacka
Huhtikuu 2021

TIIVISTELMÄ

Ville Penttinen: Infrastruktuuri koodina: Pulumi-alustan arviointi
Diplomityö
Tampereen yliopisto
Tietotekniikka, DI
Huhtikuu 2021

Pulumi-alusta on infrastruktuurin hallintaan kehitetty avoimeen lähdekoodiin pohjautuva teknologia, joka mahdollistaa pilvialustainfrastruktuurin resurssien luomisen, päivittämisen ja poistamisen käyttämällä alustan tukemia ohjelmointikieliä. Tyypillisesti määritettäessä infrastruktuuria koodina käytetään deklaratiivisia täsmä- tai konfiguraatiokieliä, kun taas Pulumia käytetään yleisten ohjelmointikielten, kuten C# kanssa. Ohjelmointikielten käyttö mahdollistaa kielille tyypillisten abstraktioiden, vahvan tyypityksen ja käännösaikaisen tarkistuksen hyödyntämisen. Käännettävät ohjelmointikielet tukevat alustan omaksumista.

Tämän työn tavoitteena oli arvioida Pulumi-alustan soveltuvuutta eri arviointikriteerein ohjelmointikonsultointiyritys Profit Software Oy:n tarpeisiin. Alustan arviointiin käytetyt kriteerit jaettiin kahteen kategoriaan: alustakriteereihin ja alustan käyttöönoton kriteereihin. Alustakriteerejä olivat teknologian kypsyys, tuettujen pilvialustojen määrä, kehitysyhteisön aktiivisuus ja alustan suorituskyky. Myös kustannuksia ja Pulumia yrityksenä arvioitiin osana alustakriteerejä. Alustan käyttöönoton kriteerit olivat jatkuvan integraation tuki, alustan tukemat kehitysympäristöt, alustan konfiguroitavuus, laajennettavuus, omaksuttavuus, testattavuus, vian etsinnän tuki ja ylläpidettävyys. Käyttöönoton arvioinnissa hyödynnettiin Microsoftin kehittämää avoimeen lähdekoodiin pohjautuvaa mikropalveluarkkitehtuurin toteuttavaa referenssisovellusta. Tutkimuksessa tehtiin referenssisovellukselle kehitysympäristön pilvialustainfrastruktuuri pilvialustoille Microsoft Azure, Amazon Web Services, Google Cloud Platform ja DigitalOcean.

Työssä tehdyn tutkimuksen tulosten perusteella Pulumi-alusta toimii hyvin esimerkkisovelluksen kaltaisten sovellusten infrastruktuurin hallintaan C#-kielellä. Alusta arvioitiin helposti omaksuttavaksi. Pulumin tunnettuus ei ole samalla tasolla kuin esimerkiksi vastaavan työkalun Terraformin, mikä vaikuttaa saatavilla olevaan kehitysyhteisön tuottamaan materiaalin määrään. Työssä tehdyn alustan arvioinnin perusteella Pulumia suositellaan harkittavaksi yhtenä mahdollisena työkaluna pilvialustainfrastruktuurin hallintaan.

Avainsanat: Pulumi, infrastruktuuri koodina, C#, pilvialustat, Kubernetes, Helm, Terraform

ABSTRACT

Ville Penttinen: Infrastructure as code: evaluation of the Pulumi platform
Master of Science Thesis
Tampere University
Information Technology, MSc
April 2021

Pulumi is an open source platform developed for infrastructure management which allows to create, update and remove cloud-based infrastructure resources using programming languages supported by the platform. Typically, declarative domain-specific languages are used for defining infrastructure as code. Conversely, the Pulumi platform allows defining infrastructure resources using common programming languages, such as C#. Programming languages enable taking advantage of common abstractions, strong typing and compile-time checking. Utilizing compiled programming languages supports the adoption of the platform.

The goal of this thesis was to evaluate the Pulumi platform by using various evaluation criteria to determine if the platform is suitable for a Finnish financial consulting enterprise, Profit Software Ltd. The evaluation criteria were divided into two categories: platform criteria and platform adoption criteria. Platform criteria included the following: technological maturity, the number of supported cloud platforms, activity of the development community and performance of the platform. Furthermore, costs and Pulumi as a company were evaluated in the platform criteria. Platform adoption criteria included the following: continuous integration support, development environments supported by the platform, configurability, extensibility, adoptability, testability, support for debugging and maintainability of the platform. To evaluate the adoptability of the platform, Microsoft's open source microservice sample reference application was used. As part of the evaluation, development environment infrastructure was created for the following cloud platforms: Microsoft Azure, Amazon Web Services, Google Cloud Platform and DigitalOcean.

Based on the results, the Pulumi platform works well for infrastructure management with C# programming language when developing similar applications to the one used for evaluation in this thesis. The platform was assessed to be easily adoptable. The Pulumi platform is not as widely known as its competitors, such as Terraform, which affects how much material is available produced by the development community. Based on the evaluation, the Pulumi platform is recommended for consideration as a tool for managing cloud-based infrastructure.

Keywords: Pulumi, infrastructure as code, C#, cloud platforms, Kubernetes, Helm, Terraform

ALKUSANAT

Haluan kiittää Profit Software Oy:tä mahdollisuudesta tehdä diplomityö itse valitsemastani aiheesta. Kiitokset ohjaajalleni Karille avusta aiheen muodostamisessa sekä tuesta ja kommenteista työn teon aikana. Kiitokset myös Pulumin avoimen lähdekoodin kehitysyhteisön jäsenille, jotka ovat auttaneet työssä vastaamalla ongelmiin ja kehitykseen liittyviin kysymyksiin.

Helsingissä, 14. huhtikuuta 2021

Ville Penttinen

SISÄLLYSLUETTELO

KUVALUETTELO

TAULUKKOLUETTELO

OHJELMA- JA ALGORITMILUETTELO

LYHENTEET JA MERKINNÄT

Amazon EC2	Amazon Elastic Compute Cloud
Amazon RDS	Amazon Relational Database Service
API	Ohjelmointirajapinta (enlg. Application programming interface)
AWS	Amazon Web Services
AWS CDK	AWS Cloud Development Kit
BFF	Backend for Frontend
CD	Jatkuva toimitus (engl. continuous delivery)
CI	Jatkuva integraatio (engl. continuous integration)
CLI	Komentoriviliitäntä (engl. command-line interface)
CLR	Common Language Runtime
GCP	Google Cloud Platform
gRPC	gRPC Remote Procedure Calls
HTTP	Hypertext Transfer Protocol
IaaS	Infrastruktuuri palveluna (engl. Infrastructure as a Service)
IaC	Infrastruktuuri koodina (engl. Infrastructure as Code)
IDE	Integroitu kehitysympäristö (engl. Integrated Development Environment)
JIT	Ajonaikainen kääntäminen (engl. Just-In-Time)
JSON	JavaScript Object Notation
NIST	National Institute of Standards and Technology
PaaS	Alusta palveluna (engl. Platform as a Service)
SaaS	Sovellus palveluna (engl. Software as a Service)
SDK	Ohjelmistokehityspaketti (engl. software development kit)
SQL	Structured Query Language
URN	Yksikäsitteinen nimi (engl. Uniform Resource Name)
YAML	YAML Ain’t Markup Language

1. JOHDANTO

Tässä työssä esitellään infrastruktuuri koodina -lähestymistapaa hyödyntävä Pulumi-alusta. Pulumi-alustan esittelyn lisäksi työssä tehdään arviointi alustan soveltuvuudesta ja sen käytöstä infrastruktuurin hallinnassa.

1.1 Tavoitteet ja rajaus

Tämän työn tavoitteena on arvioida Pulumi-alustan käyttöä ja käyttöönottoa infrastruktuurin hallinnassa hyödyntämällä sen tarjoamia mahdollisuuksia C#-ohjelmointikielen avulla. Työssä arvioidaan sitä, olisiko Pulumi-alusta sellainen, joka voitaisiin ottaa käyttöön Profit Software Oy:n tekemissä pilvialustaprojekteissa.

Pulumi-alustan käyttöönoton arviointiin käytetään esimerkkisovellusta, joka esitellään tarkemmin luvussa 6.2. Esimerkkisovelluksen avulla arvioidaan Pulumin käyttöönottoa olemassa olevan sovelluslähdekoodin kanssa, sekä sitä, miten Pulumin käyttöönotto onnistuu erityisesti kontteja ja Kubernetesta hyödyntävissä ohjelmistoissa. Pulumi-alustaa arvioidaan muun muassa seuraavilla kriteereillä: teknologian kypsyys, kehitysyhteisön aktiivisuus, omaksuttavuus, testattavuus ja ylläpidettävyys.

Osana työssä tehtävää arviointia tehdään lisäksi suorituskykyvertailu Pulumi- ja Terraform-alustojen välillä.

Tässä työssä keskitytään Pulumi-alustan käyttöön C#-ohjelmointikielellä. C#-ohjelmointikieli on vahvasti tyypitetty, olio-ohjelmointia tukeva kieli. C# on valittu käytettäväksi sen takia, että kieli on yhtenä käytetyimmistä ohjelmointikielistä Profit Software Oy:ssä. Lisäksi kielen hyödyntämä .NET-alusta on käyttöjärjestelmäriippumaton, eli se toimii Windows-, Linux- sekä Mac-käyttöjärjestelmissä.

Työssä käytettävät pilvialustat rajataan Microsoft Azure-, Amazon Web Services-, Google Cloud Platform- ja DigitalOcean-pilvialustoihin, joille toteutetaan Kubernetes-klusteri, johon esimerkkisovellus asennetaan. Esimerkkisovellus asennetaan Kubernetes-klusteriin pilvialustariippumattomina kontteina. Pulumin arviointiin vaikuttavat näin ollen myös Kubernetesin ja sen paketinhallintajärjestelmän ominaisuudet.

1.2 Rakenne

Tässä luvussa käydään läpi työn rakenne yleisellä tasolla. Työn toinen luku käsittelee pilvialustoja yleisesti ja niiden käyttöä Profit Software Oy:ssä. Toisessa luvussa esitellään myös esimerkkisovelluksen käyttämät pilvialustariippumattomat Kubernetes ja Helm. Työn kolmannessa luvussa käydään läpi infrastruktuuri koodina -lähestymistapaa yleisluontoisesti sekä sen etuja ja mahdollisia haasteita. Työn neljännessä luvussa esitellään C#-ohjelmointikielen ominaisuuksia siinä määrin kuin ne liittyvät tämän tutkimuksen esimerkkisovelluksen infrastruktuurikoodin toteutukseen ja Pulumin käyttöön. Työn viidennessä luvussa esitellään Pulumi-alusta ja sen toiminnan perusperiaatteet. Alustaa vertaillaan sen kilpailijoihin kuten Terraformiin. Työn kuudennessa luvussa käydään läpi työssä tehtävän tutkimuksen arviointikriteerit ja esitellään arvioinnissa käytettävä esimerkkisovellus. Pulumi-alustan käyttöönoton tutkimus tehdään esimerkkisovelluksen perusteella. Työn seitsemännessä luvussa esitetään alustan arvioinnin tulokset. Työn viimeisenä lukuna on tutkimuksen yhteenveto.

2. PILVIALUSTAT

Profit Software Oy:ssä pilvialustojen käyttö on yleistynyt viime vuosien aikana. Eri pilvialustoista Profit Software Oy:n projekteissa on käytetty seuraavia eri pilvialustoja: Microsoft Azure, Amazon Web Services (AWS) ja Google Cloud Platform (GCP). Näiden lisäksi projekteissa on käytetty pilvialustariippumattomia teknologioita, kuten Kubernetesta.

Yleisesti pilvialustojen käytön odotetaan yleistyvän [72]. Profit Software Oy:n asiakasprojekteissa asiakkaan toiveet otetaan huomioon, mikä vaikuttaa myös pilvialustojen käyttöönottoon.

2.1 Yleisesti

Pilvialustoista puhuttaessa täytyy ensiksi määrittää, mistä pilvialustoista ja niiden tarjoamissa palveluissa on yleisesti ottaen kyse. Yhdysvaltojen kansallinen standardi- ja teknologiainstituutio NIST (engl. National Institute of Standards and Technology) määrittelee pilvipalvelut sellaisina palveluina, jotka ovat saatavilla kaikkialla tarpeen vaatiessa (engl. on-demand) [40]. Määritelmän mukaan pilvipalveluihin liittyvät seuraavat ominaisuudet:

- Tarpeen vaatima itsepalvelu (engl. on-demand self-service)
- Laajamittainen pääsy verkkoon
- Resurssien yhdistäminen (engl. resource pooling)
- Nopea elastisuus (engl. rapid elasticity)
- Palveluiden mittaus (engl. measured service)

Tarpeen vaatima itsepalvelu tarkoittaa sitä, että palveluiden käyttäjät pystyvät hankkimaan (engl. provision) resursseja ilman, että palveluntarjoajaan täytyy olla yhteydessä [40]. Tarpeen vaatima itsepalvelu mahdollistaa resurssien luomisen automatisoinnin, joka on edellytyksenä luvussa 3 esiteltävälle infrastruktuuri koodina -lähestymistavalle.

Laajamittaisella pääsyllä verkkoon tarkoitetaan sitä, että palveluiden tarjoamat ominaisuudet ovat verkon yli saatavilla riippumatta käyttävästä asiakasalustasta [40]. Resurssien yhdistämisellä tarkoitetaan sitä, että pilvialustantarjoajat hyödyntävät ja yhdistävät asiakkaiden tarvitsemia resursseja dynaamisesti [40]. Asiakkaiden kannalta käytettävien resurssien tarkkaa sijaintia ei välttämättä ole tiedossa. Joissakin tapauksissa käyttäjät

pystyvät määrittelemään maan tai mahdollisesti käytettävän palvelinkeskuksen (engl. data center).

Nopea elastisuus mahdollistaa resurssien hankkimisen dynaamisesti ja automaattisesti [40]. Nopealla elastisuudella mahdollistetaan skaalautuvuus, sillä pilvialustan käyttäjän kannalta voidaan varata lisää resursseja tarpeen vaatiessa. Vastaavasti, kun resursseja ei tarvita, ne voidaan sammuttaa ja käynnistää myöhemmin uudestaan.

Palveluiden mittaus tarkoittaa sitä, että palveluita ja niiden suoritusta voidaan seurata eli monitoroida ja mitata [40]. Monitorointi on tärkeä osa esimerkiksi DevOps-kulttuuria [25], jota käsitellään osana lukua 3.

Pilvialustat tyypillisesti tarjoavat käyttäjilleen mahdollisuuden erilaisten palvelumallien (engl. service model) käyttöön [40]. Yleisesti palvelumallit jaetaan kolmeen tasoon:

- Infrastruktuuri palveluna (engl. Infrastructure as a Service)
- Alusta palveluna (engl. Platform as a Service)
- Sovellus palveluna (engl. Software as a Service),

joista kukin palvelumallin taso tarjoaa pilvialustojen käyttäjille erilaisia mahdollisuuksia.

IaaS eli infrastruktuuri palveluna on palvelumalli, jossa alustan toimittaja vastaa alustan infrastruktuurista laitteisto- ja virtualisointitasoilla. Tyypillisesti alustan toimittaja mahdollistaa esimerkiksi virtuaalikoneiden luomisen ja alustan käyttäjän vastuulle jää käyttöjärjestelmien hallinnointi, tiedon varastointi sekä virtuaalikoneisiin asennettavat sovellukset [40]. Esimerkkejä IaaS-palveluista ovat muun muassa pilvialustojen tarjoamat virtuaalikonepalvelut, kuten Amazon EC2 tai Microsoft Azure VM.

PaaS eli pilvialusta palveluna mahdollistaa ohjelmistojen kehittämisen ja julkaisun ilman, että alustan käyttäjän tarvitsee hallinnoida käytössä olevaa infrastruktuuria [40]. Esimerkkejä PaaS-palveluista ovat muun muassa AWS RDS -tietokantapalvelu sekä Microsoft Azure SQL.

SaaS eli sovellus palveluna on palvelumalli, jossa alustan käyttäjät käyttävät pilvialustalla toimivia sovelluksia ilman, että käyttäjät hallinnoivat ohjelmistojen julkaisua tai infrastruktuuria [40]. Tyypillisesti käyttäjät vastaavat ainoastaan käyttämistään tiedoista. Esimerkkejä SaaS-palveluista ovat muun muassa Microsoft Office365 -palvelu sekä luvussa 5 esiteltävä Pulumi SaaS -palvelu.

Eri palvelumallit mahdollistavat erilaisia pilvialustojen käyttötapoja. Pilvialustan toimittajat voivat tarjota myös näitä palvelumalleja yhdistäviä palveluita. Microsoft Azure Kubernetes -palvelu on esimerkki yhdistetystä palvelusta, joka käyttäjän näkökulmasta vaikuttaa PaaS-palvelulta, koska käyttäjän ei tarvitse hallinnoida palvelun käyttämää virtuaalikoneinfrastruktuuria. Microsoft Azure Kubernetes -palvelu kuitenkin mahdollistaa myös käytettävien IaaS-resurssien hallinnoimisen ja konfiguroinnin.

2.2 Kubernetes ja Helm

Kubernetes on Googlen kehittämä konttien hallintajärjestelmä. Kubernetes on kehitetty tukemaan järjestelmien kehitystä ja hallinnointia [17], ja sitä kehitetään avoimen lähdekoodin projektina GitHub-palvelussa. Keskeisenä osana Kuberneteksen toimintaa ovat kontit. Kontit kapseloivat (engl. encapsulate) sovellusympäristön abstrahoimalla laite- ja käyttöjärjestelmätason palvelut sovelluksen kehittäjältä ja käyttöönottoinfrastruktuurilta [17]. Kubernetes hallinnoi, orkestroi ja ajastaa ajettavia kontteja [13].

Kubernetes-konttien käyttö ei ole sidottu mihinkään tiettyyn pilvialustaan. Useimmat pilvialustat tarjoavat Kubernetes-klustereiden hallintapalveluita, joilla Kubernetes-klustereita pystytään luomaan ja hallinnoimaan. Tällainen on esimerkiksi luvun 6.2 tutkimuksessa käytettävä Microsoft Azure Kubernetes -palvelu.

Kubernetesta käytetään sen sovellusrajapinnan eli API:n (engl. application programming interface) avulla [16, 17]. Tämä rajapinta mahdollistaa myös kolmansien osapuolien työkalujen, kuten tässä työssä esiteltävän Pulumin, kehittämisen. Kuberneteksen rajapinta mahdollistaa resurssien luomisen, muokkaamisen ja niiden poistamisen. Kubernetes-resurssit kuvaillaan tyypillisesti YAML-tiedostoina, joiden pohjalta Kubernetes-komentoriviliitäntä `kubectl` suorittaa tarvittavia rajapintakutsuja.

Usein resursseista on tarve luoda useampia versioita siten, että vain tietyt parametrit vaihtuvat. Yksi tätä tarkoitusta varten kehitetyistä työkaluista on Kuberneteksen paketinhallintajärjestelmä Helm, jota käsitellään seuraavaksi.

Helm sai alkunsa vuonna 2015 [18] ja se on kehitetty Kuberneteksen paketinhallintajärjestelmäksi [13]. Kuberneteksen tavoin Helmiä kehitetään avoimen lähdekoodin projektina. Helmin tavoite on helpottaa Kubernetes-resurssien luomista ja hallinnointia [13].

Helm-pakettien määrittelyssä käytettäviä tiedostoja nimitetään Helm-kaavioiksi (engl. Helm charts) [13]. Nämä kaaviotiedostot ovat YAML-kielellä kirjoitettuja. Helm-kaaviot sisältävät kuvaukset käytettävistä Kubernetes-resursseista. Nämä kaaviot tukevat malleja (engl. template), joiden avulla sovelluksen vaatimat Kubernetes-resurssit pystytään parametrisoimaan [13]. Helm-paketinhallintajärjestelmää käytetään `helm`-komentoriviliitännän kautta.

Kubernetes ja Helm mahdollistavat pilvialustariippumattomien konttipohjaisten sovellusten kehityksen ja julkaisun.

Seuraavassa luvussa käydään läpi infrastruktuuri koodina -lähestymistavan taustoja sekä siihen liittyviä käytäntöjä ja periaatteita.

3. INFRASTRUKTUURI KOODINA

Tässä luvussa määritellään, mitä yleisesti tarkoitetaan, kun puhutaan infrastruktuurista koodina (engl. Infrastructure as Code, IaC). Tämän lisäksi luvussa tuodaan esiin, mitä hyötyjä infrastruktuuri koodina -lähestymistavalla tavoitellaan sekä millaisia haasteita tähän voi liittyä.

3.1 Esittely

Infrastruktuuri koodina -lähestymistavalle ei ole yhtä ainoaa määritelmää. Kief Morris määrittelee lähestymistavan kirjassaan [41] seuraavanlaisesti: infrastruktuuri koodina on lähestymistapa infrastruktuurin automaatioon ohjelmistokehityksen käytäntöjen mukaisesti. Infrastruktuuri koodina -lähestymistavalla korostetaan johdonmukaisia ja toistettavia rutiineja järjestelmien ja niiden konfiguraatioiden päivittämiseen [41].

Artač et al. määritelmässään [9] niin ikään tuo esille ohjelmistokehityksen käytäntöjen johdonmukaisuuden tärkeyden, sillä infrastruktuuri koodina -lähestymistavassa on oleellista infrastruktuurin kuvaus lähdekoodin avulla siten, että komentorivisarjat, automaatioon ja konfiguraatioon liittyvä koodi, mallit, tarvittavat riippuvuudet ja toiminnalliset konfiguraatioparametrit voidaan ilmaista käyttämällä samaa standardikieltä kuin sovellusohjelmoinnissa. Tavoitteena on käyttää uudelleen toimivia ja yleisiä ohjelmistokehityksen käytäntöjä nopeuttamaan kehitysprosessia: infrastruktuuri voidaan versioida, mikä mahdollistaa muun muassa vian etsinnän.

Infrastruktuuri koodina -lähestymistavan keskeisenä periaatteena on määritellä infrastruktuurin vaatimat resurssit tiedostoja hyödyntämällä käyttöliittymien ja manuaalisten syötteiden sijasta [41]. Niistä resursseista, joita päivitetään samaan aikaan, käytetään tässä työssä termiä infrastruktuuripino tai pelkkä pino, jos konteksti on selvä. Resurssit ovat infrastruktuurin osia, kuten esimerkiksi virtuaalikoneet, verkot ja tietokantaklusterit.

Infrastruktuurin määrittelyyn koodina ei ole olemassa yhtä ainoaa tapaa. Tapoja ja työkaluja on monia, mutta keskeisenä osana on kuitenkin se, että määrittely tapahtuu tiedostoina.

Infrastruktuurilähdekoodissa määritellään halutut infrastruktuurin komponentit ja niiden konfiguraatio. Lähdekoodi suoritetaan esimerkiksi käyttämällä jotakin infrastruktuuri koo-

dina -työkalua, ja suorituksen lopputuloksena halutunlainen infrastruktuuri luodaan tai olemassa olevaa infrastruktuuria muokataan haluttuun tilaan. [41]

Infrastruktuurin koostavat resurssit jaetaan usein eri julkaisuympäristöihin käyttötarkoituksen perusteella [41]. Ympäristöjä voivat olla esimerkiksi kehitys-, testi- ja tuotantoympäristöt, joista jokaisessa voidaan käyttää samankaltaisia resursseja. Ympäristöjä voi olla useita ja ympäristöön voi kuulua useita infrastruktuuripinoja [41].

Infrastruktuuri koodina -lähestymistapa on kehitetty vastaamaan infrastruktuurin ylläpitoon, muuttamiseen ja kehittämiseen liittyviin haasteisiin [41]. Lähestymistapa on myös yksi DevOps-kulttuurin mahdollistavista ja tärkeistä käytännöistä. Seuraavassa alaluvussa käsitellään infrastruktuuri koodina -lähestymistavan sekä DevOps-kulttuurin yhteyttä.

3.2 Infrastruktuuri koodina ja DevOps-kulttuuri

Tammikuussa 2015 Microsoft antoi markkinointitutkimusyritykselle tehtävän tutkia, onko infrastruktuuri koodina -lähestymistavasta ja tähän liittyvistä työkaluista hyötyä ohjelmistojen kehityksessä ja niiden julkaisussa. Tutkimuksessa tehtiin verkkopohjainen kysely yhteensä 300 kehittäjälle (engl. developer, dev) ja ylläpitäjälle (engl. operations, ops). Vastaajista 50 prosenttia käytti infrastruktuuri koodina -lähestymistapaa ja 50 prosenttia oli joko ottamassa lähestymistavan käyttöön tai oli kiinnostuneita infrastruktuuri koodina -lähestymistavasta. [26]

Tutkimuksen [26] mukaan lähestymistapa vähentää kitkaa ohjelmistojen julkaisun elinkaaren haastavimmissa vaiheissa, kuten kehitys-, testaus- ja konfigurointivaiheissa, sillä nämä vaiheet liittyvät keskeisesti toisiinsa. Lisäksi se edistää yhteistyötä kehittävien ja operatiivisten tiimien välillä, sillä lähestymistapa edellyttää yhteisten työkalujen, kuten komentorivisarjakielien valintaa, sekä kehitys- että operatiivisten tiimien välillä [25, 26].

Markkinointiyrityksen tutkimuksessa tuodaan esille se, että lähestymistapaa jo nyt hyödyntävät käyttäjät, sekä käyttöä vasta suunnittelevat käyttäjät ovat samaa mieltä lähestymistavan tuomista hyödyistä [26]. Yhtenä keskeisimmistä hyödyistä esiin tuodaan yhteistyö kehittävien ja operatiivisten tiimien välillä [26].

Infrastruktuuri koodina -lähestymistapa mahdollistaa sovellusten ja infrastruktuurin konfiguroinnin nopeammin ja luotettavammin [26, 41] manuaalisiin prosesseihin nähden. Infrastruktuurin määrittäminen koodina mahdollistaa myös sen testaamisen muun sovelluslähdekoodin tavoin [41]. Kief Morris korostaa kirjassaan [41] infrastruktuurin testauksen merkitystä.

