jamk

Adoption of Infrastructure as Code (IaC) in Real World

Lessons and practices from industry

Olga Murphy

Master's thesis
December 2022
Information Technology
Full-Stack Degree Programme

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Description

Murphy Olga

Adoption of Infrastructure as Code (IaC) in Real World; Lessons and practices from industry

Jyväskylä: JAMK University of Applied Sciences, December 8th, 2022, 61 pages

Information Technology, Full-Stack development, Master's Thesis

Permission for web publication: Yes

Language of publication: English

Abstract

With increased demand for speed and software projects, teams increasingly need to automate tasks. For infrastructure that resides in the cloud, IaC is the way to automate by bringing scalability, speed, and transparency to manual tasks. As a recently emerging technology, there is a shortage of experts to meet demand.
To understand the challenges of IaC adoption, this study was composed of a literature review and a survey. The survey was designed to gather professionals' opinions on IaC usage and learning processes in their teams and projects.
The results show that IaC is not an overly difficult technology to learn, but it requires investment of time and resources due to its complexity. Although the community around IaC is growing, advanced cases lack support in official documentation. To find answers to trivial problems, developers refer to their peers and grey literature. Contrary to previous studies, developers named AWS CDK and Terraform as their most used tools. Overall, experts evaluated usage of IaC highly. Despite the challenges, all respondents, including non-users, could see benefits to employing the practice in their work. However, this is just an initial insight into what is required to support IaC adoption in companies and promote its development.

Keywords/tags (subjects)

Infrastructure as Code, IaC, configuration as code, DevOps, automation, cloud computing, technology adoption, survey

.

Contents

Figures

Tables

1 Introduction

Over the past few decades, software development has progressed rapidly. Software development processes became more dynamic. Software projects are expected to be delivered quicker, which increases the demand for applications to be deployed in a continuous manner (Hummer et al., 2013; Artac et al., 2017; Guerriero et al., 2019). Cloud and automation tools are candidates that could be of use to keep up with demands of this ever-growing digital world. Gartner identified "Optimized technologist delivery" as one of the main hype cycle themes of 2022 from more than 2000 emerging technologies. Major topics of the theme included Cloud Sustainability and Industry Cloud Platforms being led by Cloud Data Ecosystems (Gartner, 2022). They also made a prediction that usage of structured infrastructure automation is going to leap to 70% of organisations in 2025 from the 20% that was recorded in 2021. At the same time such a rapid growth requires specific sets of skills and roles to be adapted by the industry. Another Gartner prediction is that such skills in 2021 were present in less than 10% of enterprise but will grow only to 50% by 2025 (Gartner, 2022). HashiCorp (HashiCorp, 2022) found that "one in five companies (22%) are still struggling to gather the skills needed to staff a platform team".

The recent transformation of software practices has been described by Morris as *"moving from Iron Age to Cloud Age"*. From a practical point, the change brings virtualised resources in place of physical hardware and shifts processes to automated from manual. Cloud technologies help developers to learn how to adopt the change and improve continuously. As one of these cloud technologies, Infrastructure as Code (IaC) is an approach that allows provisioning and management of infrastructure programmatically in an automated manner. It promotes application of software development practices to the infrastructure that are applied to the code. When it first emerged, the main task of IaC was to configure servers with Perl and bash scripts. Now IaC is widely used to not only manage the servers but to manage full clusters, networks, and other types of resources. Microsoft emphasises that IaC uses DevOps methodologies and strategies to support resource creation automation (Microsoft, 2022). IaC can be viewed as a supporting practice to help DevOps improve collaboration and bring teams closer together (Morris, 2020). Not only companies are adopting IaC, but academics have also been incorporating both DevOps and IaC practices. Although bringing difficulties in the learning phase, it has increased experiments' repeatability and systems' effectiveness (De Bayser et al., 2015).

IaC is a relatively new trend and its practitioners have only recently started to gain the expertise to develop it (Guerriero et al., 2019). Bringing IaC into use requires effort in learning the practise and even switching mindset about risks and change (Morris, 2020). Adoption of new technologies often happens through the experience of other practitioners in the industry (Moore, 1999). Due to the novelty of IaC, there is a lack of experienced practitioners in industry, which slows its uptake to a degree. Lack of empirical research and practical information leaves companies unaware and unwilling to try new practices when there is no immediate gain in it.

This thesis focuses on finding background information from existing literature on how IaC started to emerge and what are the main benefits it brings to development teams. This study obtains details about IaC adoption by contacting companies, that either use cloud services or work with infrastructure automation, by means of survey. The rest of the thesis is structured as follows. Section 2 presents goals and methodology; Section 3 covers existing extensive literature about IaC; Section 4 combines knowledge about adoption of new technologies with a focus on IT teams; Section 5 describes survey implementation; Section 6 presents the results of the study; Section 7 contains discussion and limitations; Section 8 draws the conclusions and sets avenue for further potential research.

2 Goals and Methods

Setting infrastructure in the cloud takes time, practice and often is not straight forward to complete. Interfaces of cloud providers can be confusing to navigate due to the plethora of configurations. Repeating the same set of configurations is not an easy task either. Automating these tasks is a way to improve stability, repeatability, and knowledge about a cloud environment. Bringing those scripts under the roof of version control and structured programming language model, reshapes the practice to become Infrastructure as Code.

2.1 IaC application

The value of adopting IaC may not be apparent to stakeholders especially the ones that do not work with infrastructure (Morris, 2020). According to Morris (2020) the practice of IaC is often thought of as chaotic and unnecessary. There is no need in automating something that is going to

be built only once. However, even the best IT systems tend to get updates or become broken. Figure 1 presents an example how cloud resources are deployed with manual processes. A developer by means of UI or CLI sporadically creates, updates, or deletes resources. This process could happen a multitude of times by more than one developer. As a result, rarely anyone in the team would have a clear understanding about the state of the resources.

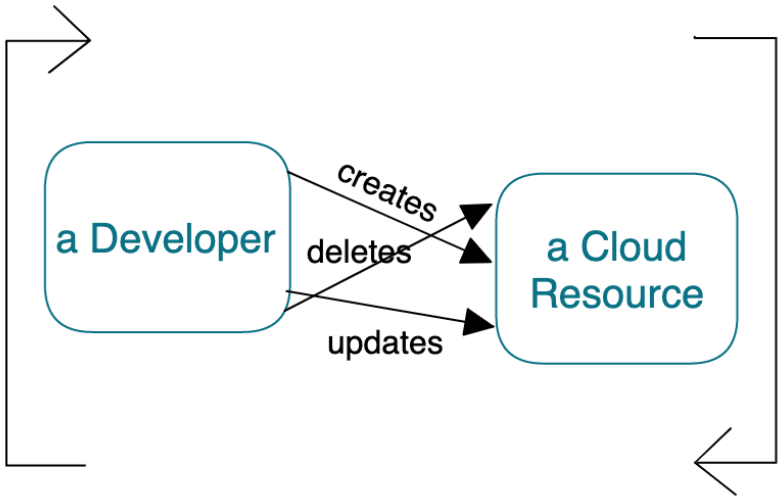

Figure 1 Manual resource creation example

In the long-term, IaC makes it easier to add repeatability to the infrastructure and understand future changes as every change follows the same deployment process. On Figure 2 an example flow chart shows how IaC could be utilised, is presented. The infrastructure gets documented, tested, and deployed automatically and it will always be pushed to version control and have a visible history of deployment history. When infrastructure code is ready, the resources can be re-created again with a click of a button in case of failure, update, or breaking change. Defining infrastructure in an automated manner removes the need for routine tasks and save developers' time to focus on improving development systems.

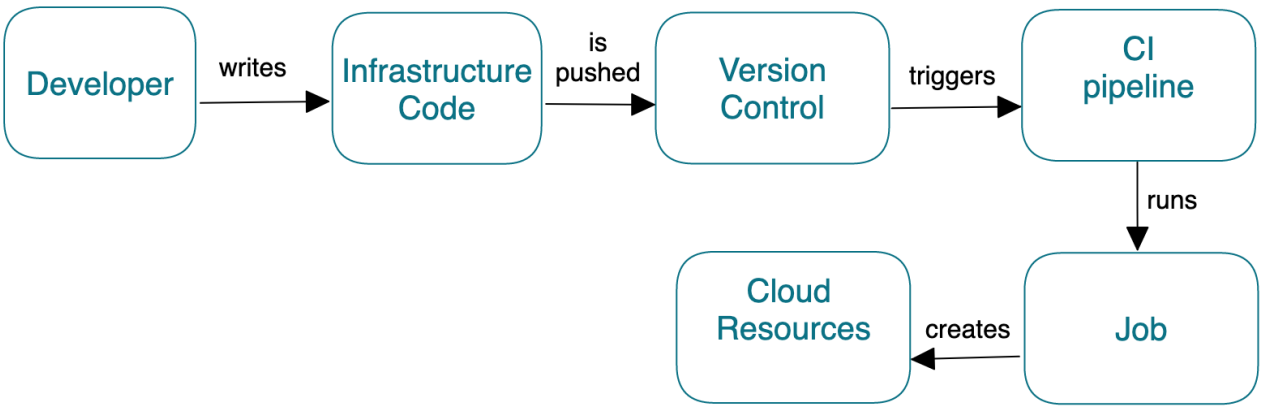

Figure 2 Resource creation flow example with IaC

Adoption of IaC is less likely to happened in old fashioned or monolithic oriented teams. In these teams, automation is not a concern, and they tend to think that infrastructure should be handled by someone else. IaC need arises in teams with certain software development practices. One of the main characteristic factors is the use of cloud computing to host applications and infrastructure. As resources' numbers starts to grow, it becomes very difficult to keep track of the state of the environment. IaC can prove itself useful and time saving to manage those resources. Teams might also be applying Agile or DevOps and find IaC to be the way to make infrastructure provisioning easier. When starting with IaC teams should be aware about version control and code review practices as both are important for knowing about the changes that are happening to the environment. To provide a full automation capability with IaC it also should be integrated with CI/CD pipelines. IaC aids developers to continuously learn and improve systems as developers learn to be afraid of breaking the infrastructure.

There is no best scenario for how to use IaC tools within a project as every project is unique. The IaC community is still in a process of learning and discovering better practices. In each case IaC should be tailored to a project's needs to fulfil its potential. IaC can be applied in projects in multiple ways: from moving a part of existing infrastructure to the cloud, to building an application based on a microservice architecture and other cloud offerings. Developers might as well start small and only deploy single resources in an automated way. It can also be beneficial to have the infrastructure code grow together with the application code.

Supporting a multi-environment and multi-tenant setup is another case where IaC provides most benefits. Instead of spending days to recreate the same environment manually, adding a new customer can be done by adding a new parameters file and setting up an additional deployment task. A test environment can also be created and destroyed on demand to reduce costs of running resources that are used only occasionally.

Starting to write infrastructure in a form of code can be challenging especially if previously GUI was used to perform these tasks. There is a long list of decisions that developers should make prior to setting up IaC. They need to decide which tool to pick from a wide array of different tools. IaC code resides in version control and developers need to plan the location and structure of their code. Every environment needs to be configured to properly store parameters and secrets. Before the code is executed it should be tested and after it is running be updated and follow upkeep rules. As developers create infrastructure, they can decide to define some resources as modules that can be used to speed up creation of future resources also in subsequent projects.

For consulting companies that operate by projects and create resources into their customer's environment, IaC can bring both advantages and problems. On the one side it allows repeatability and consistency for the infrastructure, conversely it generates difficulties related to permissions and compliment with customers' security policies. Some customers are not convinced by automated resource provisioning as it can generate additional costs. Often there is no professional on their side who is experienced enough to approve the code. For some customers it might also get difficult to explain the overhead cost that the initial IaC work requires.

IaC sometimes gets mixed with Infrastructure as a Service (IaaS). As it is defined in this thesis IaC is a way to automate resources provisioning. However, IaaS is a service that provides virtualized computing solutions through cloud providers. With IaaS companies can shift their on-premises resources to the cloud as a part of one single cloud offering. Therefore, these two concepts are different but can work side by side: one provides computing resources, while the other is a tool to automate infrastructure creation and management. Another effective pairing for IaC is Kubernetes, a container orchestration tool, as it provides a better management of a Kubernetes cluster. Thus, IaC can be utilised in pairing with other modern tools to create a vast list of resources, including virtual machines, network resources, data resources, applications, clusters, and containers.

2.2 Research Goal

Although IaC is growing a popularity among professionals, little academic literature exists that is focused on the practice. For the past several years academics have been noting that the IaC field deserves more attention (Rahman et al., 2019; Palma et al., 2020; Kumara et al., 2021). According to data from Google Trends interest in the topic has been growing steadily. Figure 3 present graph compiled data from Google Trends (2022) data for the topic "Infrastructure as Code" worldwide over the past 5 years.

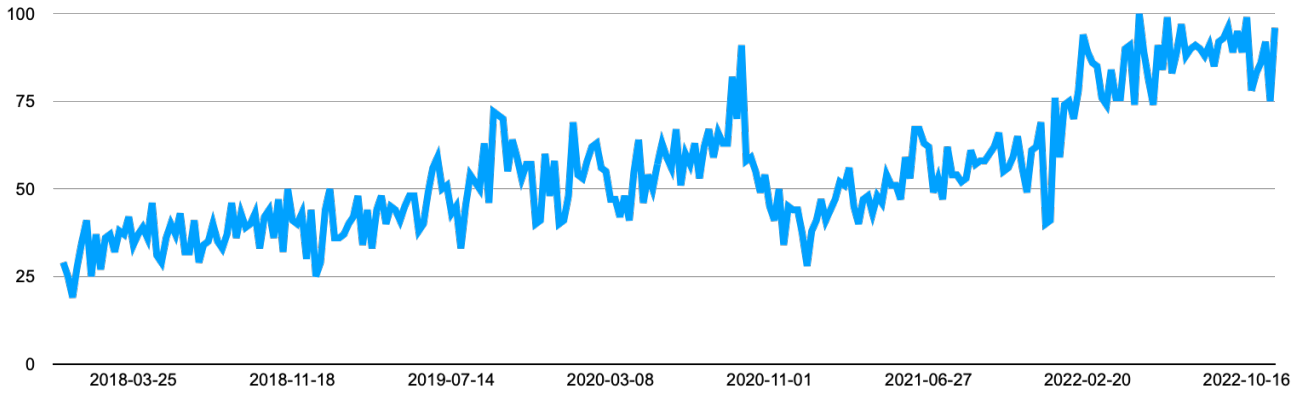

Figure 3 Google trends data

Previous research in the context of IaC has been done with focuses on (a) security (Rahman et al., 2019; Rahman et al., 2021); (b) grey literature review (Kumara et al., 2021); (c) best practices (Chen et al., 2018; Rahman & Sharma, 2022); and (d) tools evaluation (Palma et al., 2020). Lwakatare et al. (2019) conducted a case study of DevOps in multiple companies. They have observed the DevOps practices and collected feedback on how each case identified the goals and benefits of DevOps adoption. They also identified IaC practices separately from DevOps context. The authors discussed the benefits of IaC while noting that in some cases scripts have been provisioned by the operations teams. The same cases described that developers did not have much knowledge of IaC but that it could be acquired in the future from the operations team.

