Esko Salonen

Software Project Services using Infrastructure-as-Code

School of Technology and Innovations
Master's thesis in Technology
Software Engineering

Vaasa 2020

Preface

I would like to thank the case company for providing the topic and resources for the research as well as for all other interesting duties I have received during my employment. At the moment of adding my last words to the thesis, I would like to also express my gratitude to my colleagues for their continuous support and encouragement during my studies.

I would like to thank the “steering group” of the thesis for spending valuable working time directing my journey. Special thanks to Teemu Niemi for defining the topic, Olli Rajala for technical assistance and especially to Lassi Niemistö for reviewing my manuscripts and always providing valuable ideas and feedback.

Vaasa, 2.9.2020

Esko Salonen

UNIVERSITY OF VAASA
School of Technology and Innovations
Author: Esko Salonen
Title of the Thesis: Software Project Services using Infrastructure-as-Code
Degree: Master of Science (Tech.)
Programme: Software Engineering
Supervisor: Prof. Jouni Lampinen
Instructor: M.Sc. (Tech.) Lassi Niemistö
Year of Completing the Thesis: 2020 **Page count:** 68

ABSTRACT:

Infrastructure-as-code is a modern practice for IT automation as well as managing complex and large-scale infrastructure. It allows describing infrastructure and configuration in a code-like syntax easily understandable for any software developer. Unlike manual infrastructure configurations the infrastructure code can be versioned, tested, reviewed, and even compiled resulting in the actual desired systems infrastructure and configuration.

The objective of this thesis is to create a new software project establishment process for a medium sized software company. The aim is to automate and streamline a beginning phase of new projects by creating and documenting a process of establishing basic software project services by using infrastructure-as-code. A desired feature of the new process is to partly delegate the work of IT department to project teams who are most aware of the tools and services required for each project.

The thesis is divided into two parts. The first part presents relevant infrastructure-as-code and DevOps software development methodology related theory in a form of a literature review and the second part describes planning and implementing required infrastructure-as-code automations and the actual service establishment process for the case company.

The outcome of the study was a new automated project establishment process, in a form of two artifacts: process documentation and required infrastructure-as-code scripts. The study also produced a recommended way for handling the scripts and configuration files during the process, along with multiple other recommendations for similar implementations and for usage of infrastructure-as-code in general. Furthermore, results also included some recommendations for software manufacturers and a brief overview of infrastructure-as-code related risks.

Infrastructure-as-code was found to be a powerful enabler of automation and suitable also for workflow automation. However, creating infrastructure-as-code scripts and building a suitable development environment were found to be difficult. Also, several target system related challenges were identified during the implementation. The conclusion considering the service establishment process was that it is feasible to delegate only configuring the infrastructure-as-code implementation to the end users.

KEYWORDS: Infrastructure-as-Code, Software Engineering, DevOps, Ansible

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen yksikkö
Tekijä: Esko Salonen
Tutkielman nimi: Ohjelmistoprojektipalvelut käyttäen infrastruktuuri koodina -lähestymistapaa
Tutkinto: Diplomi-insinööri
Oppiaine: Ohjelmistotekniikka
Työn valvoja: Prof. Jouni Lampinen
Työn ohjaaja: DI Lassi Niemistö
Valmistumisvuosi: 2020 **Sivumäärä:** 68

TIIVISTELMÄ:

Infrastruktuuri koodina on moderni IT automaation sekä monimutkaisten ja suurten infrastruktuurien hallinnan käytäntö. Se mahdollistaa infrastruktuurin ja konfiguraation määrittelyn käyttäen ohjelmakoodia muistuttavaa syntaksia, joka on kenen tahansa ohjelmistokehittäjän ymmärrettävissä. Toisin kuin manuaaliset konfiguraatiot, infrastruktuurikoodi voidaan versioida, testata, katselmoida ja jopa kääntää, jolloin lopputuloksena saadaan luotua ja konfiguroitua määrityksen mukainen järjestelmäinfrastruktuuri.

Tämän opinnäytetyön tavoitteena on luoda uusi ohjelmistoprojektien tarvitsemien palveluiden perustamisprosessi toimeksiantajayritykselle. Tavoitteena on automatisoida ja virtaviivaistaa uusien projektien aloittamista luomalla ja dokumentoimalla uusi tavanomaisten ohjelmistoprojektipalveluiden perustamisprosessi, joka hyödyntää infrastruktuuri koodina -lähestymistapaa. Yksi uuden prosessin toivotuista ominaisuuksista on siirtää tietohallinnon työkuormaa osittain projektitiimeille, joilla on paras käsitys kunkin projektin vaatimista työkaluista ja palveluista.

Opinnäytetyö jakautuu kahteen osaan, joista ensimmäinen esittelee infrastruktuuri koodina -lähestymistapaan ja DevOps-ohjelmistokehitysmetodologiaan liittyvää teoriaa kirjallisuuskatsauksen muodossa. Toinen osa kuvaa uuden prosessin sekä siihen kuuluvien infrastruktuurikoodien suunnittelua ja toteutusta kohdeyrityksen toimeksiannosta.

Tutkimuksen lopputuloksena luotiin uusi automatisoitu projektipalveluiden perustamisprosessi, joka muodostui kahdesta artefaktista: prosessidokumentaatiosta ja vaadittavista infrastruktuuri koodina -skripteistä. Tutkimus tuotti myös suositeltavan tavan skriptien ja konfiguraatiotiedostojen hallintaan prosessin aikana sekä useita suosituksia vastaavanlaisten implementaatioiden luomiseen tulevaisuudessa ja infrastruktuuri koodina -työkalujen käyttöön yleisesti. Tulokset sisälsivät myös suosituksia ohjelmistovalmistajille sekä lyhyen katsauksen infrastruktuuri koodina -lähestymistapaan liittyvistä riskeistä.

Infrastruktuuri koodina todettiin tehokkaaksi automaation mahdollistajaksi ja sopivaksi myös työnkulkujen automatisoimiseen. Toisaalta skriptien kirjoittaminen ja tarvittavan kehitysympäristön rakentaminen havaittiin haasteellisiksi tehtäviksi. Tutkimuksen edetessä havaittiin myös useita kohdejärjestelmiin liittyviä haasteita. Johtopäätös palveluiden perustamisprosessin kannalta oli, että loppukäyttäjille on järkevää ja mahdollista delegoida ainoastaan skriptien tarvitseman konfiguraation tuottaminen.

AVAINSANAT: Infrastruktuuri koodina, Ohjelmistokehitys, DevOps, Ansible

Contents

Figures

Tables

Abbreviations

IaC	Infrastructure-as-code
IoT	Internet of things
CaC	Configuration-as-code
VCS	Version control system
YAML	Yet another markup language
CI	Continuous integration
CD	Continuous delivery
SaaS	Software as a service
SSH	Secure shell protocol
WinRM	Windows remote management protocol
OS	Operating system
VM	Virtual machine
POC	Proof of concept
REST	Representational state transfer
API	Application programming interface
UI	User interface
CSRF	Cross-site request forgery
HTML	Hypertext markup language

1 Introduction

Infrastructure-as-code is a modern approach to IT automation and continuous delivery as well as managing complex and large-scale infrastructure. It allows describing infrastructure and configuration using definition files written in code-like syntax easily understandable for any person who is familiar with software development, such as software designer or systems administrator. (Morris, 2016, pp. 5-6)

Using IaC tools has several benefits: changes to IT-infrastructure are less troublesome, discussing and documenting changes is easier, and users can be involved in the process of defining changes. One of the most useful aspects of IaC is the ability to treat infrastructure definition as any regular text file containing a piece of code: they can be stored to version control system (VCS), reviewed, tested, combined, edited, re-used, compiled, and automatically deployed. (Arundel, 2017, pp. 2-3; Morris, 2016, pp. 5-6; Spinellis, 2012)

The interest towards IaC is in most cases driven by a shift to more complex and large-scale IT systems as well as the rise of DevOps software development methodology (see Artač, Borovšak, Di Nitto, Guerriero, & Tamburri, 2017). DevOps-related requirements for service scaling, deployment speed, and uptime along with continuous delivery drive many software companies to search for new and better solutions for infrastructure management.

Infrastructure-as-code is still emerging innovation, but it has already been deployed and used at the core business of largest software companies around the world (Morris, 2016, pp. 5; Parnin et al., 2017, pp. 91). The outbreak of IaC has started from the large scale, because automation is the only decent way of handling infrastructures consisting of thousands or even millions of servers. Despite proving its usefulness at the leading edge, IaC practices are currently considered as ‘a best practice’ and therefore a recommended default setting for managing any size of IT infrastructure (Parnin et al., 2017, pp. 91).

1.1 Research Background and Motivation

This thesis is done as an assignment for a medium-sized Finnish software company that has an employer relationship with the author. The main businesses of the case company are software development services, digitalization solutions, and IoT. The IT department of the company is facing the problem of modern IT operations: constantly increasing number of different systems and environments and therefore increasing amount of work and required human resources. The company has started to look for a next level solution for configuration management and provisioning automation. In the phase of searching practices and patterns the company has decided to also investigate possibilities of IT workflow automation by harnessing beneficial features of infrastructure-as-code.

The company has selected Red Hat Ansible (see chapter 2.5) as the infrastructure-as-code platform, that will be taken into use. Ansible was chosen because it is already widely used, has extensive community and offers a large catalog of features and plugins. Furthermore, Ansible is designed to be agentless, which means that no client software is required to control managed hosts (Red Hat Inc., 2019b). The company also values the fact that Ansible is open source and the basic version is free also for commercial use.

1.2 Research Questions and Objectives

The objective of the thesis is to form a process for establishing software project services by using infrastructure-as-code. The aim is to streamline and automate the beginning stage of a new software project to gain competitive advantage by reducing lead times and workload of the IT department. Besides automation, the idea is to use IaC to partly delegate project tools establishment to employees who best know what is required in each case - project team members.

The initial idea of the process includes project team writing the definition of desired project services, IT-department reviewing the definition and then executing the approved definition to provision required software project services. Forming the process

requires creating instructions of building the definition for project team members, deciding a suitable platform for the review, and creating example scripts to technically verify the possibilities of IaC-based automation on required software project services.

The research process of this thesis consists of following phases (in chronological order):

1. Literature review

2. Designing and defining IaC-based automations and outlining the service establishment process

3. Implementing required Ansible playbooks (IaC scripts)

4. Designing the service establishment process

5. Documenting the process and reflecting the research process

In addition to creating and documenting the process, the study also aims answering research questions: “are the current IaC tools of the case company suitable for that kind of a purpose” (RQ1) and “what is the best available platform for handling IaC scripts during the process” (RQ2). The study also aims providing guiding information and recommendations for other similar implementations in the future.