3.3 Työkalut

Komentorivisarjoja (engl. scripts) on käytetty jo pitkään infrastruktuurin automatisointiin [41]. Pelkkien komentorivisarjojen käyttäminen infrastruktuurin määrittämiseen johti siihen, että komentorivisarjoissa otettiin kantaa sekä siihen, mitä infrastruktuuriresursseja luodaan, että siihen, miten ne tulisi luoda [41]. Näiden komentorivisarjojen ylläpito ja kehittäminen johtivat yhä monimutkaisempien komentorivisarjojen luomiseen, mikä hankaloitti niiden käyttöä.

Nykyisten infrastruktuuri koodina -työkalujen historia johtaa vuoteen 1993, kun Oslon yliopistossa tutkijana toiminut Mark Burgess kehitti Configuration Enginen (CFEngine) [14]. Configuration Enginen kehityksen taustalla oli se, että Burgess oli kyllästynyt ratkaisemaan ongelmia manuaalisesti ylläpitäessään hänen vastuullaan ollutta infrastruktuuria. Käytössä olleilla Unix-järjestelmillä oli kullakin omat erityispiirteensä, minkä vuoksi komentorivisarjojen kirjoittaminen oli työlästä ja niitä jouduttiin ylläpitämään jokaiselle järjestelmälle erikseen [3].

CFEnginestä teki edistyksellisen sen tapa hyödyntää halutun tilan (engl. desired state) mallia [15]. Samat komennot pystyttiin ajamaan useampaan kertaan siten, että vältyttiin virheiltä ja lopputuloksena järjestelmä oli halutussa tilassa. CFEngine hoiti taustalla tarvittavat tarkistukset muun muassa resurssien luomiselle [15].

CFEnginen jälkeen on kehitetty useita työkaluja, joilla on pyritty tekemään infrastruktuurin ja konfiguraation hallinnasta lähestyttävämpää ohjelmistojen kehittäjille. Tällaisia työkaluja ovat esimerkiksi vuonna 2005 julkaistu Puppet [84] ja vuonna 2009 julkaistu Chef [6].

Yleisesti infrastruktuurin hallintaan käytetyt työkalut voidaan jakaa pinonhallintatyökaluihin (engl. stack management) ja konfiguraationhallintatyökaluihin (engl. configuration management) [41]. Pinojen hallintaan käytetyt työkalut pohjautuvat tyypillisesti halutun tilan malliin, jossa työkalulle kuvataan tila, jossa halutaan infrastruktuurin olevan ja työkalun tehtävänä on varmistaa, että infrastruktuurin tila vastaa kuvausta. Pinohallintatyökaluja ovat esimerkiksi Terraform ja tässä työssä tutkittava Pulumi. Pinonhallintatyökaluilla hallinnoidaan infrastruktuuriresursseista koostettuja infrastruktuuripinoja.

Konfiguraationhallintaan tarkoitettuja työkaluja ovat vuorostaan esimerkiksi aiemmin mainitut Puppet ja Chef. Konfiguraationhallinnantyökaluja käytetään pystytettyjen palvelimien ja sovellusten konfigurointiin. Konfiguraationhallintaan käytettävät työkalut eivät ota kantaa siihen, kuinka infrastruktuuri pystytetään. Koska konfiguraationhallinnan työkalut eivät ota kantaa infrastruktuurin pystyttämiseen, voidaan niitä käyttää yhdessä pinohallintatyökalujen kanssa.

3.4 Hyödyt ja haasteet

Yhtenä haasteena ohjelmistoprojekteissa on se, että ohjelmistokehittäjien käyttämät kehitysympäristöt voivat poiketa testi- ja tuotantoympäristöistä [26]. Yhtenäistämällä ympäristöjen määritelmät, lisäämällä ne versionhallintaan muun sovelluslähdekoodin lisäksi, sekä automatisoimalla infrastruktuurin julkaisu pystytään parantamaan infrastruktuurin johdonmukaisuutta, vähentämään virheitä, säästämään aikaa sekä parantamaan jäljitettävyyttä (engl. auditability) [26].

Infrastruktuurin automatisointi mahdollistaa nopean ja jatkuvan toimituksen [41]. Keskeistä on, että infrastruktuuria pystytään päivittämään ja kehittämään jatkuvasti. Infrastruktuuri koodina -lähestymistapa ei edellytä pilvialustojen käyttöä, mutta se vaatii kuitenkin nykyajan pilvialustojen mukaisen ajattelumallin, jonka keskiössä on jatkuva muutos [41]. Infrastruktuurin ei tule olla muuttumaton osa ohjelmistoa, jonka muuttamista ja päivittämistä pidetään pelottavana, vaan infrastruktuuria tulee kehittää, ylläpitää ja päivittää samoja ohjelmistokehityksen käytäntöjä noudattamalla kuin muitakin ohjelmistoja.

Infrastruktuurin automatisointi myös helpottaa infrastruktuurin automaattista testaamista [41]. Näitä automaattisia testejä voidaan ajaa muiden automaattisten testien lisäksi, jotta ohjelmistosta voidaan testata sen koko ympäristö. Kief Morris nostaa kirjassaan esille sen, että automatisoinnin tulisi olla osa kehitysprosessia heti alusta alkaen [41].

Infrastruktuurin määritelmän ollessa tiedostoina voidaan sen kanssa hyödyntää samoja versionhallinta- ja ohjelmistokehityskäytäntöjä kuin muun sovelluslähdekoodin kanssa [9]. Infrastruktuurilähdekoodin ylläpito ja kehitys ovat tärkeässä osassa lähestymistapaa. Infrastruktuurinlähdekoodin ylläpidossa tulee huomioida se, että muutokset voivat pahimmillaan poistaa käytössä olevia palveluita, joten on tärkeää, että infrastruktuuriin tehtävät muutokset ovat jäljitettävissä.

4. C#-OHJELMOINTIKIELI

Tässä luvussa käydään läpi Microsoftin kehittämää C#-ohjelmointikieltä [80] siltä osin kuin tässä työssä tehtävässä tutkimuksessa on tarpeellista. C#-ohjelmointikielen kaikkia ominaisuuksia ei siis tämän työn puitteissa käydä läpi.

4.1 Esittely

C#-ohjelmointikieli on Microsoftin kehittämä ja julkaisema vahvasti tyypitetty ja olio-ohjelmointia tukeva moniparadigmainen ohjelmointikieli [80]. C# hyödyntää automaattista roskienkeruuta (engl. garbage collection) muistinhallintaan, mikä tarkoittaa sitä, että ohjelmoijan ei pääsääntöisesti tarvitse huolehtia muistin varaamisesta eikä muistin vapauttamisesta.

Kaikki C#:n tietotyypit perivät `object`-tyypistä ja sitä voidaan käyttää paluuarvona metodeissa, jonka halutaan voida palauttavan minkä tahansa kielen tukemista tietotyypeistä. C# tukee myös geneeristä ohjelmointia [82]. Geneerinen ohjelmointi mahdollistaa koodin uudelleen käyttämisen siten, että algoritmi voidaan kirjoittaa kertaalleen ja käännösaikana siitä muodostetaan tarvittavia ilmentymiä.

4.2 Tyypitys

C# hyödyntää vahvaa tyypitystä, joka tarkoittaa sitä, että kaikilla vakioilla, muuttujilla ja lausekkeilla (engl. expression), jotka palauttavat jonkin arvon, on jokin tyyppi, joka on viimeistään käännösvaiheessa kääntäjän tiedossa. C# tukee tyyppipäättelyä (engl. type inference) paikallisten muuttujien määrittelyyn `var`-avainsanan avulla. Tyyppipäättelyllä tarkoitetaan kääntäjän automaattista tyyppien päättelyä käännösaikana. C#-kielessä tämä automaattinen päättely tapahtuu muuttujan alustajan lausekkeen paluuarvon perusteella [80]. Esimerkiksi kääntäjä tulkitsee ohjelmassa 4.1 esitetyn `merkkijono`-muuttujan tyypiksi `string` ja vastaavasti `numerotaulukko`-muuttujan tyypiksi `int[]`.

```
1 var merkkijono = "Merkkijonoesimerkki";
2 var numerotaulukko = new[] { 1, 2, 3 };
```

Ohjelma 4.1. *C#-kielen tyyppipäättely*

4.3 Olio-ohjelmointi

Olio-ohjelmoinnin periaatteiden mukaisesti C# tukee perintää, abstrakteja kantaluokkia ja rajapintoja (engl. interface). Ohjelmassa 4.2 on esitetty kaksi luokkaa `Henkilo`, joka sisältää `Nimi`-nimisen merkkijonotyyppisen ominaisuuden, joka on mahdollista asettaa vain olion rakentajassa, sekä `Henkilo`-luokasta perivä `Opiskelija`-luokka. C#-kielessä olion rakentajalla on aina sama nimi kuin luokalla, johon se kuuluu. `Opiskelija`-luokka perii `Henkilo`-luokasta ja täten sisältää sekä `Nimi` että `Opiskelijanumero` ominaisuudet, joista molemmat asetetaan `Opiskelija`-olion rakentajassa. On huomattava, että kantaluokan `Henkilo`-rakentajaa kutsutaan käyttämällä `base`-avainsanaa, jolla viitataan perittävän luokan rakentajaan.

```
1  public class Henkilo
2  {
3      public string Nimi { get; }
4
5      public Henkilo(string nimi)
6      {
7          Nimi = nimi;
8      }
9  }
10
11 public class Opiskelija : Henkilo
12 {
13     public string Opiskelijanumero { get; }
14
15     public Opiskelija(string nimi, string opiskelijanumero)
16         : base(nimi)
17     {
18         Opiskelijanumero = opiskelijanumero;
19     }
20 }
```

Ohjelma 4.2. *Olioiden määrittely C#-kielelllä*

Tätä työtä kirjoittaessa C#-kielen uusin versio on versio 9. C#-kielen yhdeksäs versio sisältää joitakin ominaisuuksia, joita hyödynnetään tässä työssä tehtävässä tutkimuksessa [83]. Näitä ovat muun muassa init only -alustajat, jotka sallivat ominaisuuksien asettamisen vain oliota rakennettaessa, sekä tietoluokkatyypit (engl. record), jotka ovat muuttumattomia (engl. immutable) olioita, eli niiden arvoja ei voida päivittää suoraan, vaan ominaisuuksien arvojen muuttaminen vaatii uuden ilmentymän luomisen. Ohjelmassa 4.3 on esitetty ohjelmaa 4.2 vastaavat luokat hyödyntämällä tietoluokkia.

```
1  public record Henkilo(string Nimi);
2
3  public record Opiskelija(string Nimi, string Opiskelijanumero)
4      : Henkilo(Nimi);
```

Ohjelma 4.3. *Paikkasidonnaisten tietoluokkatyyppien määrittely*

Ohjelmassa 4.3 esitetty `Opiskelija`-tietoluokka perii `Henkilo`-tietoluokasta. Käytettäessä paikkasidonnaista tietoluokan (engl. positional record) määritelmää kantatietoluokan rakentajaa kutsutaan käyttämällä sen nimeä `base`-avainsanan sijasta.

On huomionarvioista, että tietoluokkien samanarvoisuuden vertailu eroaa normaalien luokkien samanarvoisuuden vertailusta. Tietoluokat käyttävät arvopohjaista samanarvoisuuden vertailua (engl. value-based equality) [80]. Arvopohjainen samanarvoisuus tarkoittaa sitä, että kahta samantyyppistä muuttujaa pidetään samoina, jos niiden sisältämät arvot ovat samat. Oletusarvoisesti luokat käyttävät viitepohjaista samanarvoisuutta (engl. reference-based equality) ja ainoastaan samaan olioon viittaavia muuttujia pidetään samanarvoisina. Tietoluokille C#-kääntäjä luo automaattisesti ominaisuuksien arvoja vertailevan `Equals`-metodin.

Muuttumattomat ja arvopohjaista samanarvoisuutta hyödyntävät tietoluokat soveltuvat hyvin arvo-olioiden [27] määrittelyyn. Arvo-oliot ovat sellaisia olioita, joilla ei ole yksilöivää tunnistetta, vaan niitä pidetään samanarvoisina, jos ne sisältävät samat arvot.

4.4 Tyyppimuunnokset

C#-kieli tukee sekä implisiittisiä että eksplisiittisiä tyyppimuunnoksia (engl. type conversions) [81]. Implisiittiset tyyppimuunnokset tapahtuvat automaattisesti. Esimerkiksi `int`-tyypin muuttuja voidaan implisiittisesti muuttaa `long`-tyypin muuttujaksi, sillä 64-bittinen `long`-tyyppi pystyy pitämään 32-bittisen `int`-tyypin arvon sisällään ilman tiedon menetystä. Samoin perittyä luokkaa voidaan aina käyttää implisiittisesti kantaluokan tyypin sisältävän muuttujan kautta.

Eksplisiittiset tyyppimuunnokset ovat sellaisia, joissa on riskinä kadottaa tietoa, esimerkiksi muutettaessa desimaaliarvoja sisältävän `decimal`-tyypin muuttuja kokonaislukutyyppiin `int`. Tämän vuoksi eksplisiittiset tyyppimuunnokset tapahtuvat aina eksplisiittisellä muunnosoperaatiolla (engl. casting).

C#-kieli tukee käyttäjän määrittämiä tyyppimuunnoksia. Ohjelmassa 4.4 on esitetty sähköpostiosoitetta mallintava `Email`-tietoluokka, jolle on määritelty sekä implisiittinen muunnos merkkijonoksi, eli `string`-tyyppiin. Tämän lisäksi tietoluokalle on määritelty eksplisiittinen muunnosoperaattori merkkijonotyypistä `Email`-tietoluokan ilmentymäksi. Eksplisiittisessä muunnosoperaattorissa tarkistetaan, että annettu merkkijono sisältää ainakin yhden `@`-merkin.

Ohjelman 4.4 rivillä 11 käytetään implisiittistä tyyppimuunnosta `Email`-tyypistä `string`-tyyppiin. Tämä implisiittinen muunnos onnistuu ilman ajonaikaisia virheitä. Vastaavasti rivillä 13 käytetään eksplisiittistä tyyppimuunnosta, jossa merkkijono yritetään muuttaa muunnosoperaattorilla `Email`-tyypiksi. Tämä muunnos epäonnistuu, sillä annettu merkkijono ei sisällä eksplisiittisen muunnosoperaattorin odottamaa `@`-merkkiä.

```
public record Email(string EmailAddress)
{
    public static implicit operator string(Email email) => email.EmailAddress;
    public static explicit operator Email(string address)
    {
        if (!address.Contains('@')) { throw new ArgumentException($"expected '{address}'
            to contain '@'."); }
        return new Email(address);
    }
}
// Onnistuu, käytetään yllä määriteltyä implisiittistä muunnosta
string email = new Email("olli@opiskelija.com");
// Aiheuttaa virheen, koska @-merkki puuttuu
Email email = (Email)"olli.opiskelija.com";
```

Ohjelma 4.4. *Tyyppimuunnosten määrittely*

4.5 .NET-ajoympäristö

C#-ohjelmat käännetään välikieleksi (engl. intermediate language, IL), joka suoritetaan virtuaalisessa .NET-ajoympäristössä, ja josta puhutaan myös termillä Common Language Runtime eli CLR [80]. CLR käyttää ajonaikaista kääntämistä (engl. Just-In-Time compilation, JIT), jossa välikieli käännetään alustakohtaiseksi konekieleksi [80].

C# ei ole ainoa kieli, jota .NET-ajoympäristö tukee. Muita tuettuja ohjelmointikieliä ovat esimerkiksi Visual Basic ja F#. Tämä tarkoittaa sitä, että C#-kielellä tehtyjä kirjastoja voidaan hyödyntää esimerkiksi F#-kielellä ja päinvastoin.

Luvussa 6.2 hyödynnetään C#-kieltä ja sen ominaisuuksia infrastruktuurilähdekoodin määrittämisessä käytännössä. Seuraavassa luvussa esitellään työn tutkimuksen kohteena oleva Pulumi-alusta.

5. PULUMI

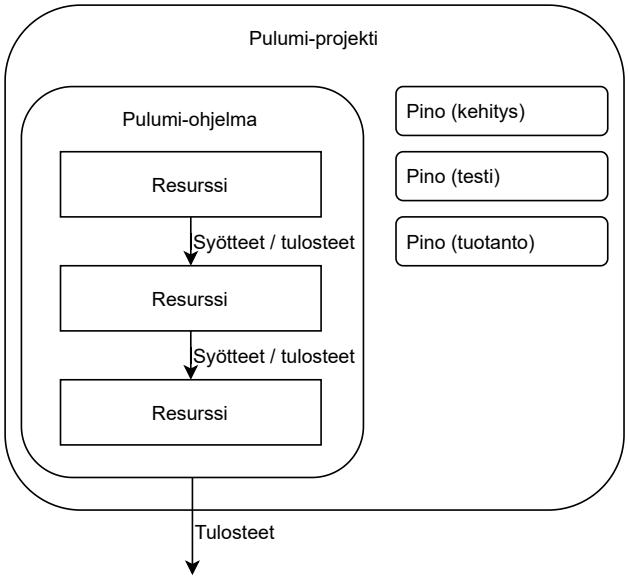
Tässä luvussa esitellään työn arvioinnin ja tutkimuksen kohteena oleva Pulumi infrastruktuuri koodina -alusta. Pulumi on moderni infrastruktuurin hallintaan ja konfigurointiin kehitetty alusta [52]. Pulumi-alustaa kehitetään avoimen lähdekoodin projektina GitHub-palvelussa [51].

Pulumi-alusta koostuu seuraavista osista [8]: komentoriviliitäntä (engl. command-line interface, CLI), ajoympäristö (engl. runtime) ja ohjelmointikielikohtaiset ohjelmistokehityspaketit. Nämä ohjelmistokehityspaketit mahdollistavat eri pilvipalveluiden käytön. Tätä työtä kirjoittaessa Pulumi tukee muun muassa seuraavia ohjelmointikieliä [36]: JavaScript, TypeScript, Python, Go sekä tässä työssä käytettävä C#. Pulumin tukemista pilvipalveluista puhuttaessa tarkoitetaan sekä pilvialustoja kuten Microsoft Azure, Amazon Web Services ja Google Cloud Platform, että muita palveluita, kuten Docker ja Kubernetes, joita voidaan hallinnoida Pulumilla. Näiden lisäksi Pulumia kehittävä yritys Pulumi Corporation tarjoaa Pulumi SaaS -palvelua, jota on mahdollista käyttää infrastruktuurin tilanhallintaan.

5.1 Käsitteet

Pulumi-alustaan liittyvät keskeiset käsitteet [8] on esitetty kuvassa 5.1. Pulumi-projekti sisältää ohjelmointikielellä kirjoitetun Pulumi-ohjelman. Pulumi-ohjelmassa kuvataan halutut infrastruktuuriresurssit sekä niiden konfiguraatio. Resurssit saavat parametreinaan syötteitä ja tuottavat tulosteita, joiden arvoihin muissa resursseissa voidaan viitata. Pulumi-ohjelma itsessään voi myös tuottaa tulosteita, joihin voidaan viitata sekä ohjelmallisesti toisista Pulumi-ohjelmista, että komentoriviliitännällä, joka esitellään alaluvussa 5.2. Syötteitä ja tulosteita käydään tarkemmin läpi alaluvussa 5.7.

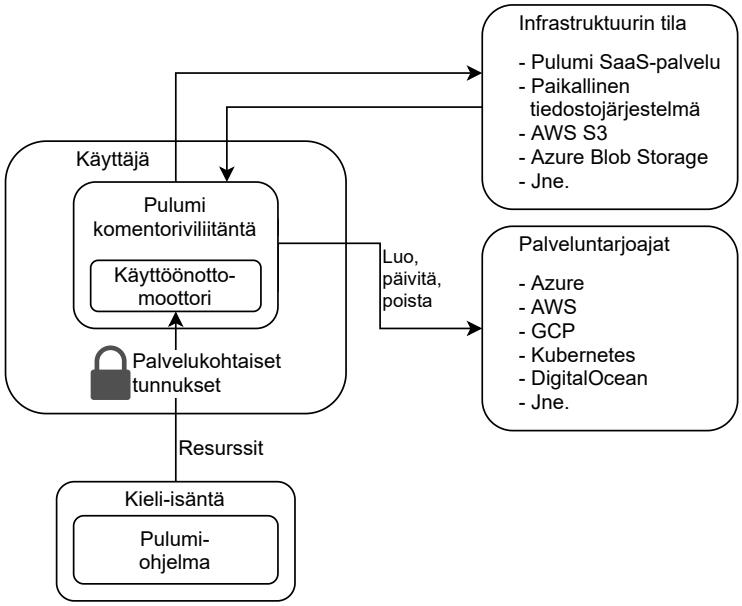
Pulumi-ohjelman ilmentymistä käytetään termiä Pulumi-pino [69]. Pulumi-pinoon kuuluvat käytettävien pilvialustojen Pulumi-resurssit, joita voivat olla esimerkiksi luvussa 3 esitetyt virtuaalikoneet tai verkot tai luvussa 6.2 esiteltävän esimerkkisovelluksen käyttämä Microsoft Azure Kubernetes -klusteri. Pulumi-pinoja voidaan käyttää esimerkiksi eri julkaisuympäristöjen erotteluun [69]. Termiä Pulumi-pino ei pidä sekoittaa luvussa 3 esitettyyn termiin infrastruktuuripino. Infrastruktuuripino on yläkäsite samanaikaisesti päivitettävien resurssien kokoelmalle, joka Pulumia käytettäessä määritellään Pulumi-ohjelmassa. Pulumi-pinot ovat projektikohtaisia Pulumi-ohjelmassa määritellyn infrastruktuuripinon il-



Kuva 5.1. *Pulumin käsitteet [8]*

mentymiä. Pulumi-pinon viimeisin tila tallennetaan komentoriviliitännän käyttämään tilanhallintapalveluun [70]. Tilanhallintaa käydään tarkemmin läpi alaluvussa 5.3.

Kuvassa 5.2 on esitetty komentoriviliitännän, infrastruktuurin tilan ja infrastruktuuripalveluiden keskinäiset suhteet. Komentoriviliitäntä kommunikoi sekä käytettävän tilanhallintapalvelun että käytettävien palveluntarjoajien kanssa. Komentoriviliitäntä ei tallenna tai ylläpidä palveluiden käyttämiä tunnuksia, vaan se käyttää samoja tunnuksia kuin palveluiden omat työkalut [70]. Palvelukohtaiset tunnukset asetetaan siis käyttäjän ympäristön perusteella. Joissakin tapauksissa tunnukset voidaan myös konfiguroida pinokohtaiseen konfiguraatiotiedostoon, jota käsitellään tarkemmin alaluvussa 5.8.



Kuva 5.2. *Komponenttien keskinäiset suhteet*

Kieli-isäntä (engl. language host) suorittaa jollakin Pulumin tukemista ohjelmointikielistä kirjoitettuja Pulumi-ohjelmia [53]. Esimerkiksi käytettäessä C#-ohjelmointikieltä kieli-isäntä kääntää ajettavan ohjelman automaattisesti, ellei Pulumi-projektitiedostoon ole erikseen määritelty käytettävän valmiiksi käännettyä ohjelmaa. Ohjelman suorituksen aikana kieli-isäntä muodostaa kuvauksen infrastruktuurin halutusta tilasta [53], joka välitetään käyttöönottomoottorille (engl. deployment engine). Käyttöönottomoottori vertailee haluttua tilaa infrastruktuurin tämänhetkiseen tilaan ja päättelee tämän perusteella, mitä operaatioita täytyy tehdä, jotta infrastruktuurin tila vastaa uutta kuvausta [53]. Kaikkia resursseja ei voida päivittää suoraan, vaan päivitysoperaatio vaatii ensiksi resurssin poistamisen. Poistamisen jälkeen samanniminen resurssi luodaan uudestaan päivitetyin arvoin. Käyttöönottomoottori on osa Pulumi-komentoriviliitäntää [53].

Käyttöönottomoottori hyödyntää eri resurssintarjoajia (engl. resource provider), jotka ovat pilvialustakohtaisia. Nämä resurssintarjoajat mahdollistavat eri Pulumin tukemien alustojen käytön, ja niiden tehtävänä on vastata tietyn alustan resurssioperaatioista. Pilvialustakohtaiset resurssintarjoajat koostuvat ohjelmalisäkkeistä (engl. plugin) ja ohjelmointikieliriippuvaisesta kirjastosta [53]. Näitä kirjastoja voidaan käyttää kielten tyypillisesti tukemien paketinhallintajärjestelmien kautta.

5.2 Komentoriviliitäntä

Pulumi-alustan keskeisenä osana on sen komentoriviliitäntä, jota käytetään alustaa hyödyntävän infrastruktuurin hallintaan. Komentoriviliitäntä [54] on Go-ohjelmointikielellä kehitetty alustariippumaton (engl. cross-platform) ohjelma, jonka suorittamiseen käytetään komentorivikehoitetta, kuten Microsoft PowerShell [5] tai GNU Bash [12]. Ennen kuin komentoriviliitäntää voidaan käyttää, täytyy se asentaa. Asentamiseen löytyvät käyttöjärjestelmäkohtaiset ohjeet Pulumin verkkosivuilta [23].

Komentoriviliitännän avulla pystytään muun muassa [54]:

- vaihtamaan Pulumin käyttämää tilanhallintaa [57]
- luomaan Pulumi-projekteja [58]
- luomaan ja poistamaan pinoja [60]
- muokkaamaan Pulumi-konfiguraatioita [55]
- suorittamaan Pulumi-ohjelmia, jotka luovat resursseja [62]
- esikatselemaan muutosten vaikutusta [59]
- poistamaan resursseja [56]

Komentoriviliitännän asentamisen jälkeen täytyy varmistaa, että `pulumi`-komento löytyy käytettävän komentorivikehoitteen ympäristömuuttujasta `PATH`.

5.3 Tila ja tilanhallinta

Jotta Pulumi pystyy hallinnoimaan infrastruktuuria, täytyy sen tallentaa infrastruktuuriin liittyvää metatietoa johonkin tietovarastoon. Tätä infrastruktuurimetatietoa nimitetään infrastruktuurin tilaksi. Jokaisella Pulumi-pinolla on oma tilansa, jonka perusteella komentoriviliitäntä pystyy lukemaan, luomaan, päivittämään ja poistamaan resursseja. [70]

Pulumi-pinojen metatietojen tietovarastona voidaan käyttää paikallista tiedostojärjestelmää [70], mutta tässä tapauksessa pinojen tilat ovat saatavilla vain sillä koneella, jolla komento on suoritettu. Paikallisen tiedostojärjestelmän lisäksi pinojen tilaa voidaan pitää esimerkiksi AWS S3 -sangossa tai Azure Blob Storagessa. Tätä tilanhallintatapaa kutsutaan itsehallinnoitavaksi tilaksi [70].