Guerriero et al. (2019) conducted an empirical study investigating adoption, support, and challenges of IaC. As their study was mainly focused on experienced developers of IaC, the researchers identified not taking novice adoptees as a limitation and a potential avenue for either replication study or future direction of research.

Rahman et al. (2020) conducted a study to find anti-patterns in IaC scripts' development. In the survey the authors identified that applying as code practices to infrastructure scripts, such as collaboration and code review are crucial for writing a better-quality infrastructure code. They also identified the lack of version control usage, inadequate logging and testing as potential anti-patterns that should be investigated further.

Rahman et al. (2019) completed a systematic mapping study of IaC and concluded that IaC as a trend is growing but is still under-researched. They identified gaps in the existing literature connected to IaC. The authors recommend multiple directions for the researchers that are investigating the IaC subject. They emphasise a lack of empirical studies, defect analysis, security, anti-patterns, and knowledge and training.

IaC supports DevOps processes by simplifying resource provisioning. While DevOps is mostly about organisational culture, IaC is focused on the tools and how those tools are used by practitioners within teams and projects. As various tools are emerging quickly it takes efforts to learn them. While there is a vast number of sources and grey literature available, there is no de facto standard on how IaC should be implemented. This is due to multiple reasons: tools are diverse, use cases for IaC are broad and often are not well defined.

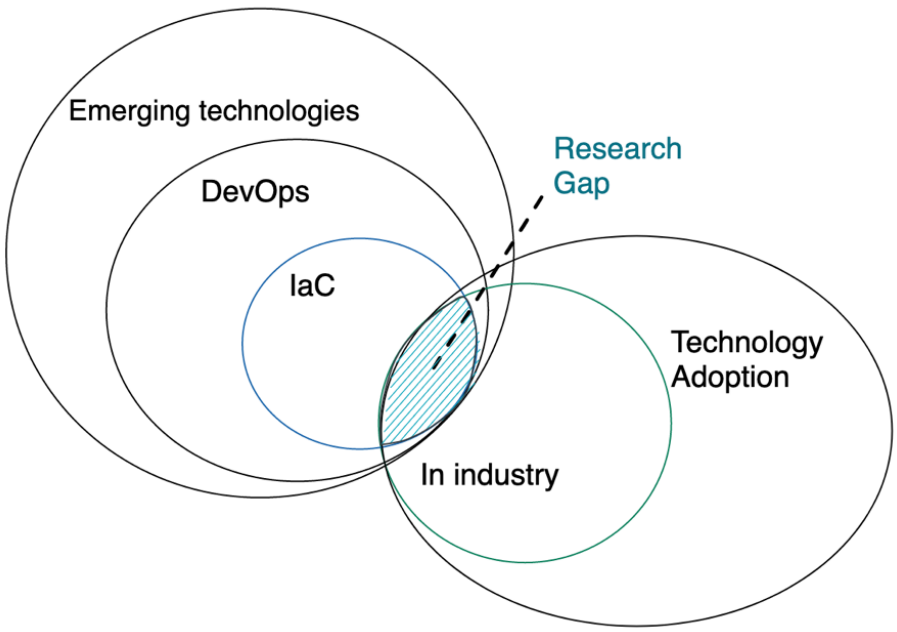

Figure 4 IaC adoption research gap

This study's research focuses on the state of adoption of IaC in an industry setting. Like other emerging technologies adoption of IaC can be challenging and is under-researched. Figure 4 presents the research gap that this thesis is aiming to examine. Over the course of the research developers are asked about their team's tools and organisational practices. A key focus of this study is to gain insight into developers' learning experience with IaC. It also explores how satisfied developers are with the technology usage and the state of adoption of IaC. A literature review about technology adoption, background of IaC and a selection of tools are presented here to support the understanding of the technology itself and challenges that practitioners often face when dealing with emerging technologies.

The goal of this thesis is to aid developers and teams to understand the IaC trend and help its adoption techniques. These research questions were:

RQ1. How easy it is for the companies to shift to IaC and why?

RQ2. What are the driving factors for a team/company to start applying IaC?

RQ3. How do developers start learning IaC?

2.3 Methods Selection

To answer these questions survey-based research was chosen as a primary approach for data collection. In this research, practitioners' knowledge and experience are the main sources of information. To obtain more insights from the available data mixed methods research with embedded design was implemented by adding complementary qualitative questions.

Due to the lack of empirical research in the field of IaC it is difficult to find previous studies to support methods selection. In those few available studies, the research methods are often unclear. For this reason, Rahman et al. (2019) recommends defining data collection and analysis more clearly for future research. Therefore, in this thesis methods are specified in detail to contribute to the research validity and allow replication. Due to the complexity and subjectivity of developers' experiences, mixed methods were chosen to provide the means to collect a more diverse dataset.

Mixed methods research incorporates parts of both quantitative and qualitative research methods (Creswell & Plano Clark, 2017). The quantitative approach aims to collect numerical and precise data that can be qualified and analysed statistically. Findings from quantitative research are unbiased and often predictable (Williams, 2007). Qualitative research provides less structured data with more details. This approach is used for describing the phenomenon and can be used to generate a hypothesis (Williams, 2007).

Mixed methods design is applicable when both types of data have meaningful connections between each other (Williams, 2007). Adding another data type in such ways helps to improve generalisability, contextualisation, and credibility (George, 2022). Creswell and Plano Clark (2017) describe multiple ways how mixed research could be implemented. Triangulation design is used to obtain qualitative and quantitative data concurrently. Collecting and analysing both quantitative and qualitative data simultaneously creates embedded design, while if the results are analysed separately design is convergent parallel. If quantitative data was collected after qualitative it follows an exploratory sequential approach, otherwise it would be explanatory sequential. In all the cases, both types of data are collected to enrich the findings.

To obtain opinions from software developers about technologies' state of adoption, various qualitative studies have been conducted. Surveys, interviews, and case studies are widely used in the domain of software engineering in numerous research activities. Nuottila et al. (2016) used a case study with semi structured interviews to identify challenges of adopting Agile in a public organisation. Riungu et al. (2016) chose case studies with supporting interviews to gather data about DevOps adoption from three software development organisations. Lwakatare et al. (2019) completed a multiple case-study of five companies to explore implementation of DevOps in practice. Guerriero at al. (2019) completed semi-structured interviews with 44 developers to study expert practitioners' opinions of IaC about adoption, challenges, and support of the practice. Bosu et al. (2019) conducted a survey to gather insight about practices of blockchain software developers. Rahman et al. (2020) used survey as a part of their methods to obtain developers opinion about the challenges of IaC. Private technology companies commission surveys on regular basis to identify their market needs and observe the state of technology adoption. HashiCorp commissioned a survey from Forrester Inc. to understand the needs of multi-cloud operating model (HashiCorp, 2022).

While conducting interviews, both Guerrero et al. (2019) and Nuotilla et al. (2016) had at least two researchers participating in the interviews. Additionally, they split the interviews transcribing and coding to improve validity of data. Rather than taking the interview as the main method, choosing a survey-based approach in this thesis takes any chance of the researcher's biased opinion away from the data collection process. The researcher is an active adoptee of IaC and a developer in a consulting company. A survey was favoured over other methods to avoid a potential conflict of interest while trying to conduct full interviews with other developers.

Surveys in research are used as instruments to collect and analyse information by questioning individuals who represent the population or are the whole population (also called census) (Pickard, 2017). The purpose of a survey is to study relationships between specific variables that are identified in the research hypothesis. When conducted well, survey results can represent opinions over the generalised population (Pickard, 2017). However, questions' design, proper sampling and equitable data analysis hold considerable restrictions to such generalisation. Surveys can be descriptive, explanatory, or a combination of both. The main goal of a descriptive survey is to describe the phenomenon as precisely as possible, while an explanatory survey is focused on finding the cause and effect of the phenomenon. In this thesis, survey-based research was identified as the most suitable instrument for gathering data as it provides a wider access to IaC adoptees in a limited timeframe. The selected method allowed us to gather responses from multiple developers per team, including novice adoptees, to identify challenges that developers face while starting to learn IaC. While this research is aimed at both novice and advance practitioners of DevOps and IaC, practitioners do not fall into discrete categories labelled as such. Instead, they fall onto a continuous spectrum of experience and ability. It was not preferential to target only very experienced practitioners but to explore the state of adoption of IaC across the industry. The survey's structure also allows data collection from developers that are only getting familiar with IaC via trainings or side projects. Throughout the survey, insight into the field can be found such as gaps in the existing knowledge base and practices. Or in this case, a lack of static adoption techniques. It allows insight into companies' learning cultures and IaC adoption on a wider scale while targeting a larger number of companies. Surveys provide directly comparable results as each participant answers the same questions, particularly semantic differential type of questions. Such structured data is the basis for true statistical analysis.

Thus, survey-based research provides structure, flexibility, confidentiality, and fits to the time constraints set by a master's thesis. In this research, a survey is the primary method for data collection, and it includes both quantitative and open-ended questions.

The extensive literature review was completed as a part this thesis to provide the student with the most recent knowledge of IaC. The list of citations includes most recent publications that are related to the topic of IaC. Research papers and books referenced in this thesis were found from Google Scholar, ACM Digital Library, Janet Online Library, and Knovel Online Library. Similar queries were used in all the mentioned databases that resulted in wide array of queries, which included following terms: "infrastructure as code", "IaC", "IaC adoption", "infrastructure automation", "IaC practices", "IaC challenges", "DevOps", "Agile practices", "terraform", "Bicep", "CloudFormation", "cloud adoption".

3 Background of Infrastructure as Code

Infrastructure as Code technology is a new and dynamic automation technique that organically appeared from the emergence of cloud adoption, need for rapid software change and IT teams' cooperation. This chapter includes literature review related to the history of IaC, its benefits, tools, and challenges.

3.1 The beginning of Infrastructure as Code

DevOps and cloud-native architecture are two paradigms, which led to IaC. The rise of cloud computing and cloud-native application started to pick up in 2014 (Labouardy, 2021) and have created a need to find new ways for resource provisioning. Cloud providers such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure are driving the change in cloud infrastructure. Cloud Native Foundation was established in 2015 to help advance container technology as well supporting the shift to the cloud world. Managing IT infrastructure used to mean hardware handling, setting up servers and connecting router wires. Cloud computing has changed that by providing the developers with the necessary resources in one place. In a cloud environment computing resources are allocated by cloud providers who are taking ownership over underlying hardware resources.

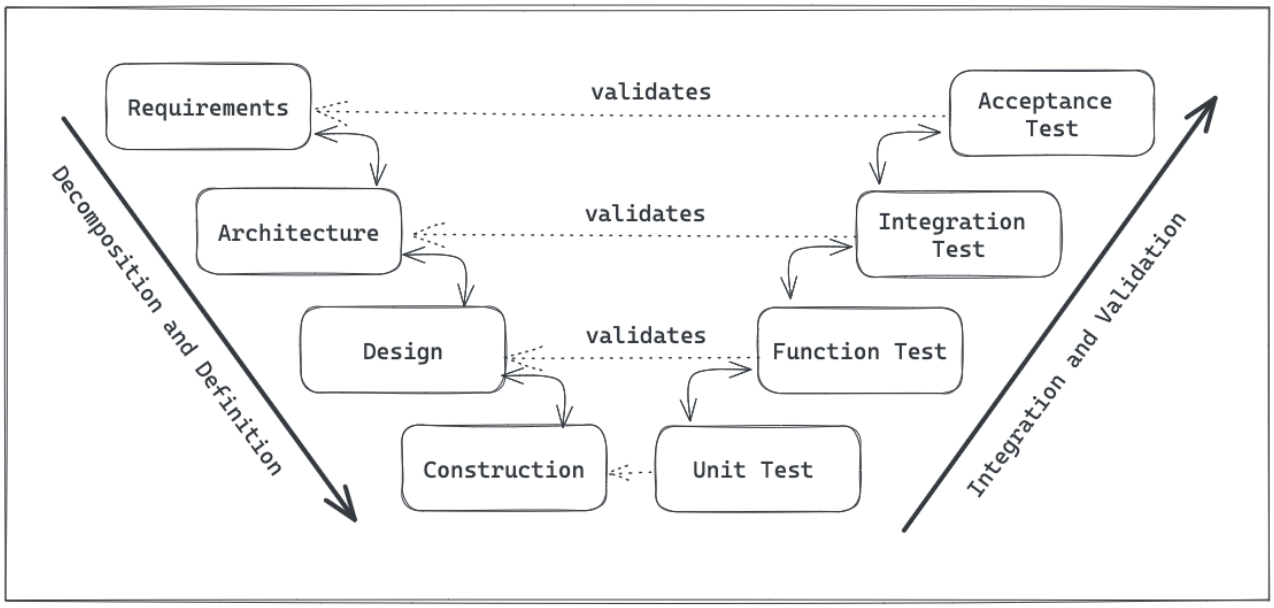

Figure 5 Traditional software development lifecycle

On the organisation level, software development practices have been shifting equally quickly. Traditional software development cycle models such as Waterfall and V-shape (Figure 5), although successful in some cases, are lengthy and bulky (Ambler & Lines, 2020). The release process extends delivery time even further - after the development is completed, code is handed to the operations team to create the environment on which to run it. Usually, developers have little knowledge about their code's impact on infrastructure (De Bayser et al., 2015). Operations side had to specify the environment, fight for server space and credentials, therefore being in most cases much slower to install the change than developers could or would like to (Berg et al., 2021).