The scope of the study is limited to a single process in a single case company. The study covers forming and documenting the service establishment process, but extensive testing and evaluation of resulting artifacts is left for further research.

1.3 Structure of the Thesis

The study is divided into two main parts: the first part defines and presents required theory and background of the application area. The theory is presented as a literature review and consists of separate chapters for infrastructure-as-code and DevOps. Both sections present the topics by using relevant literature, theory, and research.

The second part, which is done in a co-operation with the case company, presents the authors practical contribution to the subject. It consists of designing and implementing IaC-based software project services establishment automation and process for the case company.

Results, observations, and conclusions of the study are presented and described at the end of the thesis right after the second part. The final part also provides authors recommendations for relevant stakeholders and for similar implementations in the future as well as provides answers to research questions.

2 Infrastructure-as-code

Infrastructure-as-code (IaC) is a modern technology that is designed to help with operating high number of different systems and environments. It is a concept of infrastructure provisioning and configuration management tools that use simple text files, often called “recipes”, written in code-like syntax to define desired infrastructure environment and configuration state (Artač, Borovšak, Di Nitto, Guerriero, & Tamburri, 2017; Spinellis, 2012). Like a piece of traditional code, the “infrastructure code” or “recipe” can be compiled - the result is the desired infrastructure and configuration: networks, virtual machines, containers, and other resources deployed and configured according the definition (Farley & Humble, 2011, pp. 292; Hüttermann, 2012, pp. 135; Morris, 2016, pp. 5-6).

Configuration-as-code is a sister term for infrastructure-as-code, often mentioned along with or instead of IaC in related literature. In comparison to IaC, CaC as a term focuses more on configuration management aspect of IaC (see Morris, 2016, pp. 82). It is also often used as a marketing term for IaC-related products that lack some of the characteristic IaC features (Morris, 2016, pp. 82). Despite the configuration management orientation, it is also often referred as a synonym for infrastructure-as-code (see e.g. A. Rahman, A. Partho, P. Morrison, & L. Williams, 2018; Rahman, Mahdavi-Hezaveh, & Williams, 2019).

2.1 A Brief History of Managing IT Infrastructure

At the old times of IT, all software systems were tightly coupled with physical hardware: servers, storage devices, expansion cards, network adapters, switches, and routers (Farley & Humble, 2011, pp. 277). All these devices required distinct manual configuration and maintenance to keep systems online and running. Especially provisioning new infrastructure or deploying new services used to be very time consuming and laborious task – all changes and maintenance was required to be well

designed and planned beforehand because misconfigurations and system failures were expensive (Morris, 2016, pp. 4).

As time has passed, the tight coupling between systems and physical hardware started to slowly decay. By utilizing new technologies such as virtualization, containers, software-defined networking, cloud services, serverless technologies, and IT automation, systems started to be all the time less dependent of the hardware used beneath them (Morris, 2016, pp. 4). Harnessing new tools and technologies led to a significant reduction of effort and time consumed to maintaining current and provisioning new infrastructure and system environments.

As provisioning new systems become easier and faster, the amount of different systems started to increase rapidly and the traditional problem of IT departments, slow and laborious deployment changed to a problem of maintaining high number of different systems, devices, applications, and networks without a remarkable amount of repetitive manual work. At the same time different cloud services become more and more common and started to complicate and diffuse traditional IT infrastructure that used to be based only on private server hardware of each organization.

2.2 Common Problems of Manual Infrastructure Management

The traditional way of manually managing growing infrastructure very soon leads into different kinds of problems. As the infrastructure becomes larger and more complex, maintaining it starts to consume more and more resources, the risk of problems increases, resilience decays, and recovery becomes more and more difficult. Manual infrastructure management is often considered as an antipattern (e.g. Morris, 2016, pp. 153). This section describes three common problems of manual infrastructure management.

2.2.1 Server Sprawl

Modern Cloud and virtualization technologies have made deployment of new servers a lot easier and faster than it was before. In many organizations that has led to a situation of number of servers rising faster than the ability of IT team to manage them optimally. That can lead to notable wasting of resources when handling repetitive routine tasks such as installing updates without a proper automation. (Morris, 2016, pp. 6-7)

2.2.2 Configuration Drift and Snowflake Servers

Configuration drift is a phenomenon which starts to arise when one out of multiple identically configured servers receives manual configuration modifications because of a problem or update that cannot be deployed across whole server group. Over time these undocumented configuration modifications start to accumulate across the whole infrastructure and eventually that leads to each server having unique configuration that nobody is able to reproduce. A server containing that kind of configuration is called a snowflake server – a computer that 'should not be touched'. (Morris, 2016, pp. 7-8)

2.2.3 Fragile Infrastructure

Fragile infrastructure is a situation in which a snowflake server problem has spread over entire catalog of different systems. The infrastructure that consists of multiple snowflake servers is hard to maintain, has a low resilience against disruptions and failures and recovering it is painful and slow. (Morris, 2016, pp. 8)

2.3 Infrastructure Management Using Infrastructure-as-Code

By using infrastructure-as-code, it is possible to define the whole infrastructure and configuration using simple definition files (Morris, 2016, pp. 5). Using IaC for creating and maintaining the infrastructure and configuration helps to ensure that for every component there is all the time a complete set of instructions for creating and configuring it. Those infrastructure definitions are a rough documentation of the desired

systems state and they can then be used to easily re-create whole infrastructure or pieces of it when necessary (Arundel, 2017, pp. 3; Morris, 2016, pp. 7-8).

Using IaC helps maintaining changes, preventing configuration drift and snowflake server problem (see chapter 2.2.2). Also, deploying identical changes across multiple pieces of infrastructure is effortless and repeating requires minimal manual work (Arundel, 2017, pp. 3). That is an important advantage when dealing with rising number of systems and larger infrastructures.

Infrastructure definition and configuration for specific application can be saved together with a source code to the version control system and it can be deployed at the same way as the program itself (Morris, 2016, pp. 5). Managing changes is easier, because a VCS can provide a complete log of every change also for IaC files. In case of problems IaC and VCS can be very helpful when the infrastructure needs to be reverted to a previous (working) state (Arundel, 2017, pp. 4).

IaC together with a VCS is also a key enabler of more than one person working on changes to the same infrastructure. Using VCS reduces a risk of multiple developers creating conflicting changes. Also, the VCS ensures that a complete log of every change is always available: what was changed, who authored the change, and when it was created. (Arundel, 2017, pp. 4)

2.4 Use Cases for IaC

Infrastructure-as-code has already proven its usefulness while being used at the core business of many largest software companies, such as Google, Facebook, Twitter, PayPal, and Netflix (Miglierina, 2014; Morris, 2016, pp. 5; Parnin et al., 2017). Large companies do not publish detailed descriptions of which tools they are using and how, but it is obvious that IaC is an essential tool of running and provisioning global large-scale IT infrastructure.

In current literature infrastructure-as-code has two primary use cases: infrastructure automation for continuous integration/continuous delivery (CI/CD) (see e.g. Artač et al., 2017; Farley & Humble, 2011; Miglierina, 2014) and general IT infrastructure management (see e.g. Lavriv, Klymash, Grynkevych, Tkachenko, & Vasylenko, 2018; Morris, 2016).

IaC being used at CI/CD is often related to DevOps methodology (see chapter 3.3). Artač et al. (2017) discusses infrastructure-as-code as a vital element of DevOps, as well as Spinellis (2012), who considers configuration management tool as an essential DevOps enabler. The whole release engineering process and IaC as an integral part of it is described by Adams and McIntosh (2016) as well as in a book by Farley and Humble (2011).

From release engineering point of view, container technologies have been an innovation that has reduced a need for IaC (see Adams & McIntosh, 2016; Miglierina, 2014). However, there are also related processes that can benefit from IaC, for example creating and controlling the infrastructure for running containers as well as building and updating container images (Arundel, 2017, pp. 4, 170-171). Container technologies also have features that are related to IaC or even use the same idea of defining configuration in a form of code. For example, Docker tool has IaC-like codefiles (called “dockerfiles”), which are used to define containers (see Miglierina, 2014).

IaC being used as a tool for IT infrastructure management is often related to cloud environments and very large-scale infrastructure. For example, Miglierina (2014) describes multiple IaC-related tools and use-cases for cloud provisioning and deployment. The article is also related to DevOps, especially the operations part. Aiftimiei et al. (2017) describes cloud automation in general covering many different use-cases and tools on infrastructure-as-a-service cloud infrastructure. IaC driven cloud provisioning can be used to deploy applications, but also to conduct comparisons of cloud applications costs and performance on different service providers services as

described by Scheuner, Leitner, Cito, & Gall (2014; 2015). According to Lavriv et al. (2018) IaC could also be used to automate disaster recovery on cloud infrastructure.

2.5 Ansible

Ansible is a versatile infrastructure-as-code tool which is designed to meet multiple different use-cases. It can be used for IT automation, release engineering, building development environments, controlling large infrastructures, cloud provisioning, and many more purposes. (Red Hat Inc., 2019a)

On the contrary to many other IaC tools, Ansible is designed to operate without any Ansible-specific client software running on managed systems i.e. it is “agentless”. By default, Ansible uses OpenSSH for communication between control node and managed hosts. The lack of proprietary communication protocol and client software simplifies the command structure, reduces possible attack surface against managed systems, and enables zero resource consumption on managed hosts when no tasks are being conducted. (Red Hat Inc., 2019)

Ansible control node is also designed to be *stateless,* which implies it does not use a database or any other mechanism to save information of current state of the infrastructure. Each task performed on a managed host usually starts with a check phase which ensures that the operation is executed only if the desired state is not yet achieved. (Red Hat Inc., 2019)

To locate managed hosts on a network Ansible uses a list of hosts called *inventory*. The default inventory is a text file which defines hosts by hostnames or ip-addresses and host groups, that can be used to provision multiple hosts at once. Inventory files can also be used to set variables for individual hosts or host groups. Ansible supports several different inventory file formats and even dynamic inventories, which are automatically

fetched e.g. from a cloud service. (Red Hat Inc., 2019) A basic sample inventory file (e.g. hosts.ini) containing four hosts and two host groups is presented below:

```
testnode.corp.com

[webservers]
10.0.1.4
webserver1.corp.com

[databases]
10.0.1.5
```

The core of using Ansible is working on IaC scripts called playbooks. Playbooks are written in YAML syntax and they mainly consist of definitions and tasks. Playbooks can target a single host, a machine invoking the script, a group of hosts or all hosts in an inventory. A sample Ansible playbook (e.g. apache.yml) for installing Apache2 webserver on a webservers host group is presented below:

```
---
  - hosts: webservers # Define target hosts
    become: yes # Use root privileges

    tasks:
    - name: Install Apache2
      yum:
        name: httpd
        state: latest

    - name: Start and enable firewalld and Apache2
      service:
        name: "{{ item }}"
        state: started
        enabled: yes
      with_items: # Loop the task with these items
      - firewalld
      - httpd

    - name: Allow Apache access through firewall
      firewalld:
        service: http
        permanent: yes
        state: enabled
      notify: Restart firewalld # Trigger firewall restart
if the configuration was changed
```

```
    handlers:
    - name: Restart firewalld # Restart firewall if rules
were changed
      service:
        name: firewalld
        state: restarted
```

The sample playbook consists of four tasks. The first task ensures that Apace2 is installed. If an existing installation is not found on the current host, Ansible installs the package from package repository. The second task ensures that the Apache2 service is started and sets it to auto-start on reboot. The third task modifies the configuration of firewall daemon to allow http traffic between network and Apache2. The script also contains a handler for restarting firewall daemon to apply possible configuration modifications. The handler is executed only if the firewall configuration task actually made a modification to the firewall configuration.