Itsehallinnoitavan tilan lisäksi Pulumia kehittävä Pulumi Corporation tarjoaa SaaS-palvelua [62], jossa tila tallennetaan Pulumin hallinnoimaan (engl. hosting) palveluun. Tällöin SaaS-palvelu huolehtii tilan lukituksesta ja usean samanaikaisen päivityksen estosta.

Oletusasetuksilla komentoriviliitäntä käyttää tilanhallintaan Pulumi SaaS -palvelua. Pulumin käyttämää tilanhallintaa voidaan vaihtaa `pulumi login <tilapalvelu>` -komennolla [57]. Ajamalla `pulumi login` -komento ilman parametreja kirjaudutaan Pulumi SaaS -palveluun. Tietty tilanhallintapalvelu voidaan myös ottaa käyttöön projektikohtaisesti.

5.4 Projektit

Pulumi-projekti on kansio, joka sisältää Pulumi-ohjelman ja metatietoa, joka kertoo, miten ohjelma suoritetaan [49]. Tyypillisesti komentoriviliitännällä luodaan `Pulumi.yaml`-tiedosto ajamalla `pulumi new <projektin-tyyppi>` -komento.

```
1 name: azure-esimerkki-1
2 runtime: dotnet
3 description: Azure esimerkki
4 backend:
5   url: https://api.pulumi.com
```

Ohjelma 5.1. *Projektitiedosto Pulumi.yaml*

Ohjelmassa 5.1 on esimerkki `Pulumi.yaml`-tiedostosta. Tiedosto sisältää komentoriviliitännän tarvitsemat metatiedot, joiden perusteella Pulumi-ohjelma suoritetaan. `Pulumi.yaml`-tiedosto kertoo muun muassa projektin käyttämän ohjelmointikielen `runtime`-kentässä. Ohjelmassa 5.1 on käytössä `dotnet`-ajoympäristö. Seuraavaksi käydään läpi projektiin kuuluvaa Pulumi-ohjelmaa, jonka suorituksen perusteella luodaan resurssit.

5.5 Ohjelmat

Pulumi-ohjelma koostuu tiedostoista, jotka sisältävät lähdekoodin infrastruktuurin hallintaa varten [49]. Pulumi-ohjelman voi kirjoittaa millä tahansa Pulumin tukemalla ohjelmointikielellä. Pulumi-ohjelmat käyttävät ohjelmointikieli- ja pilvialustakohtaisia kirjastoja, joiden avulla voidaan määritellä haluttuja infrastruktuuriresursseja ja muodostaa näistä infrastruktuuripino. Kieli-isäntä suorittaa Pulumi-ohjelman, ja esimerkiksi C#-kielen oletusasetuksilla Pulumi-ohjelmat käännetään ennen niiden suoritusta.

```
1 using Pulumi;
2
3 return await Deployment.RunAsync<Esimerkki.AzureEsimerkkipino>();
```

Ohjelma 5.2. *C#-kielellä kirjoitettu Pulumi-ohjelma*

Ohjelmassa 5.2 on esitetty Pulumi-ohjelma, joka käyttää liitteen A ohjelmassa A.2 esitettyä pinoluokkaa. Ohjelman suorituksen aikana pinoluokassa määriteltyjen resurssien tiedot välitetään käyttöönottomoottorille, joka suorittaa resurssien luomisen, päivittämisen ja poistamisen [53]. Seuraavaksi käydään tarkemmin läpi resursseja.

5.6 Resurssit

Pulumi-resursseja ovat kaikki Pulumin luomat infrastruktuurin eri osat [65]. Komentoriviliitäntä hallinnoi näitä resursseja ja niiden tilaa, jota ylläpidetään käytössä olevassa tilanhallintapalvelussa.

C#-kielellä resursseja kuvataan ohjelmallisesti kahtena `Resource`-luokan aliluokkana: `CustomResource` ja `ComponentResource` [65]. `CustomResource`-luokan oliot ovat resurssintarjoajien hallinnoimia resursseja [65], kuten esimerkiksi liitteen A ohjelmassa A.2 käytetty resurssiryhmä ja varastotili. `ComponentResource`-luokan oliot ovat muiden resurssien loogisia yhdistelmiä, jotka koostuvat muista resursseista [65]. Tässä työssä `ComponentResource`-luokan olioista käytetään nimitystä komponenttiresurssit. Komponenttiresursseilla on mahdollista muodostaa korkeamman tason abstraktioita ja kapseloida resurssien luomisen yksityiskohtia [65]. Liitteen A ohjelmassa A.1 on esitetty komponenttiresurssi, joka kapseloi staattisen verkkosivun luomisen Azure Blob Storageen.

Jokaisella Pulumin hallinnoimalla resurssilla on sekä looginen nimi, joka annetaan resurssille sen rakentajassa, että fyysinen nimi, joka on resurssin nimi siinä palvelussa, johon se kuuluu [65]. Resurssin loogista nimeä käytetään etuliitteenä (engl. prefix) resurssin fyysiselle nimelle, jos fyysistä nimeä ei ole erikseen määritelty [65]. Lisäksi resurssin loogista nimeä käytetään resurssin yksikäsitteisessä nimessä (engl. Uniform Resource Name, URN). Yksikäsitteisiä nimiä käytetään resurssien tilan seurannassa, ja niiden tulee olla Pulumi-ohjelman sisällä uniikkeja.

Pulumi-ohjelmien juuriresurssina on pinoresurssi, jonka alle kaikki ohjelmassa luodut resurssit kuuluvat [65]. Resursseista voidaan myös muodostaa hierarkioita asettamalla niiden vanhemmaksi jokin toinen resurssi (engl. parent resource) [65]. Hierarkialla ei ole vaikutusta resurssien luomiseen [65], mutta sillä voidaan kuvata niiden keskinäisiä suhteita, esimerkiksi mikä resurssi luo jonkin toisen resurssin. Resurssihierarkioita käytetään usein komponenttiresurssien kanssa. Komponenttiresurssiin kuuluvat resurssit voidaan asettaa komponenttiresurssin lapsiksi.

```
TYPE                                                    NAME
pulumi:pulumi:Stack                                     azure-esimerkki-1-azure-esimerkki-1.dev
├─ azure-esimerkki:resurssi:Verkkosivu                  verkkosivu
│  └─ azure-native:storage:StorageAccountStaticWebsite  verkkosivu-web
│     └─ azure-native:storage:Blob                      index.html
├─ azure-native:resources:ResourceGroup                 resurssiryhma
├─ azure-native:storage:StorageAccount                  varastotili
└─ pulumi:providers:azure-native                        default_0_8_0
```

Kuva 5.3. *Pulumi-ohjelman luomat resurssit*

Kuvassa 5.3 on esitetty ohjelman 5.2 luomat resurssit ja niiden muodostama resurssihierarkia. Kuvasta nähdään esimerkiksi liitteen A ohjelman A.1 verkkosivukomponenttiresurssin muodostama hierarkia.

5.7 Syötteet ja tulosteet

Pulumi-resurssien parametreja nimitetään syötteiksi (engl. input) ja resurssien palauttamia arvoja nimitetään tulosteiksi (engl. output). Tässä työssä käytetään lisäksi termejä syötearvo ja tulostearvo näiden ilmentymien nimittämisessä.

Pulumi-resurssit tyypillisesti ottavat kuvassa 5.1 esitettyjä syötteitä parametreinaan [32]. Nämä syötteet voivat olla sellaisia, jotka saadaan vasta ohjelman ajon aikana selville. Tällainen on esimerkiksi liitteen A ohjelmassa A.2 esitetyn resurssiryhmän nimi (rivi 23), jonka lopullinen arvo selviää vasta ohjelman suorituksen aikana. Muut resurssit voivat kuitenkin viitata resurssiryhmän nimeen, vaikka sen arvo ei olisikaan selvillä kuin vasta suorituksen aikana. Syötteiden mallintamiseen käytetään C#-ohjelmointikielellä geneeristä `Input<T>`-tyyppiä [32]. Liitteen A ohjelmassa A.1 hyödynnetään syötetyyppejä verkkosivukomponenttiresurssin parametrien resurssiryhmän nimen ja verkkosivun sisällön kanssa (rivit 6 ja 7).

Resurssien ja pinojen palauttamista arvoista käytetään nimitystä tuloste. Tulosteet ovat asynkronisia arvoja, joiden lopullinen arvo selviää vasta ohjelman suorituksen aikana [32], kun infrastruktuuriresurssit on päivitetty. Tulosteet edustavat kahta asiaa: tulosteen lopullista tulostearvoa sekä riippuvuutta tulosteen syötteisiin [32]. Tulostearvo voidaan muuttaa syötearvoksi, ja sitä voidaan käyttää sellaisenaan muiden resurssien syötteenä. Vastaavasti syötearvo voidaan muuttaa tulostearvoksi, ja syötearvo voidaan myös palauttaa tulostearvona.

Tulosteiden mallintamiseen C#-kielellä käytetään geneeristä `Output<T>`-tyyppiä [32]. Koska tulosteet ovat asynkronisia, ei niiden tulostearvoja voida käsitellä sellaisenaan ohjelmakoodissa. Tulostearvon käsittelyyn täytyy käyttää tulostetyypin `Apply`-metodia.

```
1               // Palautetaan varaston avain salaisena tulosteena
2               this.VarastonAvain = Output
3                   // Käytetään Output.Tuple:a yhdistämään resurssiryhmän ja varastotilin nimitulosteet
4                   .Tuple(resurssiryhma.Name, varastotili.Name)
5                   // Haetaan varastotilin avain nimitulosteita hyödyntäen ja merkitään se salaiseksi
6                   .Apply(names => Output.CreateSecret(HaeVarastotilinAvain(names)));
```

Ohjelma 5.3. *Tulostearvojen käsittely Pulumilla*

Ohjelmassa 5.3 on ote liitteen A ohjelman A.2 pinoluokan tulostearvojen käsittelystä. Ohjelman rivillä 4 käytetään `Output.Tuple`-metodia yhdistämään kaksi eri tulostetta, minkä jälkeen rivin 6 Apply-metodin parametrina saadaan yhdistetty tulostearvo. Yhdistettyä tulostearvoa käytetään hakemaan varastotilin avain ohjelman suorituksen aikana, kun resurssit on luotu ja niiden tulosteet ovat saatavilla. Resurssien lisäksi myös Pulumi-pinot voivat palauttaa tulosteita. Pinoluokkien tulosteet merkitään C#-kielellä `Output<T>`-tyypin lisäksi `Output`-attribuutilla, jota Pulumi käyttää pinoluokan tulosteiden keräämiseen. Liitteen A ohjelmassa A.2 käytetään pinoluokan tulosteita palauttamaan muun muassa verkkosivun osoite.

```
Current stack outputs (3):
    OUTPUT              VALUE
    ResurssiryhmanNimi  resurssiryhma8cf9bd77
    VarastonAvain       [secret]
    VerkkosivunOsoite   https://varastotilid538d73a.z16.web.core.windows.net/
```

Kuva 5.4. *Pinon tuottamat tulosteet*

Kuvassa 5.4 on esitetty ote komentoriviliitännän tulostamista pinokohtaisista tiedoista, joista nähdään ohjelmassa 5.2 luodun pinoluokan tulosteet. Seuraavaksi käydään läpi Pulumi-pinoja tarkemmin.

5.8 Pinot

Yksi Pulumi-pino on yhden Pulumi-ohjelman ilmentymä. Pinot ovat toisistaan irrallisia ja itsenäisesti konfiguroitavia [69]. Pinoja voidaan käyttää esimerkiksi eri julkaisuympäristöjen erottelussa. Pulumi-ohjelman suorituksen aikana luodut resurssit kuuluvat siis siihen pinoon, joka suorituksen aikana on asetettu aktiiviseksi.

Keskeisenä osana Pulumi-pinoja on pinojen konfigurointi [20]. Konfiguraation avulla tuetaan pinojen uudelleenkäytettävyyttä [41]. Pulumi-pinon konfigurointi tapahtuu `pulumi config` -komennolla. Esimerkiksi AWS-alue voidaan asettaa ajamalla `pulumi config set aws:region <alue>`. Ohjelmassa 5.4 on esitetty pinon konfiguraatiotiedosto, jonka `pulumi config` -komento luo, jos pinolla ei ole sellaista jo olemassa.

```
config:
  aws:region: eu-north-1
  tunniste:tunnus: esimerkki-tunnus
  tunniste:salasana:
    secure: AAABANOMqawyvcyEj7vOO/AXUVdeMD0eZHzZJLo9VsG19ITEiw==
```

Ohjelma 5.4. *Pulumi-pinon konfiguraatiotiedosto*

Ohjelmassa 5.4 esiteltävässä pinon konfiguraatiotiedostossa on määritelty kolme konfiguraatioarvoa. Konfiguraatiotiedostossa konfiguraatioarvojen avaimet ovat muotoa `nimiavaruus:avaimen-nimi`, jossa nimiavaruus on esimerkiksi pilvialustan tunniste tai projektin nimi. Rivillä 2 on määritelty käytettävä AWS-alue. Rivillä 3 on määritelty `tunniste`-nimiavaruuteen `tunnus`-niminen avain, jonka arvona on `esimerkki-tunnus`. Rivillä 4 on määritelty `tunniste`-nimiavaruuteen `salasana`-niminen salattu arvo, joka on tallennettu salatussa muodossa konfiguraatiotiedostoon. Koska salaisuudet ovat salattuina konfiguraatiotiedostoissa, voidaan ne tallentaa versionhallintaan turvallisesti.

Konfiguraatiotiedostoissa ja tulosteissa on mahdollista määritellä salaisuuksia, jotka salataan käyttäjän valitsemalla salausmekanismilla [66]. Salaisuuksia voidaan käyttää esimerkiksi palvelukohtaisten salasanojen tai muiden tietojen salaukseen. Salausmekanismia käytetään konfiguraation sekä pinojen tilanhallintaan liittyvien tiedostojen salaukseen. Pulumi-ohjelmassa voidaan tulostearvoja merkitä salaisiksi `Output.CreateSecret`-metodilla. Mikäli tuloste on merkitty salaiseksi, merkitään myös kaikki siihen viittaavat arvot salaisiksi.

Pinon konfiguraatioon voidaan myös viitata ohjelmallisesti Pulumi-ohjelmassa. C#-kielellä konfiguraatioon viittaaminen tapahtuu `Pulumi.Config`-luokkaa käyttäen.

```
var config = new Pulumi.Config("tunniste");
var tunnus = config.Get("tunnus") ?? "vakioarvo";
var salasana = config.RequireSecret("salasana");
```

Ohjelma 5.5. *Pulumi.Config-luokan käyttö C#-kielellä*

Ohjelmassa 5.5 luodaan uusi `Config`-luokan ilmentymä, jonka rakentajassa annetaan konfiguraation nimiavaruus, joka viittaa konfiguraatiotiedostossa asetettuun nimiavaruuden arvoon. Jos rakentajalle ei anneta parametria, Pulumi käyttää projektin nimeä nimiavaruutena. `Config`-luokkaa käyttäen voidaan konfiguraatiotiedostosta hakea arvoja `Get`-alkuisilla metodeilla. Pakolliset konfiguraatioarvot voidaan hakea `Require`-alkuisia metodeja. Salaisuuksia haettaessa käytetään `Secret`-päätteisiä metodeja, jotka palauttavat ne `Output<T>`-tyyppinä ja merkitsevät ne salaisiksi [66].

5.9 Lisenssi

Pulumi-komentoriviliitäntä ja resurssikohtaiset lähdekoodit on lisensioitu Apache License 2.0 -lisenssillä [63]. Apache License 2.0 -lisenssi on salliva avoimen lähdekoodin lisenssi

[68]. Lisenssi sallii muun muassa lähdekoodin kaupallisen käytön, muuttamisen, jakamisen ja yksityisen käyttämisen. Lisenssi ei aseta rajoitteita alustan käyttämiselle esimerkiksi yrityksen omissa kaupallisissa tuotteissa.

5.10 Automatisointirajapinta

Pulumin automatisointirajapinta (engl. Pulumi Automation API) on ohjelmistokehityspaketti, joka mahdollistaa komentoriviliitännän käytön Pulumin tukemien ohjelmointikielten kautta [75]. Automatisointirajapinnan C#-kielen versio on tätä työtä kirjoittaessa vielä esikatselussa, eli sitä ei ole vielä virallisesti julkaistu.

Automatisointirajapinta mahdollistaa Pulumi-projektien, -ohjelmien ja -pinojen hallinnoimisen ohjelmallisesti Pulumin tukemia ohjelmointikieliä hyödyntäen. Automatisointirajapintaa voidaan käyttää sekä olemassa olevien Pulumi-projektien ja -ohjelmien kanssa että dynaamisesti luotujen Pulumi-projektien ja -ohjelmien kanssa. Automatisointirajapinta on yksi merkittävä tekijä, joka erottaa Pulumin sen kilpailijoista mahdollistamalla Pulumin käytön esimerkiksi erilaisten julkaisukäytäntöjen hallintaan [75]. Automatisointirajapintaa hyödyntäen voidaan rakentaa abstraktioita infrastruktuurin julkaisu- ja hallintaprosesseihin.

Liitteen B ohjelmassa B.1 on esitetty automatisointirajapintaa hyödyntävä ohjelmaesimerkki. Ohjelmaesimerkissä esitetty C#-ohjelma hyödyntää aikaisemmin liitteen A ohjelmassa A.2 esiteltyä pinoluokkaa. Liitteen B ohjelmaesimerkkiä B.1 voidaan käyttää Pulumi-pinon hallintaan komentoriviliitännän sijasta.

5.11 Testaus

Lähtökohtaisesti Pulumi tukee kolmen tyyppisiä testejä: yksikkötestejä, resurssiominaisuustestejä ja integraatiotestejä [74]. Yksikkötestit testaavat yksittäisiä pinoja. Resurssiominaisuustestit testaavat infrastruktuuriresurssien syötteitä ja tulosteita. Integraatiotestit testaavat järjestelmän eri osien yhteistoimintaa. Eri tyyppisiä testejä voidaan hyödyntää erilaisissa tilanteissa.

Pulumi-infrastruktuurin yksikkötestit suoritetaan simuloidussa ympäristössä, jossa käyttöönottomoottori on korvattu mock-oliolla [79]. Mock-oliota käytetään eristämään testauksessa käytettävä palvelu tai resurssi sen todellisesta vastineesta [76]. Käyttöönottomoottorin mock-olion palauttamia arvoja on mahdollista käsitellä yksikkötesteissä. Yksikkötestejä ajettaessa ei infrastruktuuriresursseja oikeasti luoda, ja niiden suoritus on tyypillisesti nopeaa. Pulumi-infrastruktuurin yksikkötestit soveltuvat etenkin resurssien syötteiden oikeellisuuden tarkasteluun [74].

Resurssiominaisuustestit ovat testejä, jotka ajetaan Pulumi-ohjelman suorituksen aika-

na ennen resurssien pystyttämistä ja pystyttämisen jälkeen [74]. Resurssiominaisuustesteissä käytössä ovat infrastruktuuriresurssien syötteet ja niiden palauttamat tulosteet. Resurssiominaisuustesteillä voidaan muun muassa varmistaa, että käytössä olevat resurssit noudattavat tiettyjä käytäntöjä [50].

Integraatiotestejä ajettaessa Pulumi-ohjelmassa määritellyt resurssit luodaan käytössä oleville pilvialustoille [33]. Infrastruktuurin integraatiotestien suorittaminen vie tyypillisesti huomattavasti enemmän aikaa kuin yksikkötestien suoritus [41]. Integraatiotesteillä on mahdollista saada varmuutta Pulumi-infrastruktuurin koko elinkaaren toiminnasta. Koska integraatiotesteissä luodaan varsinaiset infrastruktuuriresurssit, voidaan niiden suorituksen aikana myös suorittaa sovellukseen kohdistuvia testejä.

Yksikkötestejä on mahdollista kirjoittaa kaikilla Pulumin tukemista ohjelmointikielillä [79]. Resurssiominaisuustestejä on mahdollista kirjoittaa ainoastaan JavaScript-, TypeScript- ja Python-kielillä [74]. Integraatiotestaukseen on virallisesti tarkoitukseen kehitetty testiviitekehys vain Go-kielelle, mutta sitä pystytään käyttämään kaikkien Pulumi-ohjelmien kanssa riippumatta siitä, millä kielellä ne on kirjoitettu [33].

5.12 SaaS-palvelu

Pulumi SaaS -palvelu on valinnainen osa alustan käyttöä. Sitä voidaan käyttää vaihtoehtona itsehallinnoitavalle tilalle. Pulumi SaaS -palvelu tarjoaa näkymän organisaatiokohtaisesti palvelussa oleviin Pulumi-projekteihin ja näiden pinoihin. Palvelusta nähdään Pulumi-pinoihin liittyviä tietoja, joita myös komentoriviliitännällä pystytään näyttämään.

Pulumi SaaS -palvelusta on myös saatavilla yrityksille suunnattu versio, joka mahdollistaa palvelun hallinnoimisen yrityksen omassa ympäristössä [67]. Yrityksille suunnattu versio mahdollistaa Pulumin käytön sellaisissa tilanteissa, joissa esimerkiksi viranomaisvaatimukset edellyttävät tietojen säilytystä tietyllä tapaa tai tietyn maan rajojen sisällä.

5.13 Vertailu kilpailijoihin

Tässä alaluvussa vertaillaan Pulumia joihinkin tämän kilpailijoista. Vertailussa keskitytään Pulumin merkittävimpään kilpailijaan Terraformiin, sillä se vastaa toiminnallisuuksiltaan Pulumia. Lisäksi alaluvussa esitellään myös pilvialustakohtaiset AWS CloudFormation ja Azure Resource Manager.

5.13.1 Terraform

Terraform on HashiCorp-nimisen yrityksen kehittämä infrastruktuurin hallintaan käytettävä työkalu ja alusta [73]. Pulumin tavoin Terraform tukee useita pilvialustoja. Terraformin käyttöön on kehitetty täsmäkieli HashiCorp Configuration Language eli HCL [45]. HCL on

deklaratiivinen kieli, joka on kehitetty infrastruktuuriresurssien määrittämistä varten [45].

Pulumin tavoin myös Terraform pohjautuu halutun tilan malliin [35], jossa infrastruktuuriresurssimäärittelyiden pohjalta muodostetaan kuvaus siitä, mitä operaatioita resursseille täytyy tehdä, jotta infrastruktuurin tila vastaa kuvausta. Terraform ylläpitää infrastruktuurin tilaa tilanhallintapalvelussa, joka oletusasetuksilla käyttää paikallista tiedostojärjestelmää.

Ennen infrastruktuuritilan päivitystä Terraform muodostaa toteutussuunnitelman (engl. execution plan), jossa on kuvattu ne operaatiot, jotka tehdään päivityksen aikana [19]. Toteutussuunnitelma voidaan luoda ilman, että infrastruktuurin tilaa päivitetään. Suunnitelma voidaan tarkastaa ennen sen toteutusta. Pulumissa toteutussuunnitelmaa vastaava toiminnallisuus on esikatselukomento `pulumi preview`, jolla voidaan tarkistaa, mitä operaatioita tapahtuu, jos Pulumi-pino päivitetään [59].

Pulumin tavoin Terraform tukee tulosteita, joita Terraform-moduulit voivat palauttaa [44]. Tulosteiden lisäksi Terraform tukee syötearvoja, joita voidaan käyttää konfigurointiin [31]. Terraform-syötteitä on mahdollista antaa tiedostoissa, mutta niissä olevia mahdollisesti salaisia arvoja ei salata, joten syötetiedostoja ei kannata tallentaa sellaisenaan versionhallintaan. Sen sijaan Pulumi-pinon konfiguraatiotiedostoja voidaan tallentaa versionhallintaan, koska niissä salaiset arvot on salattu.

Pulumin kanssa on mahdollista käyttää Terraformin resurssintarjoajia. Tämän lisäksi Pulumia on mahdollista käyttää Terraformin tilatiedostojen kanssa, ja Terraformin hallinnoimaa tilaa voidaan hyödyntää myös Pulumi-pinoissa [30]. Pulumilla hallinnoitavaa infrastruktuuria ei voida käyttää Terraformin kanssa.

Käsitteiltään ja toiminnallisuuksiltaan Pulumi ja Terraform ovat samankaltaisia. Merkittävin eroavaisuus niiden välillä on se, että Pulumia käytetään yleisten ohjelmointikielten kanssa, kun taas Terraformia käytetään sen omalla HCL-kielellä. Ohjelmointikielten käyttö mahdollistaa aikaisemmin esitetyn automatisointirajapinnan kaltaisten laajennuksien käyttämisen Pulumilla, kun taas vastaavaa ei ole mahdollista tehdä Terraformilla. Myöhemmin luvussa 7 esitetään Pulumin ja Terraformin suorituskykyvertailun tulokset sekä vertaillaan niiden kehitysyhteisöjä.

5.13.2 Muut

Useita pilvialustoja tukevien työkalujen lisäksi pilvialustoilla on myös pilvialustakohtaisia infrastruktuuri koodina -lähestymistapaan tarkoitettuja työkaluja. Näiden alustakohtaisten työkalujen käyttö rajoittuu kyseiseen alustaan. Tällaisia ovat esimerkiksi AWS CloudFormation ja Azure Resource Manager.

AWS CloudFormation mahdollistaa AWS-pilvialustan resurssien määrittämisen infrastruktuuri koodina -lähestymistavan mukaisesti YAML- tai JSON-tiedostoina [10]. AWS Cloud-

Formation tukee ainoastaan Amazon Web Services -pilvialustaa. Pulumi tukee AWS CloudFormation -pinojen tulosteita ja niiden arvoihin voidaan viitata Pulumi-ohjelmassa [28]. AWS CloudFormation:illa luotuja resursseja voidaan myös tuoda Pulumin hallinnoimaksi.