Agile came as a rejection of traditional software development practices. It emerged in a form of a manifesto in 2001 (Berg et al., 2021). For the teams that adopted it, advantages such as increased quality, adaptability, and better communication were promised. Agile has a controversial history regarding its adoption. It was noticed that the first agile principals were created by a very homogenous group of people, who were very far from a typical team of software developers or practices that they require for success. After harsh criticism Agile did not just finish, it evolved together with the teams that adopted it. Agile 2 addresses the concerns from its predecessor and improves itself. In the new version, the set of practices is built on collaboration, people empowerment and continuous delivery. Supporting these is one of Agile 2's principals**:** Integrate early and often (Berg et al., 2021). In Agile deployment practices, shift left continues throughout the whole project to allow more automation and to support continuous delivery.

During adoption, continuous integration and continuous delivery are complementing each other's practices and are often taken in together. Continuous Integration's set of practices include using a centralised code repository and handling the changes through pipelines. After the code is committed, CI launches automated tests which improve the quality of the code and can signal if something is not going right. Paired up with code review, CI helps to foster discussion within the team and share the knowledge. CI practices promote keeping the changes separately on small branches and integrating them into master (Labouardy, 2021). Continues Deployment/Delivery (CD) handles the other side of the process – all the changes that pass required stages are automatically deployed to production (Rahman et al., 2015). CI and CD are software engineering practices that focus on delivering production-ready changes quickly (Rahman et al., 2015) and are now required to be implemented in a cloud-native way to successfully run a cloud-native application (Labouardy, 2021).

CD methods and Agile ideas led to the beginning of DevOps (Berg et al., 2021). DevOps emerged from a talk at the Agile 2008 conference (Morris, 2020) focusing on improving culture of development (Dev) and operations (Ops) teams and removing the barriers between those teams. In the traditional software development team, the main goal of developers is to push as many features as possible, while for the operations it is to support as stable an environment as possible. Such different sets of goals have created friction between the teams. To avoid that, DevOps provides guidelines for developers and operators to use the same toolset, make developers more production aware, apply shorter release cycles, promote automation tools, and create environments that are more reproducible (De Bayser et al., 2015). Like Agile, DevOps appeared to break the norms of software development processes. Occasionally DevOps is even identified as part of the Agile movement (Berg et al., 2021). DevOps requires changes to organisation structure and roles (Lwakatare et al., 2019). While IT operations are an important part of the Software Lifecycle, DevOps is proving to be a successful way to simplify development and release processes by integrating planning, development, delivery, and operations as one continuous application lifecycle (Microsoft, 2022). If used adequately, DevOps enables scalability to build more complex products, technical agility, costs-optimisation, and enhanced monitoring that help to find and fix problems. DevOps allows teams to create more rapid deployments while reducing risks of faulty pipelines. Berg et al. (2021) describes it as a "dial" to turn up for higher quality. In practice what happens is that as testing environments get easier to set up, more tests can be created and tested, therefore reducing risks to deploy faulty code as all changes pass through automated tests. Less faulty code

creates a more stable and reliable product. The case study of Lwakatare et al. (2019) found that DevOps adoption in one team improved organisational culture and mindset across the other teams within the organisation.

While the DevOps community focused on applying the above listed techniques to their development processes, they have also started to accumulate and share knowledge about coding practices and applying them to infrastructure, therefore incorporating Infrastructure as Code (Artac et al., 2017). Morris (2020) identified the application of code practices to infrastructure as one of the important steps in implementing IaC. These principals include (a) version control usage, (b) ways to reuse code, (c) task automation, (d) code review, and (e) naming conventions. Morris has found the first use of the term "Infrastructure as Code" to be used by A. Clay-Shafer and John Willis in 2009 from the talk "Agile Infrastructure" and its summary. However, the techniques came much earlier than the term itself - scripts have always been used by system administrators to manage systems. The way in which DevOps focused on getting the development practices closer to operations and vice-versa, it created the space for improving and developing those techniques jointly. Infrastructure can and should be created with no manual configuration but by specifying rules how that infrastructure should function. Those rules can be created for setting up new systems or modifying the old ones – all while collaborating, keeping the change history logs, and making sure systems are steady (Spinellis, 2012). With adoption of IaC, another layer is added to the automation concept - CI pipeline is enriched with the code that enables provisioning of the whole infrastructure automatically. Palma et al. (2020) describes IaC as practice to implement Continuous Integration and Artac et al. (2017) even refer to IaC as a key practice of automation. Wurster et al. (2020) have identified infrastructure as code as one of the main search phrases to find sources that identify automation technologies.

3.2 Benefits using IaC

Moving to cloud technologies simplifies computing and data resources provisioning processes – instead of acquiring or renting them, it is easy to set infrastructure up in the cloud as it could be ready to use after only a few minutes. Setting up data centres used to be available only to big companies but even a start-up can now provision every environment it needs, including a database management system, virtual machine, or serverless computing resources at a reasonably low

cost. However, many cloud environments are often provisioned by manual efforts. It is accomplished by writing commands to CLI or going to the cloud self-serving portal (Ambler & Lines, 2020). By adopting IaC those manual tasks can be automated. Many common problems that happen while managing infrastructure by hand such as inconsistencies and human errors can be resolved using IaC.

When infrastructure is deployed manually it frequently falls into a configuration drift (Lwakatare et al., 2019). Avoiding configuration drifts is one of the driving factors to use IaC (Been et al., 2022). Configuration drifts could happen for whole environments or for resources that should be otherwise identical between environments. This configuration misalignment is more prone to happen due to incomplete, unexpected, or incorrectly executed change to an infrastructure. In a fictional infrastructure, two identical servers were provisioned manually with scripts to host two application instances in two different environments. In the environment A an issue was found that was fixed by manually changing configurations but due to the high load of work the same change has not been carried over to environment B, so it was forgotten. After a few months the load to environment B had picked up and the same issue started to appear there as well. As the fix was left undocumented, it again took time to discover the reason for application failure. This is just one small example on how configuration drift can reduce stability and slow down deployments, but usually industry cases are much more complicated. Some might include intricate network systems or 20 microservices that must be stable enough to ensure an application's complete functionality.

A deployment process with IaC in use is different. An example of automated infrastructure provisioning process can be described as follows: the software developer creates infrastructure scripts and pushes them to version control (Git), at the same time the developer creates CI pipeline (using any of the CI tools) and pushes it to the repository. After the pipeline is configured correctly, every change to the infrastructure that is pushed to the same repository will trigger the pipeline to run all the configured tasks, which could include validations, tests, and preview of resources to deploy. Depending on how the pipeline is configured it will also deploy the changes to the specified environment automatically or after developer's review.

Morris (2020) defined three core practices to successfully implement IaC: define everything as code, continuously test and deliver and build small parts. Support for all three practices is widely

present across other academic literature sources (Guerriero et al., 2019; Ambler & Lines, 2020; Kumara et al., 2021).

Major benefits for defining everything as code includes reusability, consistency, and transparency. When the code resides on the local machine of a single person, they become the keeper of the code. When someone else in the team needs to change the code, it could take a long time to understand how the system is built and perform debugging. Sharing the code saves time and increases team awareness about the infrastructure by reducing the knowledge gaps about the system with mandatory code review practices. As all the infrastructure's code moves to version controls, the whole team takes ownership of that infrastructure. This process could also be thought of as a documenting the environment by using code to provide audit trails and specifications inside commit messages. When every resource or step of the infrastructure is declared in code that is committed to a version control, even more difficult infrastructure setups become subsequently easier to understand (De Bayser et al., 2015).

As developers create infrastructure code, they could (and should) split it into smaller pieces. Those simple pieces are easier to understand compared to a heavy, coupled system. Each piece is much easier to update, test and deploy independently. They also could be reused for future deployments. Thus, infrastructures which would require multiple modules, and intricate configurations would become easier to replicate. Lwakatare et al. (2019) found that complex infrastructures with more manual steps are prone to errors and require more testing and validation during the deployment process. When a similar system is required to be created again, it could take hours instead of days using IaC (De Bayser et al., 2015). Reducing complexity is one of the reasons IaC is proven to provide reusability for complex projects set by scientific experiments. The same argument can be applied to improving infrastructure reliability - breaking things into smaller parts makes it easier to find and fix problems as opposed to sorting through a much larger block of code that taeks the developer's time and attention to sort through.

Automation allows dynamic scaling and supports distributed architectures. Once an IaC module is written, it could be used to create as many instances of the resource as needed with minimum effort. In addition, the same automation script provides guaranteed outcomes as across each installation it results in the same outcome, providing consistency. Morris (2020) notes that stability

comes from making changes. When the system is automated enough for developers to gain trust in each deployment, it allows more frequent updates, as each change is much quicker to deliver. Developers spend less time setting up the environment and more time on the development itself. Handling network configurations by hand can be daunting to create, and IaC makes it easier to handle. It is also possible to configure RBAC (role-based access control) with IaC while creating infrastructure.

Been et al. (2022) give an example that cost optimisation from IaC could be significant when an environment is only deployed when the testing occurs compared to running it all the time. Otherwise, it is difficult to justify the initial costs of infrastructure automation, as planning it requires more time and initial resources. Most cost savings are going to be realised in the longer term, or even be hidden in seamless application deployments and stable production environment. Similarly, Berg et al. (2021) mention no direct cost benefit of DevOps but on the contrary, they state that as the features are being deployed more rapidly teams would try to push more features in, providing greater value for the customer.

IaC can help to build a system that is more reliable, more scalable, and transparent, and as a result provides higher product quality. How these IaC benefits are being achieved depends predominantly on the tools and how those tools are being used.

3.3 Available Tools

IaC allows defining and configuration of infrastructure based on software needs programmatically. IaC can be declared in various ways - plain JSON, domain specific language (DSL) or a common programming language such as Python, C# or Golang. Infrastructure code specifies both resources to create and configurations to apply to those resources.

There are two major approaches to writing infrastructure code – it can be either imperative or declarative. With the imperative approach, each step in a specific order is written about how to create the infrastructure. This approach requires a lot of details to be specified and might fail at any step thus failing the complete task. At the same time, it provides a higher control over the processes. Figure 6 presents imperative way of creating a bucket in AWS. Every command should be specified including resource check as if bucket were to be already present an error would be

thrown during the execution. With the declarative approach, only the desired state of a resource is specified, as presented in Figure 7. When the execution system is processing the code, it will take care of how to manage the state to fit the required definition. To provide that functionality, the execution engine keeps a record of the infrastructure state (Been et al., 2022). The declarative way provides a cleaner way to overview the infrastructure which makes it is easier to manage as the infrastructure size is extended by multiple environments. It is still easy to apply coding practice created in the declarative way: split complex configurations into modules while hiding unnecessary information but leaving the module dependencies visible (Spinellis, 2012). In addition, this approach supports idempotency out of the box while with imperative scripts the user is responsible for their maintenance (Hummer et al., 2013). An idempotent task always yields the same result, no matter how many times it is executed. With IaC scripts, it is important that there is only one template that represents the desired state of the resources rather than creating one for each update (Hummer et al., 2013). Some organisations prefer a declarative approach to orchestrate their infrastructure others might be mixing both ways. While many tools are associated with only one style of code they also support them both. There are no best practice or rules in choosing between imperative and declarative. Morris (2020) emphasises that as field of IaC is still evolving developers are in a state of discovering what works best in their own case.

```
$ BUCKET_NAME="examplebucket"
$ if ! `aws s3api list-buckets --query Buckets[*].Name | grep $BUCKET_NAME >
/dev/null`;  \
    then aws s3api create-bucket --bucket $BUCKET_NAME;  \
    else echo "Bucket with the same name already exists";  \
    fi
```

Figure 6 Imperative AWS CLI

```
S3Bucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketName: examplebucket
```

Figure 7 Declarative AWS CloudFormation

Although IaC practices are still in their early adoption there is a huge number of available solutions (Guerriero et al., 2019). There are various tools that can be used for IaC, some of which are cloud provider specific. Amazon was one of the first providers to offer CLI tools to manage infrastructure

(Campbell, 2020). Each major cloud provider offers their own configuration management tools. At present, AWS uses CloudFormation as the default tool, Google Cloud can be managed by Google Cloud Deployment Manager, and for Microsoft Azure it is ARM templates, Bicep and JSON.

CloudFormation is claimed to be an easy to use, declarative tool that has good support for the core services. Templates are written with JSON or YAML and they could be deployed from AWS CLI or staged to S3 bucket. The advantage of using CloudFormation is that it is an AWS service, which provides a better level of support. Users of CloudFormation have been reporting several drawbacks related to release lag for new features, lack of expressiveness of DSL, limited support for macros and functionalities, and inconsistencies (Campbell, 2020). Other tools to mention in relation to CloudFormation are Scepture, that is focused on stacks management, and Troposphere that allows creation in AWS CloudFormation with Python. One of the more popular tools that is based on CloudFormation is AWS Cloud Development Kit (CDK). AWS CDK ais imperative and uses CloudFormation's functionality for its orchestration functionality by creating an abstraction layer to support multiple programming languages such as Node.js, Typescript, C# and Python. Compared to its ancestor AWS CDK provides more structure and is more testable.

Azure supports two native ways to deploy infrastructure: ARM templates and Bicep. ARM template is a JSON file written in a specific format to represent Azure resources. As that format became difficult to maintain, Microsoft introduced Bicep to declare resources. Bicep is a DSL that uses transpilation to communicate with ARM API. Bicep allows the user to define infrastructure in a declarative and human readable manner. Compared to ARM or YAML templates Bicep files are more concise and is easy to use. Considering how young this DSL is, its structure is easy to understand even with a sizable number of resources. Bicep allows use of variables and parameters to allow environments' flexibility and as it is native to Azure it is integrated with Azure DevOps and other Azure Services, e.g., KeyVault for storing secrets. Bicep also has support for modules, what-if check to preview resources to be deployed and conditionals support. There is no need to define a separate state as it is handled by Azure. In addition, some Bicep extensions to IDEs provide a visualisation support that allows the user to gain a better understanding about existing cloud resources.

Other tools like Terraform by HashiCorp are vendor independent. Those tools could be used with any cloud provider, which makes them very intuitive to choose for a multi-cloud environment. Multiple cloud providers and multiple tools make it difficult to decide what to use. Currently there is no standard available for defining IaC. OASIS TOSCA was predicted to become a technology standard (Artac et al., 2017). However, there is only a limited proof available of how popular TOSCA became over the years, as it is rarely mentioned in existing studies (Guerriero et al., 2019). Nonetheless, the development of openTOSCA was funded by German government and German based companies.