3 DevOps

DevOps is a software development methodology that is commonly used among new projects at the case company. Therefore, it is a key to understand software project services that are the target systems of the new process. This chapter presents DevOps -related theory, main principles, relations between DevOps and IaC as well as the software project services in more detail.

3.1 DevOps for Software Development

"*DevOps is not a group and it is not a role. DevOps is a culture shift and a new way of thinking about how we develop and release software.*" (Kavis, 2014, pp. 153)

Traditionally organizations have divided their software production work to two different segments: software development and IT operations. Software development teams implement new features, write code, make changes, and fix defects whereas operations teams handle infrastructure and keep the software online, available, and operational. This kind of separation allows arise of barriers, silos, and communication blocks between different teams, because both kind of teams have different and sometimes conflicting missions. Development team aims creating changes as fast as possible while operations team tries to avoid changes to keep everything steadily functioning. (Hüttermann, 2012, pp. 5-6; Kavis, 2014, pp. 163-164)

DevOps is a software development methodology that aims combining these two segments: software development ("dev") and IT operations ("ops"). The core of DevOps is a set of practices that aim removing barriers, silos, and friction between these two functions to ensure fluent operation, collaboration, and communication. In many cases this involves "one team approach": all required experts sitting next to one table instead of traditional separate development and operations teams. (Farley & Humble, 2011, pp. 28; Hüttermann, 2012, pp. 4-5, 8-9; Kavis, 2014, pp. 163-165)

The four important aspects of DevOps are *culture, automation, measurement,* and *sharing ("CAMS")*. In DevOps methodology, people are considered more important than processes or tools, automation is used to enable fast and accurate feedback, quality is continuously measured, and knowledge such as best practices, ideas, and tools are shared among the whole team. (Hüttermann, 2012, pp. 4; Kavis, 2014, pp. 164)

The DevOps culture considers building software as a highly collaborative process: regardless of roles or titles, everyone in the project team is responsible for the whole system that is being developed and maintained. Everyone in the team is sharing the same objective and the responsibility for delivery and quality. Every team member should also be familiar with the whole system, not only the part they are working on themselves. (Kavis, 2014, pp. 164-165)

3.2 Continuous Integration and Continuous Delivery

Continuous integration is a common practice within DevOps. Instead of integrating work of each developer or development branch at the end of the software development, continuous integration obligates integrating the work to shared mainline as often as possible – even for every change. The code at the mainline should always be deployable, build successfully, and pass all tests. This is usually ensured by using a CI server which automatically builds and tests all modifications which get pushed to VCS. (Adams & McIntosh, 2016; Farley & Humble, 2011, pp. 55-56; Hüttermann, 2012, pp. 56-57; Kavis, 2014, pp. 168-170)

Continuous delivery takes the CI concept even further adding automated integration testing and automated deployment from development to production. This ensures that the software at the mainline of development stays always in a releasable state and changes and new features reach production stage as fast as possible. (Hüttermann, 2012, pp. 57; Kavis, 2014, pp. 168)

Continuous integration and continuous delivery are both important DevOps practices. They both improve the quality of the product by ensuring that each modification does not break a build and passes all test. CI/CD also ensures that decent testing is performed all the time during the development, not just at the end. They also move team culture to a desired direction towards developer's sense of ownership and responsibility for the code they are creating. (Kavis, 2014, pp. 168-169) The lack of CI/CD is sometimes even considered as an antipattern of manual release engineering (see Farley & Humble, 2011, pp. 5).

3.3 DevOps and IaC

A common use case for IaC within a DevOps project is managing infrastructure and configuration while doing continuous integration and -delivery. Both key environments, development and production, can be created, developed, and documented by using IaC. (Hüttermann, 2012, pp. 135-136)

The traditional way of handling environments involved every developer first creating own local development environments which were evolving quite a freely along the development. If a new developer entered the project team, he/she had to manually build own compatible local development environment after finding out the details from a colleague of from a documentation. (Hüttermann, 2012, pp. 136-137)

When a software was ready for deployment to production, the operations team stepped in and tried to find out the structure of the environment, which was used to develop the software by inspecting, asking questions, or reading documentation. After finding out the details, the team had to replicate them on production infrastructure as closely as possible. (Hüttermann, 2012, pp. 136-137)

While maintaining a software the same environment replication process had to be initiated again before each deployment – the operations team had to return and find out

how the environment had changed since last deployment and again replicate those changes on the previously built production environment. (Hüttermann, 2012, pp. 136-137)

Infrastructure-as-code automates the infrastructure handling. The code is at the same time a readable documentation and a recipe that can be run to build the environment. When the software is deployed to production or a new developer needs an environment for local development, the process includes usually only two steps: checking out the definition from VCS and compiling it to get the desired configuration for the development environment. (Farley & Humble, 2011, pp. 10; Hüttermann, 2012, pp. 137-138)

3.4 Software Project Services

The DevOps methodology is often seen only through related or purpose-built tools. Tools and automation have a role within DevOps, but the real power of the methodology is way more than any set of DevOps-labeled tools – it lies within people and processes (Hüttermann, 2012, pp. 11, 29). Software development tool manufacturers often like to label their products with a prefix “DevOps” but usually that only signifies that the tool can be used by both development and operations teams (Hüttermann, 2012, pp. 11).

Tools itself are not important, but as a general principle automation (which is often achieved by utilizing different tools) is (Hüttermann, 2012, pp. 29-30). Automation is used to reduce manual work and to ensure that specific actions such as running tests, building the project, and measuring performance are performed each time the same way (Hüttermann, 2012, pp. 30) . According to Hütterman (2012, pp. 29-30), it is recommended to have at least some automation for building, unit testing, acceptance testing, deploying, and configuring the application.

In the scope of this thesis the focus is mainly over tools and project services that support DevOps software projects. Therefore, it is important to define and describe what kind of tools are used, recommended, and required at the case company. In the following parts each of the most important tools and services are presented and shortly described in general level.

All presented tools can be provided from a private server of an organization, from a server of a cloud service provider or they can be purchased from outside as a turnkey SaaS product (for SaaS definition see Kavis, 2014, pp. 79-80; Mell & Grance, 2011). There are also multiple "stack" products that bundle multiple software project services as a single product or service, for example Azure DevOps (https://azure.microsoft.com/en-us/services/devops/) and GitLab (https://gitlab.com/).

3.4.1 Project Management Tool

Project management tool is used to plan, track, and manage project flow. An issue tracker is a simple project management tool, but usually they have more features like task boards, roadmaps, plans, visualizations, and reporting.

The project management tool of the case company is a versatile application. It can be used to support multiple different kinds of projects including scrum and kanban. The tool is usually used by all project team members to keep the status of a project up to date. It is also connected to other project tools to receive detailed information of the project state.

3.4.2 Documentation Tool

Documentation tool enables collaboration on documentation: creating, editing, and storing documents. Documentation tools - like the tool used at the case company – usually support multiple different types of pieces of documentation and multiple import

and export formats for various documents. They usually also provide features for organizing, indexing, and searching the documentation.

The documentation tool of the case company uses entities called spaces to organize documentation. Spaces have distinct permissions and they are created per project basis.

3.4.3 Version Control System

Version control system (VCS) is a key tool of software development, basically used to store source code and other artifacts related to a software project. A VCS enables multiple teams to work simultaneously on a source code of a single application by keeping a record of current codebase along with previous versions of all project files. He VCS tracks modifications and their authors. (Farley & Humble, 2011, pp. 32, 381-382)

Many modern VCS products have more features than only version control. Usually they also have a graphical user interface that allows viewing source code and conducting VCS related tasks. The VCS used at the case company does both.

3.4.4 CI/CD tool

CI/CD tool is a software connected to version control system. Each time a code change is committed to VCS, the CI/CD tool downloads a copy of the modified project, compiles it, executes automated tests, and reports results. A CI server is often used as an initial quality gate before the code is sent for a review or merged to a shared mainline. (Farley & Humble, 2011, pp. 63; Hüttermann, 2012, pp. 63-64)

The CI/CD tool can consist of a single software running on a single server or a separate controller software and multiple subordinate hosts located on distinct machines. These subordinate hosts (“build machines”, “build agents”) are used to execute actual CI/CD tasks orchestrated by a controller. The whole CI/CD system can also be provided SaaS

individually or as a part of a “stack” solution, which takes care of the controller and possible agent hosts.

The CI/CD tool of the case company consist of centralized controller and separate agent hosts. For information security reasons agent hosts are created and configured per project basis and they are each dedicated to only handle one specific project.

4 Designing Software Project Service Establishment Process

A current manual software project services establishment process at the case company consists of four high level steps visualized in Figure 1:

1. User creates a support request (ticket) describing a project that needs to be created. A case company has created a template that directs request to specific form and ensures that user provides all necessary details such as project name, participating users, access rights, and services that are needed.

2. The request is forwarded to a foreman or a project manager who needs to take a responsibility of the requested actions. He/she either approves or declines the request.

3. If the request is approved, it gets assigned to an IT department employee, who manually executes required actions based on provided project details. If an additional information is required, the IT specialist contacts an author of the request.