Azure Resource Manager eli ARM on Microsoft Azure -pilvialustalle kehitetty teknologia, joka mahdollistaa resurssien määrittämisen JSON-kieleen pohjautuvan täsmäkielen avulla [11]. ARM-tiedostoja nimitetään malleiksi. Pulumi tukee ARM-mallien käyttöä [29]. ARM-malleja voidaan käyttää yhdessä Pulumi-ohjelman kanssa, sillä Pulumi-ohjelmassa voidaan viitata mallin tulostearvoihin. ARM:in luomia resursseja voidaan myös tuoda suoraan Pulumin hallinnoitavaksi joko `pulumi import` -komennolla tai määrittelemällä resurssille import-ominaisuus. ARM-mallit voidaan myös muuttaa Pulumi-lähdekoodiksi käyttämällä tarkoitukseen tehtyä komentorivityökalua `arm2pulumi`[1].

Seuraavassa luvussa esitetään työssä tehtävä tutkimus arviointikriteereineen.

[1] https://www.pulumi.com/arm2pulumi/

6. TUTKIMUS

Tässä luvussa esitetään työssä tehtävä Pulumi-alustan arviointiin liittyvä tutkimus sekä arvioinnissa käytettävät arviointikriteerit. Tutkimuksessa käytetyt arviointikriteerit valittiin Profit Software Oy:n tarpeiden sekä avoimen lähdekoodin ohjelmistojen pitkäjänteistä kehitystä käsittelevän tutkimuksen [38] perusteella. Jatkossa tässä luvussa viitataan Pulumi-alustaan pelkästään alustana.

6.1 Arviointikriteerit

Alustan arviointiin käytettävät kriteerit on jaettu kahteen kategoriaan: alustakriteereihin ja alustan käyttöönoton kriteereihin. Alustakriteereissä keskitytään alustan toimintaan yleisellä tasolla. Alustakriteereillä arvioidaan muun muassa alustan kustannuksia, teknologian kypsyyttä sekä kehitysyhteisön aktiivisuutta. Alustan käyttöönoton kriteereillä arvioidaan alustan käyttöä ohjelmistokehittäjän näkökulmasta. Alustan käyttöönoton kriteereihin kuuluvat muun muassa testattavuus, alustan tukemat koodieditorit, vian etsintä sekä muut kehittämiseen vaikuttavat tekijät. Seuraavaksi esitetään kumpaankin kategoriaan liittyvät kriteerit.

6.1.1 Alustakriteerit

Teknologian kypsyydessä otetaan huomioon alustan ikä ja sen päivitystahti, eli kuinka usein alustaan tulee päivityksiä. Teknologian kypsyyteen vaikuttavat myös kehityksessä käytettävät käytännöt sekä se, kuinka alustaa ylläpidetään. Tässä tarkastellaan alustan GitHub-palvelun tietoja, kuten virheiden, auki olevien vetopyyntöjen (engl. pull request) ja alustan ylläpitäjien määrää [38, 39]. Teknologian kypsyyteen vaikuttaa myös, kuinka alustaa testataan.

Infrastruktuuri koodina -työkalun käytettävyyteen vaikuttavat myös työkalun tukemat pilvialustat ja muut palvelut. Tällä kriteerillä arvioidaan Pulumin tukemien pilvialustojen ja muiden palveluiden määrää. Eri pilvialustojen käyttö voi olla perusteltua siinä tilanteessa, että halutaan hajauttaa infrastruktuuria ongelmien varalta. Jos käytössä on esimerkiksi kaksi eri pilvipalveluntarjoajaa, toiseen kohdistuvat ongelmat eivät välttämättä vaikuta toiseen.

Alustaa kehitetään avoimen lähdekoodin projektina. Kehitysyhteisön aktiivisuus -kriteerin avulla arvioidaan kehitysyhteisön kokoa ja sitä, mitä alustalle tapahtuisi, jos sitä kehittävä Pulumi Corporation -yritys lopettaisi toimintansa. Yhteisön aktiivisuutta mitataan muun muassa Stack Overflow'hun kirjoitettujen kysymysten määrällä sekä GitHub-palvelussa olevien tähtien määrällä [38].

Suorituskykykriteereillä on tarkoitus arvioida alustan toimintaa sen suorituskyvyn perusteella. Suorituskyvyllä on vaikutusta esimerkiksi alustan käyttöön jatkuvan integraation ja toimituksen palveluissa. Suorituskykyä arvioidaan suorituskykytesteillä.

Kustannuksissa ja yritysprofiilissa arvioidaan alustaa sitä kehittävän yrityksen ja alustaan liittyvien kustannusten kautta. Alustan kustannukset vaikuttavat siihen, kuinka helposti alusta voidaan ottaa käyttöön kokeilutarkoituksessa.

6.1.2 Alustan käyttöönoton kriteerit

Automatisointi on keskeisenä osana infrastruktuuri koodina -lähestymistapaa. Automatisointikriteerissä arvioidaan alustan tukemia jatkuvan integraation ja toimituksen palveluita. Jatkuvan integraation ja toimituksen käyttö on myös tärkeä osa DevOps-kulttuuria [25]. Jatkuvaa integraatiota käytetään esimerkiksi automaattisten testien ajamiseen vetopyynnöille [25]. Jatkuva toimitus vie jatkuvan integraation julkaisuun asti [25].

Alustan tukemia kehitysympäristöjä arvioidaan. Esimerkiksi integroidut kehitysympäristöt (engl. integrated development environment) tarjoavat käyttäjilleen toiminnallisuuksia, kuten automaattinen tekstinsyöttö, dokumentaation integraatio koodieditoriin ja navigointi koodin eri osa-alueiden välillä. Integroitujen kehitysympäristöjen tukema automaattinen tekstinsyöttö ja siirtyminen koodin eri osa-alueiden välillä helpottaa omaksumista [87].

Konfiguraatiokriteerillä arvioidaan alustan konfiguroitavuutta, eli sitä, kuinka helposti voidaan tehdä esimerkiksi ympäristökohtaisia konfigurointeja. Ympäristökohtainen konfigurointi tukee uudelleenkäytettävyyttä, sillä sen avulla vähennetään toistoa eri ympäristöjen välillä [41]. Hyödyntämällä konfigurointia voidaan ympäristöjen välillä poikkeavat arvot asettaa konfiguroitaviksi. Ohjelmistoympäristöjä voidaan hyödyntää eri julkaisuympäristöjen, kuten testi- tai tuotantoympäristön, hallinnassa.

Laajennettavuudella arvioidaan alustan laajennettavuutta, eli kuinka hyvin alustaa voidaan muokata tai kuinka hyvin alustalle voidaan luoda omia laajennuksia tai ohjelmistokehityspaketteja. Laajennettavuus vaikuttaa myös avoimen lähdekoodin kehitysyhteisön muodostumiseen ja tukee pitkäjänteistä kehitystoimintaa [43].

Omaksumisella tarkoitetaan sitä, kuinka helposti ja nopeasti alusta on mahdollista ottaa käyttöön. Omaksumiseen vaikuttavat muun muassa alustan dokumentaatio ja sen tukemat ohjelmointikielet. Dokumentaation laatua arvioidaan, sillä hyvä dokumentaatio tukee

alustan parissa toimimista [47].

Testattavuus on tärkeässä osassa infrastruktuuri koodina -lähestymistapaa. Tämän kriteerin avulla arvioidaan alustaa sen perusteella, miten sitä voidaan testata infrastruktuuri koodina -lähestymistavan mukaisesti [41]. Testeillä pystytään luomaan varmuutta järjestelmän toiminnasta sekä helpottamaan muutosten tekemistä [24, 71]. Testit voivat myös toimia osana ohjelmiston dokumentaatiota, sillä testeissä kuvataan ohjelmiston toimintaa ja samalla tuetaan ohjelmiston ylläpidettävyyttä.

Vian etsintä (engl. debugging) on ominaisuus, jota voidaan tyypillisesti käyttää ohjelmointikielten kanssa. Tämän kriteerin avulla arvioidaan vian etsinnän käyttöä alustalla kirjoitettujen ohjelmien kanssa. Vian etsintä tapahtuu tarkoitukseen soveltuvilla työkaluilla, jotka ovat usein osa integroitua kehitysympäristöä [87] ja joilla on mahdollista muun muassa seurata ohjelman toimintaa, pysäyttää ohjelman suoritus esimerkiksi tiettyyn lähdekoodiriviin sekä tarkastelemaan lähdekoodissa olevien muuttujien tilaa. Vian etsintää käytetään tavallisesti ohjelmistossa raportoitujen ongelmien selvittämisessä.

Ylläpidettävyyskriteerillä arvioidaan alustalle kirjoitetun infrastruktuurilähdekoodin ylläpidettävyyttä muun muassa lähdekoodin luettavuuden kannalta. Ohjelmistojen ylläpito on usein se vaihe, jossa havaitaan, että ohjelmiston arkkitehtuuri ja toteutus tekevät sen laajentamisesta ja muuttamisesta vaikeampaa [64]. Esimerkiksi testien puute vaikeuttaa ohjelmiston ylläpidettävyyttä, sillä muutokset täytyy testata manuaalisesti, mikä on hidasta, jos muutoksia on paljon.

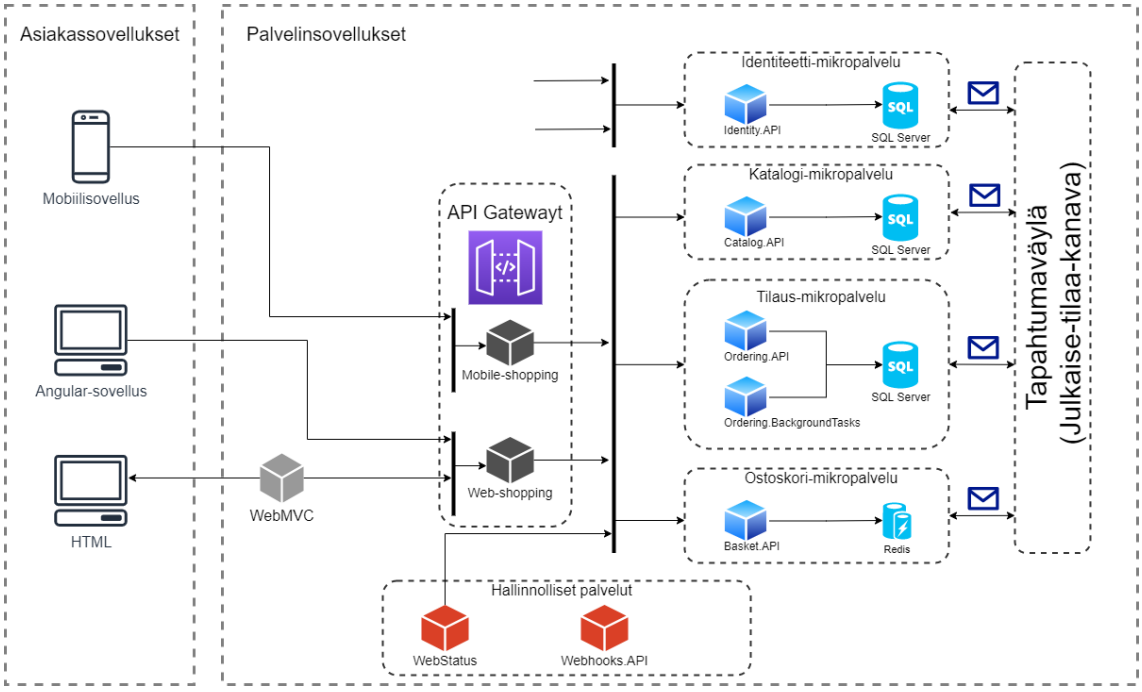
6.2 Esimerkkisovellus

Tässä alaluvussa esitetään esimerkkisovellus, jota käytetään alustan käyttöönoton arviointiin. Esimerkkisovellukseksi valittiin Microsoftin kehittämä C#-ohjelmointikieltä ja .NET 5 -ohjelmistoviitekehystä hyödyntävä mikropalveluarkkitehtuurin toteuttava referenssisovellus eShopOnContainers, jota kehitetään avoimen lähdekoodin projektina GitHub-palvelussa [1]. Microsoft on myös julkaissut e-kirjoja [77, 78] referenssisovelluksen rinnalle.

Referenssisovellus valittiin tähän diplomityöhön sen vuoksi, että kyseessä on Microsoftin kehittämä ja ylläpitämä referenssisovellus. Referenssisovelluksen käyttämiä teknologioita on myös Profit Software Oy:ssä käytössä. Referenssisovellus pohjautuu Microsoftin kehittämiin uusimpiin teknologioihin ja yleisesti hyvinä pidettyihin käytäntöihin. Sovellus sisältää myös kattavan ohjeistuksen, jolla sovelluksen voi suorittaa paikallisesti ja Microsoft Azure -pilvialustalla hyödyntämällä Azure Kubernetes -palvelua [22, 34]. Tässä työssä tehtävässä tutkimuksessa käytettiin eShopOnContainers-referenssisovelluksen `dev`-haaran versiota, jossa on lisätty tuki uusimmalle C#-ohjelmointikielen versiolle sekä .NET 5:lle. Seuraavaksi esitetään referenssisovelluksen mikropalveluarkkitehtuuri.

6.2.1 Sovellusarkkitehtuuri

Työssä käytetty esimerkkisovellus pohjautuu mikropalveluarkkitehtuuriin. Mikropalveluarkkitehtuuri on ohjelmistoarkkitehtuuri, jossa ohjelmiston toiminallisuudet jaetaan toisistaan erillisiin mikropalveluihin [37]. Näitä mikropalveluita voidaan päivittää, skaalata ja testata itsenäisesti [37].



Kuva 6.1. *Esimerkkisovelluksen sovellusarkkitehtuuri [1]*

Esimerkkisovelluksen arkkitehtuuri on esitetty kuvassa 6.1. Esimerkkisovellus sisältää kolme asiakassovellusta: mobiilisovellus, joka on toteutettu Xamarin.Forms-teknologialla; TypeScriptiä ja Angular-viitekehystä hyödyntävä yhden sivun sovellus (engl. single page application) sekä ASP.NET Core MVC -viitekehystä hyödyntävä WebMVC-sovellus. Tämän työn puitteissa infrastruktuurin pystyttämisen testauksessa käytettiin ainoastaan WebMVC-sovellusta.

Esimerkkisovelluksessa on käytössä Backend for Frontend (BFF) -käytäntö [42], joka on tapa luoda jokaiselle asiakassovellukselle oma taustajärjestelmä, jonka tehtävänä on palvella vain tätä tiettyä asiakassovellusta. Esimerkkisovelluksessa on käytössä kaksi BFF API Gateway -rajapintaa, joista toinen palvelee mobiilisovellusta ja vastaavasti toinen palvelee sekä Angular-, että WebMVC-sovelluksia. BFF-rajapinnat keskustelevat käytettävien mikropalveluiden kanssa HTTP- ja gRPC-protokollia hyödyntäen. Mikropalveluista ainoastaan identiteetti-mikropalvelu on sellainen, joka ei vaadi API Gateway:n käyttöä. Referenssisovelluksessa jokainen mikropalvelu omistaa omat tiedonlähteensä, kuten tietokannan. Tämä käytäntö on valittu sen perusteella, että se tekee kehitysympäristön pystyttämisestä ja kehittämisestä helpompaa [78].

Esimerkkisovelluksen mikropalvelut eivät kommunikoi suoraan keskenään, vaan käytössä on tapahtumaväylä (engl. event bus), joka toimii julkaise-tilaa -kanavan päällä (engl. publish-subscribe channel). Mikropalvelut julkaisevat tapahtumia, joita toiset mikropalvelut voivat tilata ja täten reagoida tapahtumiin. Mikropalveluiden lisäksi sovellukseen kuuluu hallinnollisia palveluita, joilla voidaan esimerkiksi tarkastella sovelluksen tilaa.

Alustan käyttöönotto esimerkkisovelluksen kehitysympäristön infrastruktuurin hallintaan tapahtui neljässä vaiheessa. Ensimmäisessä vaiheessa alustaa hyödynnettiin sovelluksen käyttämän Azure Kubernetes -klusterin ja tähän asennettavan kuormantasaajan pystyttämisessä. Toisessa vaiheessa kuvassa 6.1 esitettävät palvelinsovellukset asennettiin ensimmäisessä vaiheessa pystytettyyn Kubernetes-klusteriin. Kolmannessa vaiheessa sovelluksen infrastruktuuri pystytettiin myös muille tutkimukseen valituille pilvialustoille: Amazon Web Services, Google Cloud Platform ja DigitalOcean. Viimeisessä vaiheessa infrastruktuurille tehtiin automatisoituja yksikkö- ja integraatiotestejä.

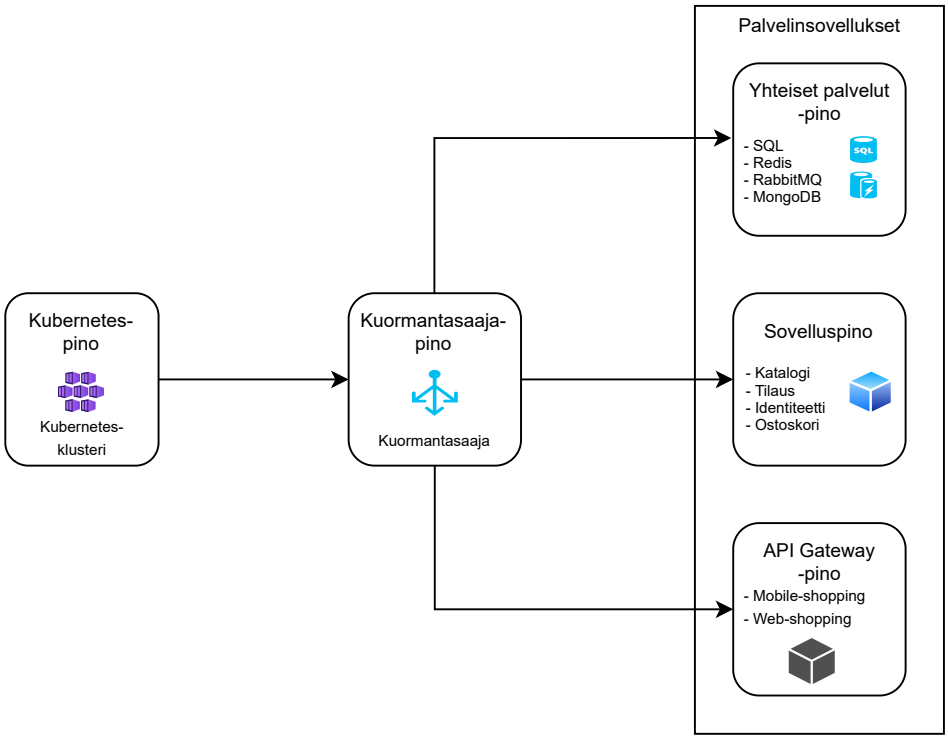
Seuraavassa alaluvussa esitellään tutkimuksessa käytetty kehitysympäristön infrastruktuuripinoarkkitehtuuri.

6.2.2 Infrastruktuuripinoarkkitehtuuri

Esimerkkisovelluksen infrastruktuurin lopullisen arkkitehtuurin pohjana hyödynnettiin eri infrastruktuuripinoarkkitehtuureja [41]. Monoliittista infrastruktuuripinoarkkitehtuuria ei käytetty, koska sitä käyttäessä kaikki infrastruktuuripinon resurssit jakaisivat saman elinkaaren [41]. Tämä olisi johtanut siihen, että pienetkin muutokset voivat vaikuttaa koko infrastruktuurin toimintaan. Tutkimuksen lopullinen infrastruktuuripinoarkkitehtuuri muodostui tutkimuksen aikana käytännön kokemusten pohjalta. Tässä sekä seuraavissa alaluvuissa esitetään tutkimuksessa käytetty infrastruktuuripinoarkkitehtuuri ja miten se toteutettiin alustalla.

Kuvassa 6.2 on esitetty tutkimuksessa käytetty infrastruktuuripinoarkkitehtuuri. Ensimmäisessä infrastruktuuripinossa luodaan muiden infrastruktuuripinojen tarvitsema Kubernetes-klusteri. Toisena infrastruktuuripinona on pino, jossa luodaan Kubernetes-yhteensopiva kuormantasaaja, joka mahdollistaa palveluiden käytön verkon ylitse verkkotunnuksella (engl. domain). Palvelinsovellukset jakautuvat kolmeen erilliseen infrastruktuuripinoon. Kaikki palvelinsovellusinfrastruktuuripinot viittaavat kuormantasaajapinon tulosteisiin. Kuvan 6.2 palvelinsovellusten infrastruktuuripinot jakautuvat seuraavanlaisesti:

- Yhteiset palvelut -pinossa luodaan sovellusten käyttämät palvelut, kuten SQL-tietokanta ja kehitysympäristössä käytetty RabbitMQ-tapahtumaväylä
- Sovelluspinossa luodaan sovellustason mikropalvelut
- API Gateway -pinossa luodaan sovellusten käyttämät API-yhdyskäytävät, jotka ohjaavat verkkoliikennettä sovelluspalveluihin



Kuva 6.2. *Esimerkkisovelluksen infrastruktuuripinoarkkitehtuuri*

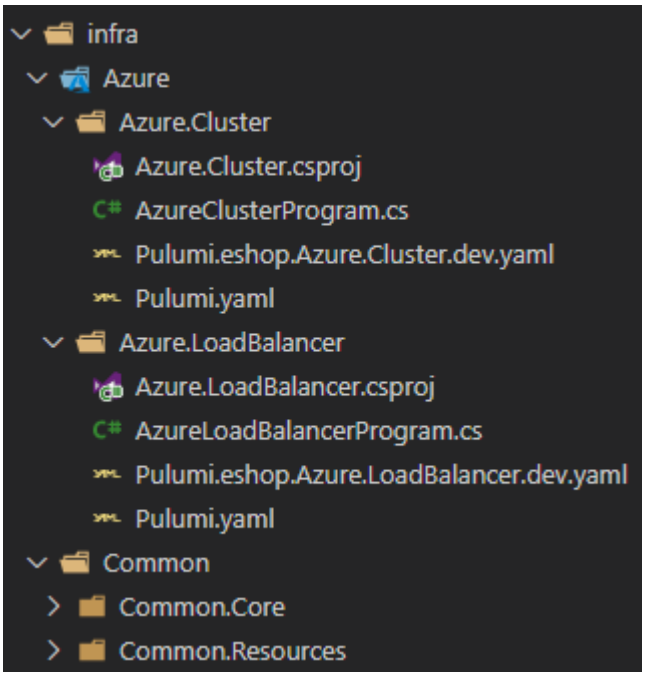
Palvelinsovellusten infrastruktuuripinojen yhtenä kriteerinä oli vastata esimerkkisovelluksen käyttämää kehitysympäristön rakennetta [1]. Koska työssä keskityttiin kehitysympäristöön soveltuvan infrastruktuurin luomiseen, ei työssä käytetty infrastruktuuri sovellu sellaisenaan tuotantoympäristössä käytettäväksi. Merkittävin muutos, joka täytyisi tehdä tuotantoympäristöä ajatellen, olisi muuttaa yhteiset palvelut -pinon eri palvelut käyttämään pilvialustariippuvaisia palveluita, kuten hajautettua SQL-tietokantaa konttien sijasta.

Seuraavissa luvun 6 alaluvuissa käydään läpi sitä, miten alustaa hyödynnettiin käytännössä tässä alaluvussa esitellyn infrastruktuuripinoarkkitehtuurin toteuttamisessa.

6.2.3 Alustan käyttöönotto: vaihe 1

Ensimmäisessä vaiheessa alustalla pystytettiin ne resurssit, jotka aikaisemmin pystytettiin manuaalisesti Microsoft Azure -pilvialustan verkkokäyttöliittymästä. Ensimmäisessä vaiheessa pystytettiin edellisessä alaluvussa esitetyt Kubernetes- ja kuormantasaajapinot.

Kuvassa 6.3 on esitetty infrastruktuuripinojen hakemistorakenne ensimmäisessä vaiheessa. Tutkimuksessa käytetty infrastruktuuripinoarkkitehtuuri toteutettiin alustalla siten, että `infra`-hakemiston alle luotiin jokaiselle tutkimuksessa käytetylle pilvialustalle oma hakemisto, jonka alihakemistoihin luotiin kaikille infrastruktuuripinoille Pulumi-projektit. Kaikille pilvialustoille yhteiset Pulumi-resurssit ja muut luokat lisättiin `Common`-hakemiston alihake-

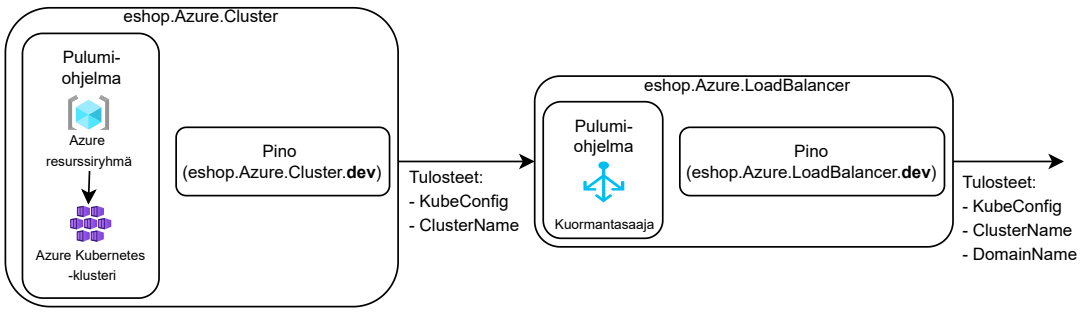


Kuva 6.3. *Infrastruktuurilähdekoodin Microsoft Azure -pilvialustalle tehdyn toteutuksen hakemistorakenne ensimmäisessä vaiheessa*

mistoihin. Kuvasta 6.3 nähdään Microsoft Azure -pilvialustalle tehtyjen Kubernetes-pinon sekä kuormantasaajapinon hakemistorakenne. Hakemistorakenne suunniteltiin siten, että jokainen infrastruktuuripino toteutettiin omana Pulumi-projektinaan. Koska Pulumi-projekti on sellainen hakemisto, josta löytyy `Pulumi.yaml`-tiedosto, nimettiin infrastruktuuripinojen hakemistot siten, että ne sisältävät käytetyn pilvialustan nimen ja infrastruktuuripinon nimen. Tämä helpottaa esimerkiksi tietyn Pulumi-projektin projektitiedoston etsimistä.