IaC tools assessment has been a topic of multiple studies. Most existing studies are focused on mainly Puppet, Chef and Ansible (Rahman et al., 2019; Rahman et al., 2021; Rahman & Sharma, 2022). The main reason that previous studies mention picking those tools, is their popularity across the community. However, in the last few years the market grew and there are multiple options that are gaining a lot of attention. Terraform has been gaining popularity with only 20 000 repositories behind Ansible while conducting GiHhub search for terms “Terrafom” and “Ansible” at the time of writing. However, many companies tend to keep their code in private repositories, therefore it is difficult to judge on the true level of their popularity in industry. When conducting an advanced google search with search query "terraform vs. " OR "ansible vs:" OR "bicep vs." OR "puppet vs." OR "chef vs." OR "AWS CDK vs." OR "IaC tools comparison", the number of grey literatures that are comparing tools has grown over the past three years compared to the three years prior. 5 000 results were found for period of 1.01.2016 to 1.01.2019 compared to 18 500 results found between 1.01.2019 and 1.10.2022.

Terraform is an open-source, declarative tool written in Golang. Terraform has support for modules to abstract common pieces of the infrastructure into reusable components. To see what changes will be deployed to the infrastructure after applying a script, Terraform has a plan functionality. It also has a wide support for variables’ configurations, secrets, and use of locals. Data resources can also be added to the infrastructure scripts. These resources gather tags or parameters are needed for dependencies without a need to track their state. Terraform has its own mechanism for managing the state – it requires a state to be kept in some store (S3 bucket, storage account or local, though not recommended) that it could compare to the desired resource state that is expressed by code. Figure 8 presents an example of a terraform file that could be a part of the

<main.ft> file. To create terraform scripts, users should be acquainted with HCL (HashiCorp configuration language). Terraform is built to provide resource deployment to most cloud environments. For AWS support it depends on AWS CDK to interact with AWS API. For Azure, it is communicating directly with ARM. If a provider does not yet exist "out of the box", there is functionality to create ones' own plugins to extend providers' support.

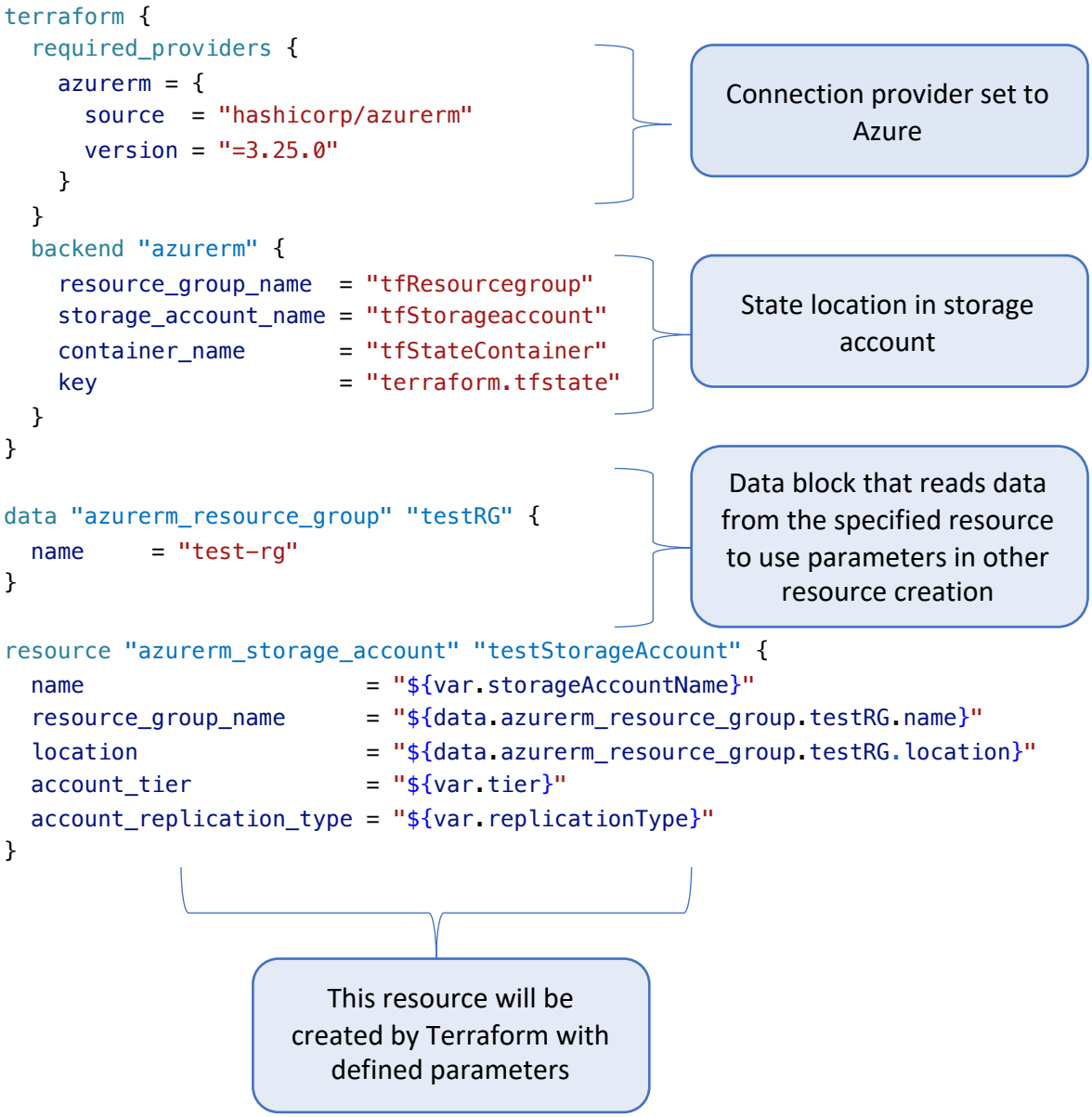

Figure 8 Terraform base structure

Pulumi tool is a new addition to declarative IaC tooling. It operates by wrapping Terraform providers and using them in runtime for API communication. Pulumi was created to overcome Terraform's shortcomings (Campbell, 2020). Instead of learning another DSL, Pulumi users can create configurations by using Golang, JavaScript (or any JavaScript transpiled language), or Python.

Chef is an example of another open-source IaC tool that mostly supports imperative ways of defining infrastructure (Hummer et al., 2013). It is written in Ruby on the client side and Erlang on the server side. When specifying resources, users deal with a Ruby-based DSL. For this tool the authors followed the "naming convention" in its fullest – infrastructure definitions are called *recipes* that are organised in *cookbooks* and nodes are managed by admin client called *knife.* Chef supports cookbooks' referencing and nesting. It also has a template library from which the recipes can be easily imported (Wurster et al., 2020). Puppet is one of the oldest tools on the market. It uses its own DSL to identify resources that are collected into Puppet modules. It also supports variables, expressions and even has an "object-oriented" feature for class definition and inheritance (Chen et al., 2018). Both Chef and Puppet require a client to manage the servers. All the commands go through the client, which is then communicating with the server to push commands and store the state of the infrastructure.

Ansible is a declarative YAML-based tool that shares Chef's concepts. Ansible is claimed to be an easy tool due to its agent-less model connecting to servers through SSH (Palma et al., 2020). Ansible automation engine works by connecting to nodes and pushing the scripts to apply which are called modules. Configuration and orchestration steps are declared with playbooks.

Present IaC tools, although similar, differ in their mechanism and application. Puppet, Chef and Ansible are configuration management systems that were designed to install and manage existing servers' software. While Terraform and CloudFormation are examples of orchestration tools that were created to provision the servers that the software is installed later (Brikman, 2019). Such categorisation means that before picking the tools, it is necessary to identify the focus of automation processes. Both sets of tools can do both tasks but some of them are easier to work with (Brikman, 2019). Wurster et al. (2020) attempted to create a deployment metamodel (EDMM) to compare different tools that they have identified as the most popular used. They consider the top 13 available deployment technologies, including tools such as Puppet, Chef, Terraform, ARM, CloudFormation and others. A few years prior, Rahman et al. (2019) identified 12 IaC tools. Palma et al. (2020) has identified Ansible metric for the code quality.

Tool	Type (manily)	Language	Used For	Owned by
Chef	imperative	Ruby-based DSL	Configuration management tool	Progress
Puppet	declarative	Ruby-based DSL	Configuration management tool	Perforce
Ansible	declarative	YAML	Orchestration and configuration tool	Red Hat
Pulumi	declarative	Python, JavaScript, C#, Go, and TypeScript	Provisioning and managing cloud resources	Pulumi
AWS Cloud Formation	declarative	JSON, YAML	Amazon Cloud resource provisioning	AWS
AWS CDK	imperative	Node.js, Typescript, C# and Python	Amazon Cloud resource provisioning	AWS
Bicep	declarative	Own DSL	Azure resources provisioning	Microsoft Azure
Terraform	declarative	HashiCorp Configuration Language (HCL)	Provisioning and managing cloud resources	HashiCorp

Table 1 IaC tools summary

To get started with a technology one needs to choose the right tools. There already exists a wide variety of tools which are used to implement IaC summary of is presented in Table 1. Tools them-

selves do not provide benefits; it is developers that should be able to use them in the best applicable way (Morris, 2020). With such a wide number of options available and no one-solution-fits all answer, it is important for the industry to open discussions and develop more unified practices for a wider, more general approach. Every tool comes with its drawbacks and issues that users are not satisfied about, which adds another point of difficulty for users to base their decision upon.

3.4 Challenges of IaC

Adopting IaC has been proven to bring benefits to projects and teams. At the same time, there are challenges when it comes to changing practices and learning new technologies, including IaC.

Practitioners can have difficulties changing systems that are already in a functioning state. Applying automation brings other risks to already established processes. One of the main resistances to adopting IaC is the fear of change. Organisations can have a certain mind-set, in some cases only focusing on either speed or quality of their code. As a result of such trade-off, reducing quality by focusing on speed creates fragile code. The other side of the scale creates outdated systems, which are overly bound by lengthy processes. The reality might come as a surprise that, quality and speed are supporting qualities of development practices and IaC introduces ways to implement that (Morris, 2020). Regular automated updates to infrastructure adds consistency to the resources, which increases confidence in running automation more frequently. Often adoption of IaC is not related to technical challenges, but requires changes of work practices, namely embracing DevOps (Guerriero et al., 2019).

Another challenge during early adoption of IaC is the wide array of available solutions and tools. Although it shows that the technology is still in its infancy, at the same time it obscures understanding of best practices and adoption techniques (Guerriero et al., 2019). Challenges in learning IaC has been identified as one of the future research directions by Rahman et al. (2019). Learning a newly emerging technology gets more difficult due to materials that are sprinkled over blogs, papers, and comments, providing no single source of concrete information. Books and trainings related to “common practices” of IaC started to emerge only recently (references). Online examples and tutorials are easy to follow but they do not necessarily provide the best practices. They are merely a way to get the development started. In addition, projects, teams, and technologies are distinct entities, so each IaC solution should be tailored to fit each purpose.

Kumara et al. (2021) conducted a systematic grey literature review of IaC practices. They included challenges that practitioners face while developing infrastructure code as a part of the selection process. Resources that were reviewed included online sources that only cover the sources written before 2019. Such scope is expected as the grey literature is growing more rapidly than it is possible to process in adequate time. It is also proving the point of IaC being an emerging technology which is still evolving at a rapid pace. As part of the study Kumara et al. (2021) identified readability, stability related to coding practices, security, code organisation, testability, and monitoring among many more challenges that practitioners faced while using IaC tools. Yet, as the authors mention, focusing only on three tools is a major limitation of the study.

On the peer reviewed side results correlate partially to those findings from industry. Practitioners identified testability, readability, consistency, and portability as main challenges of IaC (Guerriero et al., 2019). Testing is described by Morris (2020) as one of the of three core practices to implementing IaC. Morris also addresses the issue that testing IaC can bring challenges for teams as it is difficult to apply the same testing practices to IaC code that are applied to the application code. Some of the challenges in testing appear due to the declarative nature of IaC tools, slowness of tests, and overly complex dependencies to create tests. For the IaC community both manual and automated testing support the development of better testing practices. These testing practices in Terraform are extensively covered by Brikman (2019) in a rare case of white literature providing advanced knowledge for practitioners of IaC. Code organisation, team practices and infrastructure monitoring are also addressed by Morris (2020) as challenging and important parts of working in IaC settings.

In addition to these challenges, security in the context of automation has been a topic of increased focus, including IaC. IaC code, as often seen in regular code, is susceptible to defects (Rahman et al., 2020). Rahman et al. (2019) presented seven security issues in Puppet scripts that potentially could lead to security breaches. Those weaknesses included default admin access, empty passwords, hard-coded secrets, invalid IP address binding, suspicious comments, use of HTTP without TLS, and use of weak cryptography algorithms. To avoid said security issues the authors propose multiple strategies including development of a guideline to assist developers with writing IaC scripts in a more secure way. Later Rahman et al. (2021) followed with a study of security smells in

Chef and Ansible. In these studies, security smells included hard-coded secrets, empty passwords, overuse of admin privileges, network management and others.

Moving to IaC often means moving into the cloud. Cloud providers promote extensive ways to implement security, but users are quick to use cloud components in the least secure way (Morris, 2020). After creating a storage account and successfully stablishing connection to it from the application there is a chance that developer will forget about access restrictions and firewalls. Therefore, especially for new adoptees, security should not be added on top of the existing environment but instead it should be thought about already in the planning phase. When getting started with IaC is it easy to follow similar security anti-patterns. For example, when writing Terraform script for resource deployment to Azure, it is easy to write sensitive variables into the code, especially when working on a proof of concept. While it is acceptable for learning and testing after the code is committed into version control or run inside of a pipeline the secret variable would be visible to anyone who manages to get access to repository or deployment service. Therefore, it is important to investigate and plan resources' accesses, identify security rules, secrets' usages, and permissions.