4. The request is marked as completed and the author is notified of successfully fulfilled request.

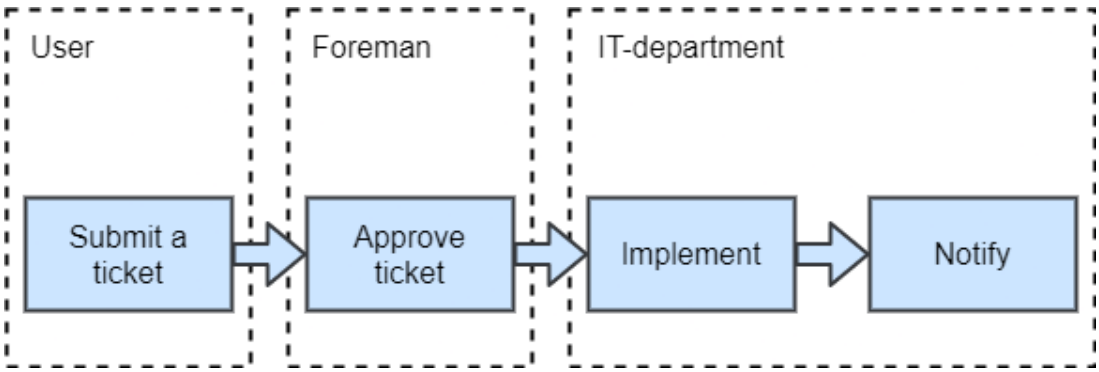


Figure 1. Current process flow.

The most time consuming and expensive phase of the process is step three. It consists of numerous sub-steps that all need to be executed manually by an IT specialist. These steps involve using multiple different user interfaces across multiple different systems, which makes it complex and difficult. From a technical point of view the manual implementation is the step that has the greatest potential to benefit from automation and therefore it will be in a center of focus in this research.

Because of the complexity, the case company has created a comprehensive and detailed internal documentation of the required actions. While designing the IaC implementation and overall automated service establishment process, the documentation will be used as a guideline and a main source of information.

A new process for establishing project services automatically was planned in co-operation with IT department and software development management. They both represent different views to the process (software development as a user and IT department as a provider) and therefore can provide valuable ideas and opinions for the design.

4.1 Target Systems

The target systems' architecture consists of multiple components that require interaction during project services establishment. Majority of these services are provided by the internal IT department in a software as a service manner (later referred as "SaaS services") and need only be provisioned and configured when a new project is started. For example, inside a project management tool a new project needs to be created, specific settings need to be applied and correct access rights need to be set before it is taken into use in a new project.

Build agents are the only component that requires a complete installation of all required software including the build agent software and its dependencies. Virtual machines for

build agents are created per project bases and each of them is required to be used only for building a single dedicated project. The OS of the build agents can be either Windows or Linux depending on the project. Both operating systems need to be supported.

In the scope of the thesis three first steps of the build agent installation, creation of a virtual machines, installing operating system and configuring control channel for IaC tool are omitted. An empty virtual machine containing only operating system and software required by the IaC tool is assumed to be present before the project establishment is started.

All components and a rough level architecture of the whole target system stack is presented at Figure 2. Most important communication channels between different systems are visualized by arrows.

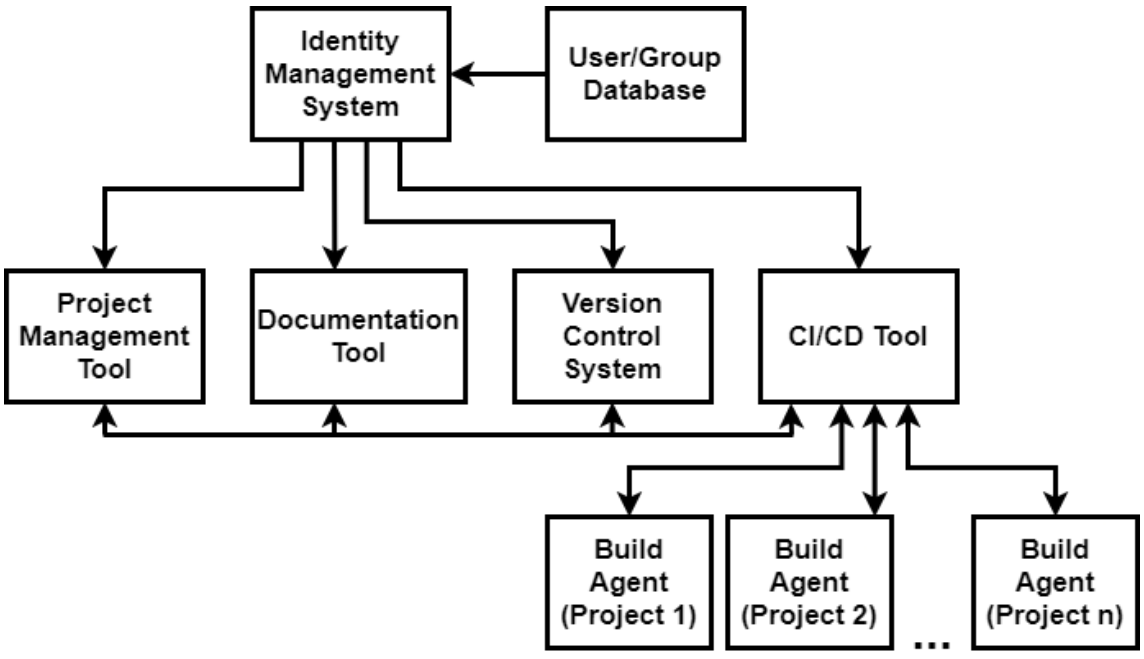


Figure 2. Architecture of target systems.

All target systems are controlled by a centralized IaC tool (Ansible control node) that will be used to execute all automated actions of the new process. Target systems are divided into three groups based on the communication channel that can be used: SaaS services, Linux hosts, and Windows hosts.

Services that are provided software-as-a-service do not allow any connections directly to the underlying server. These services will be controlled only through HTTPS by using different API endpoints and preferably REST API. Ansible has an official uri-module that allows playbooks to interact with webservices.

Linux and Windows hosts are handled by directly communicating with the operating system of the underlying server. For that purpose, Ansible supports a variety of different connection methods (see Red Hat Inc., 2019). In this case, Linux hosts will be controlled using Secure Shell (SSH) and Windows hosts using Windows Remote Management (WinRM). Ansible has different, however in many cases equivalent task modules for controlling both operating systems (Red Hat Inc., 2019).

Control- and feedback channels between IaC tool (Ansible control node) and all different services are visualized in Figure 3:

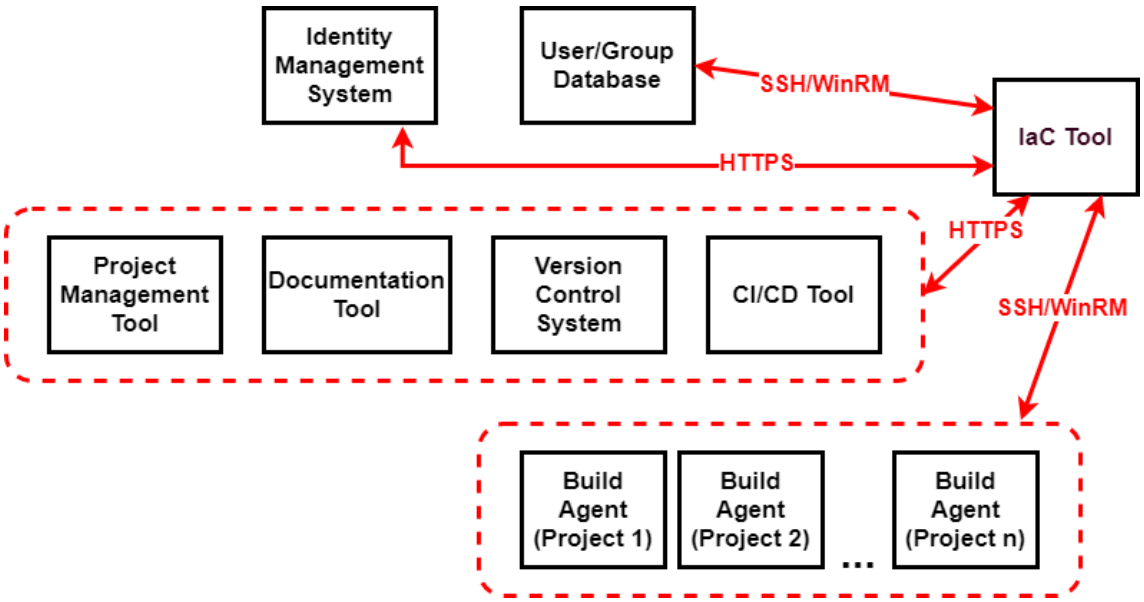


Figure 3. Control and communication channels of IaC tool.

4.2 Requirements

During the design phase several requirements for the implementation were identified and listed. These requirements were divided into three categories: functional requirements, nonfunctional requirements, and process requirements. All initial requirements together formed a requirements specification that was followed and updated during the implementation phase.

During the development phase (see chapter 5.2) the requirements specification was updated multiple times, mostly when new requirements were identified. All requirements were sorted to the relevant order and numbered after the final requirements specification was formed.

4.2.1 Functional Requirements

Functional requirements describe features that the infrastructure-as-code implementation must provide. Following functional requirements were identified during the design phase:

Table 1. Functional requirements.

Requirement identifier	**Requirement description**
FR1	Necessary user groups should be created to user/group database automatically
FR2	Users should be added to user groups automatically
FR3	User directory synchronization between user/group database and identity management system should be triggered before assigning any permissions

FR4	User directory synchronization between identity management system and each subordinate system should be triggered before assigning permissions in that specific system
FR5	A documentation space should be created automatically in the documentation tool
FR6	Documentation space permissions should be set automatically
FR7	A project should be created in the version control system automatically
FR8	Version control system project permissions should be set automatically
FR9	A project should be created automatically in the project management tool
FR10	Project permissions should be set automatically at project management tool
FR11	A project should be created automatically in the CI/CD tool
FR12	Project permissions should be set automatically at CI/CD tool
FR13	Build agents should be installed automatically
FR14	Build agents should be dedicated to a single project automatically
FR15	A git capability should be installed to an agent automatically if needed
FR16	A docker capability should be installed to an agent automatically if needed

4.2.2 Nonfunctional Requirements

Infrastructure-as-Code should be used as an automation tool for building required functionality. Characteristics of unattended IaC (see Morris, 2016, pp. 44-45) should be followed. The most important requirements are idempotency and failure visibility.

Idempotency is a property of an operation, which can be performed multiple times consequently without changing the result (Morris, 2016, pp. 44-45). For example, running an IaC script should be idempotent operation – even when executed multiple times the script should always result the identical desired state without causing problems such as duplicate entries.

Failure visibility in this case is a requirement that includes implementing checks for pre- and post-conditions of important tasks and notifying the user when a failure is occurred or when scripts cannot proceed successfully towards desired state for example because of unexpected initial state of a system (see Morris, 2016 pp. 44, 87-88).