Jokainen infrastruktuuripino toteutettiin omana Pulumi-projektinaan ja jokaisella Pulumi-projektilla oli yksi tai useampi Pulumi-pino, joita käytettiin eri julkaisuympäristöjen erottelussa. Pääasiassa työssä käytettiin kuitenkin yhtä kehitysympäristöä. Pulumi-projektien nimet suunniteltiin siten, että Pulumi-projektin nimen tulee sisältää sen hakemiston nimi, josta `Pulumi.yaml`-projektitiedosto löytyy sekä työhön valittu etuliite `eshop`, joka tulee esimerkkisovelluksen nimestä. Esimerkiksi Azure Kubernetes -klusterin luomiseen käytetyn Pulumi-projektin nimi oli `eshop.Azure.Cluster`. Pulumi-pinot nimettiin siten, että pinojen nimet sisälsivät etuliitteenä sen Pulumi-projektin nimen, johon pino kuului ja loppuosana nimessä oli julkaisuympäristö. Esimerkiksi Azure Kubernetes -klusterin kehitysympäristön Pulumi-pino oli nimeltään `eshop.Azure.Cluster.dev`.

Kuvassa 6.4 on esitetty infrastruktuuripinojen Pulumi-projektirakenne ensimmäisessä vaiheessa. Ensimmäisenä kuvassa vasemmalla on Azure Kubernetes -klusterin pystyttämiseen käytetty infrastruktuuripino. Kubernetes-pino palauttaa tulosteinaan sen pystyttämän Kubernetes-klusterin nimen sekä Kubernetes-konfiguraation. Kuormantasaajan pystyttämiseen käytetty kuormantasaajapino viittaa Kubernetes-pinoon ja käyttää tämän tu-

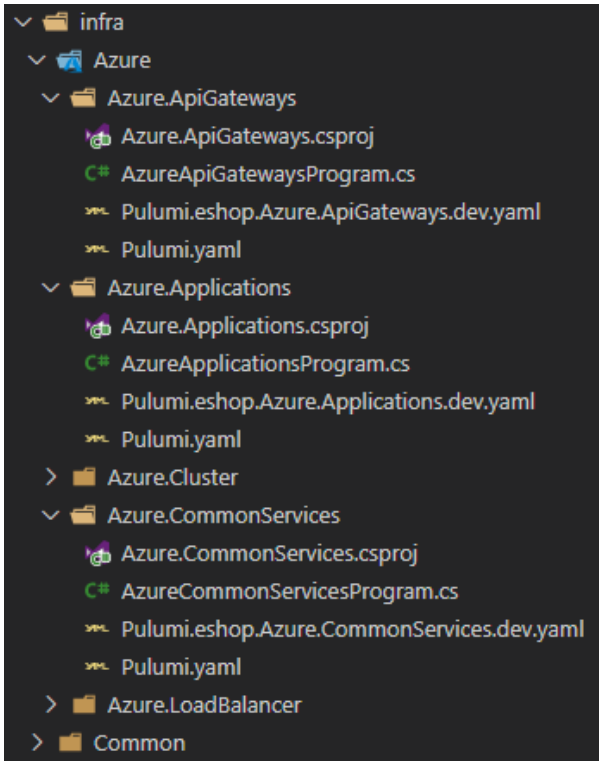


Kuva 6.4. *Pulumi-projektirakenne ensimmäisessä vaiheessa*

losteita pystytyksen aikana Kubernetes-klusterin päivittämiseen. Kuormantasaajapinon tulosteina olivat Kubernetes-pinon tulosteet, joiden lisäksi kuormantasaajapino palauttaa myös kuormantasaajalle varatun verkkotunnuksen. Seuraavaksi käydään tarkemmin läpi palvelinsovelluksien infrastruktuuripinoja, jotka hyödynsivät tässä luotuja Kubernetes-klusteria ja kuormantasaajaa.

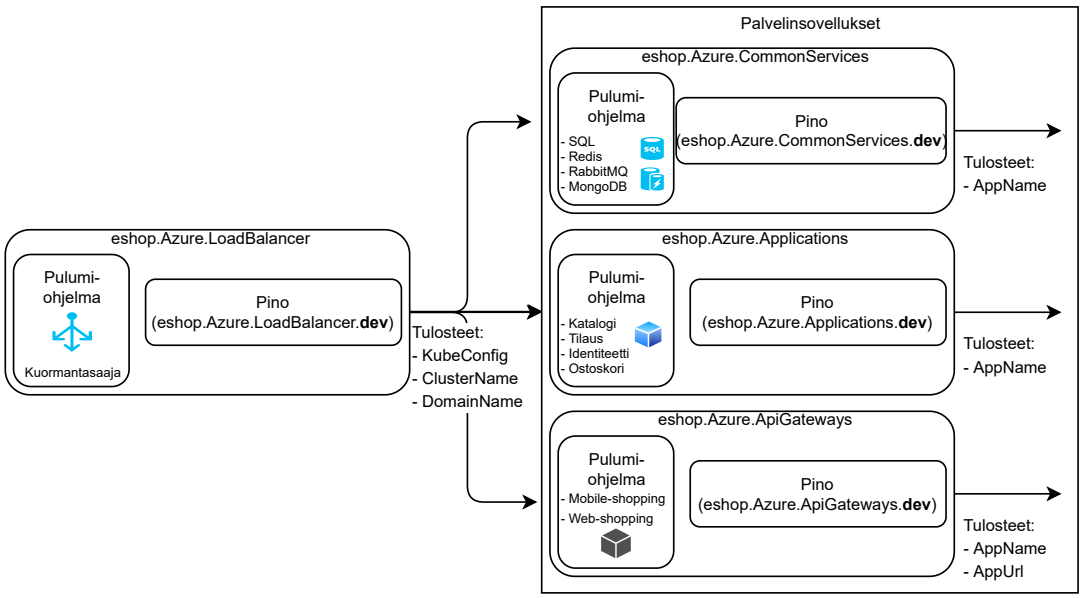
6.2.4 Alustan käyttöönotto: vaihe 2

Kun Kubernetes-klusteri ja klusteriin asennettu kuormantasaaja olivat toiminnassa, otettiin alusta käyttöön palvelinsovellusten kanssa. Esimerkkisovelluksessa käytettiin Helm-kaavioita palvelinsovellusten asentamiseen Kubernetes-klusteriin.



Kuva 6.5. *Infrastruktuurilähdekoodin Microsoft Azure -pilvialustalle tehdyn toteutuksen hakemistorakenne toisessa vaiheessa*

Kuvassa 6.5 on esitetty Azure-pilvialustalle tehdyn Pulumi-infrastruktuurin hakemistorakenne toisessa vaiheessa. Toisessa vaiheessa lisättiin kolme uutta Pulumi-projektia hakemistoineen aiemmin esiteltyjä hakemistorakennetta ja nimeämiskäytäntöjä hyödyntäen. Toisen vaiheen infrastruktuuripinot hyödynsivät esimerkkisovelluksessa valmiina olleita Helm-kaavioita, joita pystyttiin käyttämään alustalla. Nämä toisen vaiheen infrastruktuuripinot ovat pilvialustariippumattomia, eli ne eivät ole suoraan sidoksissa käytössä olevaan pilvialustaan, koska pinot ovat riippuvaisia ainoastaan Kubernetes-klusterista ja tähän asennetusta kuormantasaajasta. Pilvialustariippumattomuudesta huolimatta samaa hakemistorakennetta hyödynnettiin myös näiden infrastruktuuripinojen kanssa.



Kuva 6.6. *Palvelinsovellusten Pulumi-projektirakenne toisessa vaiheessa*

Kuvassa 6.6 on esitetty palvelinsovellusten Pulumi-projektirakenne. Kuvassa ei näy kuormantasaajapinon käyttämää Kubernetes-klusteria. Kaikki palvelinsovellusinfrastruktuuripinot käyttävät ensimmäisessä vaiheessa luodun kuormantasaajan tulosteita ensimmäisessä vaiheessa luodun Kubernetes-klusterin konfigurointiin. Oikealla ylimpänä on yhteiset palvelut -infrastruktuuripinon Pulumi-projekti. Yhteiset palvelut -pinon tulosteina saadaan sovellukselle konfiguroitu nimi. Oikealla keskellä on sovelluspinon Pulumi-projekti. Sovelluspinon tulosteina saadaan sama sovellukselle konfiguroitu nimi. Oikealla alimpana on API Gateway -pinon Pulumi-projekti. API Gateway -pinon tulosteina saadaan samainen sovellukselle konfiguroitu nimi, mutta lisäksi pino palauttaa myös kuormantasaajan tulosteena saadun verkkotunnuksen pohjalta luodun julkisen sovellusosoitteen, jonka voi avata selaimella.

Palvelinsovellusten infrastruktuuripinot eivät riipu toisistaan. Esimerkiksi sovelluspinon palvelut voidaan asentaa Kubernetes-klusteriin, vaikka yhteiset palvelut -pinoa ei ole vielä asennettu. Esimerkkisovelluksessa on otettu huomioon palvelun kestävyys erilaisten vikatilojen käsittelyssä, joten sovelluspalvelut voidaan asentaa, vaikka tietokantaa ei oli-

sikaan asennettu, mutta sovellus ei tällöin kuitenkaan tue kaikkia toiminnallisuuksia. Samoin sovellukseen ei pääse käsiksi verkon ylitse verkkotunnuksella, ellei API Gateways -infrastruktuuripinoa ole asennettu.

Seuraavaksi esitetään Pulumin käyttöönoton kolmas vaihe, jossa hyödynnettiin ensimmäisessä ja toisessa vaiheessa esitettyä infrastruktuuripinoarkkitehtuurin rakennetta useamman pilvialustan tukemisessa.

6.2.5 Alustan käyttöönotto: vaihe 3

Kolmannessa vaiheessa lisättiin tuki muille tutkimukseen valituille pilvialustoille, joita olivat AWS, GCP ja DigitalOcean. Kolmannen vaiheen tavoitteena oli saada esimerkkisovellus onnistuneesti toimimaan kaikilla tutkimukseen valituilla pilvialustoilla.

Kolmannessa vaiheessa hyödynnettiin aikaisemmin esiteltyä projektin hakemistorakennetta. Jokaiselle tuetulle pilvialustalle luotiin `infra`-hakemiston alle oma pilvialustakohtainen alihakemisto, johon tehtiin kuvaa 6.5 vastaava rakenne. Hakemistot ja Pulumi-projektit nimettiin pilvialustakohtaisesti aiemmin esiteltyjä käytäntöjä hyödyntäen. Käytännössä uuden pilvialustan lisääminen esimerkkisovellukseen tässä tutkimuksessa käytetyllä hakemistorakenteella vaati seuraavat toimenpiteet:

1. Luodaan kuvaa 6.5 vastaava hakemistorakenne uudelle pilvialustalle
2. Luodaan Kubernetes-pinossa pilvialustakohtainen Kubernetes-klusteri
3. Konfiguroidaan kuormantasaajapino ohjelman 6.1 esittämällä tavalla
4. Konfiguroidaan palvelinsovellusten infrastruktuuripinot osoittamaan kyseisen pilvialustan kuormantasaajapinoon
5. Alustetaan uuden pilvialustan Pulumi-pinot

Ohjelmassa 6.1 on esitetty komentoriviliitännällä ajettava komento, jolla pystytään konfiguroimaan kuormantasaajapino. Komento ajetaan kuormantasaajapinon Pulumi-projektitiedoston sisältävässä hakemistossa. Esimerkiksi konfiguroitaessa AWS-pilvialustaa korvattaisiin ohjelmassa korostettu <Pilvialusta>-teksti pilvialustan projektiin valitulla nimellä Aws. Lisäksi ajettavaan komentoon täytyy korvata käytössä oleva DNS-palvelu, DNS-palveluun konfiguroitava verkkotunnus sekä DNS-palvelukohtaiset tunnukset. Esimerkiksi DigitalOcean-alustan DNS-palvelua hyödyntäessä täytyy konfiguraatioon lisätä ohjelmassa esitetty API-avain.

```
1 pulumi config set-all --stack eshop.<Pilvialusta>.LoadBalancer.dev `
2     --plaintext eshop:DnsProvider=<DNS-palvelu> `
3     --plaintext eshop:Domain=<verkkotunnus> `
4     --plaintext eshop:ClusterProject=eshop.<Pilvialusta>.Cluster `
5     --secret digitalocean:token=<API-avain>
```

Ohjelma 6.1. *Kuormantasaajakehityspinon konfigurointi komentoriviliitännällä*

Hyödyntämällä tutkimuksessa esitettyä infrastruktuuripinoarkkitehtuuria ja tämän Pulumi-projektirakennetta pystyttiin uusia pilvialustoja lisäämään helposti. Yhteiset luokat ja Pulumi-resurssit olivat kaikkien projektien käytettävissä, ja niitä hyödyntäen pystyttiin minimoimaan pilvialustakohtainen konfigurointi.

Tässä luvussa esitetty Pulumi-projekti- ja hakemistorakenne eivät ole ainoa tapa toteuttaa tätä työtä vastaava toiminnallisuus. Tähän työhön valittu rakenne osoittautui kuitenkin käytännölliseksi ja helposti laajennettavaksi. Rakenteeseen vaikutti merkittävästi se, että referenssisovellus tuki pilvialustariippumattomia Kubernetesta ja Helm-kaavioita jo lähtökohtaisesti. Näiden ollessa pilvialustariippumattomia pystyttiin merkittävästä osaa Pulumi-projekteistakin tekemään pilvialustariippumattomia.

Esimerkkisovelluksen infrastruktuurin pystyttämisessä ei käytetty Pulumin automatisointirajapintaa, koska työtä aloittaessa se ei ollut vielä käytettävissä C#-kielellä. Automatisointirajapinnan käyttäminen tutkimuksessa esitetyn infrastruktuurin hallintaan olisikin hyvä jatkokehitysidea. Automatisointirajapintaa käytettiin kuitenkin infrastruktuurin testauksessa. Seuraavaksi käydään läpi tutkimuksessa tehdyn Pulumi-infrastruktuurin yksikkö- ja integraatiotestausta.

6.2.6 Infrastruktuurin testaus

Kehitysympäristön infrastruktuurin pystyttämisen lisäksi tutkimuksessa haluttiin myös kerätä käytännön kokemusta Pulumilla määritellyn infrastruktuurin testaamisesta. Testaus on tärkeässä osassa Profit Software Oy:ssä käytettäviä ohjelmistokehityskäytäntöjä.

Liitteessä D on esitetty yksikkötestejä, joita käytettiin infrastruktuurin testauksessa. Ohjelmassa D.1 on kaksi yksikkötestiä, joista ensimmäinen varmistaa, että pinon luomisen yhteydessä luodaan Azure-resurssiryhmä, ja toinen yksikkötesti varmistaa, että resurssiryhmälle on asetettu `Tags`-ominaisuudelle arvo. Ohjelmassa D.2 on esitetty kuormantasaajalle tehty yksikkötesti, jolla testataan, että pinoluokka palauttaa testissä simuloidun Kubernetes-palvelun IP-osoitteen.

Yksikkötestien lisäksi tässä työssä tehtiin koko kehitysympäristön infrastruktuurille integraatiotestejä. Integraatiotestauksessa hyödynnettiin Pulumi-automatisointirajapinnan esikatselussa olevaa C#-versiota. Koska automatisointirajapinta mahdollistaa Pulumi-pinojen hallinnoinnin, voidaan sitä hyödyntää myös testauksessa. Pulumin omaa Go-kielellä tehtyä integraatiotestiviitekehystä ei otettu käyttöön sen takia, että Go-kieltä ei tyypillisesti käytetä Profit Software Oy:n projekteissa ja myös siksi, että tässä työssä oli tarkoitus keskittyä Pulumin C#-kielen toiminnallisuuksiin.

Integraatiotestit toteutettiin siten, että automatisointirajapinnalla päivitettiin pinot niiden edellyttämässä järjestyksessä, eli ensimmäisenä testeissä pystytetään Kubernetes-klusteri, minkä jälkeen asennetaan klusteriin sopiva kuormantasaaja. Tämän jälkeen voidaan päi-

vittää palvelinsovellusten pinot eli yhteiset palvelut -pino, sovelluspino ja API Gateway -pino. Kun kaikki pinot on pystytetty, ajetaan päästä päähän -testi (engl. end-to-end test), joka suorittaa sovellukseen kirjautumisen ja tilauksen tekemisen selaimella, jota käsitellään ohjelmakoodin kautta. Testin lopuksi resurssit poistetaan käänteisessä järjestyksessä, eli ensimmäisenä poistetaan viimeisenä pystytetty API Gateway -pino ja viimeisenä poistetaan Kubernetes-klusteri. Poistaminen täytyy tehdä käänteisessä järjestyksessä, jotta resurssit poistetaan alustan hallinnasta onnistuneesti. Jos Kubernetes-pino poistetaan ensimmäisenä, ei alustan ole enää mahdollista onnistuneesti poistaa muiden pinojen resursseja Kubernetes-klusterista, koska Kubernetes-klusteri ei ole enää käytettävissä.

Tässä tutkimuksessa valittu infrastruktuuriarkkitehtuurin rakenne osoittautui hyödylliseksi myös testauksessa, sillä sama hakemistorakenne oli käytössä kaikilla työssä käytetyillä pilvialustoilla ja integraatiotestit voitiin parametrisoida toimimaan pilvialustakohtaisesti.

7. TULOKSET

Tässä luvussa esitetään Pulumi-alustan arvioinnin tulokset arviointikriteerikohtaisesti.

7.1 Teknologian kypsyys

Pulumi-komentoriviliitännästä julkaistaan versioita aktiivisesti. Taulukossa 7.1 on esitetty komentoriviliitännän 20 viimeisintä julkaisua. Julkaisuja tapahtuu aktiivisesti, mutta säännöllistä julkaisuaikataulua ei taulukon perusteella vaikuta olevan. Julkaisuja tapahtuu tarpeen mukaan, jos korjauksia tai uusia taaksepäin yhteensopivia muutoksia on valmiina.

Taulukko 7.1. *Pulumi-komentoriviliitännän 20 viimeisintä julkaisua*

Versio	Julkaisupäivämäärä	Versio	Julkaisupäivämäärä
v2.24.1	02.04.2021	v2.19.0	28.01.2021
v2.24.0	01.04.2021	v2.18.2	22.01.2021
v2.23.2	25.03.2021	v2.18.1	21.01.2021
v2.23.1	18.03.2021	v2.17.2	15.01.2021
v2.23.0	17.03.2021	v2.17.1	13.01.2021
v2.22.0	03.03.2021	v2.17.0	06.01.2021
v2.21.2	22.02.2021	v2.16.2	23.12.2020
v2.21.1	18.02.2021	v2.16.1	23.12.2020
v2.21.0	17.02.2021	v2.16.0	21.12.2020
v2.20.0	03.02.2021	v2.15.6	12.12.2020

Pulumia kehitetään pääasiassa yhteen päähaaraan nimeltä master, joka sisältää ne muutokset, joiden pohjalta komentoriviliitännän viimeisimmät versiot julkaistaan. Työtä tehdessä kehitteillä on Pulumin versio 3, joka sisältää taaksepäin yhteensopimattomia muutoksia, minkä takia versiota kehitetään omassa haarassa nimeltä feature-3.0.

Muutosten tekeminen Pulumiin tapahtuu GitHub-palveluun tehtävien vetopyyntöjen kautta. Kuka tahansa voi tehdä vetopyynnön, mutta ainoastaan Pulumi GitHub -organisaatioon kuuluvat henkilöt voivat hyväksyä vetopyyntöjä ja suorittaa vetopyynnön hyväksymiseen vaadittavia hyväksymistestejä. Pulumi GitHub -organisaatioon kuuluvat kaikki Pulumin omat ohjelmistokehityspaketit, jotka ovat erillisissä tietovarastoissaan.

Ennen vetopyynnön hyväksyntää täytyy suorittaa automatisoidut hyväksymistestit. Hyväksymistesteihin kuuluvat alustan yksikkö- ja integraatiotestit, joilla varmistetaan alustan toimivuus. Testejä on kirjoitettu kaikilla Pulumin tukemilla ohjelmointikielillä, ja jokaiselle Pulumin tukemalle ohjelmointikielelle on kielikohtaisia yksikkö- ja integraatiotestejä. Päätietovarastossa ajettavat hyväksymistestit eivät kuitenkaan testaa kaikkia Pulumin ohjelmointikirjastoja ja niiden sisältämiä resursseja. Testejä voidaan myös ajaa paikallisessa kehitysympäristössä, mutta osa testeistä kuitenkin olettaa, että kehittäjällä on pääsy integraatiotestien hyödyntämään Pulumi SaaS -palveluun.

GitHub-palvelussa[1] olevien tietojen mukaan kuukauden aikana [2] Pulumin päätietovaraston päähaaraan tuli yhteensä 108 vetopyyntöä, joista 92 yhdistettiin (engl. merge) päähaaraan. Suurin osa vetopyynnöistä on todennäköisimmin Pulumi-kehittäjien tekemiä, eli Pulumi Corporation -yrityksen työntekijöitä. Kaikkia Pulumin kehittäjiä ei ole kuitenkaan merkitty Pulumi-organisaatioon, joten täsmällisiä lukuja ei ole saatavilla.

Pulumi-alustaa kehitetään ohjelmistokehityskäytäntöjen mukaisesti. Alustaa testataan hyödyntämällä automaattisia yksikkö- ja integraatiotestejä.

7.2 Kehitysyhteisön aktiivisuus

Yhtenä kriteereistä kehitysyhteisön aktiivisuuden arviointiin käytettiin Stack Overflow'hun lähetettyjen kysymysten määrää[3]. Vertailuun otettiin Pulumin lisäksi Terraform, AWS CloudFormation, AWS CDK sekä Azure Resource Manageriin liittyvät kysymykset. Kuvassa 7.1 on esitetty infrastruktuuri koodina -lähestymistavan työkaluihin liittyvien Stack Overflow'n kysymysten määrän kasvu kvartaaleittain. Kuvaajasta nähdään, että Pulumista on tehty Stack Overflow'hun selvästi vähiten kysymyksiä, mutta maltillista kasvua on havaittavissa vuoden 2020 loppupuolen jälkeen. Vastaavasti kuvaajasta nähdään, että Terraformiin liittyvien kysymyksien määrä on kasvanut huomattavasti vuoden 2018 alun jälkeen.

Stack Overflow ei ole ainoa kehitysyhteisön aktiivisuuden ja koon mittari. Toisena mittarina käytettiin GitHub-palvelussa annettujen tähtien määrää. Tätä työtä kirjoittaessa Pulumilla on GitHubissa 8 100 tähteä vastaavasti kilpailevalla Terraformilla on 26 200 tähteä. Rekisteröityneet käyttäjät voivat antaa tietovarastolle tähden GitHub-palvelussa, jos he haluavat seurata kyseistä tietovarastoa. Tällä hetkellä ei ole todennäköistä, että kehitysyhteisö pystyisi jatkamaan Pulumin kehitystä, jos Pulumi Corporation lopettaisi toimintansa.

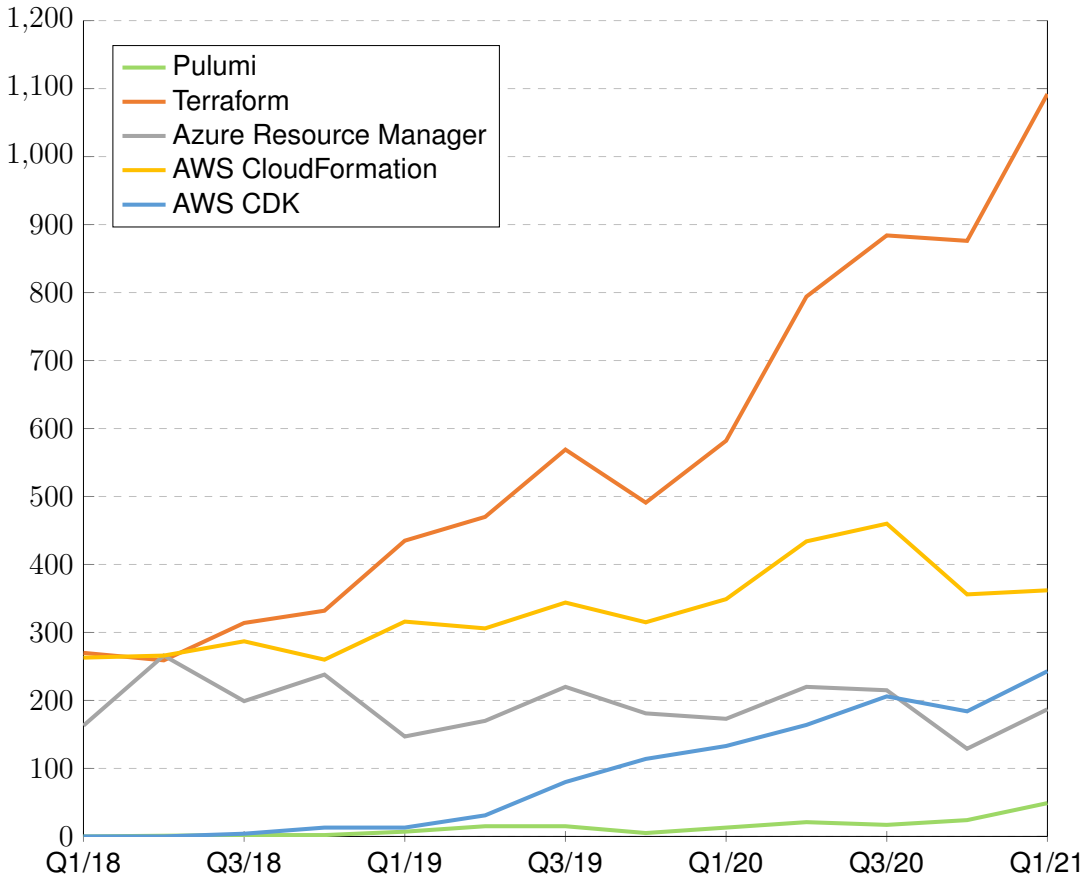
Pulumilla on käytössä kehitysyhteisölle Slack-viestipalvelu[4], jossa alustan kehittäjät ovat

[1] https://github.com/pulumi/pulumi/pulse/monthly

[2] Välillä 6.3.2021 - 6.4.2021

[3] Data pohjautuu https://data.stackexchange.com/ sivustolta saatuihin tuloksiin. Haussa käytettiin seuraavia tag-merkintöjä: amazon-cloudformation, arm-template, aws-cdk, azure-resource-manager, azure-template, pulumi, terraform. Azure Resource Manageriin liittyvät tag-merkinnät laskettiin yhteen.