Another challenge of using IaC is finding people that are going to do it. IaC is regularly applied in DevOps and Agile settings, where full-stack developers are already responsible for most of the processes. Due to the nature of IaC, it requires investments of both monetary and time. According to Morris (2020) getting started with infrastructure as code is time consuming. Developers are expected to simultaneously learn about available tools, new working practices, services, and if the team is moving to the cloud, also a new platform. When developer is applying IaC code in practice they do not only write the code, but they also become its reviewer. During infrastructure's lifecycle developers are continuously improving, supporting, and employing it in their projects. Thus, building reliable automation systems requires knowledgeable people – they are the main asset (Morris, 2020).

4 Adoption of New Practices in Software Development

Adoption of new technologies often is not an easy task and change itself can be a continuous process. In software development learning new practices is never ending as new technologies emerge and replace obsolete practices. In this way, the goals of innovation are never reached but instead

they evolve (Berg et al., 2021). If learning efforts stop, companies run the risk of becoming uncompetitive and obsolete. Those processes can equally affect big and small technology companies.

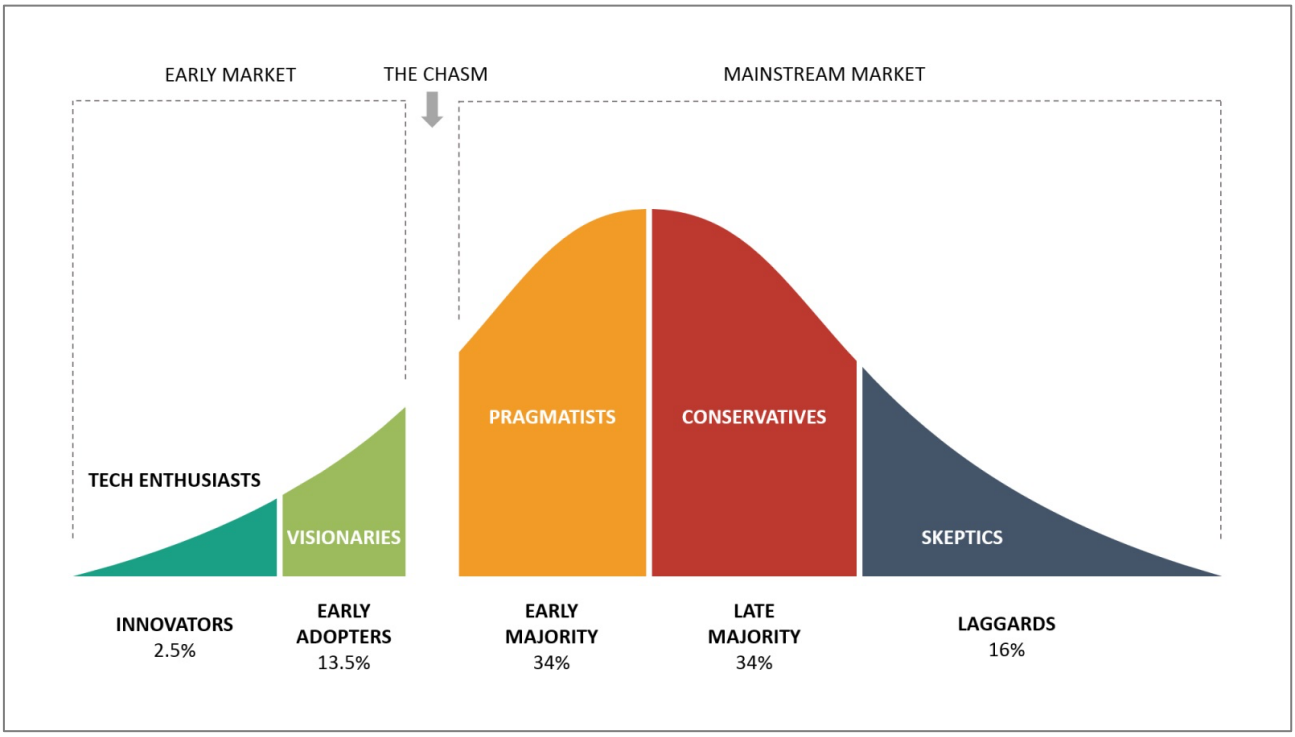

Figure 9 Moore's Adoption Curve

Academics have been addressing technology adoption for over two decades. Moor's "Crossing the Chasm" (Moore, 1999) is a widely cited book in relation to technology adoption. According to Moore (1999), technology adoptees can be divided into five groups: (a) Innovators; (b) Early Adopters; (c) Early Majority; (d) Late Majority; and (e) Laggers (Figure 9 from (Think Insights, 2022)). However, these stages do not come together seamlessly but are surrounded by gaps. The biggest gap that resides between the Early Adopters and Early Majority, is referred as the Chasm. The reason that it is difficult to cross the chasm is due to the difference in the needs of early adoptees and the majority (Think Insights, 2022). During the chasm gap, users are looking for more proof of how a technology could help them. They are focused more on stability and use cases. According to Moore (1999) while crossing the chasm, software practitioners learn from other software practitioners in their field.

Sultan and Chan (2000) present a model that describes factors affecting adoption of new technologies within companies. The model identifies (1) the scale at which this happens through individuals, groups, and in the company context; (2) technology perceptions and risk groups; and (3) experience factor as main dimensions that can measure technology adoption.

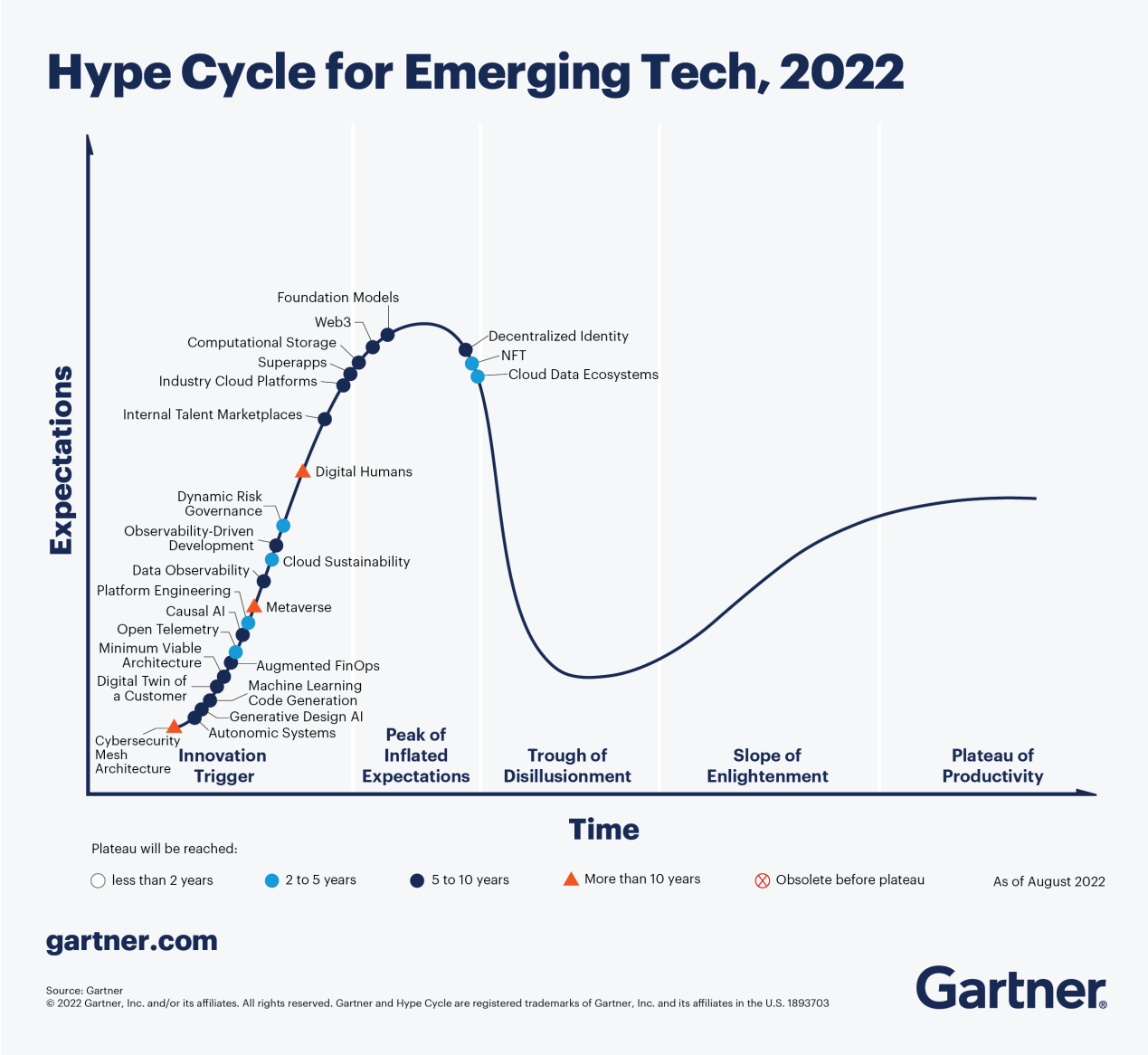

Figure 10 Gartner's Hype Cycle 2022 (Pending image usage approval from publisher)

The Gartner Hype Cycle is another way in which the technology life cycle model could be presented. According to Gartner's research methodology there are five phases of technology adoption that many technologies follow (Figure 10). Early adoptees are the ones who usually are the first one to try the technology, thus bringing it to the hype point. Adopting a technology too early, however, presents risks. After the expectations are not met, technology adoption falls into a

trough of disillusionment with only about 5% of the audience adopting it. During this phase, adoptees question the potential of a technology as challenges and failures are being addressed at a larger scale. As the dust of the fall settles and adoptees find more proven solutions and industrial use cases for the technology it sets into the Slope of Enlightenment, a more mature phase of the cycle. The last phase is the Plateau of Productivity that all technologies aim to reach as it is a solid pool of 30% of potential users as an audience with a growing community that have found ways to use the technology in their everyday business. Takeaways from Gartner's theory are (1) do not invest too early in the technology; (2) do not give up too quickly when the majority become disappointed with its applications and development; (3) do not wait until too late when no competitive edge remains to be gained from investing in the technology. Gartner makes a yearly report to present the Hype Cycle for Emerging Tech. For the year 2022 Cloud Data Ecosystems is identified to be starting the fall from the hype peak and is predicted to get to the plateau in two to five years (Figure 10).

Technologies that emerged in the software development field in the past 25 years have been driving industry to an increased rate of technology adoption. Cloud technologies, Agile, XP, Scrum, Lean, and DevOps are only a few additions that impact companies' willingness to change. For cloud-native systems, the learning curve is steep due to the number of available resources and their configurations, however it is gaining high levels of adoption (Labouardy, 2021). Practitioners have been struggling with Agile adoption (Silva & Goldman, 2014). Mahanti (2006) studied early adoption of Agile in enterprise companies. They addressed challenges of Agile adoption in connection to Chasm theory, identifying that Agile was at the Early Adopters phase. Some challenges were related to fear of change, closed mindedness, specialised, or outdated skills, and excessive documentation habits. Nuottila et al. (2016) studied factors related to adoption challenges of Agile in public companies. They present among other findings that challenges were related to habits of over-documenting, lack of education, experience or commitment, lack of stakeholders' communication, and lack of knowledge of roles in Agile set-up. Chen (2017) identified "lack of understanding" as an adoption challenge of CI. Lwakatare et al. (2019) studied adoption of DevOps in five companies. Adoption of DevOps was more successful when the change was supported by both senior managers and customers. In some cases, developers found the IaC part of DevOps adoption learning curve very steep which resulted in developers' reluctance to pursue further development. As a whole, DevOps adoption was challenged by a steep learning curve, automation practices, balancing between speed and quality, monitoring, and resource constrains. In their case study,

Riungu-Kalliosaari et al. (2016) connect difficulty in adopting DevOps to its vague definitions. Practitioners were lacking fixed practices for DevOps' usages. One of the biggest barriers to adoption was found to be insufficient communication. DevOps also highlighted a need for changing the company's culture to make people more receptive to change. According to Berg et al. (2021) companies' transformation to both Agile and DevOps is a learning journey for both developers and management. Developers should ask questions and be allowed to experiment, while leaders should embrace discussion and learn as much as they can about these practices. Both continuous improvement and learning should be incorporated into the working routine to improve the team's ability to adopt the new practices.

5 Implementation

5.1 Participants selection

To acquire participants, 16 companies were contacted to participate in the survey, from which five did not respond back. Companies that participated in the survey included Finnish based companies that specialise in consulting, banking, and engineering. Finalised set of companies consisted of 1) companies that were identified either to follow DevOps and/or IaC; 2) consulting companies or 3) personal contacts of the author that worked in the field.

Rather than sending the survey to a wider population, for example by means of social media, sampling was applied to preselected suitable participants. As the research focuses on IaC adoption in industry, participants from specific organisations that utilise IaC were a substantial part of participants' selection. Consulting companies and companies that develop web-based applications and services in the cloud were primary target groups of this survey. In a consulting practice, projects last only a short amount of time providing a quicker feedback loop. Teams have a chance to learn from the experience and gain more opportunities to adopt and use new technologies with each consecutive iteration. On the other side, teams that are using cloud providers to deploy their applications present another group of interest. While being active adoptees of cloud native architecture they could provide insights into integration of cloud infrastructure and potentially IaC as a part of their daily work.

Collecting data from industry is a challenging task due to companies, especially software consulting companies, often dealing with confidential data of their customers. Busy schedules often prevent resource allocation for documentation of developers' learning practices in a structured way. As a result of market competition and not wanting to lose potential competitive edges, software companies are not willing to share their practices publicly and therefore they do not keep their projects in open access repositories.

Participants' selection consisted of multiple steps. First, middle size Finnish based consulting companies were identified. The first round of filtering included companies that have information on their web pages related cloud services and/or DevOps. The second filter occurred while selecting companies that provided their contact information on their web page. Companies that only allowed contact through forms and sales' quotations were discarded. Another set of potential respondents were colleagues of the researcher or personal contacts of author from similar consulting companies. Lastly, a small set of selected companies included ones that shared some of their GitHub projects for public use and clearly stated it to be a part of IaC practice with a chosen tool.