Nonfunctional requirements for the IaC implementation are listed and described in following Table 2:

Table 2. Nonfunctional requirements.

Requirement identifier	**Requirement description**
NFR1	Ansible should be used as the IaC automation tool for implementing required scripts for the process
NFR2	All Ansible playbooks should be idempotent

NFR3	Checks should be implemented to detect failures, errors, and unexpected states
NFR4	Failures should be clearly indicated and reported to the end user
NFR5	When it is required to interact with a service instead of a managed host a REST API should be used if possible
NFR6	All attributes that might experience a change later should be easily configurable by the end user
NFR7	Ansible implementation needs to be able to control both Linux and Windows based build agents

4.2.3 Process Requirements

The most important feature of the process is usability. The process must not be too complicated for end user and the required effort at IT department needs to be as low as possible. The aim of the new process is to serve customers remarkably faster than previously by using manual efforts. The process needs to reduce the amount of required manual effort and working time for establishing project services at the IT department. Process requirements are presented in Table 3:

Table 3. Process requirements.

Requirement identifier	**Requirement description**
PR1	The process should be easy and usable for both the end user and IT department
PR2	The new process should consume less time and manual effort than the previous one

5 Implementing and Documenting the New Process

This chapter describes and presents the implementation and documentation phase of the new software project services establishment. The phase consists of building a development environment, creating required Ansible playbooks, building the establishment process, and writing the process documentation.

5.1 Development Environment

Building development and testing environments for IaC is often a difficult task. Testing infrastructure-as-code usually requires a separate emulated copy of whole target infrastructure or a targeted subset (Morris, 2016, pp. 207-208). Morris (2016, pp. 208) recommends building test infrastructure virtually so, that it is possible to easily revert it to a clear beginning state to avoid interference of previous test runs.

The implementation phase of the research was started by building a suitable development and testing environment for infrastructure-as-code implementation. It was clear from the beginning that no production services and servers could be used for that purpose. Instead, a local closed development environment including all required tools had to be created.

A first challenge was to find a suitable software to mimic production environment as closely as necessary. All open source software such as Linux OS was trivial to obtain and install, but commercial software caused more problems. Luckily, all software vendors offered trial licenses that could be used also for development purposes during their validity period. Eventually all required software products were successfully obtained to build the development environment.

The environment was built on top of virtual local area network (LAN) and virtual machines running on VirtualBox virtualization tool. VirtualBox allowed single installation

of both Windows and Linux to own virtual machines and then copying these template machines to form required amount of VMs. VirtualBox also had a useful snapshot feature that allowed saving previous states of each VM and returning to them when needed.

A first virtual machine had to be installed and configured manually, but after successful installation of Ansible control node, it could be used to perform all post-installation tasks to all following virtual machines including local installations of required SaaS services.

5.2 Implementing Project Establishment Automation

Implementation of infrastructure-as-code scripts started by creating a simple POC of managing a Windows host with Ansible. The case company had used Ansible to control only Linux machines and there was no previous experience of configuring WinRM or establishing communication with Windows. The POC was successful and the requirement NFR7 (*Ansible implementation needs to be able to control both Linux and Windows based build agents*) was initially found achievable.

5.2.1 Build Agent Installation

The first feature developed was an installation of a build agent on Windows host (FR13, NFR7). The development workflow was started by first gaining an understanding of the manual process. Therefore, the first build agent was installed manually by following the internal documentation of the case company. After understanding the manual installation process, it was implemented as an Ansible playbook.

Implementing the build agent installation was a straightforward task as well as connecting it to the CI/CD tool. After the implementation was ready, it was tested and handed to IT department to verify by using it to install a real build agent with it. During the verification, a couple of more requirements were identified. After installation tasks, dedicating the agent to a single project and installing a few most common capabilities were required to also be automated (FR14-16). These new requirements were addressed

by an implementation that used the official REST API of the CI/CD tool and interacted with local configuration files of agent software via Ansible lineinfile-module.

5.2.2 Provisioning of SaaS Services

The following step was to implement project establishment across all four SaaS services: creating a project to project management tool (FR9), VCS (FR7), and CI/CD tool (FR11) and creating a documentation space to documentation tool (FR5).

Despite being manufactured by the same company, all these tools required a different implementation and one could not be provisioned via official REST API. Instead an alternative way had to be invented to achieve required functionality (see chapter 5.3.2).

The next challenge was creating permission groups (FR1) and assigning correct permissions to all four SaaS services (FR6, 8, 10, 12). Three of the tools allowed permissions to be set through an API and one of the systems was lacking the API endpoint for assigning permissions. Again, an alternative way was required to be found to achieve the functionality.

Testing and developing the IaC implementation on SaaS services was more simplistic than developing build agent installation, because reverting virtual machines to an initial state was not required to conduct testing – instead creating a new project with another name was sufficient to clean the environment. Co-existence of different projects should not cause interference in any situation and it is a problem that must be solved if discovered.

5.2.3 Discovering a Workflow for Implementing Infrastructure-as-Code

During the development, a simple workflow for implementing automations with infrastructure-as-code was discovered. The workflow consists of five steps:

1. Identifying and defining a required action

2. Manual discovery of algorithm to perform the required action

3. Implementation of the algorithm as IaC script (Ansible playbook)

4. Testing the implementation at development environment

5. Verification of the feature by a customer (IT department in this case)

It has similarities with the waterfall software development model, but it is still an iterative process: if testing fails the process needs to be continued from implementation and if verification fails, the process needs to return back to step that must be reiterated to fix found deficiencies.

During the process, one might face problems implementing the algorithm by using an IaC tool. In that case the process must return to step 2 and another algorithm for the required action needs to be discovered if possible.

5.2.4 Introducing a Role-Based Playbook Structure

As the codebase got larger and larger, at certain point it was necessary to re-organize multiple individual scripts to a single entity. This was done by utilizing a role-based project structure, which is Ansibles default way of organizing complex and large playbook structures (see Red Hat Inc., 2019). Ansible roles are reusable components that can be used to build playbooks.

Multiple playbooks related to different functions of different services were split to separate roles. Eventually the implementation had separate roles for each SaaS system, a role for creating users and groups on user/group database, and three roles for installing

and configuring build agents: a role for Windows hosts, another for Linux hosts, and a generic role that points to a right role depending on actual detected OS of a target system.

Along with the role-based project structure a configuration procedure was also implemented. Eventually the configuration was performed by providing three different configuration files, each intended to serve different category of configuration options: project configuration, project specific technical configuration, and static technical configuration. A project configuration contains all options that can be defined by the end user. Both technical configurations will be provided by the IT department.

5.2.5 User Directory Synchronization

Ansible scripts were already partially tested during the development by the IT department to fulfill real-life project establishment tasks. During these tests, it was noticed that creating users and groups to user/group database is not sufficient. By default, the user directory is synchronized across services at certain intervals and during the establishment process the automation could not wait for the next synchronization. Instead, the process had to be triggered synchronously (FR3-4).

The forced user directory synchronization had to be implemented in two phases. First synchronizing the directory from user/group database to identity management system and after that synchronizing it to each four subsystems. Even though all these four systems are manufactured by the same vendor, they were observed to have major differences. All these systems eventually required a different implementation for triggering the synchronization by Ansible.

5.3 Implementation Challenges

Several challenges were faced during the implementation phase. This chapter presents the most significant problems and actions that were taken to solve or work around them during the implementation.

5.3.1 Computing Resources

During the development there were not enough computing resources available for the project. A development environment containing all components of target infrastructure (described in chapter 4.1 and Figure 2) was too resource heavy for a laptop computer used to run the virtual development infrastructure. Luckily, in this case it was possible to split the assignment into clearly discrete parts which were not dependent on each other.

Because of limited computing power the development and testing were performed in parts so that the focus was on specific smaller subset of the whole systems stack at a time while only relevant components were powered on and others switched off. Luckily, VirtualBox had a feature to suspend running virtual machine temporarily by saving the execution state to disk. The suspend feature mitigated the time required to switch between different development contexts. Target systems had some relationships between each other, but despite having limited resources it was possible to have all required systems powered on simultaneously while developing each part.

5.3.2 Restrictions of API Endpoints

During the development of Ansible implementation several difficulties were faced while trying to comply with requirement NFR5 (*When it is required to interact with a service instead of a managed host a REST API should be used if possible*). Many SaaS services that were contacted indirectly via HTTPS had very minor support of different API features. In many cases REST APIs provided did not contain endpoints for all actions that were required to conduct the service establishment. In these cases, other ways needed to be invented to successfully implement all desired functions by working around API deficiencies.

An important advancement with solving the problem was an observation that the UIs of different services use API-like endpoints to communicate with the service on the background. By binding directly to them it was found possible to mimic user doing

actions via the UI, which is a practice related to Robot Process Automation (for definition see Willcocks, Lacity, & Craig, 2015).

However, these internal endpoints of the applications were not designed to be used automatically: they had different features and data structures than real API endpoints, return values were not easily readable by a piece of software, and no documentation was available. Because the inspection of program source code was not possible, the only way to discover these endpoints was through runtime inspection of communication between the UI and the back-end service.

Using internal endpoints of the service also resulted in two new problems: authentication and cross-site request forgery protection. It was also noted that supporting scripts containing binds to these unsupported endpoints will very likely require more maintenance in the future as the interface compatibility is not guaranteed between different versions of the software.

Official REST APIs of the SaaS services typically use HTTP basic authentication to authenticate user and authorize access to the API. Because internal API endpoints are designed to be used only from the UI, the authorization is performed using a session token in a cookie header of each HTTP request. To use these endpoints, Ansible implementation had to start with a call to an endpoint that was used to authenticate users by username and password and assign the session token to a cookie. A couple of actions required also a second authentication to grant access to administrator features. The authentication could be automated by calling a separate endpoint with a payload containing administrator password. After successful authentication, the endpoint assigned a second session token for authorizing administrator actions.

Cross-site request forgery (XSRF or CSRF) is an attack against users of a webservice, that allows attacker to create arbitrary HTTP requests that the service considers as intended user actions belonging to a valid session. In many cases a successful attack involves

stealing a session identifier assigned during authentication or launching arbitrary request directly from a device of a victim. (N. Jovanovic, E. Kirda, & C. Kruegel, 2006)

To prevent CSRF attacks the manufacturer of the SaaS services has supplied products with a system that appends all forms and action hyperlinks with a per-request token, which is unique to each loaded page and allows invoking only the next possible user actions. Navigating between non-action pages does not require a valid token but on each request that results in an action the token is verified before the requested action performed.