[4] https://slack.pulumi.com/



Kuva 7.1. *Infrastruktuuri koodina -työkaluihin liittyvien Stack Overflow'n kysymysten määrän kasvu kvartaaleittain vuosina 2018-2021*

tavoitettavissa. On mahdollista, että ainakin osa kysymyksistä on esitetään suoraan yhteisön Slack-kanavilla Stack Overflow'n sijasta.

7.3 Tuetut pilvialustat ja muut palvelut

Tätä työtä kirjoittaessa Pulumi tukee dokumentaation perusteella yhteensä 53 eri pilvialustaa ja muuta palvelua C#-ohjelmointikielellä [7]. Profit Software Oy:n käyttämistä suosituimmista pilvialustoista tuetaan kaikkia, joita käytettiin tässä tutkimuksessa.

Kattava pilvialustatuki mahdollistaa yhden työkalun käytön eri pilvialustojen infrastruktuurin hallintaan. Pulumi mahdollistaa myös useamman pilvialustan käytön yhdessä Pulumi-projektissa. Tällöin on mahdollista hyödyntää monipilvimallia, jossa hyödynnetään useita pilvialustoja ja niiden palveluita [46].

7.4 Suorituskyky

Pulumi-alustan suorituskyvyn arviointiin käytettiin suorituskykytestausta sekä vertailtiin sitä toiminnallisuuksiltaan vastaavaan Terraform-alustan suorituskykyyn. Suorituskyky-

vertailuun käytettiin liitteen A ohjelmassa A.2 esitettyä Pulumi-pinoluokkaa ja liitteessä E esitettyjä Terraform-moduuleita. Testauksessa käytettiin toiminnallisuuksiltaan samankaltaisia infrastruktuuripinoja, joilla pystytettiin staattinen verkkosivu Microsoft Azure -pilvialustalle. Suorituskykyvertailussa käytettiin kuutta eri testitapausta, joista neljä suoritettiin Pulumilla ja kaksi Terraformilla. Testitapaukset laadittiin seuraavanlaisesti:

Testitapaus 1. Pulumi, SaaS-palvelu

Testitapaus 2. Pulumi, SaaS-palvelu, valmis ohjelmabinääri

Testitapaus 3. Pulumi, paikallinen tila

Testitapaus 4. Pulumi, paikallinen tila, valmis ohjelmabinääri

Testitapaus 5. Terraform, paikallinen tila

Testitapaus 6. Terraform, paikallinen tila, valmis suunnitelma

Testitapaukset jaettiin siten, että kummankin työkalun kanssa hyödynnettiin oletusasetuksilla ajoa sekä etukäteen tehtyjä toimenpiteitä. Toimenpiteillä tarkoitetaan tässä Pulumi-ohjelman kääntämistä valmiiksi suoritettavaksi etukäteen ja Terraform-suunnitelman luomista etukäteen. Pulumin kanssa käytettiin sekä SaaS-palvelua että paikallista tiedostojärjestelmää tilanhallintaan. Vastaavasti Terraformin kanssa käytettiin pelkästään paikallista tiedostojärjestelmää tilanhallintaan.

Taulukko 7.2. *Suorituskykyvertailussa käytettyjen työkalujen ja pakettien versiot*

Työkalu/Paketti	**Versio**
Pulumi-komentoriviliitäntä	v2.24.1
Pulumi.AzureNative-kirjasto	v0.8.0
Terraform-komentoriviliitäntä	v0.14.10
Terraform provider azurerm	v2.55.0
Terraform provider random	v3.1.0

Taulukossa 7.2 on esitetty vertailussa käytettyjen komentoriviliitäntöjen ja alustojen käyttämien pakettien versiot. Suorituskykyvertailussa käytettiin tietokonetta, jossa on Intel Core i7-8665U 1.90GHz -prosessori ja 32 gigatavua muistia sekä 200 megabitin verkkoyhteyttä. Suorituskykyvertailuun käytettiin C#-kielelle tehtyä BenchmarkDotNet-kirjastoa[5] ja komentorivityökalujen ajamiseen käytettiin CliWrap-kirjastoa[6], jolla kumpikin työkalu voitiin suorittaa samaan tapaan kuin ne ajettaisiin manuaalisesti komentorivikehoitteessa.

Yksi suorituskykymittaus suoritettiin ajamalla jokainen testitapaus yhteensä 6 kertaa, joista ensimmäinen kerta toimi lämmittelykertana eikä sen tuloksia otettu huomioon laskennassa. Yhden suorituskykymittauksen aikana eri infrastruktuuripinoja pystytettiin yhteensä 36 kertaa. Jokaisen testitapauksen suoritus pystytti infrastruktuuripinon alusta asti, eli

[5] https://benchmarkdotnet.org/
[6] https://github.com/Tyrrrz/CliWrap

mitään resursseja ei aikaisemmista suorituksista ollut saatavilla. Pulumilla tämä tehtiin siten, että jokaiselle suoritukselle luotiin oma Pulumi-pino. Vastaavasti Terraformin kanssa hyödynnettiin eri tiedostoja tilan tallennukseen. Yhden mittauksen jälkeen kaikki luodut resurssit poistettiin, minkä jälkeen mittaus toistettiin. Näin testitapaukset olivat toisistaan täysin erillisiä.

Mittaus toistettiin yhteensä 17 kertaa, joista 5 hylättiin karkean virheen vuoksi. Karkeaa virhettä aiheuttivat esimerkiksi tilapäiset verkkoyhteysongelmat tai resurssien poistamisen epäonnistuminen, joka näkyi tuloksissa suurena hajontana. Lopullisessa tulosten laskennassa käytettiin 12 suorituskykymittauksen tulosta. Yhteensä yhden testitapauksen infrastruktuuripino pystytettiin 72 kertaa, joista 12 oli lämmittelykertaa, eli tulosten laskennassa huomioitiin 60 infrastruktuuripinon pystytystä.

Suorituskykyvertailun tuloksissa on huomioitava käytetyt infrastruktuuriresurssit sekä käytetty pilvialusta, jotka vaikuttavat tuloksiin, mutta joiden tarkkaa vaikutusta on vaikea arvioida. Infrastruktuuriresurssien pystyttämiseen kuluva aika on riippuvainen muun muassa pilvialustaan kohdistuvasta kuormasta ja verkkoyhteyden latenssista. Tuloksissa on myös huomioitava se, että käännettäessä Pulumi-ohjelmaa ei ohjelmaa käännetty tyhjästä jokaisella suorituskerralla, koska valmiiksi käännettyjä tiedostoja ei poistettu testitapausten suoritusten välillä.

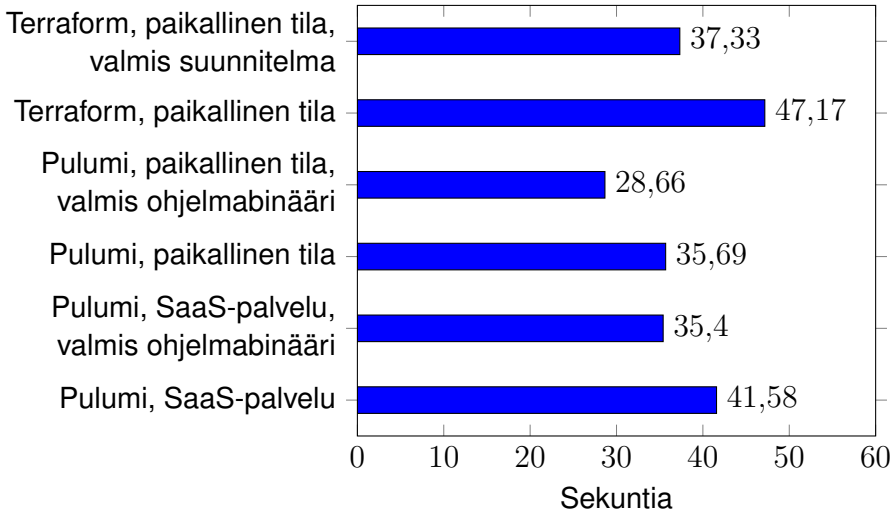
Taulukossa 7.3 on esitetty suorituskykyvertailun mittauksen tulokset. Taulukkoon on laskettu keskimääräinen infrastruktuuripinon pystyttämiseen kulunut aika sekä keskiarvon keskivirhe testitapauksittain liitteessä F esitettyjen mittaustulosten perusteella. Taulukon tulosten perustella Pulumi oli suorituskykyvertailussa nopein, kun käytettiin paikallista tilaa ja valmista ohjelmabinääriä. Terraform oli suorituskykyvertailussa hitain, kun se suoritettiin oletusasetuksilla. Oletusasetuksilla Terraformin toteutussuunnitelma luodaan ennen kuin muutokset tehdään.

Taulukko 7.3. *Suorituskykyvertailun tulokset*

Testitapaus	**Keston keskiarvo ja keskiarvon keskivirhe (s)**
Pulumi, SaaS-palvelu	$41{,}58 \pm 0{,}43$
Pulumi, SaaS-palvelu, valmis ohjelmabinääri	$35{,}40 \pm 0{,}23$
Pulumi, paikallinen tila	$35{,}69 \pm 0{,}54$
Pulumi, paikallinen tila, valmis ohjelmabinääri	$28{,}66 \pm 0{,}26$
Terraform, paikallinen tila	$47{,}17 \pm 0{,}48$
Terraform, paikallinen tila, valmis suunnitelma	$37{,}33 \pm 0{,}27$

Taulukon 7.3 tulosten perusteella voidaan tehdä karkeaa arviota Pulumi-ohjelman kääntä-

miseen kuluvasta ajasta sekä Terraform toteutussuunnitelman luomiseen kuluvasta ajasta. Taulukon tulosten perusteella Terraformin toteutussuunnitelman luomiseen kuluu keskimäärin noin 9,84 s. Pulumi-ohjelman kääntämiseen kuluva aika voidaan laskea ensiksi laskemalla tilanhallintapalvelukohtaisten tulosten erotus, jonka jälkeen tulokset lasketaan yhteen ja jaetaan kahdella. Lopputuloksena Pulumi-ohjelman kääntämiseen kuluu keskimäärin noin 6,61 s.



Kuva 7.2. *Suorituskykyvertailun tulokset kuvaajana*

Kuvassa 7.2 on esitetty taulukon 7.3 tulokset kuvaajana. Tässä alaluvussa tehdyn suorituskykyvertailun tuloksia ei voida pitää absoluuttisena totuutena vertailuun käytettyjen työkalujen suorituskykyeroista. Suorituskykyvertailun tulokset paljastavat kuitenkin sen, että ainakin samanlaista infrastruktuuria pystyttäessä Pulumi suoriutuu tehtävästään hyvin ilman suorituskykyongelmia.

7.5 Kustannukset ja yritysprofiili

Pulumia kehittävä Pulumi Corporation [2] on pääomasijoittajien tukema yhdysvaltalainen startup-yritys, joka on perustettu vuonna 2017 ja johon yrityksen omien verkkosivujen[7] mukaan kuuluu 43 työntekijää[8]. Tilastokeskuksen määritelmän[9] mukaan yritystä voidaan pitää pienenä yrityksenä sen henkilöstömäärän perusteella. Pulumi Corporation keräsi vuoden 2020 lokakuussa 37,5 miljoonaa dollaria B-sarjan rahoitusta[10]. Yrityksen liikevaihdosta tai muista taloudellisista tunnusluvuista ei löytynyt julkista tietoa.

Pulumin kehityksen lisäksi Pulumi Corporation tarjoaa Pulumi SaaS -palvelua. Pulumi SaaS -palvelu on ilmainen henkilökohtaiseen käyttöön [48] ja maksullinen yrityksille. Tätä työtä kirjoittaessa edullisin vaihtoehto kustantaa 50 dollaria kuukaudessa, jos laskutus

[7] https://www.pulumi.com/about/

[8] Työntekijäksi on laskettu ne henkilöt, jotka ovat osa Pulumi tiimiä 7.4.2021

[9] http://www.stat.fi/meta/kas/pienet_ja_keski.html

[10] https://info.pulumi.com/press-release/series-b-announcement

tehdään vuosittain, muussa tapauksessa kustannus on 60 dollaria kuukaudessa. Edullisin vaihtoehto sisältää tuen enintään kolmelle käyttäjälle ja enintään 20 Pulumi-pinolle. Itsehallinnoitavan tilan käyttö on kaikille ilmaista [48].

7.6 Automatisointi

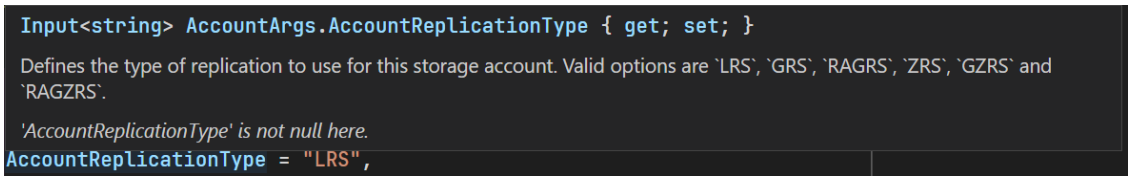
Pulumin komentoriviliitäntä on suunniteltu ja toteutettu siten, että sitä on mahdollista käyttää automatisointiin. Komentoriviliitäntää on mahdollista käyttää jatkuvan integraation ja toimituksen palveluissa. Pulumi tukee useita jatkuvan integraation ja jatkuvan toimituksen palveluita, kuten Azure DevOps, GitHub Actions ja GitLab CI [21]. Pulumin tukemia jatkuvan integraation palveluita voidaan käyttää esimerkiksi infrastruktuurimuutosten esikatseluun [21]. Jatkuvaa toimitusta hyödyntämällä on mahdollista automatisoida infrastruktuurin muutosten julkaisu.

Tässä työssä ei tehty erillistä tutkimusta Pulumin tukemien jatkuvan integraation ja toimituksen palveluista.

7.7 Kehitysympäristö

Integroidut kehitysympäristöt mahdollistavat usein jonkinlaisen navigoinnin lähdekooditasolla esimerkiksi luokkien määrityksiin, ja koodieditorissa voidaan tutkia resurssien ominaisuuksia. Ohjelmointikieliä käytettäessä voidaan käyttää samoja koodieditoreita infrastruktuurilähdekoodin kirjoittamiseen kuin muutakin sovelluslähdekoodia kirjoittaessa.

Hyödyntämällä vahvasti tyypitettyjä ohjelmointikieliä voidaan syötteiden täyttämisessä tyypillisesti hyödyntää kehittäjän käytössä olevaa integroitua kehitysympäristöä. Vahvasti tyypitetyt resurssit helpottavat niiden oikeanlaista täyttämistä. Kenttiä täyttäessä voidaan käytettävästä koodieditorista nähdä esimerkiksi kenttään liittyvää dokumentaatiota sekä kentän tietotyyppi. Kuvasta 7.3 nähdään, että `AccountReplicationType`-ominaisuuden tyyppinä on `Input<string>`. Kuvassa myös nähdään koodieditorin työkaluvihje, jossa on kenttään liittyvää dokumentaatiota.



Kuva 7.3. *Pulumi-koodin Visual Studio Code -editorin työkaluvihje (engl. tooltip)*

7.8 Konfigurointi

Pulumi-pinot mahdollistavat erilaisia käyttötapoja. Yksi tapa käyttää Pulumi-pinoja on käyttää niitä erottelemaan eri julkaisuympäristöt toisistaan. Tällaisia ympäristöjä voivat olla esimerkiksi kehitys-, testi- ja tuotantoympäristöt.

Pulumi tallentaa konfiguroinnin pinokohtaiseen tiedostoon, joka voidaan lisätä versionhallintaan. Versionhallinnan käyttö mahdollistaa pinon ja sen konfiguraation jakamisen. Versionhallinnan käyttö tukee myös jäljitettävyyttä, sillä muutoksia tehdessä versionhallintaan tyypillisesti tallennetaan myös käyttäjän nimi tai muu tunniste. Koska Pulumin käyttämät salaiset konfiguraatioarvot on salattu, voidaan myös salattuja arvoja sisältävät konfiguraatiot tallentaa versionhallintaan.

Pulumin komentoriviliitäntä mahdollistaa usean konfiguraatioarvon yhtäaikaisen asettamisen `pulumi config set-all` -komennolla. Tämä tekee esimerkiksi tarvittavien pinokohtaisten konfiguraatioiden dokumentoimisesta helpompaa, sillä dokumentaation voidaan laittaa yksi `pulumi config set-all` -komentoesimerkki, jonka kehittäjä voi muokata tarpeen mukaiseksi.

7.9 Laajennettavuus

Pulumi mahdollistaa omien resurssiabstraktioiden rakentamisen. Resursseja voidaan koostaa komponenttiresursseiksi.Komponenttiresursseista voidaan kehittää esimerkiksi yhteisiä ohjelmistokehityskirjastoja, joita voidaan käyttää eri projektien välillä. Hyödyntämällä komponenttiresursseja sekä jaettuja pinoluokkia voidaan rakentaa uudelleenkäytettäviä pinoja [41]. Komponenttiresurssit sellaisinaan rajoittuvat kuitenkin vain sen ohjelmointikielen kanssa käytettäväksi kuin se, millä ne on kirjoitettu.

Kehitysyhteisön ja ekosysteemin kannalta on tärkeää, että alustan päälle voidaan rakentaa omia laajennuksia [38]. Näitä laajennuksia voitaisiin esimerkiksi jakaa lähdekoodina avoimen lähdekoodin lisenssillä, ja kaikki alustaa käyttävät voisivat hyödyntää niitä.

Monet Pulumin omista ohjelmointikirjastoista pohjautuvat Terraform-lähdekoodiin, jonka pohjalta on mahdollista generoida Pulumi-ohjelmointikirjastoja [61]. Tämä myös mahdollistaa uusien Terraform-resurssien tuomisen Pulumin käyttöön. Terraform-tuen lisäksi Pulumiin on mahdollista rakentaa omia resurssintarjoajia.

Oman resurssintarjoajan voi rakentaa dynaamisena resurssintarjoajana (engl. dynamic provider) [65], joka ei vaadi erillistä ohjelmalisäkettä ja joka suoritetaan sen sisältävän ohjelman kontekstissa. Dynaamisia resurssintarjoajia ei kuitenkaan ole tällä hetkellä mahdollista tehdä muilla kuin Python-, JavaScript- ja TypeScript-kielillä. Dynaamiset resurssintarjoajat soveltuvat esimerkiksi mukautettujen resurssien tai uusien alustojen tukemiseen. Dynaamisia resurssintarjoajia ei voida helposti jakaa paketteina [65]. Dynaamisten

resurssien lisäksi on mahdollista luoda omia resurssintarjoajia Pulumin Go-kielen ohjelmointikirjastoa hyödyntämällä [86]. Tällaiset resurssintarjoajat soveltuvat jaettaviksi, sillä resurssintarjoajan määrityksen pohjalta generoidaan tarvittavat ohjelmointikielikohtaiset kirjastot [86]. Komponenttiresursseille on myös mahdollista rakentaa oma resurssintarjoaja, jonka avulla komponenttiresursseja voidaan myös hyödyntää kaikilla Pulumin tukemilla ohjelmointikielillä [85]. Komponenttiresurssintarjoajia on mahdollista tätä työtä tehdessä tehdä Go-, JavaScript- ja TypeScript-kielillä.

Pulumiin on siis mahdollista tehdä omia laajennuksia ja uusien palveluiden tukemiseen on useampia tapoja. Kaikki kielet eivät kuitenkaan tue vielä kaikkia laajennusvaihtoehtoja, mikä täytyy huomioida, jos on tarve omille resurssintarjoajille.

7.10 Alustan omaksuminen

Pulumi-alustan dokumentaatio [7] ja ohjelmointikielten käyttö tukevat Pulumi-alustan omaksumista. Pilvialusta- ja resurssikohtainen dokumentaatio sisältää usein esimerkkejä resurssien käyttämisestä.

Pulumi tarjoaa hyvän dokumentaation ainakin suosituimmille pilvialustoille. Pulumin dokumentaatio ei kuitenkaan itsessään riitä infrastruktuuriresurssien kehittämiseen, vaan pilvipalvelualustojen ja muiden käytettävien alustojen dokumentaatiota täytyy myös hyödyntää. Tämä on kuitenkin odotettavissa, sillä Pulumi on vain yksi tapa käyttää sen tukemia palveluita.

Pulumin GitHub-palveluun julkaistut esimerkkiprojektit[11] ovat hyviä lähtökohtia eri pilvialustojen käyttöön. Esimerkkiprojektien lisäksi Pulumin sivustoilta löytyy ohjeita [4] Pulumin käyttöönottoon, jos käytössä on jokin toinen infrastruktuuri koodina -työkaluista.

Pulumi tukee C#-ohjelmointikieltä riittävällä tasolla. Pulumin C#-kielen kirjastot eivät tätä työtä tehdessä tukeneet vielä kaikkia ominaisuuksia, joita esimerkiksi Pulumin TypeScript-versio tukee. Arviointiin tehdyssä tutkimuksessa ei kuitenkaan havaittu sellaisia ongelmia, jotka olisivat vaatineet toisen Pulumin tukeman ohjelmointikielen käyttöä.

Koska Pulumia käytetään ohjelmointikielillä täsmä- tai konfiguraatiokielten, kuten YAML tai JSON sijasta, voidaan sen kanssa hyödyntää käytössä olevan ohjelmointikielten rakenteita ja käytäntöjä. C#-ohjelmointikielen kanssa tämä tarkoittaa sitä, että resurssien ja pinojen kanssa voidaan hyödyntää muun muassa perintää (engl. inheritance) ja koostamista (engl. composition). Näiden lisäksi ohjelmointikielille tyypilliset rakenteet, kuten ehtolauseet sekä silmukat, ovat käytettävissä.

Tutkimuksessa hyödynnettiin perintää muun muassa luomalla liitteen C ohjelmassa C.2 esitetty `StackBase`-kantaluokka tutkimuksessa käytetyille pinoluokille. Tämä kantaluokka

[11] https://github.com/pulumi/examples

sisältää pinoluokissa yleisesti käytettyjä staattisia ominaisuuksia, kuten Pulumi-pinon nimen, Pulumi-projektin nimen, sekä organisaation nimen, jota käytetään siinä tapauksessa, että Pulumi-tilanhallintaan käytetään Pulumi SaaS -palvelua. Edellä mainittujen ominaisuuksien lisäksi `StackBase`-kantaluokan rakentajassa tarkistetaan, että Pulumi-pinon nimi vastaa työhön valittuja käytäntöjä.

Vahvasti tyypitettyä ohjelmointikieltä käytettäessä saadaan jo käännösvaiheessa osa virheistä kiinni. Pulumi voisi hyödyntää C#-kielen ominaisuuksia laajemmin. Esimerkiksi resurssien parametrit annetaan kaikki olioalustajassa (engl. object initializer), joka ei käännösaikana pysty erottamaan pakollisia parametreja valinnaisista. Tämä johtaa siihen, että puutteellisista parametreista saadaan virheet vasta ajonaikaisesti. Pulumi sisältää kattavat virheilmoitukset, joten virheiden korjaus on usein nopeaa ja selkeää. Listauksessa 7.1 on esitetty Pulumi-komentoriviliitännän ajonaikainen virhe, kun pakollista resurssiryhmän nimeä ei annettu parametrina `Azure.Storage.Account`-resurssille.

```
1  Diagnostics:
2    pulumi:pulumi:Stack (<Pinon nimi>):
3      error: Running program '<polku>' failed with an unhandled exception:
4      System.ArgumentNullException:
5        [Input] Pulumi.Azure.Storage.AccountArgs.ResourceGroupName is required
6        but was not given a value (Parameter 'ResourceGroupName')
7            at Pulumi.InputArgs.ToDictionaryAsync()
8            at <leikattu>
```

Listaus 7.1. *Ajonaikainen virhe resurssiryhmän nimen puuttuessa*

Listauksen 7.1 rivillä 5 ja 6 nähdään, että virhe sisältää sen luokan nimen, josta parametri puuttuu. Tässä tapauksessa `AccountArgs`-luokan `ResourceGroupName`-kentälle ei annettu arvoa. Useamman puutteellisen kentän tapauksessa Pulumi ei raportoi kuin yhden kentistä kerrallaan, joten komento täytyy ajaa puutteellisen kentän lisäämisen jälkeen.

Virheellisten syötearvojen tapauksessa Pulumi raportoi näistä useamman. Virheellisten syötearvojen virheilmoitukset eivät kuitenkaan sisällä enää suoraa viittausta alkuperäiseen lähdekoodiin, vaan virheet viittaavat resursseihin, joita ollaan luomassa. Listauksessa 7.2 on esitetty ote Pulumin komentoriviliitännän tulostamasta virheilmoituksesta, kun `Azure.Storage.Account`-resurssille annettiin tarkoituksellisesti väärin kirjoitetut `AccountReplicationType`-parametrin sekä `AccountTier`-parametrin arvot. Väärin kirjoitetut arvot ovat korostettuna listauksessa.

```
1  Diagnostics:
2    azure:storage:Account (storage):
3      error: azure:storage/account:Account resource 'storage' has a problem:
4        expected account_replication_type to be one of
5          [LRS ZRS GRS RAGRS GZRS RAGZRS], got LRSS
6      error: azure:storage/account:Account resource 'storage' has a problem:
7        expected account_tier to be one of [Standard Premium], got Standardd
```

Listaus 7.2. *Ajonaikainen virhe, kun resurssille on syötetty virheelliset arvot*

Listauksesta 7.2 nähdään, että Pulumi tulostaa virheellisesti kirjoitetun resurssin odotetut arvot. Tämä helpottaa virheiden korjausta.