Potential respondents were personally contacted via email, social media, or messaging applications. With every request the researcher asked respondents to forward the survey to their colleagues who would be willing to contribute to the study. In the beginning of the survey participants were informed about the reason behind the data collection and its purpose as stated in the data management plan (ref. Appendix 1). Respondents were also asked if they wanted to take part in further inquiries by the researcher regarding possible follow-up questions. If participants were interested to receive the results from the study, they were asked to leave their email address. However, there was no other requirement for providing personal data as answers were collected anonymously.

5.2 Survey Design and Data Analysis

The survey was created with the Webropol 3.0 Survey & Reporting tool. The tool is available for online use for students enrolled at JAMK University of Applied Sciences and is recommended by the institution for data collection purposes. The survey was sent to participants over the course of approximately one month. After the initial data collection with the survey and preliminary data analysis, follow up inquiries were sent to some participants via email. A data management plan

was created to support data collection and is available in Appendix 1. Full survey listing is available in Appendix 2.

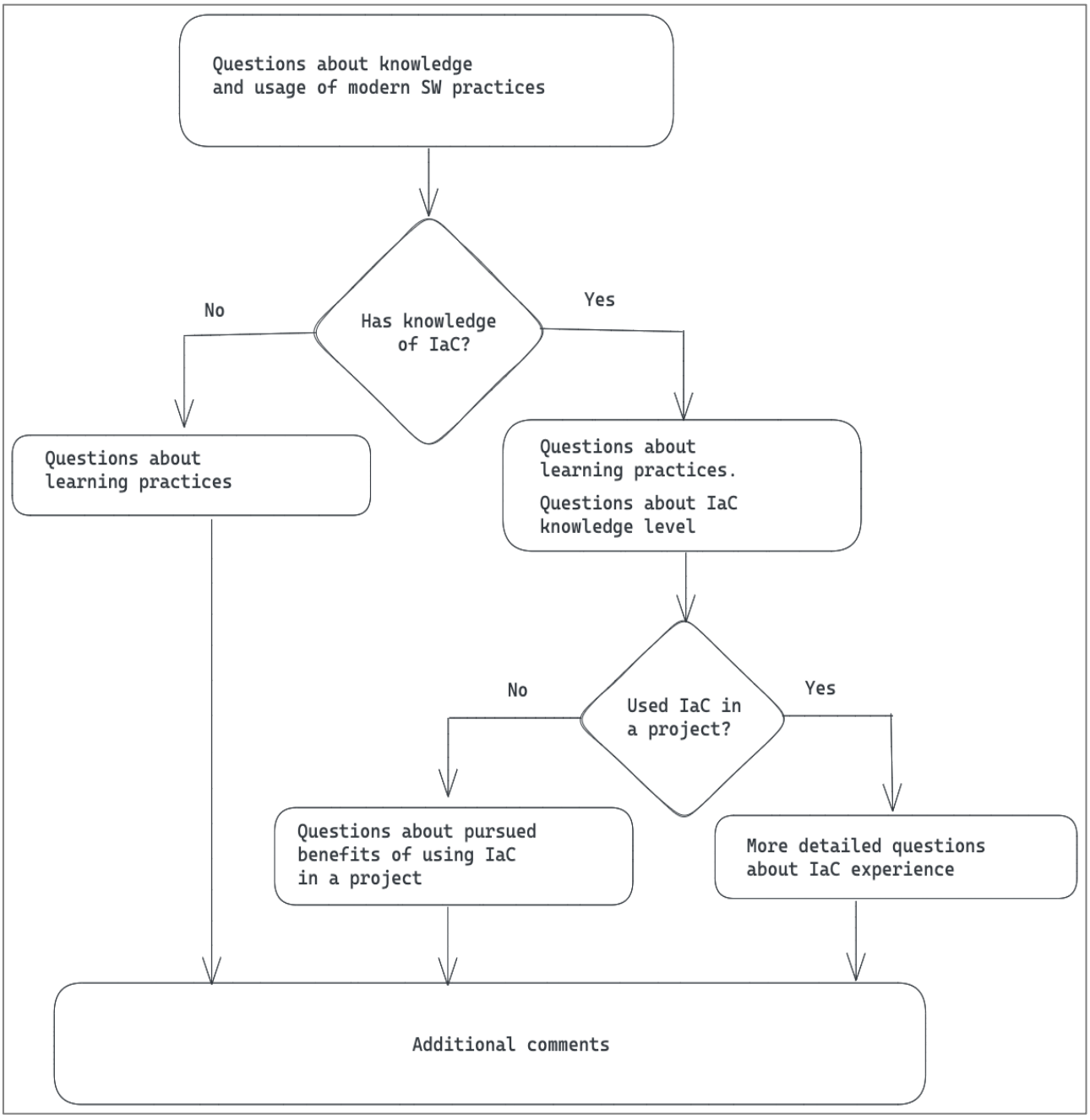

Figure 11 Survey design flow chart

The goal for designing the survey was to keep the survey short while obtaining as much meaningful data as possible. The survey was structured to gather both knowledge of IaC adoption in projects and information on awareness of the practice. To separate developers that had no knowledge of IaC, the survey included rules that tailored the order of questions and their appear-

ance by hiding irrelevant questions based on previous answers. For the developers that are familiar with the technology but do not use IaC in their projects a separate set of questions was presented to enquire their opinion about potential benefits of using IaC. The main logical complements are presented in Figure 11. The survey starts with a presentation of the research. The author included information about data anonymity and a field to submit an email address for potential enquiries. The first data collection questions asked respondents about their knowledge and usage of modern software practices, including Agile, DevOps, Version Control, CD/CI, etc. A question about cloud practice adoption was included to confirm that modern use of IaC happens primarily in the cloud environment setting. Semantic differential scale type questions were asked to obtain quantitative data related to practitioners' experience of learning and using IaC. At the end of survey, a question with a free comment field was included to provide respondents with an opportunity to leave other thoughts about IaC that were not asked during the survey. Additionally, the survey included a short introduction to IaC practice and a link to material from RedHat (Red Hat, Inc., 2022) introducing them to IaC. The material was included to provide information to those developers who are not yet familiar with the practice.

Answers from practitioners that were actively using the technology in the scope of a project within organisations were identified as core data. Respondents with no experience with IaC were still able to provide valuable data to describe other practices in use, teams' awareness of IaC, and overall knowledge sharing practices in companies. Answers from the developers that have adopted IaC in a project scope were taken for the further analysis. Table 2 contains survey questions that were directly motivated by previously delineated research questions of this thesis. Open-ended questions are listed in the table without answers options.

Q#	**RQ#**	**Question text**	**Answer choices**
8	RQ2	Do you think that IaC could benefit your team?	Yes, No
9	RQ2	What barriers do you think have stopped your team from implementing IaC?	-

13	RQ2	How often is the topic of automation processes mentioned in your team?	[Weekly, Monthly, Sometimes, Never]
15	RQ3	What was the reason that you started using IaC?	-
16	RQ2	Who proposed the technology?	[Developers, Management, Customer]
19	RQ1	What are the main difficulties that you have faced with IaC (if there were any)?	-
20	RQ3	What sources do you check to resolve difficulties and questions about IaC?	[Colleagues, Books, articles, Official documentation, Blog posts Chats, stack overflow, Other. Please, specify]
22	RQ1	How would you rate the difficulty to start learning IaC?	Scale 1-10
23	RQ1	How would you rate getting more advance knowledge of IaC?	Scale 1-10
24	RQ1	How would you evaluate your overall experience with IaC?	Scale 1-10

Table 2 Survey questions

To improve understandability of the questions, the survey was reviewed and tested before opening it to respondents. A language expert, a lecturer experienced in surveys and a software expert were asked to evaluate the questions, ensuring that they were complete and clear. Questions were corrected and extended after three review iterations. Changes included minor corrections to language used, questions type and additional information provided at the start of the survey.

Survey data was gathered with the Webropol tool which also provided later storage, reporting and partial analysis. Before the data analysis process, raw data was backed up to OneDrive and saved on a laptop for further analysis. Before data analysis answered surveys were checked for validity,

ensuring that each of the respondents provided serious replies, and obviously falsely answered surveys were disregarded.

The next step in data processing included compilation of the descriptive statistics from the dataset. Descriptive statistics provide a summary of the data and present basic features of the data often with graphics analysis (Trochim, 2022). Inferential statistics in this data analysis consisted of paired T-tests. A T-test is used to compare different groups among the respondents by measuring the difference between the mean values of the groups. The resulting statistic informs whether the differences between the groups are real or are purely by chance. In question 6 respondents were asked to describe their knowledge of IaC within a set scale. Based on the knowledge declaration, the respondents' answers to the evaluation questions no. 22, no. 23 and no. 24 (listed in Table 2) were assigned to three groups (experts, advanced and intermediates). The groups were compared to each other using the T.TEST function in Excel. The same grouping was used to compare the number of actively used tools each respondent indicated in question 17. An alternative grouping of the respondents was based on the number of years they used IaC for, as answered in question 14 and it resulted in three groups (more than 3 years, 2-3 years, less than 1 year). The answers to the evaluation questions no. 22, no. 23 and no. 24 were compared based on this alternative grouping. Corelation analysis was done by Webropol Insights functionality for questions that would be informative (see Figure 14). The questions selected for comparison provided insight into potential correlations into individuals knowledge about IaC in relation to the learning practices of their workplace. And how proactive individuals are in pursuing IaC knowledge sharing among their colleagues.

Qualitative data was analysed separately due to its unstructured nature. The survey included five open ended questions related to IaC adoption reason, difficulties, barriers for IaC implementation and learning practices. All qualitative answers were categorised using a Reflexive Thematic Analysis (TA) process. Presented by Braun and Clarke (2019) a Reflexive TA process applies an iterative coding technique to textual data for defining themes within the unstructured data. For each text item, the researcher assigns labels to identify the most important feature which generates potential themes. Labels and themes can be renamed and updated with each iteration. Each finalized theme is described as objectively as possible so it can be compared to existing literature. Reflexive TA can be used for both short qualitative answers and longer texts, such as observations, notes, or

interviews. Besides statistical analysis, the Webropol tool provided a text mining feature for qualitative data. This feature was used to confirm the researcher's initial observations from the respondents' answers to the open-ended questions. Thematic analysis was also applied to open comments question.

6 Results

The survey link was opened 97 times, 33 people started answering the questions, among which 27 respondents submitted the survey. As not all the questions were mandatory and therefore skipped by some developers and the survey rules applied filtering based on respondents' experience with the practice, the number of questions answered varied.

Practice	Familiar with		Usage in project	
	#	%	#	%
Agile	27	96.4%	23	82.1 %
Lean	14	50.0%	4	14.3 %
Scrum	24	85.7%	17	60.7 %
DevOps	26	92.9%	23	82.1 %
CI/CD	26	92.9%	25	89.3 %
Automated deployments	24	85.7%	21	75.0 %
TDD	21	75.0%	8	28.6 %
Code review practices	23	82.1%	21	75.0 %
Version control practices	26	92.9%	26	92.9 %

Table 3 Modern SW practices awareness and usage

Most of the respondents have a solid knowledge base of modern SW practices (full listing of the practices is presented in Table 3). Only one respondent did not indicate knowledge of Version Control. Four respondents did not know about Code Review practices, while six did not use it in their daily work. IaC practice was a familiar concept to 92% of the respondents. 100% of active adoptees reported using version control for storing their code. All the respondents that practice IaC also used cloud computing. It supports the idea that IaC is an important part of cloud eco system management. In the majority of cases (90%) adoption of IaC was initiated by the developers themselves. In 85% of the responses, either the respondent themselves or their colleagues use IaC in

projects. Respondents that did not used IaC as part of their projects recognise potential benefits as a result of the practice. However, they found it problematic to adopt IaC due to a lack of knowledge and time. One respondent mentioned that they do not believe the practice is mature enough yet to be employed in their work.

Overall, respondents thought starting to learn IaC was of average difficulty, being rated 4.0 out of a 10 points scale. They recognised getting more advance knowledge of IaC to be noticeably more difficult, rating it 5.6 out of 10. Although on average it can be observed that more experienced users thought it is easier to start learning IaC than beginners, this difference was not statistically significant ($P = 0.305$). The difference in ratings given by experts compared to beginners ($P = 0.186$) and advanced users compared to beginners ($P = 0.191$) for getting advance knowledge was not significant either. Overall experience with IaC was rated 7.2 out of 10 across the groups. Here the difference between the groups is significantly different with both experts and advanced practitioners evaluating IaC much higher than intermediate users (experts vs intermediates $P = 0.013$ and advanced vs intermediates $P = 0.031$). 18 out of 19 respondents said that they will push their team to use IaC in the next project.

In their reasons for adoption, 23% of respondents identify IaC as an established practice. Other respondents mentioned various reasons for starting to develop IaC, including reproducibility and audit trail, which reflects well-established benefits of using the practice. This knowledge supports the notion that IaC can be successfully applied in industry and that practitioners have a good understanding of benefits of its implementation. In three cases respondents mentioned that they started using IaC to manage multiple environments.

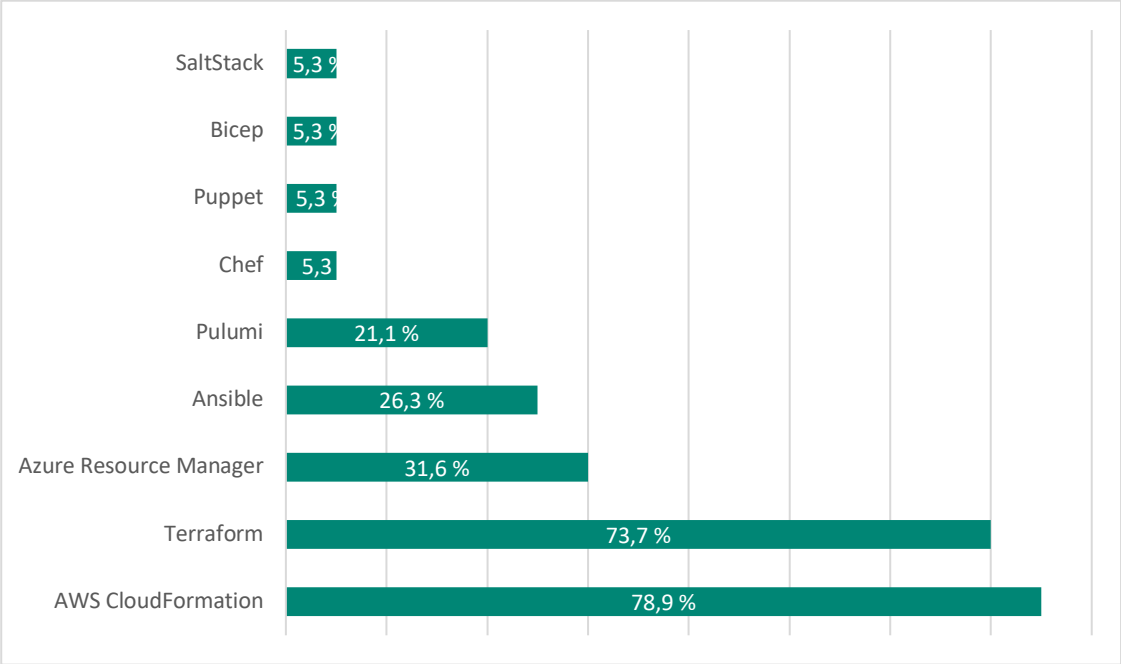

Figure 12 IaC tools usage

Practitioners used a variety of tools to manage infrastructure. AWS CloudFormation was the most commonly mentioned tool, followed by Terraform (Figure 12). In the AWS community, practitioners prefer using AWS CDK to CloudFormation, which they specifically mentioned as a part of their reply about tooling. From the tooling selection it could be concluded that in developers used either AWS as their main cloud environment or multi-cloud. Developers with more experience with IaC used a higher variety of tools. Experts declared that they used on average 4.3 tools, advanced 3.5 tools, and intermediates only 2.0 tools.