While using internal endpoints the implementation had to be able to gather and handle the CSRF token. This was achieved by implementing additional steps before executing any actions:

1. Requesting a view that contains a link to a desired action(s)

2. Parsing the CSRF token from the HTML content of the response

3. Invoking a request to a desired action with a valid CSRF token

The most complex interaction flow between Ansible and an example SaaS target system is presented in Figure 4.

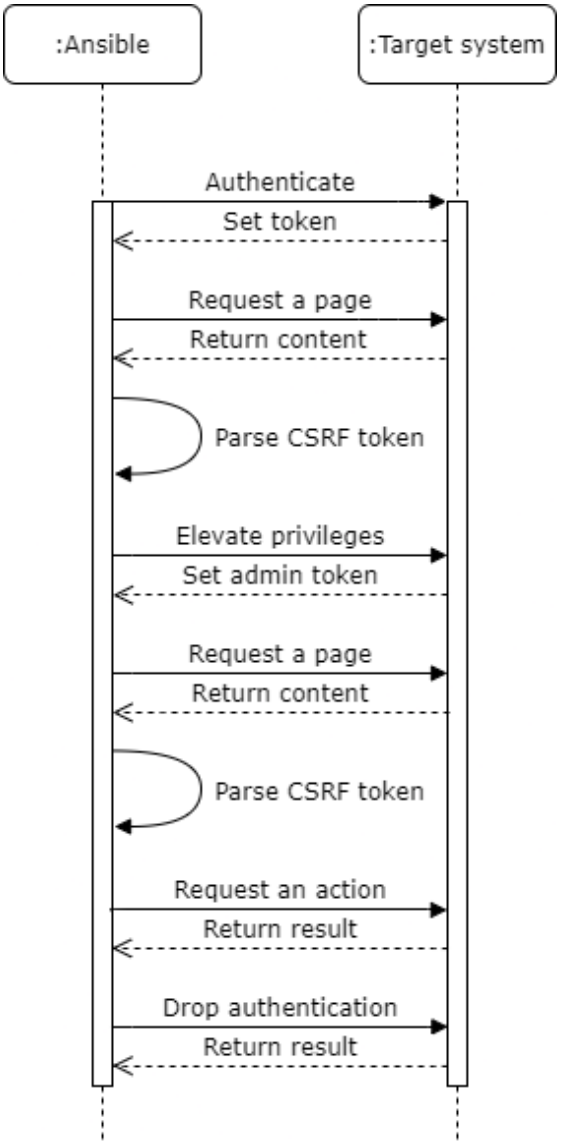


Figure 4. The most complex internal API sequence of a single internal API call.

5.3.3 Maintaining Idempotency

Idempotency was specified as a mandatory requirement for Ansible playbooks (NFR2: *All Ansible playbooks should be idempotent*). To comply with the requirement additional steps had to be implemented to almost all actions that were based on API calls to check the current status and take actions only when it is required. This is demonstrated in Figure 5, which shows an idempotent project creation flow that was used with multiple SaaS services.

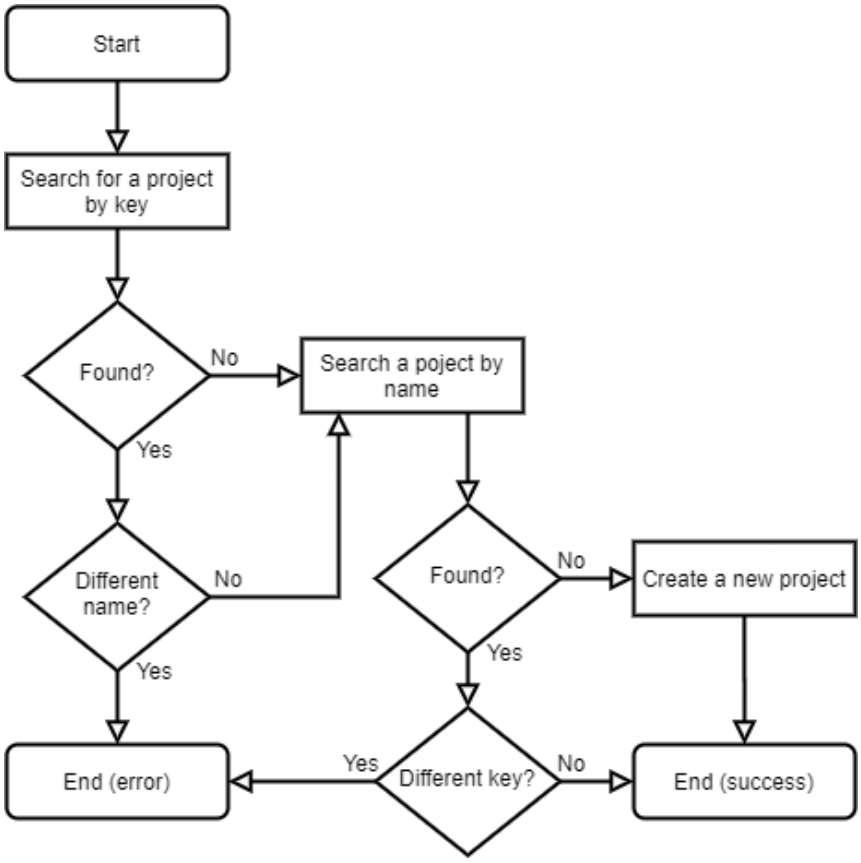


Figure 5. Idempotent project creation flow.

On each system the project has properties that must be unique: project key and project name. The project creation cannot succeed if the new project has identical properties with some existing project, thus the situation must be checked each time. If there already is a project that has both identical name and key the project is considered as the same project that already exists. In that case the system is already at a desired state. If the match is only partial e.g. a project with a correct name is found, but it has a different key, the script is unable to reach desired state and the error message is returned (as required by NFR4).

5.3.4 Obscure Permissions

While testing and deploying software project establishment scripts to production, several permissions related issues had to be solved on SaaS services. During the development, an account with system-wide administrator access rights was used to authenticate to all development systems inside development environment. For

information security reasons, production systems need to comply with least privilege principle (see J. H. Saltier & M. P. Schroeder, 1975), which disallows using full administrative privileges when possible. Instead in each system a separate user account should be created with as low access rights as possible to conduct the automated tasks.

Carrying out least privilege principle introduced several deficiencies in the documentations of multiple target systems. In many cases it was difficult to figure out minimal permission settings that allowed user to perform required actions inside each system. That led to additional work - conducting multiple tests to find out the correct permission settings for each specific action.

A part of systems did have only very broad access control options, while the least privilege principle would have greatly benefited for much more precise access control. Many target systems eventually required a use of a systems administrator user account with highest possible permissions, even though required actions were not assumed to require that high level of access.

5.4 Managing Scripts and Configuration

According to Morris (2016, pp. 182-183) using a version control system is an essential software development practice also for infrastructure-as-code. Even though on this study there was only a single developer working on IaC scripts, a VCS was used already during the development for storing Ansible playbooks and related configuration files.

Eventually the configuration for IaC scripts consisted of three files used for different purposes. The only end-user provided configuration file is project.yml. It contains all project settings that user can define while submitting a new project request. The technical configuration for the IT was split into two files: it-project.yml and it-static.yml. Project configuration file allows IT department to customize project specific settings such as build agent host addresses. Static IT configuration contains general static settings such

as usernames and passwords, addresses of target system, permission settings, and other non-project dependent settings.

At the beginning phase of the new proposed process, a configuration file that is provided by the end used is stored to a support request system of the IT department of the case company. The configuration file is saved along with the other information to the request system and retained also after the process is finished. Practice of using and storing tickets to the support request system is identical to the current process and therefore it is easy to adapt to the new workflow.

Storing the user defined part of the configuration is trivial, but the technical part requires more attention. The IT provided static technical configuration file contains routine setting such as API addresses of the services, but also sensitive data such as administrator account usernames and passwords. Therefore, it is important to store it properly, for example not directly to a VCS as a plaintext (Morris, 2016, pp. 183). For storing secrets, Ansible has a feature called Vault, which can be used to encrypt files that contain secrets. After encryption it should be reasonably safe to store ciphertext configuration files to a VCS (Red Hat Inc., 2019).

5.5 Project Establishment Process

During the implementation phase it was observed that creating required IaC scripts is a complicated task. It requires considerable knowledge of Ansible as a tool and playbook structure as well as an extensive development environment. Therefore, it was found obvious that to comply with a requirement PR1, it is not feasible to delegate responsibility for creating these scripts to end users. Instead, the IaC implementation was divided into multiple modules that could be easily configured by using a separate configuration file, which is simple enough to be created and modified by the end user (see NFR6). The configuration file is also easy to store to a support request system as an attachment of a new project request.

The new proposed software project establishment process is presented at Figure 6. The process flow is highlighted with blue color and required artifacts with gray. Dotted squares behind the process present the main actors of each process phase.

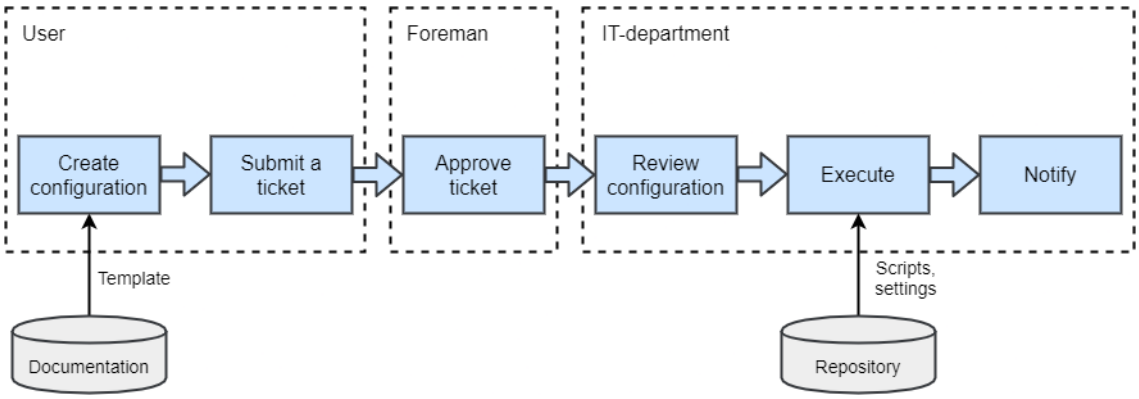


Figure 6. New process flow.

The process is started by user who is a responsible person of a new project. The user creates a project service configuration based on a template provided by a process documentation (see chapter 5.6 and appendix 1). Then the configuration is submitted along with a support ticket to a support request system. The system automatically forwards the ticket to foreman, who either approves or denies it.