Tätä työtä kirjoittaessa Pulumin C#-kielen ohjelmistokehityskirjastot käyttävät paljon merkkijono- eli `string`-tyyppisiä resurssiparametreja, myös sellaisissa tilanteissa, kuten yllä, joissa parametrille on sallittu vain tietyt arvot. Tämä tarkoittaa sitä, että virheellisesti kirjoitetut arvot havaitaan vasta infrastruktuurikoodia suoritettaessa.

Omissa komponenttiresurssi- ja pinoluokissa on puolestaan mahdollista käyttää vahvasti tyypitettyjä arvoja. Esimerkiksi tutkimuksessa hyödynnettiin liitteen C ohjelmassa C.3 esitettyä `StackName`-tietoluokkaa Pulumi-pinon nimen kapseloinnissa. Tällä tavoin pystytään helpommin varmistumaan siitä, että projektiin valittuja nimikäytäntöjä noudatetaan.

Yhtenäisyyden vuoksi on suotavaa hyödyntää omien Pulumi-resurssien parametrien kanssa samankaltaisia rakenteita, kuin Pulumin ohjelmistokehityskirjastot käyttävät. Tämä tekee resurssien jakamisesta ja niiden käytöstä yhtenäisempää.

7.11 Testattavuus

Pulumi tukee yksikkö- ja integraatiotestausta infrastruktuuri koodina -lähestymistavan mukaisesti. Pulumi-alustaa käytettäessä ei ole esteitä infrastruktuurin testaukselle. Infrastruktuurin testaus on hyvä tapa varmistua siitä, että sovelluksen koko elinkaari toimii oletetusti.

Integraatiotestejä ajettaessa on olemassa riski, että testien aikana luotavat resurssit jäävät kummittelemaan, eli niitä ei poisteta onnistuneesti. Tämä voi tapahtua esimerkiksi siinä tilanteessa, että testejä ajettaessa tietokone tai testiä suorittava prosessi kaatuu. Tässä tapauksessa testin aikana luotavia resursseja ei poisteta, joten niiden poistaminen täytyy tehdä manuaalisesti. Pahimmassa tapauksessa kummittelevat resurssit jäävät huomaamatta, minkä vuoksi siihen voi myös liittyä taloudellinen riski. Riski on suurempi siinä tapauksessa, että testit ajetaan käyttämällä paikallista tilaa. Paikallista tilaa käytettäessä infrastruktuurin tilan tiedot ovat ainoastaan sillä koneella, jolla testit on ajettu.

Kummitusresurssiriskiä ei pystytä täysin poistamaan, mutta sitä voidaan kuitenkin pienentää eri tavoin. Yksi tapa on käyttää paikallisen tilan sijasta joko Pulumin tarjoamaa SaaS-palvelua tai vaihtoehtoisesti jotakin Pulumin tukemista tiedostonvarastointivaihtoehdoista, kuten AWS S3 -sankoa tai Azure Blob Storagea. Yhteistä tilanhallintaa käyttäessä voidaan havaita mahdollisesti poistamatta jääneet resurssit ja poistaa ne manuaalisesti.

Pinokohtaisten yksikkötestien kirjoittaminen on mahdollista C#-kielellä. Yksikkötestit soveltuvatkin esimerkiksi resurssien syötearvojen tarkasteluun. Yksikkötestien kirjoittaminen vaatii kuitenkin tietämystä Pulumin toiminnasta, jotta niitä voidaan kirjoittaa pinoille, jotka käyttävät esimerkiksi pinoviittauksia. Samoin esimerkiksi Helm-kaavioita käyttävien

pinojen testaus vaatii tietämystä Pulumin toiminnasta.

7.12 Vian etsintä

Vian etsinnän käyttäminen Pulumi-ohjelman suorituksen aikana vaatii tätä työtä kirjoittaessa räätälöidyn toteutuksen tekemistä. Vian etsintää kannattaa käyttää vain esikatselutilassa, koska tällöin resursseja ei oikeasti luoda. Jos nimittäin vian etsintää käytetään samaan aikaan kun resursseja luodaan, on mahdollista aiheuttaa virheitä resurssien luomisessa.

Liitteen C ohjelmassa C.1 on esitetty vian etsintää tukemaan luotu staattinen C#-luokka, jonka ainoaa julkista metodia kutsumalla voidaan käynnistää vian etsintä, jos jompikumpi luokan käyttämistä ympäristömuuttujista on määritelty. Luokan käyttämät ympäristömuuttujat eivät ole Pulumin virallisesti tukemia, vaan ne pohjautuvat kehitysyhteisön ehdotukseen[12]. Tätä käytettäessä vian etsintä tapahtuu siten, että asetetaan luokan käyttämän ympäristömuuttujan arvoksi `true`, minkä jälkeen suoritetaan Pulumi-ohjelma esikatselutilassa ajamalla komento `pulumi preview`.

7.13 Ylläpidettävyys

Pulumia käyttävää infrastruktuurilähdekoodia voidaan ylläpitää samoja käytäntöjä hyödyntäen kuin muuta sovelluslähdekoodia. Infrastruktuurilähdekoodia ylläpidettäessä on kuitenkin tärkeä ymmärtää Pulumin toiminnan perusteet, jotta vältytään esimerkiksi resurssien poistamiselta väärästä pinosta.

Infrastruktuurilähdekoodin ylläpidon kannalta on myös tärkeä tietää käytössä olevista palveluista ja niiden toiminnasta ennen muutosten tekemistä. Pulumi ei aseta rajoitteita infrastruktuurilähdekoodin ylläpidolle. Ohjelmointikielten käytäntöjen ja abstraktioiden hyödyntäminen tukee ylläpidettävyyttä sekä omaksumista.

[12] https://github.com/pulumi/pulumi/issues/1372

8. YHTEENVETO

Pulumi on moderni infrastruktuuri koodina -lähestymistapaan tehty alusta, joka mahdollistaa sen tukemien ohjelmointikielten käytön infrastruktuurin määrittämiseen. Pulumi tukee useita pilvialustoja ja muita palveluita, joista suurin osa rajattiin tämän työn tutkimuksen ulkopuolelle.

Luvussa 6 tehdyssä tutkimuksessa perehdyttiin Pulumin toimintaan ja sen käyttöön käytännössä esimerkkisovelluksen avulla. Työhön valittu esimerkkisovellus toimi hyvänä esimerkkinä pilvialustariippumattoman Kuberneteksen käytöstä yhdessä Pulumin ja eri pilvialustojen kanssa.

Koska Pulumia käytetään ohjelmointikielillä, mahdollistaa se käytössä olevalle ohjelmointikielelle tyypillisten abstraktioiden rakentamisen. Näitä ovat C#-kieltä käytettäessä muun muassa perintä ja koostaminen. Abstraktioiden avulla voidaan infrastruktuurilähdekoodista tehdä tarkoitukseen sopivaa ja kuvaavaa. Abstraktioista voidaan tehdä omia kirjastoja, joita voidaan jakaa käytettävälle ohjelmointikielelle tyypilliseen tapaan.

Abstraktioiden lisäksi Pulumiin voi lisätä omia laajennuksia. Pulumin laajennettavuus mahdollistaa myös sellaisten palveluiden käytön, joille ei virallisesti ole vielä tukea. Tätä työtä kirjoittaessa kaikki kielet eivät kuitenkaan tukeneet samoja ominaisuuksia eikä kaikkia laajennuksia voida tehdä pelkällä C#-kielellä. Resurssitarjoajalaajennuksia voidaan tehdä käyttämällä Go-ohjelmointikieltä.

Yksi merkittävä lisä Pulumi-alustaan on sen automatisointirajapinta, joka tarjoaa uudenlaisia käyttötapoja ja menetelmiä infrastruktuurin hallintaan. Automatisointirajapinnan tutkiminen ja sen käyttö olisivatkin hyviä jatkokehitysideoita.

Pulumin omaksuminen oli helppoa, sillä sitä käytetään ohjelmointikielellä konfiguraatiokielten sijasta. Kattava dokumentaatio ja hyvät esimerkit tukivat eri pilvialustojen käyttöönottoa. Etenkin valmiit palvelukohtaiset esimerkit toimivat hyvänä lähtökohtana toteutukselle. Pulumi kuitenkin vaatii myös käytettävien palveluiden tuntemusta.

Pulumi on tätä työtä kirjoittaessa aktiivisessa kehityksessä ja työn tekemisen aikana alustaan ja sen ohjelmistokirjastoihin tuli useita päivityksiä. Alustan kehitystä voidaan seurata GitHub-palvelussa, johon voidaan myös ilmoittaa vioista ja kehitysideoista. Pulumi ei ole vielä saavuttanut suurta kehitysyhteisöä kehittäjien tyypillisesti käyttämissä pal-

veluissa, kuten Stack Overflow'ssa, joten ongelmien selvityksessä täytyy turvautua joko GitHub-palvelun kautta vian ilmoittamiseen tai käyttää Pulumin kehitysyhteisön Slack-viestipalvelua. Pulumin kehittäjät osallistuvat keskusteluun Slack-palvelussa ja heiltä saakin usein tarvittaessa apua.

Luvun 6 tutkimuksen perusteella voidaan Pulumia suositella harkittavaksi käyttöön Kubernetesta hyödyntävissä projekteissa. Pulumin avulla on mahdollista hallinnoida sekä Kubernetes-klusteriin liittyvää infrastruktuuria, että Kubernetes-klusterissa ajettavia sovelluksia. Pulumi-alusta tarjoaa yhtenäisen käyttökokemuksen kumpaankin käyttötapaukseen.

Työssä tehdyn tutkimuksen perusteella Pulumia suositellaan harkittavaksi yhtenä vaihtoehtona Profit Software Oy:ssä käytettävän pilvialustainfrastruktuurin hallintaan. Sen käyttö voitaisiin aloittaa käyttämällä sitä kehitys- tai testiympäristön infrastruktuurin pystyttämisessä. Käyttö olisi mahdollista aloittaa ilman kustannuksia hyödyntämällä itsehallinnoitavaa tilaa. Itsehallinnoitavan tilan hyödyntäminen edellyttää kuitenkin jonkinlaisten yhteisten käytäntöjen sopimista. Pulumi SaaS -palvelun käyttöä voitaisiin harkita projektikohtaisesti. Pulumi tukee kaikkia suosituimpia Profit Software Oy:ssä käytettäviä pilvialustoja. Yrityksen nykyisiä kehityskäytäntöjä voidaan hyödyntää Pulumin C#-kielellä kirjoitettujen toteutusten kanssa.

Pulumin käyttöönotossa täytyy huomioida kehitysyhteisön koko. Vaikka alusta onkin teknologialtaan hyvä, voi sen käyttöönotto pysähtyä, jos ongelmien selvitykseen ei löydy tarvittavia resursseja. Koska Pulumi on avoimen lähdekoodin projektina GitHub-palvelussa, vianselvitys ja korjausten tekeminen on tietysti mahdollista myös itse. Tämä vaatii ajan lisäksi perehtymistä Pulumi-alustan toimintaan ja sen kehitykseen.

LÄHTEET

[1] *.NET Microservices Sample Reference Application*. URL: https://github.com/dotnet-architecture/eShopOnContainers (viitattu 6.2.2021).

[2] *About*. Pulumi. URL: https://www.pulumi.com/about/ (viitattu 7.4.2021).

[3] *About Us - CFEngine - Distributed Configuration Management*. URL: http://web.archive.org/web/20160326035531/https://auth.cfengine.com/the-history-of-cfengine (viitattu 4.2.2021).

[4] *Adopting Pulumi*. Pulumi. URL: https://www.pulumi.com/docs/guides/adopting/ (viitattu 7.4.2021).

[5] Aiello, J. *What is PowerShell? - PowerShell*. URL: https://docs.microsoft.com/en-us/powershell/scripting/overview (viitattu 26.2.2021).

[6] *Announcing Chef*. Chef Blog. Jan. 15, 2009. URL: https://blog.chef.io/announcing-chef (viitattu 28.3.2021).

[7] *API Reference*. Pulumi. URL: https://www.pulumi.com/docs/reference/pkg/ (viitattu 7.4.2021).

[8] *Architecture & Concepts*. Pulumi. URL: https://www.pulumi.com/docs/intro/concepts/ (viitattu 16.1.2021).

[9] Artač, M., Borovšak, T., Di Nitto, E., Guerriero, M. ja Tamburri, D. A. DevOps: introducing infrastructure-as-code. *Proceedings of the 39th International Conference on Software Engineering Companion*. ICSE-C '17. Buenos Aires, Argentina: IEEE Press, 20. toukokuuta 2017, s. 497–498. ISBN: 978-1-5386-1589-8. DOI: 10.1109/ICSE-C.2017.162. URL: http://doi.org/10.1109/ICSE-C.2017.162 (viitattu 13.2.2021).

[10] *AWS CloudFormation - Infrastructure as Code & AWS Resource Provisioning*. Amazon Web Services, Inc. URL: https://aws.amazon.com/cloudformation/ (viitattu 13.4.2021).

[11] *Azure Resource Manager overview*. URL: https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview (viitattu 13.4.2021).

[12] *Bash - GNU Project - Free Software Foundation*. URL: https://www.gnu.org/software/bash/ (viitattu 26.2.2021).

[13] Block, A. ja Dewey, A. *Learn Helm*. 1st edition. Packt Publishing, 2020. ISBN: 1-83921-429-5.

[14] Burgess, M. Configuration Engine V2.0 (7. syyskuuta 1993). URL: https://web.archive.org/web/20130723160143/http://www.iu.hio.no/~mark/papers/cfengine_history.pdf (viitattu 4.2.2021).

[15] Burgess, M. A Site Configuration Engine. *The USEENIX Association, Computing Systems* 8.3 (1995).

[16] Burns, B., Beda, J. ja Hightower, K. *Kubernetes: Dive into the Future of Infrastructure*. Sebastopol: O'Reilly Media, Incorporated, 2019. ISBN: 978-1-4920-4653-0.

[17] Burns, B., Grant, B., Oppenheimer, D., Brewer, E. ja Wilkes, J. Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade. *Queue* 14.1 (1. tammikuuta 2016), s. 70–93. ISSN: 1542-7730. DOI: 10.1145/2898442.2898444. URL: http://doi.org/10.1145/2898442.2898444 (viitattu 28.2.2021).

[18] *CNCF Helm Project Journey Report*. Cloud Native Computing Foundation. URL: https://www.cncf.io/cncf-helm-project-journey-report/ (viitattu 28.2.2021).

[19] *Command: plan*. Terraform by HashiCorp. URL: https://www.terraform.io/docs/cli/commands/plan.html (viitattu 12.4.2021).

[20] *Configuration*. Pulumi. URL: https://www.pulumi.com/docs/intro/concepts/config/ (viitattu 12.3.2021).

[21] *Continuous Delivery*. Pulumi. URL: https://www.pulumi.com/docs/guides/continuous-delivery/ (viitattu 13.4.2021).

[22] *Deploy to Azure Kubernetes Service (AKS)*. URL: https://github.com/dotnet-architecture/eShopOnContainers/wiki/Deploy-to-Azure-Kubernetes-Service-(AKS) (viitattu 6.2.2021).

[23] *Download and Install*. Pulumi. URL: https://www.pulumi.com/docs/get-started/install/ (viitattu 26.2.2021).

[24] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, Petersen, K. ja Mäntylä, M. V. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. *2012 7th International Workshop on Automation of Software Test (AST)*. 2012 7th International Workshop on Automation of Software Test (AST). Kesäkuu 2012, s. 36–42. DOI: 10.1109/IWAST.2012.6228988.

[25] Ebert, C., Gallardo, G., Hernantes, J. ja Serrano, N. DevOps. *IEEE Software* 33.3 (toukokuu 2016), s. 94–100. ISSN: 1937-4194. DOI: 10.1109/MS.2016.68.

[26] Forrester. *Infrastructure As Code: Fueling The Fire For Faster Application Delivery*. Forrester, maaliskuu 2015. URL: https://www.microsoft.com/en-us/download/details.aspx?id=46403.

[27] Fowler, M. *ValueObject*. URL: https://martinfowler.com/bliki/ValueObject.html (viitattu 18.2.2021).

[28] *From AWS CloudFormation*. Pulumi. URL: https://www.pulumi.com/docs/guides/adopting/from_aws/ (viitattu 27.3.2021).

[29] *From Azure Resource Manager (ARM)*. Pulumi. URL: https://www.pulumi.com/docs/guides/adopting/from_azure/ (viitattu 27.3.2021).

[30] *From Terraform*. Pulumi. URL: https://www.pulumi.com/docs/guides/adopting/from_terraform/ (viitattu 27.3.2021).

[31] *Input Variables - Configuration Language*. Terraform by HashiCorp. URL: https://www.terraform.io/docs/language/values/variables.html (viitattu 12.4.2021).

[32] *Inputs and Outputs*. Pulumi. URL: https://www.pulumi.com/docs/intro/concepts/inputs-outputs/ (viitattu 4.2.2021).

[33] *Integration Testing*. Pulumi. URL: https://www.pulumi.com/docs/guides/testing/integration/ (viitattu 12.4.2021).

[34] *Introduction to Azure Kubernetes Service - Azure Kubernetes Service*. 2019. URL: https://docs.microsoft.com/en-us/azure/aks/intro-kubernetes (viitattu 6.2.2021).

[35] *Introduction to Terraform*. Terraform by HashiCorp. URL: https://www.terraform.io/intro/index.html (viitattu 10.4.2021).

[36] *Languages*. Pulumi. URL: https://www.pulumi.com/docs/intro/languages/ (viitattu 15.1.2021).

[37] Larrucea, X., Santamaria, I., Colomo-Palacios, R. ja Ebert, C. Microservices. *IEEE Software* 35.3 (toukokuu 2018), s. 96–100. ISSN: 1937-4194. DOI: 10.1109/MS.2018.2141030.

[38] Liao, Z., Deng, L., Fan, X., Zhang, Y., Liu, H., Qi, X. and Zhou, Y. Empirical Research on the Evaluation Model and Method of Sustainability of the Open Source Ecosystem. *Symmetry* 10.12 (Dec. 2018), p. 747. DOI: 10.3390/sym10120747. URL: https://www.mdpi.com/2073-8994/10/12/747 (viitattu 5.4.2021).

[39] Liao, Z., Qi, X., Zhang, Y., Fan, X. and Zhou, Y. How to Evaluate the Productivity of Software Ecosystem: A Case Study in GitHub. *Scientific Programming* 2020 (Aug. 3, 2020), e8814247. ISSN: 1058-9244. DOI: 10.1155/2020/8814247. URL: https://www.hindawi.com/journals/sp/2020/8814247/ (viitattu 5.4.2021).

[40] Mell, P. and Grance, T. *The NIST Definition of Cloud Computing*. NIST Special Publication (SP) 800-145. National Institute of Standards and Technology, Sept. 28, 2011. DOI: https://doi.org/10.6028/NIST.SP.800-145. URL: https://csrc.nist.gov/publications/detail/sp/800-145/final (viitattu 24.1.2021).

[41] Morris, K. *Infrastructure As Code*. Sebastopol: O'Reilly Media, Incorporated, 2021. ISBN: 1-09-811467-1.

[42] Newman, Sam. *Backends For Frontends*. URL: https://samnewman.io/patterns/architectural/bff/ (viitattu 19.2.2021).

[43] Nyman, L. ja Lindman, J. Code Forking, Governance, and Sustainability in Open Source Software. *Technology Innovation Management Review* (January 2013: Open Source Sustainability 2013), s. 7–12. ISSN: 1927-0321.

[44] *Output Values - Configuration Language*. Terraform by HashiCorp. URL: https://www.terraform.io/docs/language/values/outputs.html (viitattu 12.4.2021).

[45] *Overview - Configuration Language*. Terraform by HashiCorp. URL: https://www.terraform.io/docs/language/index.html (viitattu 10.4.2021).

[46] Petcu, D. Multi-Cloud: expectations and current approaches. *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*. MultiCloud '13. New York, NY, USA: Association for Computing Machinery, 22. huhtikuuta 2013, s. 1–6. ISBN: 978-1-4503-2050-4. DOI: 10.1145/2462326.2462328. URL: http://doi.org/10.1145/2462326.2462328 (viitattu 7.4.2021).

[47] Plösch, R., Dautovic, A. ja Saft, M. The Value of Software Documentation Quality. *2014 14th International Conference on Quality Software*. 2014 14th International Conference on Quality Software. Lokakuu 2014, s. 333–342. DOI: 10.1109/QSIC.2014.22.

[48] *Pricing*. Pulumi. URL: https://www.pulumi.com/pricing/ (viitattu 16.1.2021).

[49] *Projects*. Pulumi. 2021. URL: https://www.pulumi.com/docs/intro/concepts/project/ (viitattu 23.1.2021).

[50] *Property Testing*. Pulumi. URL: https://www.pulumi.com/docs/guides/testing/property-testing/ (viitattu 12.4.2021).

[51] *Pulumi*. URL: https://github.com/pulumi/pulumi (viitattu 3.2.2021).

[52] *Pulumi - Modern Infrastructure as Code*. Pulumi. 2021. URL: https://www.pulumi.com/ (viitattu 15.1.2021).

[53] *Pulumi Architecture*. Pulumi. URL: https://www.pulumi.com/docs/intro/concepts/how-pulumi-works/ (viitattu 27.2.2021).

[54] *Pulumi CLI*. Pulumi. URL: https://www.pulumi.com/docs/reference/cli/ (viitattu 26.2.2021).

[55] *pulumi config*. Pulumi. URL: https://www.pulumi.com/docs/reference/cli/pulumi_config/ (viitattu 26.2.2021).

[56] *pulumi destroy*. Pulumi. URL: https://www.pulumi.com/docs/reference/cli/pulumi_destroy/ (viitattu 26.2.2021).

[57] *pulumi login*. Pulumi. URL: https://www.pulumi.com/docs/reference/cli/pulumi_login/ (viitattu 26.2.2021).

[58] *pulumi new*. URL: https://www.pulumi.com/docs/reference/cli/pulumi_new/ (viitattu 26.2.2021).

[59] *pulumi preview*. Pulumi. URL: https://www.pulumi.com/docs/reference/cli/pulumi_preview/ (viitattu 26.2.2021).

[60] *pulumi stack*. Pulumi. URL: https://www.pulumi.com/docs/reference/cli/pulumi_stack/ (viitattu 26.2.2021).

[61] *Pulumi Terraform Bridge*. URL: https://github.com/pulumi/pulumi-terraform-bridge (viitattu 7.4.2021).

[62] *pulumi up*. Pulumi. URL: https://www.pulumi.com/docs/reference/cli/pulumi_up/ (viitattu 26.2.2021).

[63] *Pulumi-komentoriviliitännän lisenssi*. URL: https://github.com/pulumi/pulumi/blob/master/LICENSE (viitattu 29.1.2021).

[64] Reifer, D. J. *Software maintenance success recipes*. 1st edition. Boca Raton, Fla: CRC Press, 2012. ISBN: 0-429-10704-8.

[65] *Resources*. Pulumi. URL: https://www.pulumi.com/docs/intro/concepts/resources/ (viitattu 12.3.2021).

[66] *Secrets*. Pulumi. URL: https://www.pulumi.com/docs/intro/concepts/secrets/ (viitattu 12.3.2021).

[67] *Self-Hosted Pulumi Service*. Pulumi. 2021. URL: https://www.pulumi.com/docs/guides/self-hosted/ (viitattu 3.2.2021).

[68] Sinclair, A. Licence Profile: Apache License, Version 2.0. *International free and open source software law review* 2.2 (2010), s. 107–114. ISSN: 1877-6922. DOI: 10.5033/ifosslr.v2i2.42.

[69] *Stacks*. Pulumi. 2021. URL: https://www.pulumi.com/docs/intro/concepts/stack/ (viitattu 3.2.2021).

[70] *State and Backends*. Pulumi. 2021. URL: https://www.pulumi.com/docs/intro/concepts/state/ (viitattu 3.2.2021).

[71] Taipale, O., Kasurinen, J., Karhu, K. and Smolander, K. Trade-off between automated and manual software testing. *International Journal of System Assurance Engineering and Management* 2.2 (June 1, 2011), pp. 114–125. ISSN: 0976-4348. DOI: 10.1007/s13198-011-0065-6. URL: https://doi.org/10.1007/s13198-011-0065-6 (viitattu 5.4.2021).

[72] Taleb, N. ja Mohamed, E. A. Cloud Computing Trends: A Literature Review. *Academic journal of interdisciplinary studies* 9.1 (2020), s. 91. ISSN: 2281-3993. DOI: 10.36941/ajis-2020-0008.

[73] *Terraform by HashiCorp*. URL: https://www.terraform.io/ (viitattu 16.1.2021).

[74] *Testing*. Pulumi. URL: https://www.pulumi.com/docs/guides/testing/ (viitattu 12.4.2021).

[75] *The Pulumi Automation API - The Next Quantum Leap in IaC*. Pulumi. URL: https://www.pulumi.com/blog/automation-api/ (viitattu 28.3.2021).

[76] Thomas, D. ja Hunt, A. Mock objects. *IEEE Software* 19.3 (toukokuu 2002), s. 22–24. ISSN: 1937-4194. DOI: 10.1109/MS.2002.1003449.

[77] Torre, C. de la. *Containerized Docker Application Lifecycle with Microsoft Platform and Tools*. Microsoft Corporation.

[78] Torre, C. de la, Wagner, B. and Rousos, M. *.NET Microservices: Architecture for Containerized .NET Applications*. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/ (viitattu 5.2.2021).

[79] *Unit Testing*. Pulumi. URL: https://www.pulumi.com/docs/guides/testing/unit/ (viitattu 12.4.2021).

[80] Wagner, B. *A Tour of C# - C# Guide*. URL: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (viitattu 3.2.2021).

[81] Wagner, B. *Casting and type conversions - C# Programming Guide*. URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/casting-and-type-conversions (viitattu 26.2.2021).

[82] Wagner, B. *Generics - C# Programming Guide*. URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/ (viitattu 27.2.2021).

[83] Wagner, B. *What's new in C# 9.0 - C# Guide*. URL: https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-9 (viitattu 26.2.2021).

[84] *Welcome to Puppet 7.5.0*. URL: https://puppet.com/docs/puppet/7.5/puppet_index.html (viitattu 28.3.2021).