Respondents that have more experience with IaC pushed their teams more to use IaC. Having more experience with IaC inversely correlated to the frequency of discussions around automation (Figure 13). Automation was a topic of discussion in teams of all the respondents at least in a semi-regular way. Developers that have been adopting IaC for longer tended to discuss it less often than other practitioners. Experience overall did not correlate to the other answers. However, experience and knowledge were inversely correlated (Figure 13).

	5	6	9	10	12	13	21
5	1	**0.61**	**0.68**	**0.44**	**0.43**	-0.24	**0.81**
6	**0.61**	1	**0.43**	**0.45**	0.37	0.06	-0.31
9	**0.68**	**0.43**	1	0.31	**0.42**	-0.19	**0.66**
10	**0.44**	**0.45**	0.31	1	0.22	0.19	0.33
12	**0.43**	0.37	**0.42**	0.22	1	-0.39	-0.11
13	-0.24	0.06	-0.19	0.19	-0.39	1	-0.12
21	**0.81**	-0.31	**0.66**	0.33	-0.11	-0.12	1

Figure 13 Correlations matrix

5. What best describes your knowledge about IaC?
6. Have you ever used IaC to manage infrastructure in a project?
9. How many other people in your team are aware about IaC practices?
10. Do you have knowledge sharing activities in your team?
12. How often is the topic of automation processes mentioned in your team?
13. When was the first time you used IaC?
21. Would you push your team to use IaC in the next project?

Figure 14 Questions used for correlation analysis

There were multiple problems facing developers when they were implementing IaC. After multiple rounds of coding activity, the answers to question 19 were categorised in following themes: 1) debugging – there are no easy debugging possibilities and error messages are often either not documented or are unclear 2) complexity – initial setup required significant amount of time especially if the environment were more complicated, included modules, handled multiple environments or cloud providers; 3) permissions – while handling infrastructure for customers, permissions were difficult to request; 4) learning – lack of documentation about the functionality and tools selection; and 5) non users of IaC – difficulties to work with resources that were created manually and convincing developers to switch to IaC. While resolving the problems during infrastructure setup developers rely mostly on their peers and official documentation (Figure 15) according to how they answered question 20. The sources used by developers connect well with the problems that they experience, as debugging problems are often very specific and there is a chance that other

community members would have experience of similar situations. More complex setups are usually not described in the basic documentation, therefore developers again depend on the support of their peers. An additional answer to this question was a response of “trial and error”.

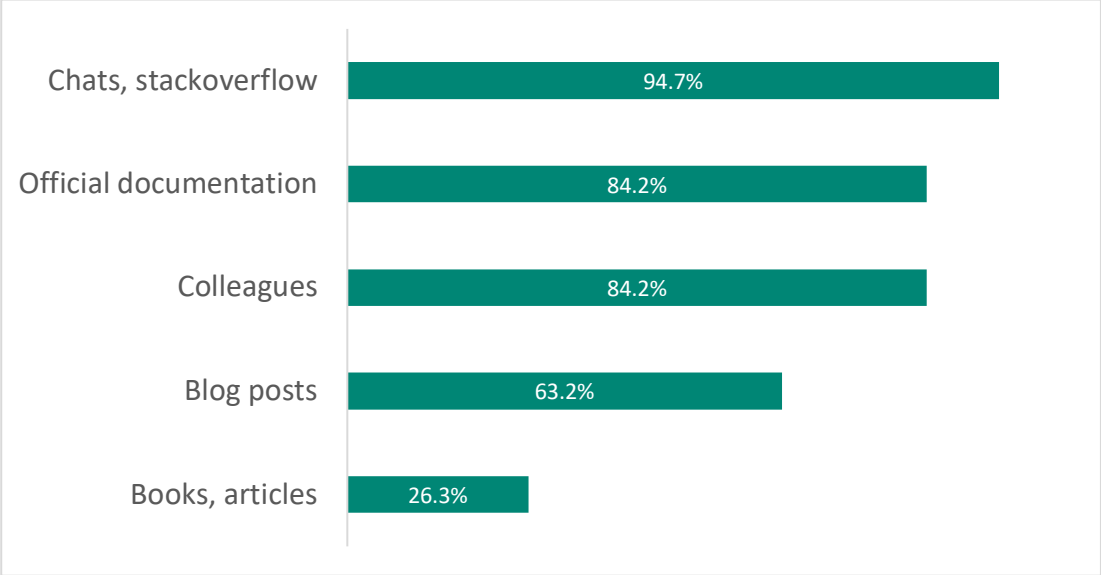

Figure 15 Source of knowledge

Considerable number of respondents (82%) reported an active learning culture at their workplaces. Learning activities varied from regular meetings (weekly or monthly) to learning groups and ad hoc training sessions. Respondents that mentioned regular meetings responded that the whole or at least half of the team were familiar with IaC practices in question 10. One team organised group trainings in their team and the developer specifically mentioned the adoption of AWS CDK into use during those sessions.

In question 21 developers shared their insights about the skills needed to start implementing IaC. Here the answers were complementary with each other and formed well defined groups. Developers responded that it is equally important to have at least a basic development background and knowledge of the environment they are working with. Being familiar with yaml, json and scripting were mentioned as major benefits as well as information seeking skills. This observation connects well with answers to questions 19 that internet sources were useful and official documentation was lacking information.

7 Discussion

This thesis aimed to investigate the state of adoption of IaC by means of gathering developers' opinion about the practice via a survey. As part of the thesis, a literature review was completed to examine the emergence of IaC practice as well as common tools and technology adoption theories. A target of this thesis was to narrow the gap in the empirical research related to the learning of IaC practices. Though having limitations this research investigated the manner in which developers begin learning IaC and the difficulties potentially slowing its adoption.

The findings of this thesis show that IaC as a practice is well known, and it is receiving active attention from practitioners in Finland. This research focused on finding insights into IaC adoption in teams that actively used IaC as part of their development process. Although adoption of IaC can be complex, there are still clear benefits for practitioners to implement the practice in their projects. Despite IaC practices not yet being widely applied, teams are starting to find usages for it in real life projects which matches well to be in the chasm state of adoption across the industry.

In line with initial assumptions, the data reveals that IaC is not overly difficult to learn in comparison with other technology practices, but it still requires effort from developers. It could be concluded that practitioners are satisfied with the practice as after becoming accustomed to using IaC, developers forget there are other way to manage their infrastructure as noted in at least five responses to the survey. As there is no connection between the years of experience in using IaC and developing a solid foundation of knowledge in its application, it can be concluded that becoming confident in IaC knowledge does not require a significant amount of time. However, developers admit that some structures of infrastructure automation, such as modules and multi environment support, could be more complex to build and maintain. As a result, this could be intimidating to developers who are just starting to employ IaC in their work in addition to getting used to working in the cloud. Advancing IaC skills is impeded by the lack of reported use cases and advanced examples in the official documentation. IaC tools providers should update their documentation more regularly to accommodate evolution of the field. Alternatively, courses that are run by more specialised consulting companies could help reduce this gap, on top of which they can create their own documentation. Results of this study suggest that it could be relatively easy for a team to shift to IaC. Developers agreed that when starting to learn IaC, it is important to have at least a basic development background along with good information seeking and troubleshooting skills. But they

also experience challenges connected to learning and convincing others to start using IaC. If a single developer without support of the team is starting to use IaC, they would not be able to gain its benefits. On the contrary, they would become less productive by trying to keep up with the changes that were done manually, resulting in even less stable infrastructure than the one they started from. Gaining correct permissions to access customers' infrastructure was mentioned by developers as one of the difficulties while creating infrastructure. In consulting companies, infrastructure is managed by their customers who have their own requirements but are not yet adapted themselves to allow easy automation processes.

Teams start applying IaC to projects to overcome shortcomings related to infrastructure audit, supporting multi cloud environments and scalability. These benefits developers reported seeing in IaC align with those viewed in prior studies (Guerriero et al., 2019). Teams that are most aware of IaC in this study were also the ones that reported the most structured learning and knowledge sharing activities which suggests that peer support is an important step in learning IaC. Developers who would like to introduce this practice to their workplace need strong learning and knowledge sharing culture.

The findings support earlier studies that IaC practitioners tend to be polyglots (Guerriero et al., 2019). It also reveals that the more developers know about IaC, the greater the number of tools they find themselves using. As there are no best practices established yet, developers are still figuring out which way of work fits better to certain projects. Between the respondents, there is a clear tendency for using AWS cloud as they were AWS specific tools. However, as Terraform is used nearly as often, that could be a way in which practitioners go around the challenge of multicloud environments. According to HashiCorp's annual report multicloud/cloud agnostic tools are getting more attention from industry in 2022 due to its ability to help organizations with their business goals (HashiCorp, 2022). It is possible that a cloud agnostic tool is used as a backup in case the cloud infrastructure would be transferred elsewhere. Some teams could also be applying different tools to different projects.

Despite Rahman et al. (2020) describing the lack of version control usage as an anti-pattern that was reported by developers, the results of this thesis show the opposite, with developers using

version control as a part of their best IaC practices. This shows the change in the field since Rahman et al.'s study. This contrary finding could also be the result of alternative tools being reported by respondents, namely Terraform and AWS CDK as opposed to Puppet in the previous study. In addition to Rahman et al. findings of lack of logging and testing being a concern for developers, results observed in this thesis separate debugging as one of the main challenges of IaC users. As in many cases logging is used to assist debugging processes, these two challenges are closely intertwined.

As part of the open comments, five developers stated that IaC practice is a crucial practice to have. Some of the respondents provided open comments in the end of the survey. In their statements developers shared insights regarding the IaC adoption process in their teams and some have described the throwbacks that they have experienced while starting with IaC. They emphasise the importance of not trying to start with a full automation but start with small steps. Another comment highlights the importance of writing tests as early as possible, thus suggesting another best practice of testing being applied.

As this study implements survey-based data collection, there are multiple factors that may affect the validity of the results due to the uncontrollable nature of who responds to the survey and those that ignore it. Even though the study was trying to include respondents from across the spectrum of knowledge on IaC, in practise this proved very difficult to execute properly in the available timeframe and resources. The survey was predominantly responded to by experienced users of IaC, despite not explicitly seeking experts in IaC. Another potential threat to validity is that the survey gathers a snapshot of current opinions from a limited number of users in the field. Hence, the reported findings may not be representative of industry wide attitudes towards IaC. The low response rate can be interpreted in multiple ways. Some respondents might have opened the survey multiple times on different devices. It can also be expected that after initially opening the link, respondents did not want to participate as they had no knowledge of IaC and felt intimidated by a topic in which they feel they cannot contribute much to. The survey should have stated more clearly that no prior knowledge of the practice was required for answering the survey. It is equally possible that the survey description and the title did not generate enough interest and failed to engage potential respondents to the point that they would be more willing to participate. A small number of respondents did not complete the survey after starting to answer questions. It

is possible that participants did not complete the survey due to their time constraints or interruptions. Another threat to the validity of the survey results is the design of the survey itself. Improvements could be applied to some of the survey questions by changing scale or reforming a question's type and text. As there were multiple limitations in this study, it would be interesting to view how a new cohort of respondents would answer the survey and if it would differ or complement the current collected data. An important part of this would be to include more novice practitioners among a wider range of skill levels.

Based on the results of this thesis, the general consensus is a recommendation for improvement of IaC adoption. Companies need to invest into developers' knowledge of IaC and promote their learning for it to reach its potential at a team scale. For the developers it is important to take time in understanding the infrastructure, implement tests and start small when just getting started to implement IaC. Finally, IaC tools providers should focus on improving their official documentation to support more complex use of the infrastructure with use case scenarios. This process would continue to push the development and adoption of the practice to the next level, unleashing greater potential for wider usage, bringing greater benefits to its adoptees.

8 Conclusions

Manual processes require more time and are error prone, but IaC provides developers with the ability to automate creation of infrastructure and support Cloud Data Ecosystems in an automated manner. Adoption of IaC has a potential to lower companies' expenses while improving time efficiency, consistency, and transparency of their cloud infrastructure. As mentioned by Gartner and HashiCorp, there is clearly a lack of professionals that are competent enough to automate more complex tasks and environments. Without these professionals, successful implementation of IaC is greatly reduced and the potential benefits to industry would be negligible. To get more IaC competent and fluent developers, industry wide adoption will be necessary. A big part of adoption will include successfully learning IaC and opening discussion for its development. From both the literature review and the results of the survey the best strategies for learning IaC are to partition the subject into smaller, easily absorbed pieces and convey full teams to follow said practices. However, this learning will require time and repetition, along with commitment from management, customers and developers in companies.

IaC generates significant interest from both academia and industry. There has been an increasing number of publications and blogs on the subject in recent years. However, more structured research is required to develop best practices and elaborate on the usage of tools. Furthermore, more in-depth case studies and advanced knowledge is needed to broaden insights and expertise in the field. One possible research avenue could be analysing learning materials provided by IaC tools' vendors to see how successfully they convey core IaC practices and how coherent their examples are to the needs of the field.