If the request ticket is approved, it is moved to a task backlog of IT department. When the handling of the request starts, the first step is a review. The configuration is checked to ensure it is feasible, the syntax is correct, and all relevant details have been provided. When a review is passed, an IT-specialist clones IaC scripts and a separate technical configuration from a code repository, appends user configuration and executes the playbook. After successfully handling the request, a ticket is closed and the user is notified.

If the project requires more services/tools in the future, the user can use the original configuration as a template, add configuration for new tools and pass it to the establishment process again. Due to the idempotency of the IaC scripts only the required changes are performed while existing tools are left intact.

The process could be taken into use in two phases with a transition time. In the first phase only the manual provisioning of different systems in current process is replaced with an equivalent automated phase from the new one. This involves IT department to temporarily create all required configurations for project establishment but enables safe evaluation of automations and resolving possible issues. Even though an additional step is required, it is very likely that the IT department workload is already reduced without any visible changes for end users.

The second phase involves replacing the whole process with a new one by publishing the process documentation and requiring users to create project configuration files. If there are any serious problems encountered during the second phase, it is always possible to roll back to manual establishment process by using the configuration file as a source of project information for manual project establishment.

5.6 Documenting the Process

The process documentation was created on a basis of the process developed at the implementation phase. The process was documented to a brief instructions document that could directly replace the existing piece of documentation at the intranet of the case company when the new process is fully taken into use. The documentation is provided as an appendix 1.

The documentation provides user with general information on the process along with two document templates: one for creating a ticket and another for creating IaC project configuration file. The process is presented as a figure at the end of the documentation. In addition to instructions of new project establishment, the documentation contains also general information on software project services and use of different tools at the case company. These parts however are not relevant for the study and therefore they are omitted from the presented documentation.

The writing style of the documentation was adopted from the previous documentation which also was brief and aimed providing only the required necessary details for the end users. The aim was to use figures and keep the text as brief as possible, so that it is easy for users to find out the relevant information.

6 Results and Observations

Conducting the study gave the author a good position to observe forming the new process and gain technological knowledge of implementing IaC. This chapter presents results of the study along with observations and recommendations by the author. Also, a brief overview of IaC-related risks is included at the end.

6.1 Observations During the Study

The holistic conclusion of the literature review was that automation in general is a practice that should be harnessed to an increasing extent. The argument got support from IaC-related literature (see e.g. Morris, 2016, pp. xvi; Spinellis, 2012) as well as from the DevOps side (see e.g. Hüttermann, 2012, pp. 8, 29; Parnin et al., 2017). Lack of automation is generally considered as an antipattern by e.g. Morris (2016, pp. 153) and Faley & Humble (2011, pp. 5). Automation is the desired path also inside the case company who is seeking for new practices and tooling to minimize repetitive or laborious manual tasks and to release human resources to more meaningful tasks.

During the practical part of the study infrastructure-as-code was found to be a powerful enabler of automation and Ansible a versatile IaC tool. A rise of container technologies has reduced the use of IaC in release engineering – also in the case company – but it is still a valid technology for multiple automation purposes (see e.g. Arundel, 2017, pp. 2-5). IaC was also found to be a suitable for automating SaaS services. However, building that kind of capabilities requires more effort and might have notable challenges. IaC and Ansible were also found suitable for workflow automation – at least in the case presented in this study.

Considering required infrastructure environment, developing and testing IaC scripts was found to be a troublesome task. In this case it required a complete replica of target systems, which introduced problems with licenses, insufficient computing resources and replicating the manually created environment. According to Artač e.al (2017) IaC should

make testing easier, but in this case the target systems were not initially provisioned and installed automatically and therefore the development environment had to be created manually. This caused multiple differences and a configuration drift between production target systems and development environment. Building and maintaining this kind of environment was time consuming.

Creating IaC scripts in general was found to be a complicated task. Even though the syntax is simple and most functions easy to understand, creating complete IaC scripts is a task that cannot be easily delegated to a person not familiar with the technology – even if the person was a software developer. Distributed development would cause also other issues such as high consumption of human resources at review phase. Also, the need of a development infrastructure drastically reduces a feasibility of that kind of process. Developing scripts blindly without a development environment and testing possibilities is not reasonable. Therefore, creating the system as ready-made discrete building blocks that can be only enabled or disabled and configured by the end user was selected as the best available option during the study. Luckily, the objective and target systems were suitable for that kind of approach.

6.2 Recommendations for the Case Company

The study suggests that the case company puts the new software project establishment process into operation by using two-phase transition: first using only the automated software project services provisioning at IT department and after a transition time fully establishing the new process and involving end users.

Automation is all the time more and more important enabler of competitive advantage in IT business. The study suggests that the case company should keep on investing in automation also in the future. There are a lot of processes and workflows that could and should be automated to reduce lead times and repetitive manual work – especially among the infrastructure management, which is clearly a strength of IaC.

IaC provides great tools for many automation purposes, but it is not the only option. It is obvious that the recommendation for the case company is to utilize it even more, but also other technologies and approaches need to be taken into account while planning new automation projects. Especially for workflow automation there are multiple alternative tools, such as UiPath (https://www.uipath.com) and Robot Framework (https://robotframework.org).

During the study there came up already initial ideas of improving the process even further for example by developing the workflow towards a completely VCS-based process instead of the one based on IT support request system. This kind of thoughts and ideas to improve the process even further are important. The author of the study suggests that the case company continues improving the process even after this study is finished. The most important thing is to maintain resources for this kind of development work also in the future.

6.3 General Recommendations for Other Similar Implementations

Infrastructure-as-code and Ansible were found suitable for workflow automation in the presented case with some limitations. Special attention needs to be paid to the process usability for end users, who cannot be expected to perform too complicated tasks. In the context of this thesis, the implementation of Ansible playbooks was dedicated to a developer familiar with the technology, and only creating a simple configuration file was assigned to end users. This is a recommended practice also for other similar implementations involving end users.

Infrastructure-as-code is not an only option for implementing this kind of automations. Automated project establishment could have been also implemented by using alternative technologies such as shell scripts or Python programming language (https://www.python.org). However, then some of the benefits of IaC would have been lost. For example, executing tasks that need to be performed directly on servers would

have been more complicated and a developer should have addressed idempotence on all parts of the task chain. Provisioning SaaS services, however, is less technology dependent and it already required special measures to maintain idempotence (see chapter 5.3.3).

It may be possible to use third-party tools with Ansible. The implementation presented in this study contained parts that could also be built by partially using e.g. a third-party tool and/or another programming languages. For example, handling REST API calls to SaaS services could have benefited from another tool purpose-built for requesting web resources. In this case API calls were all conducted by using Ansible uri-module, but for new similar implementations it is recommended to evaluate alternatives for example Selenium (https://www.selenium.dev) The most important feature of a third party tool would be the ability to integrate to the software which is used to orchestrate the automation (in this case Ansible).

If a project is a commercial one and if it contains integrations to SaaS services, it is important to evaluate and consider API support of target services while estimating the amount of work required. During the implementation phase of this study, the lack of API features (see chapter 5.3.2) caused remarkable amount of additional work and effort for implementing alternative ways to achieve required functionality that had no supported endpoints available at target systems APIs.

6.4 Recommendations for Software Vendors

The most troublesome problems faced during the study were caused by deficient API features of target systems. Therefore, it is suggested that companies manufacturing development tools should pay attention to the extent of API features and the ability to automate actions on their products. Good automatability will likely become an increasingly important feature in the future. Not only is automatability beneficial for

end-users, but also for software manufacturers who are required to test their products. The same automation capabilities could be used for conducting automated testing.

During the study, problems with access rights management were also encountered on target systems. A recommended access rights management approach would be consistent and clear instead of complex and obscure. The study suggests all software vendors to implement clear and precise access control mechanisms to ensure their products can be secured and configured according to least privilege principle. It is not enough to let an administrator only choose a single permission for example for all administrative functions. When automation is used to perform specific tasks or there are multiple persons responsible for different administrative functions, it is beneficial to be able to restrict permissions to allow only required actions.

Licensing is also an area where increased automation should be better considered. It is recommended that users of a software product could have a permission to build separate environment for automation development and testing without additional costs. Respectively, production environment licenses should include a permission to have a few additional user accounts dedicated for automation purposes.

Vendors of IaC tools should investigate the need and evaluate the current extent of features related to REST API connectivity and improve them if it is found necessary. At least features provided by Ansible were found sufficient for the purposes of this study, but there is always room for improvement. One concrete suggestion for Ansible developers would be providing a development tool that could help with building and testing REST integrations. The tool could be similar to existing graphical REST API clients that allow exporting the current request as a shell script or a programming language implementation.

6.5 Risks of Using IaC

Infrastructure-as-code like all other automation practices involves risks that need to be considered while planning IaC-related projects or deployment of IaC tools. This chapter shortly presents risks of using IaC from the perspective of the author supported by literature and the practical experience gained during the research process.

IaC tools are very powerful and they can be used to control infrastructure in a very large scale (Parnin et al., 2017, pp. 91). Because of the great power there is also a possibility of disastrous mistakes and propagating smaller mistakes to a large amount of different systems (see e.g. A. Rahman, 2018). The risk is high especially if the scripts are developed carelessly and tested poorly. Therefore, it is necessary to invest in testing and building proper test environments for IaC development, even though it might be a challenging task.

Automation and using IaC tools require investments. Even when the used tool is free, using an IaC tool to implement an operation is likely to consume more time than performing identical actions manually on a single server once. Using IaC also creates work time overhead by requiring an extensive development and testing environment. Even though IaC is a recommended practice, it is likely that in some rare corner cases IaC can be cost-ineffective alternative in a comparison with manual actions or other automation tools. Therefore, a company planning to utilize this kind of tool is required to evaluate reduced productivity in terms of several benefits such as documentability, repeatability, ability to re-use, and traceability.

Also, selecting a tool for IaC is an important decision. After investing high amount of effort to make the most of specific tool, it is hard and expensive to switch to another. The cost of change consists of the price of the tool itself and the effort required to operate it: installing the tool, installing possible remote agent software as well as implementing, testing, and deploying scripts and other automation capabilities. Because

changes are expensive, it is important to evaluate and select the toolset carefully before purchasing required software and deploying IaC in large scale.

7 Conclusions

Infrastructure-as-code is a modern approach to infrastructure automation and configuration management. IaC tools can be harnessed far beyond automatizing only traditional administration of virtual servers at IT department - they can be useful and beneficial also for many other areas in a software company.

Among infrastructure-as-code and DevOps-related research and literature numerous examples of preceding IaC applications were presented. Most of existing use cases were related to IT automation and establishing continuous integration and delivery, but the study demonstrated that IaC has also possible use cases in the field of workflow automation.