[85] *xyz Pulumi Component Provider (TypeScript)*. URL: https://github.com/pulumi/pulumi-component-provider-ts-boilerplate (viitattu 7.4.2021).

[86] *xyz Pulumi Provider*. URL: https://github.com/pulumi/pulumi-provider-boilerplate (viitattu 7.4.2021).

[87] Zayour, I. and Hajjdiab, H. How Much Integrated Development Environments (IDEs) Improve Productivity? *Journal of Software* 8.10 (Oct. 1, 2013), pp. 2425–2431. ISSN: 1796-217X. DOI: 10.4304/jsw.8.10.2425-2431. URL: http://ojs.academypublisher.com/index.php/jsw/article/view/9619 (viitattu 5.4.2021).

LIITE A: PULUMI-INFRASTRUKTUURIPINOESIMERKKI

Tässä liitteessä on esitetty komponenttiresurssi ja pinoluokka, joita käyttämällä voidaan luoda staattinen verkkosivu Microsoft Azure -pilvialustalle.

```
using Pulumi;
using Storage = Pulumi.AzureNative.Storage;

namespace Esimerkki {
    public record VerkkosivuKomponenttiresurssiArgs {
        public Input<string> ResurssiryhmanNimi { get; init; }
        public Input<string> Sisalto { get; init; }
        public Storage.StorageAccount Varastotili { get; init; }
    }

    // Komponenttiresurssit perivät Pulumi.ComponentResource-luokasta
    public class VerkkosivuKomponenttiresurssi : ComponentResource {
        [Output] public Output<string> VerkkosivunOsoite { get; init; }

        public VerkkosivuKomponenttiresurssi(string name,
            VerkkosivuKomponenttiresurssiArgs args,
            ComponentResourceOptions? options = null)
            : base($"azure-esimerkki:resurssi:Verkkosivu", name, options) {
            // Luodaan verkkosivu resurssi
            var verkkosivu = new Storage.StorageAccountStaticWebsite($"{name}-web", new
                Storage.StorageAccountStaticWebsiteArgs {
                AccountName = args.Varastotili.Name,
                ResourceGroupName = args.ResurssiryhmanNimi,
                IndexDocument = "index.html",
                // Asetetaan komponenttiresurssi resurssin vanhemmaksi oletusarvona resurssit kuuluvat
                    pinoon
            }, new() { Parent = this });

            // Tallennetaan verkkosivun sisältö tietovarastoon
            var index = new Storage.Blob("index.html", new Storage.BlobArgs {
                AccountName = args.Varastotili.Name,
                ResourceGroupName = args.ResurssiryhmanNimi,
                ContainerName = verkkosivu.ContainerName,
                Source = args.Sisalto.Apply(ToAsset),
                ContentType = "text/html",
            }, new() { Parent = verkkosivu });

            // Tulosteena saadaan varastotilin kautta
            // haettua se osoite, josta verkkosivu on saatavilla
            this.VerkkosivunOsoite = args.Varastotili.PrimaryEndpoints
                .Apply(endpoint => endpoint.Web);
            // Komponenttiresurssin rakentajan lopussa kutsutaan RegisterOutputs-metodia,
            // joka kertoo että kyseisen resurssin luominen on valmis
            this.RegisterOutputs();

        }
        private static AssetOrArchive ToAsset(string input) => new StringAsset(input);
    }
}
```

Ohjelma A.1. *Komponenttiresurssiesimerkki*

```
using System.Threading.Tasks;
using Pulumi;
using Resources = Pulumi.AzureNative.Resources;
using Storage = Pulumi.AzureNative.Storage;

namespace Esimerkki {
    // Pinoluokkien tulee periä Pulumi.Stack-luokasta
    public class AzureEsimerkkipino : Stack {
        [Output] public Output<string> VarastonAvain { get; init; }
        [Output] public Output<string> ResurssiryhmanNimi { get; init; }
        [Output] public Output<string> VerkkosivunOsoite { get; init; }

        // Resurssien alustus tapahtuu pinoluokan rakentajassa
        public AzureEsimerkkipino() {
            var konfiguraatio = new Config("esimerkki-nimiavaruus");
            var arvoesimerkinArvo = konfiguraatio.Get("arvoesimerkki");

            // Luodaan resurssiryhmä
            var resurssiryhma = new Resources.ResourceGroup("resurssiryhma");

            // Luodaan Storage Account -resurssi
            var varastotili = new Storage.StorageAccount("varastotili", new Storage.
                StorageAccountArgs {
                ResourceGroupName = resurssiryhma.Name,
                Sku = new Storage.Inputs.SkuArgs {Name = Storage.SkuName.Standard_LRS},
                Kind = Storage.Kind.StorageV2,
                EnableHttpsTrafficOnly = true });

            // Luodaan VerkkosivuKomponenttiresurssi, joka saa parametrinaan
            // resurssiryhmän nimen, varastotili-olion sekä verkkosivun sisällön
            var verkkosivu = new VerkkosivuKomponenttiresurssi("verkkosivu", new
                VerkkosivuKomponenttiresurssiArgs {
                ResurssiryhmanNimi = resurssiryhma.Name,
                Varastotili = varastotili,
                // Tulostemerkkijonojen käsittelyyn on Output.Format-metodi,
                // jolla pystytään käyttämään C#-kielen merkkijono-interpolaatiota
                // tulosteiden ja syötteiden kanssa
                Sisalto = Output.Format($@"
<html>
<body>
    <h1>Pulumi: Esimerkkiverkkosivu</h1>
    <p>Varastotilin nimi: {varastotili.Name}</p>
    <p>arvoesimerkki: {arvoesimerkinArvo}</p>
</body>
</html>"), });

            // Palautetaan varaston avain salaisena tulosteena
            this.VarastonAvain = Output
                // Käytetään Output.Tuple:a yhdistämään resurssiryhmän ja varastotilin nimitulosteet
                .Tuple(resurssiryhma.Name, varastotili.Name)
                // Haetaan varastotilin avain nimitulosteita hyödyntäen ja merkitään se salaiseksi
                .Apply(names => Output.CreateSecret(HaeVarastotilinAvain(names)));
            // Palautetaan verkkosivun osoite Pulumi-pinon tulosteena
            this.VerkkosivunOsoite = verkkosivu.VerkkosivunOsoite;
            // Palautetaan resurssiryhmän nimi tulosteena
            this.ResurssiryhmanNimi = resurssiryhma.Name;
        }

        private static async Task<string> HaeVarastotilinAvain((string
            resurssiryhmanNimi, string varastotilinNimi) args) {
            var tilinAvaimet = await Storage.ListStorageAccountKeys.InvokeAsync(new
                Storage.ListStorageAccountKeysArgs {
                ResourceGroupName = args.resurssiryhmanNimi,
                AccountName = args.varastotilinNimi });
            return tilinAvaimet.Keys[0].Value;
        }
    }
}
```

Ohjelma A.2. *Pinoluokkaesimerkki*

LIITE B: AUTOMATISOINTIRAJAPINTAESIMERKKI

Tässä liitteessä esitetään Pulumin automatisointirajapintaa käyttävä ohjelmaesimerkki. Ohjelmaesimerkki hyödyntää liitteen A ohjelmassa A.2 esitettyä pinoluokkaa.

```
using System;
using System.Collections.Generic;
using System.Linq;
using Pulumi.Automation;

// Ajamalla ohjelma parametrilla destroy tuhotaan pino, muussa tapauksessa päivitetään
bool poistetaan = args.Any(arg => arg == "destroy");

// Luodaan Pulumi-ohjelma viitaten pinoluokkaan
var ohjelma = PulumiFn.Create<Esimerkki.AzureEsimerkkipino>();
var projektinNimi = "azure-esimerkki-1";
var pinonNimi = $"{projektinNimi}.dev-auto";

// Luodaan Pulumi-pino edellä määriteltyjen ohjelman, projektin ja pinon nimen mukaisesti
var pino = await LocalWorkspace.CreateOrSelectStackAsync(
    new InlineProgramArgs(projektinNimi, pinonNimi, ohjelma));

// Asennetaan tarvittavat pilvialustakohtaiset ohjelmalisäkkeet
// Käytettäessä automatisointiohjelman suorituksen aikana määriteltyä Pulumi-ohjelmaa
// täytyy lisäkkeet asentaa erikseen
await pino.Workspace.InstallPluginAsync("azure-native", "v0.8.0");

await pino.SetAllConfigAsync(new Dictionary<string, ConfigValue>() {
    { "azure-native:location", new ConfigValue("northeurope") },
    { "esimerkki-nimiavaruus:arvoesimerkki", new ConfigValue("Pulumi
        automatisointirajapinta-arvoesimerkin arvo") },
});

// Päivitetään pinon tila pilvialustan perusteella
await pino.RefreshAsync();

if (poistetaan) {
    // Poistetaan pinon resurssit
    var poistoTulos = await pino.DestroyAsync();
    var kesto = poistoTulos.Summary.EndTime - poistoTulos.Summary.StartTime;
    Console.WriteLine($"Poistaminen kesti {kesto.TotalSeconds} sekuntia");
} else {
    // Päivitetään pino ja sen resurssit ja tulostetaan pinon tulosteet lopuksi
    var paivitysTulos = await pino.UpAsync();
    var kesto = paivitysTulos.Summary.EndTime - paivitysTulos.Summary.StartTime;
    Console.WriteLine($"Päivitys kesti {kesto.TotalSeconds} sekuntia");
    foreach (var (avain, tuloste) in paivitysTulos.Outputs)
    {
        Console.WriteLine($"Tuloste {avain}: {(tuloste.IsSecret ? "[salainen]" : tuloste
            .Value)}");
    }
}
```

Ohjelma B.1. *Automatisointirajapintaesimerkki*

LIITE C: ESIMERKKISOVELLUKSEN INFRASTRUKTUURIN YHTEISIÄ LUOKKIA

Tässä liitteessä esitellään tutkimuksessa käytettyjä infrastruktuuripinojen yhteisiä luokkia.

```
1  namespace Infra.Common.Core
2  {
3      public static class DebugUtils
4      {
5          public static void StartDebuggingIfEnabled()
6          {
7              if (IsDebuggingEnabled())
8              {
9                  WaitOrLaunchDebugger();
10             }
11         }
12
13         private static bool IsDebuggingEnabled()
14         {
15             // PULUMI_DEBUG ja PULUMI_DEBUG_LAUNCH eivät ole Pulumin
16             // virallisesti tukemia ympäristömuuttujia.
17             // Niitä käytetään vian etsinnän mahdollistamiseksi
18             bool debug = EnvUtils.GetBoolean("PULUMI_DEBUG", whenNotFound: false);
19             bool debugLaunch = EnvUtils.GetBoolean("PULUMI_DEBUG_LAUNCH", whenNotFound:
                   false);
20
21             return Pulumi.Deployment.Instance.IsDryRun && (debug || debugLaunch);
22         }
23
24         private static void WaitOrLaunchDebugger()
25         {
26             var process = System.Diagnostics.Process.GetCurrentProcess();
27
28             Pulumi.Log.Info($"Debug ProcessId='{process.Id}' Name='{process.ProcessName
                   }'");
29
30             if (EnvUtils.GetBoolean("PULUMI_DEBUG_LAUNCH", whenNotFound: false))
31             {
32                 Pulumi.Log.Info("Launching Debugger");
33                 System.Diagnostics.Debugger.Launch();
34             }
35             else
36             {
37                 Pulumi.Log.Info("Waiting for debugger...");
38                 Pulumi.Log.Info("Attach debugger to the process manually.");
39
40                 while (!System.Diagnostics.Debugger.IsAttached)
41                 {
42                     System.Threading.Thread.Sleep(100);
43                 }
44             }
45         }
46     }
47 }
```

Ohjelma C.1. *Vian etsintään käytetty luokka*

```
namespace Infra.Common.Core
{
    /// <summary>
    /// Kantaluokka Pulumi-pinoille. Sisältää yleisesti käytettyjä tietoja, kuten
    /// pinon, projektin ja organisaation nimen.
    /// Lisäksi kantaluokka mahdollistaa pinojen vian etsinnän preview-komennon yhteydessä.
    /// </summary>
    public abstract class StackBase : Pulumi.Stack
    {
        /// <summary>Pulumi-pinon nimi</summary>
        public static StackName CurrentStack =>
            StackName.From(Pulumi.Deployment.Instance.StackName);

        /// <summary>Pulumi-projektin nimi</summary>
        public static string CurrentProject => Pulumi.Deployment.Instance.ProjectName;

        /// <summary>
        /// Käytettäessä Pulumi SaaS -palvelua pinot kuuluvat johonkin organisaatioon.
        /// Yksin käytettäessä tässä on käyttäjän käyttäjätunnus.
        /// </summary>
        public static string Organization => "esimerkkiorganisaatio";
        public static string PackageName => "eshop";
        public static string CurrentEnvironment => CurrentStack.Environment;

        protected StackBase() : base(options: null)
        {
            // Kutsumalla CurrenStack-ominaisuutta kutsutaan StackName.From metodia,
            // joka varmistaa, että pinon nimi on oikeassa muodossa.
            _ = CurrentStack;
            DebugUtils.StartDebuggingIfEnabled();
        }
    }
}
```

Ohjelma C.2. *Pulumi-pinoluokkien kantaluokkana käytetty abstrakti StackBase-luokka*

```
namespace Infra.Common.Core
{
    public record StackName(string Project, string Environment)
    {
        public static StackName From(string project, string environment) =>
            new(project, environment);

        public static StackName From(string value)
        {
            // Etsitään viimeinen .-merkki ja varmistetaan että merkkijono
            // sisältää yhden pisteen, joka ei saa olla merkkijonon ensimmäisenä merkkinä
            var index = value.LastIndexOf('.') switch {
                int it when it > 0 => it,
                _ => throw new InvalidStackNameException(
                    $"Ensure the StackName is in 'ProjectName.Environment' format. Got
                        '{value}'"
                ),
            };

            return From(project: value[..index], environment: value[(index + 1)..]);
        }

        public override string ToString() => $"{Project}.{Environment}";

        // StackName voidaan muuttaa merkkijonoksi ja merkkijonosyötteeksi
        public static implicit operator string(StackName name) => name.ToString();
        public static implicit operator Pulumi.Input<string>(StackName name) => name.
            ToString();
    }
}
```

Ohjelma C.3. *Pinon nimenä käytetty StackName-tietoluokka*

LIITE D: ESIMERKKISOVELLUKSEN INFRASTRUKTUURIN YKSIKKÖTESTEJÄ

Tässä liitteessä esitellään esimerkkisovelluksen infrastruktuurin yksikkötestauksessa käytettyjä ohjelmia.

```
1  public class AzureClusterStackTests : StackTest<AzureCluster>
2  {
3      [Fact]
4      public async Task ResourceGroup_Is_Created()
5      {
6          // Suoritetaan testi. Paluuarvona saadaan resurssit
7          var resources = await RunTestAsync();
8
9          var resourceGroups = resources.OfType<ResourceGroup>().ToList();
10         resourceGroups.Should().HaveCount(1);
11     }
12
13     [Fact]
14     public async Task ResourceGroup_Has_Environment_Tag()
15     {
16         var resources = await RunTestAsync();
17
18         var resourceGroup = resources.OfType<ResourceGroup>().Single();
19
20         var tags = await resourceGroup.Tags.GetValueAsync();
21         tags.Should().NotBeNull();
22         tags.Should().ContainKey("Environment");
23     }
24 }
```

Ohjelma D.1. *Resurssiryhmään liittyviä yksikkötestejä*

```
public class LoadBalancerStackTests : StackTest<LoadBalancerStack>
{
    [Fact]
    public async Task Creates_Resources()
    {
        const string IpAddress = "192.0.0.1";
        const string TestKubeConfig = "KubeConfigTest";
        const string TestClusterName = "TestClusterName";
        const string TestDomain = "test.invalid";

        // Asetetaan testissä käytetty pinokohtainen konfiguraatio
        WithConfig(new()
        {
            { "eshop:DnsProvider", "digitalocean" },
            { "eshop:DomainName", TestDomain },
            { "eshop:ClusterProject", "test-cluster" },
            { "digitalocean:token", "token" },
        });

        var resources = await RunTestAsync(
            configureCall: (token, args, provider) =>
            {
                // Ne resurssit, joita halutaan simuloida, täytyy Helm-kaavioita
                // käytettäessä asettaa alla olevalla tavalla
                if (token == "kubernetes:helm:template")
                {
                    args.Add("result", new object[]
                    {
                        IngressService(IpAddress),
                    }.ToImmutableArray());
                }
            }
            , configureResources: (type, name, inputs, outputs) =>
            {
                // Pinoviittaukset edellyttävät että nimillä secretOutputNames ja outputs
                // on lisätty arvot outputs-muuttujaan
                if (type.Contains("StackReference"))
                {
                    outputs.Add("secretOutputNames", Array.Empty<string>().
                        ToImmutableArray());
                    outputs.Add("outputs", new Dictionary<string, object>()
                    {
                        { "KubeConfig", TestKubeConfig },
                        { "ClusterName", TestClusterName },
                    }.ToImmutableDictionary());
                }
            }
        );

        resources.Should().NotBeEmpty();

        // Etsitään pinoluokka ja tarkistetaan sen tulosteet
        var pino = resources.OfType<LoadBalancerStack>().Single();

        var loadBalancerEndpoint = await pino.LoadBalancerEndpoint.GetValueAsync();
        loadBalancerEndpoint.Should().Be(IpAddress);

        var kubeConfig = await pino.KubeConfig.GetValueAsync();
        kubeConfig.Should().Be(TestKubeConfig);

        var clusterName = await pino.ClusterName.GetValueAsync();
        clusterName.Should().Be(TestClusterName);

        var domainName = await pino.DomainName.GetValueAsync();
        domainName.Should().Be($"{clusterName}.{TestDomain}");
    }
}
```

Ohjelma D.2. *Kuormantasaajapinoon liittyvä yksikkötesti*

LIITE E: TERRAFORM-INFRASTRUKTUURIPINOESIMERKKI

Tässä liitteessä esitetään suorituskykyvertailussa käytetty Terraform-infrastruktuuripino, joka toiminnallisuuksiltaan vastaa liitteen A Pulumi-infrastruktuuripinoa. Ohjelmassa E.1 on esitetty Terraform-juurimoduuli, jossa luodaan resurssiryhmä ja varastotili Microsoft Azure -pilvialustalle. Ohjelmassa E.2 on esitetty Terraform-sisältömoduuli, joka luo verkkosivun sisällön ja tallentaa sen varastotiliin. Ohjelmassa E.3 on esitetty Terraform-moduulin palauttamat tulosteet.

```
terraform {
  required_version = "~> 0.14.10"
}

provider "azurerm" {
  features {}
}

# Muuttuja
variable "arvoesimerkki" {
  type = string
}

# Luodaan satunnaisluku, jota käytetään resurssien nimeämisessä
resource "random_integer" "satunnaisluku" {
  min     = 1111
  max     = 9999
}

resource "azurerm_resource_group" "resurssiryhma" {
  name     = "tfresurssiryhma${random_integer.satunnaisluku.result}"
  location = "North Europe"
}

resource "azurerm_storage_account" "varastotili" {
  name                     = "tfvarastotili${random_integer.satunnaisluku.result}"
  resource_group_name      = azurerm_resource_group.resurssiryhma.name
  location                 = azurerm_resource_group.resurssiryhma.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
  account_kind             = "StorageV2"
  enable_https_traffic_only = true

  static_website {
    index_document     = "index.html"
  }
}
```

Ohjelma E.1. *Terraform-juurimoduuli*

```
# Luodaan paikallinen muuttuja verkkosivun_sisalto ja asetetaan sen arvoksi
# verkkosivun sisältö.
locals {
  verkkosivun_sisalto = <<-EOT
<html>
    <body>
        <h1>Terraform: Esimerkkiverkkosivu</h1>
        <p>Varastotilin nimi: ${azurerm_storage_account.varastotili.name}</p>
        <p>arvoesimerkki: ${var.arvoesimerkki}</p>
    </body>
</html>
  EOT
}

resource "azurerm_storage_blob" "indexhtml" {
  name                   = "index.html"
  # Viitataan varastotilin nimeen, joka muodostuu suorituksen aikana
  storage_account_name   = azurerm_storage_account.varastotili.name
  storage_container_name = "$web"
  type                   = "Block"
  # Viitataan paikallisen muuttujan arvoon
  source_content         = local.verkkosivun_sisalto
  content_type           = "text/html"
}
```

Ohjelma E.2. *Terraform-verkkosivun sisältömoduuli*

```
# Pulumin tavoin myös Terraform tukee tulosteita.
# Tulosteet merkataan output-lohkoon, jonka nimeksi annetaan haluttu tulosteen nimi.
output "ResurssiryhmanNimi" {
  value = azurerm_resource_group.resurssiryhma.name
}

output "VarastonAvain" {
  value = azurerm_storage_account.varastotili.primary_access_key
  # Varaston avain on salainen
  sensitive = true
}

output "VerkkosivunOsoite" {
  value = azurerm_storage_account.varastotili.primary_web_endpoint
}
```

Ohjelma E.3. *Terraform-tulosteet*

LIITE F: SUORITUSKYKYMITTAUKSEN TULOKSET

Tässä liitteessä on esitetty luvussa 7.4 tehdyn suorituskykymittauksen yksittäisten toistojen mitatut arvot testitapauksittain. Mittaustulokset on jaettu kahteen taulukkoon F.1 ja F.2. Infrastruktuurin pystyttämiseen kulunut aika on ilmoitettu sekunneissa.

Taulukko F.1. *Suorituskykymittausten toistojen 1–30 tulokset*

Testitapaus	**Kesto (s)**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Testitapaus 1	38,83	37,54	38,11	38,24	38,25	40,87	39,95	38,11	39,07	39,14
Testitapaus 2	34,08	33,47	34,59	34,57	33,14	33,43	35,61	34,13	33,15	33,35
Testitapaus 3	31,48	31,18	29,46	31,93	31,03	33,73	35,09	32,38	31,97	32,57
Testitapaus 4	28,09	25,50	27,28	26,12	27,83	28,31	25,69	28,42	25,98	27,67
Testitapaus 5	42,68	42,63	44,90	45,12	42,31	43,06	44,11	41,99	44,11	44,66
Testitapaus 6	35,22	35,83	35,15	34,41	34,76	34,89	36,57	36,79	35,19	34,67
Testitapaus	**Kesto (s)**									
	11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Testitapaus 1	44,07	45,41	46,26	45,11	44,86	37,23	39,05	38,39	38,57	38,56
Testitapaus 2	36,98	36,94	38,22	36,43	37,66	34,36	34,20	34,05	33,50	33,43
Testitapaus 3	39,15	39,86	38,81	38,26	39,50	32,01	29,05	31,10	31,47	29,31
Testitapaus 4	31,01	27,88	31,42	30,51	29,31	25,62	27,44	25,84	27,12	27,98
Testitapaus 5	49,02	49,64	49,17	50,55	49,66	43,09	45,16	43,75	45,46	44,45
Testitapaus 6	40,45	39,47	39,12	39,75	39,24	34,87	34,92	34,43	35,10	35,52
Testitapaus	**Kesto (s)**									
	21	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Testitapaus 1	37,84	38,62	38,20	37,43	38,75	44,81	44,00	44,35	42,90	44,79
Testitapaus 2	33,84	33,32	34,12	32,88	33,84	36,75	39,74	37,36	35,29	36,92
Testitapaus 3	32,23	31,25	32,31	31,79	32,19	37,57	37,84	38,25	36,41	37,74
Testitapaus 4	27,63	25,78	27,87	25,03	28,25	28,64	30,92	30,22	28,63	31,23
Testitapaus 5	42,90	44,30	43,14	43,17	43,47	49,33	52,30	51,91	50,79	52,33
Testitapaus 6	35,70	36,35	38,71	42,08	39,63	38,65	38,36	38,42	38,75	39,00

Taulukko F.2. *Suorituskykymittausten toistojen 31–60 tulokset*

Testitapaus	**Kesto (s)**									
	31	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
Testitapaus 1	46,99	44,54	43,97	45,12	45,22	46,30	46,86	43,71	43,85	45,38
Testitapaus 2	38,73	38,63	36,79	36,25	36,85	36,48	36,57	36,47	36,15	37,23
Testitapaus 3	40,42	39,62	40,48	40,43	42,47	38,70	39,24	38,48	37,89	38,32
Testitapaus 4	31,29	28,94	33,33	30,82	30,36	29,74	30,07	28,36	31,14	30,86
Testitapaus 5	55,12	53,36	51,85	52,32	52,13	51,07	51,49	49,44	51,46	51,10
Testitapaus 6	38,26	38,25	38,09	38,51	38,73	37,89	38,66	38,68	39,94	39,62
Testitapaus	**Kesto (s)**									
	41	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**
Testitapaus 1	38,16	37,47	38,35	37,81	37,11	43,60	44,99	43,91	43,82	44,04
Testitapaus 2	35,77	32,29	33,92	34,22	34,31	36,39	35,40	36,34	36,32	38,45
Testitapaus 3	32,72	33,31	34,43	31,38	32,20	38,56	37,78	41,90	39,72	38,44
Testitapaus 4	28,12	25,98	28,64	26,87	28,13	30,87	28,51	30,31	31,12	28,09
Testitapaus 5	43,32	43,44	43,98	44,23	45,10	51,68	49,98	48,59	48,94	49,20
Testitapaus 6	34,93	35,57	34,03	35,69	35,90	38,09	39,85	38,26	39,54	38,83
Testitapaus	**Kesto (s)**									
	51	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
Testitapaus 1	44,47	44,71	44,78	43,80	45,36	39,06	39,53	37,87	39,26	37,37
Testitapaus 2	35,98	36,79	36,60	36,66	37,85	33,18	33,26	33,54	33,97	33,21
Testitapaus 3	41,94	41,18	42,50	42,94	41,53	32,12	29,22	31,99	32,33	32,08
Testitapaus 4	31,23	30,10	31,21	31,60	29,36	28,09	25,76	27,76	25,50	27,96
Testitapaus 5	49,82	51,57	48,02	48,57	48,28	42,28	42,98	43,23	44,78	43,97
Testitapaus 6	38,96	38,63	38,81	39,36	40,26	34,87	35,99	34,72	34,34	34,70