It is through investment of time and resources into research and learning that Infrastructure as Code would become economically relevant. Once it generates enough interest from the customers it can become truly widespread and reach its final adoption stage.

References

Ambler, S. W., & Lines, M. (2020). *Choose Your WoW! - A Disciplined Agile Delivery Handbook for Optimizing Your Way of Working.* Project Management Institute, Inc. (PMI).

Artac, M., Borovssak, T., Di Nitto, E., Guerriero, M., & Tamburri, D. A. (2017). DevOps: Introducing Infrastructure-as-Code. *In 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C),* (pp. 497-498). IEEE.

Been, H., Staal, E., & Keiholz, E. (2022). *Azure Infrastructure as Code: With ARM Templates and Bicep.* Simon and Schuster.

Berg, C., Cagle, K., Cooney, L., Fewell, P., Lander, A., Nagappan, R., & Robinson, M. (2021). *Agile 2 - The Next Iteration of Agile.* John Wiley & Sons.

Bosu, A., Iqbal, A., Shahriyar, R., & Chakroborty, P. (2019). Understanding the motivations, challenges and needs of blockchain software developers: A survey. *Empirical Software Engineering, 24*(4), 2636-2673.

Braun, V., & Clarke, V. (2019). Reflecting on reflexive thematic analysis . *Qualitative research in sport, exercise and health, 11*(4), 589-597.

Brikman, Y. (2019). *Terraform: Up & Running: Writing Infrastructure as Code.* O'Reilly Media, Inc.

Campbell, B. (2020). *The Definitive Guide to AWS Infrastructure Automation: Craft Infrastructure-as-Code Solutions.* Apress.

Chen, L. (2017). Continuous Delivery: Overcoming adoption challenges. *Journal of Systems and Software, 128*, 72-86.

Chen, W., Wu, G., & Wei, J. (2018). An approach to identifying error patterns for infrastructure as code. *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 124-129.

Creswell, J. W., & Plano Clark, V. L. (2017). *Designing and conducting mixed methods research.* Sage Publications.

De Bayser, M., Azevedo, L. G., & Cerqueira, R. (2015, May). ResearchOps: The case for DevOps in scientific applications. In 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM) (pp. 1398-1404). IEEE.

Edwards, P. J., Sainfort, F., Jacko, J. A., McClellan, M., & Kongnakorn, T. (2012). CHAPTER 41: METHODS OF EVALUATING OUTCOMES . In G. Salvendy, *Handbook of Human Factors and Ergonomics.* John Wiley & Sons.

Gartner. (2022, August 10). *What's New in the 2022 Gartner Hype Cycle for Emerging Technologies*. Gartner: https://www.gartner.com/en/articles/what-s-new-in-the-2022-gartner-hype-cycle-for-emerging-technologies

Gartner. (2022, January 26). *4 Predictions for I&O Leaders on the Path to Digital Infrastructure*. Gartner: https://www.gartner.com/en/articles/4-predictions-for-i-o-leaders-on-the-path-to-digital-infrastructure

George, T. (2022, 10). *Mixed Methods Research | Definition, Guide & Examples.* Retrieved 5 11 2022, from Scribbr: https://www.scribbr.com/methodology/mixed-methods-research/

Google. (2022, 10). *Google Trends*. https://www.google.com/trends

Guerriero, M., Garriga, M., Tamburri, D. A., & Palomba, F. (2019). Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 580-589). IEEE.

HashiCorp. (2022). *HashiCorp 2022 State of Cloud Strategy Survey.* Retrieved 2 12 2022, from Making Multi-Cloud Work: https://www.hashicorp.com/state-of-the-cloud

Hummer, W., Rosenberg, F., Oliveira, F., & Eilam, T. (2013, December). Testing idempotence for infrastructure as code. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing* (pp. 368-388). Springer, Berlin, Heidelberg.

Kumara, I., Garriga, M., Romeu, A. U., Di Nucci, D., Palomba, F., Tamburri, D. A., & van den Heuvel, W. J. (2021). The do's and don'ts of infrastructure code: A systematic gray literature review. *Information and Software Technology*, *137*, 106593.

Labouardy, M. (2021). *Pipeline as Code: Continuous Delivery with Jenkins, Kubernetes, and Terraform.* Manning Publications.

Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., ... & Lassenius, C. (2019). DevOps in practice: A multiple case study of five companies. *Information and Software Technology*, *114*, 217-230.

Mahanti, A. (2006). Challenges in enterprise adoption of agile methods-A survey. *Journal of Computing and Information technology*, *14*(3), 197-206.

Microsoft. (2022, August 19). *What is infrastructure as code (IaC)?* https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code

Microsoft. (2022, October 12). *What is DevOps?* https://learn.microsoft.com/en-us/devops/what-is-devops

Moore, G. A., & McKenna, R. (1999). Crossing the chasm.

Morris, K. (2020). *Infrastructure as Code.* O'Reilly Media.

Nuottila, J., Aaltonen, K., & Kujala, J. (2016). Challenges of adopting agile methods in a public organization. *International Journal of Information Systems and Project Management*, *4*(3), 65-85.

Palma, S. D., Di Nucci, D., Palomba, F., & Tamburri, D. A. (2020). Toward a catalog of software quality metrics for infrastructure code. *The Journal of Systems & Software*.

Pickard, A. J. (2017). *Research methods in information.* Facet Publishing.

Rahman, A. A., Helms, E., Williams, L., & Parnin, C. (2015). Synthesizing continuous deployment practices used in software development. *2015 Agile Conference* (pp. 1-10). IEEE.

Rahman, A., & Sharma, T. (2022, March). Lessons from Research to Practice on Writing Better Quality Puppet Scripts. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 63-67). IEEE.

Rahman, A., Farhana, E., & Williams, L. (2020). The 'as code'activities: development anti-patterns for infrastructure as code. *Empirical Software Engineering*, *25*(5), 3430-3467.

Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, *108*, 65-77.

Rahman, A., Parnin, C., & Williams, L. (2019). The seven sins: Security smells in infrastructure as code scripts. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 164-175.

Rahman, A., Rahman, M. R., Parnin, C., & Williams, L. (2021). Security smells in ansible and chef scripts: A replication study. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *30*(1), 1-31.

Red Hat, Inc. (2022, May 11). *What is Infrastructure as Code (IaC)?* RedHat: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac

Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J., & Männistö, T. (2016). DevOps adoption benefits and challenges in practice: A case study. *In International conference on product-focused software process improvement* (pp. 590-597). Springer, Cham.

Salvendy, G. (2012). *Handbook of Human Factors and Ergonomics.* John Wiley & Sons.

Silva, C. C., & Goldman, A. (2014, July). Agile methods adoption on software development: a pilot review. In *2014 Agile Conference* (pp. 64-65). IEEE.

Spinellis, D. (2012). Don't Install Software by Hand. *IEEE Software, 29*, 86-87.

Sultan, F. & Chan, L. (2000). The adoption of new technology: the case of object-oriented computing in software companies. *IEEE transactions on Engineering Management, 47*(1), 106-126.

Think Insights. (2022, November 26). *Crossing the Chasm – Technology adoption lifecycle.* https://thinkinsights.net/strategy/crossing-the-chasm/

Trochim, W. M. (2022, 11 28). *Research Methods Knowledge Base.* https://conjointly.com/kb/research-data-analysis/

Wadia, Y., Udell, R., Chan, L., & Gupta, U. (2019). *Implementing AWS: Design, Build, and Manage Your Infrastructure : Leverage AWS Features to Build Highly Secure, Fault-Tolerant, and Scalable Cloud Environments.* Packt Publishing Ltd.

Williams, C. (2007). Research Methods. *Journal of Business & Economics Research (JBER), 5*(3).

Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., & Soldani, J. (2020). The essential deployment metamodel: a systematic review of deployment automation technologies. SICS Software-Intensive Cyber-Physical Systems, 35(1), 63-75.

Appendices

Appendix 1. Data Management Plan

jamk

Data Management Plan: Adoption of Infrastructure as Code (IaC) in Real World

Author(s) of the thesis: Olga Murphy

Commissioner of the thesis: twoday Finland

1. General description of data

A table or list of data that you collect and produce or existing data and its properties (type, file format, rights to use the data, size):

- Survey data
- .csv format
- Webropol survey tool that was collected by contacting potential participants via email or messaging clients
- 27 responses

Controlling the consistency and quality of data:
The researcher to take multiple backups of the data before modifying it.
A copy base report data in Webropol.
Answers are also stored on OneDrive in pdf, word and xls formats.

2. Personal data, ethical principles and legal compliance

Does the data contain personal data (yes/no); measures related to their processing:
Yes, partially.

Other legislative and ethical factors in the thesis and related actions:
Email address that is collected with the respondent's consent. No additional actions are required.

Rights to use, collect, or continue to use the data, possible confidentiality, and related measures:
Data is collected confidentially. However, respondents can leave their contact information if they would like to receive the results of the survey. Researcher also asks explicitly whether respondents would like to be contacted for further potential interviews.

3. Documentation and metadata

The author of the thesis makes at least his / her own notes on how the data has been processed during the research. Method of implementation:

- Copy of the raw data is processed to provide the best result applying statistical analysis (descriptive statistics, T TEST and Reflexive TA)
- Before the analysis data was checked for validity and inconsistencies
- Data processing for the analysis included grouping and calculations

jamk

4. Storage and backup during the thesis project

Storage and backup of the data:
Storage on Webropol surveys & reporting data collection tool and OneDrive.

Controlling access to your data:
Access is allowed only for the researcher via institutional login.

5. Archiving and opening, destroying or storing the data after the thesis project

Possible archiving and opening of the data and descriptive metadata for reuse:
No

Data to be destroyed and method of implementation:
Data is to be destroyed after the completion of the thesis.
Data from all the sources (Webropol, OneDrive and local laptop) will be permanently deleted.

Data to be stored for authors and / or the commissioner and storing location:
Not stored

6. Data management responsibilities and resources

Responsibilities and possible resources available:
Student is responsible to deleting the data

Plan prepared (place and time): Jyväskylä, 14.11.2022

Appendix 2. Survey

jamk.fi

JAMK University of Applied Sciences

Adoption of Infrastructure as Code (IaC)

Hello! Thank you for getting curious and following this survey link.

If you are here then the author of this survey thought that you would be able to contribute to research about Infrastructure as Code (IaC). The goal of this survey is to collect some preliminary data to investigate how IT companies adopt IaC and what is their experience with the technology.

Answering this survey will take about 15 minutes. You may leave your contact details if you would like to receive results of this study.
The data is collected anonymously in any case.

1. You can leave your contact info here*

Email _____

2. Would you be interested in answering follow up questions after the survey?

◯ Yes

◯ No

* By leaving your email you are agreeing to be contacted by the researcher. Your contact information will not be shared with third parties and will only be used for direct communication regarding inquires or the results of the survey.

3. Which of these practices are you familiar with? *

☐ Agile

☐ Lean

☐ Scrum

- [ ] DevOps
- [ ] CI/CD
- [ ] Automated deployments
- [ ] TDD
- [ ] Code review practices
- [ ] Version control practices
- [ ] None of the above

4. Which of these practices does your team follow? *

- [ ] Agile
- [ ] Lean
- [ ] Scrum
- [ ] DevOps
- [ ] CI/CD
- [ ] Automated deployments
- [ ] TDD
- [ ] Code review practices
- [ ] Version control practices
- [ ] None of the above

5. Do you use cloud computing? *

- ◯ Yes
- ◯ No

6. What best describes your knowledge about IaC? *

- ◯ No knowledge

- ◯ I have heard something about it
- ◯ Only theoretical knowledge
- ◯ Hobby projects / completed cources
- ◯ Beginner work usage
- ◯ Advanced
- ◯ Expert

7. Have you ever used IaC to manage infrastructure in a project? *

- ◯ Yes
- ◯ Yes, I use it systematically
- ◯ No, but my colleagues have
- ◯ No

8. Do you think that IaC could benefit your team?

- ◯ Yes
- ◯ No

9. What barriers do you think have stopped your team from implementing IaC?

_____
_____
_____
_____
_____

10. How many other people in your team are aware about IaC practices?

- ◯ Everyone
- ◯ More than half of team/unit/company

- ◯ Less than a half of team
- ◯ Only a few
- ◯ None

If you would like to learn more about IaC, here is a good page to start reading about it - https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac.

11. Do you have knowledge sharing activities in your team? *

- ◯ Yes
- ◯ No

12. In what format does the learning happen?

_____
_____
_____
_____
_____

13. How often is the topic of automation processes mentioned in your team? *

- ◯ Weekly
- ◯ Monthly
- ◯ Sometimes
- ◯ Never

14. When was the first time you used IaC? *

- ◯ This year
- ◯ Last year
- ◯ 2 -3 years ago

◯ Over 3 years ago

◯ Other _____

15. What was the reason that you started using IaC?

_____
_____
_____
_____
_____

16. Who proposed the technology?

◯ Developers

◯ Management

◯ Customer

17. What do you use to implement IaC?

- [ ] Terraform
- [ ] Chef
- [ ] Puppet
- [ ] Ansible
- [ ] Bicep
- [ ] SaltStack
- [ ] Pulumi
- [ ] AWS CloudFormation
- [ ] Azure Resource Manager
- [ ] Google Cloud Deployment Manager
- [ ] Kubernetes
- [ ] Other _____

18. Where do you keep IaC code?

- ◯ Local
- ◯ Version Control
- ◯ Neither. Please, specify _____

19. What are the main difficulties that you have faced with IaC (if there were any)?

_____
_____
_____
_____
_____

20. What sources do you check to resolve difficulties and questions about IaC?

- [ ] Colleagues
- [ ] Books, articles
- [ ] Official documentation
- [ ] Blog posts
- [ ] Chats, stackoverflow
- [ ] Other. Please, specify _____

21. From your experience, what skills does a developer need to start with IaC?

_____
_____
_____
_____
_____

22. How would you rate the difficulty to start learning IaC?

23. How would you rate getting more advance knowledge of IaC?

24. How would you evaluate your overall experience with IaC?

25. Would you push your team to use IaC in the next project?

◯ Yes

◯ No

26. Other comments that you would like to share about IaC in your team

_____

_____

_____

_____

_____