The thesis presented a feasible process of setting up software project services by using infrastructure-as-code -related configuration files written by project responsible persons. The proposed process consisted of project responsible filling a specific configuration file template describing desired project services, IT-department reviewing the configuration, and after successful review IT-department deploying the approved definition to required environments without manual effort. The process was formed in the scope of single case company.

Results of the study also provided a brief overview to risks of IaC and several recommendations, which were divided into three categories based on target: recommendations for the case company, for implementing other similar processes, and for software vendors. The most important recommendation was to put the proposed software project services provisioning process into operation at the case company. The study proposed a two-step transition from current manual process to the new process.

Recommendations for similar implementations in the future concerned mostly process design, selecting the toolset and evaluating target systems. Software vendors were

suggested to improve automation possibilities and access control mechanisms on their products based on challenges faced during the research.

The thesis also presented answers to two research questions described at the beginning (see chapter 1.2). The first research question (RQ1: “are the current IaC tools of the case company suitable for that kind of a purpose?”) concerned suitability of Ansible to workflow automation. Ansible was found very powerful and practical automation tool as well as suitable for automating the presented process in the context of the case company. The general conclusion of IaC suitability to workflow automation still requires further research.

The second research question (RQ2: “what is the best available platform for handling IaC scripts during the process”) was not answered directly. During the development, the focus of user interaction shifted from creating IaC scripts to the filling a configuration template. Because the user configuration consists only of a single file, a conclusion was made not to propose any new tool. Instead, the current support request system of IT department was found suitable for the purpose. The ready-made IaC implementation was handled by using an ordinary VCS.

During the study, several new research questions and topics for further research were discovered. As previously stated, to give a general conclusion of IaC as a tool for workflow automation more study on the topic is required. This thesis presented one possible use scenario, but because preceding literature did not provide more examples of IaC being used to that purpose, more research is needed on the topic.

Currently there are multiple different IaC tools available from different vendors. That is why researching different alternatives at least in terms of features, performance, and suitability for different purposes would be reasonable. That kind of knowledge could provide help for different parties considering the deployment of an IaC tool.

Also, inter-compatibility between IaC tools and different additional tools would be a research subject worth of consideration. For example, there are multiple different REST API drivers available, some of which might be more efficient to develop and use than e.g. the default Ansible uri-module. Also, compatibility of IaC and non-IaC automation tools could be explored and researched.

References

A. Rahman. (2018). *Anti-patterns in infrastructure as code.* Paper presented at the 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), 2018. https://doi.org/10.1109/ICST.2018.00057

A. Rahman, A. Partho, P. Morrison, & L. Williams. (2018). *What questions do programmers ask about configuration as code?* Paper presented at the 2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE),

Adams, B., & McIntosh, S. (2016). *Modern release engineering in a nutshell -- why researchers should care.* Paper presented at the IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Klagenfurt, Austria, 2016. https://doi.org/10.1109/SANER.2016.108

Aiftimiei, C., Costantini, A., Bucchi, R., Italiano, A., Michelotto, D., Panella, M., . . . Salomoni, D. (2017). Cloud environment automation: From infrastructure deployment to application monitoring. *Journal of Physics: Conference Series, 898*, 82016. ISSN 1742-6588. https://doi.org/10.1088/1742-6596/898/8/082016

Artač, M., Borovšak, T., Di Nitto, E., Guerriero, M., & Tamburri, D. (2017). *DevOps: Introducing infrastructure-as-code.* Paper presented at the IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, 2017. https://doi.org/10.1109/ICSE-C.2017.162

Arundel, J. (2017). *Puppet 5 beginner's guide* (3rd ed. ed.) PACKT Publishing. ISBN 978-1-78847-290-6

Farley, J., & Humble, D. (2011). *Continuous delivery: Reliable software releases through build, test and deployment automation*. Boston: Pearson Education, Inc. ISBN 978-0321601919

Hüttermann, M. (2012). *DevOps for developers* (2012th ed.). USA: Apress. ISBN 978-1430245698

J. H. Saltier, & M. P. Schroeder. (1975). Protection of information in computer systems. *IEEE CSIT Newsletter, 3*(12), 19. ISSN 2379-5654. https://doi.org/10.1109/CSIT.1975.6498831

Kavis, M. (2014). *Architecting the cloud: Design decisions for cloud computing service models (SaaS, PaaS, and IaaS)*. Hoboken, NJ: John Wiley & Sons. ISBN 978-1-118-61761-8

Lavriv, O., Klymash, M., Grynkevych, G., Tkachenko, O., & Vasylenko, V. (2018). *Method of cloud system disaster recovery based on "infrastructure as a code" concept.* Paper presented at the 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET), Lviv - Slavske, Ukraine, 2018. https://doi.org/10.1109/TCSET.2018.8336395

Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing*. Gaithersburg, MD: Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology.

Miglierina, M. (2014). *Application deployment and management in the cloud.* Paper presented at the 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 2014. https://doi.org/10.1109/SYNASC.2014.63

Morris, K. (2016). *Infrastructure as code: Managing servers in the cloud* (1st ed.). Sebastopol: O`Reilly Media, Inc. ISBN 978-1491924358

N. Jovanovic, E. Kirda, & C. Kruegel. (2006). *Preventing cross site request forgery attacks.* Paper presented at the Security and Privacy for Emerging Areas in

Communications Networks (SecureComm), Baltimore, MD, USA, 2006. https://doi.org/10.1109/SECCOMW.2006.359531

Parnin, C., Helms, E., Atlee, C., Boughton, H., Ghattas, M., Glover, A., . . . Williams, L. (2017). The top 10 adages in continuous deployment. *IEEE Software, 34*(3), 86-95. ISSN 0740-7459. https://doi.org/10.1109/MS.2017.86

Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology, 108*, 65-77. ISSN 0950-5849. https://doi.org/10.1016/j.infsof.2018.12.004

Red Hat Inc. (2019a). Ansible documentation. [An official online documentation for Ansible IaC tool]. Retrieved 18.5.2020 from https://docs.ansible.com/ansible/latest/

Red Hat Inc. (2019b). Why ansible? [A product description on a web page of a software manufacturer]. Retrieved 3.1.2020 from https://www.ansible.com/overview/it-automation

Scheuner, J., Cito, J., Leitner, P., & Gall, H. (2015). *Cloud WorkBench.* Paper presented at the 24th International Conference on World Wide Web, Florence, Italy, 2015. https://doi.org/10.1145/2740908.2742833

Scheuner, J., Leitner, P., Cito, J., & Gall, H. (2014). *Cloud work bench -- infrastructure-as-code based cloud benchmarking.* Paper presented at the IEEE 6th International Conference on Cloud Computing Technology and Science, Singapore, 2014. https://doi.org/10.1109/CloudCom.2014.98

Spinellis, D. (2012). Don't install software by hand. *IEEE Software, 29*(4), 86. ISSN 0740-7459.

Willcocks, L. P., Lacity, M., & Craig, A. (2015). The IT function and robotic process automation. *LSE Research Online Documents on Economics, London School of Economics and Political Science*. https://EconPapers.repec.org/RePEc:ehl:lserod:64519

Appendices

Appendix 1. Process Documentation for End Users

Requesting a new software project

New projects are requested by submitting a ticket through IT Servicedesk. A desired ticket template:

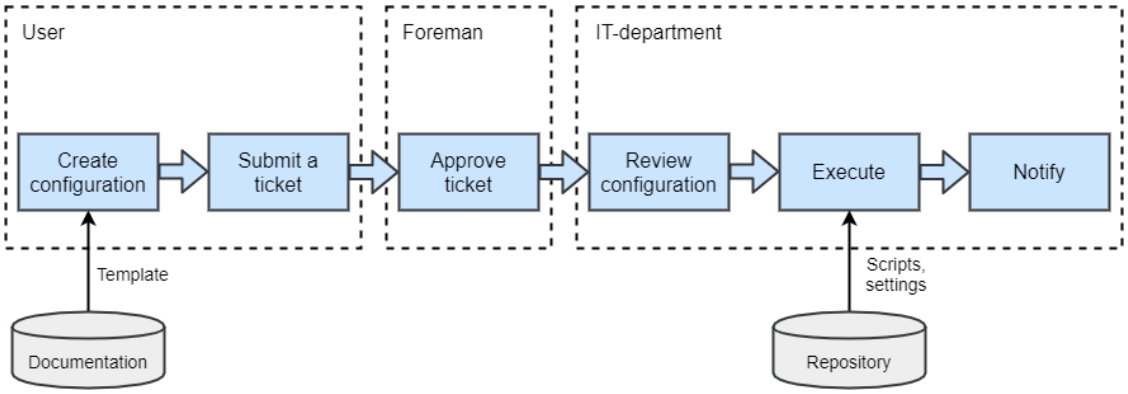
```
Hello Service Desk,

I need a following new project to be created:
Project name: FILL
Customer name: FILL

Access: By default, team leaders get access to all projects.
If this is not wanted, please indicate it in the request.
External users: FILL (full name, email, company, and public
IP from which the external user will be accessing)

Build agent resources (if needed): FILL <e.g. 2 vCPU, 4Gb
vRAM, Windows>
Build agent capabilities (excl. git and docker): FILL
```

Project tools (project management tool, documentation tool, VCS, and CI/CD tool) are provisioned by automatically processing a configuration file. Please provide a configuration file called project.yml as an attachment of your request. Use YAML syntax and a following template:

```
---
# Project configuration

project_name: ''
project_key: ''
project_description: ''

# ***** Users *****
configure_users_and_groups: yes # yes/no

project_admins:
  - admin1 # Usernames
```

```
project_developers:
  - developer1 # Usernames
  - developer2

# ***** Documentation Tool *****
documentation_tool_enabled: yes

# ***** Project Management Tool *****
project_management_tool_enabled: yes

pmt_project_lead_username: ''

pmt_project_type: 'software' # software/business based on
selected template
pmt_project_template: 'gh-scrum-template'

# Software templates:
#  gh-scrum-template
#  gh-kanban-template
#  basic-software-development-template

# Business templates:
#  core-project-management
#  core-task-management
#  core-process-management

# ***** Version Control System *****
vcs_enabled: yes

# ***** CI/CD Tool *****
cicd_tool_enabled: yes

# Build Agents
build_agent_dedicate: yes # Dedicate build agent to the
project
build_agent_capability_git: yes # Install git capability
build_agent_capability_docker:   yes   #   Install   docker
capability
```

After the ticket is submitted, it is sent to a foreman for approval. After the foreman approves the ticket, it is assigned to IT department for handling. The detailed project establishment process is presented at a following figure:


User
Foreman
IT-department
Create configuration
Submit a ticket
Approve ticket
Review configuration
Execute
Notify
Template
Documentation
Scripts, settings
Repository