Continuous integration and delivery in small business

Ammattikorkeakoulututkinnon opinnäytetyö

Visamäki, Tietojenkäsittelyn Tradenomi

Hyväksymislukukausi, 2021

Joona Isoaho

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Tietojenkäsittelyn koulutusohjelma
Visamäki

Tekijä Joona Isoaho **Vuosi** 2021

Työn nimi Continuous integration and delivery in small business

Työn ohjaaja Lauri Salminen

Tämä opinnäytetyö sisältää tutkimuksen siitä, kuinka integraatio ja käyttöönotto tapahtuu toimeksiantoyrityksessä, jotta prosessi voidaan automatisoida kokonaan tai osittain. Opinnäytetyön suunnittelu aloitettiin tekijän työharjoittelun aikana, jolloin tutustuttiin prosessiin ohjelmistokehittäjänä. Harjoittelun lisäksi tekijällä on vahva harrastuspohja työssä käytettyihin teknologioihin. Opinnäytetyön tarkoituksena on automatisoida integraatio ja käyttöönotto prosessit ja esitellä se tämän työn lukijalle.

Opinnäytetyön aikana vain muutama palvelin ja Odoo instanssi lisättiin automaatioprosessiin. Koko infrastruktuurin siirtäminen kerralla olisi liian suuri muutos ja aiheuttaisi varmasti ongelmia mahdollisesti monilla palvelimilla ja instansseilla. Kokeilimme automatisaatiota muutaman kehittäjän kanssa ja tulokset vaikuttavat lupaavilta. 8-12 askeleen sijaan askelten määrä laski muutosten työntämiseen versionhallintajärjestelmään ja niiden yhdistämiseen tiettyyn haaraan eli kahteen askelmaan. Sen jälkeen koodi on kaikilla automaatioon lisätyillä palvelimilla ja instansseilla. Prosessi kestää useita minuutteja kaikkien tarkistusten kanssa eikä sovellu nopeaan kehittysprosessiin. Se on soveliaampi käyttöönottoprosessiin. Prosessin nopeus nousee dramaattisesti, kun tarkistukset, joita Ansible tekee, otetaan pois käytöstä ja tämän jälkeen prosessi soveltuu nopeisiin kehityssykleihin.

Avainsanat Ansible, Automation, Odoo

Sivut 34 sivua, joista liitteitä 0 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Tietojenkäsittelyn koulutusohjelma
Visamäki

Author Joona Isoaho **Year** 2021

Subject Continuous integration and delivery in small business

Supervisors Lauri Salminen

This thesis consists of investigation of how the integration and deployment process takes place at the client company in order to automate all, or most of it. The planning of the thesis project was started during the internship at the client company. Work was carried out during the internship that included getting familiar with the process as a software developer. In addition to internship the writer also has strong hobbyist background with the technologies used in the project. The purpose of thesis is to automate the integration and deployment process and explain it to the reader in the thesis.

During the thesis process only a couple of servers and Odoo instances were added to the automation process. Switching the whole infrastructure all at once would be too big shock and would surely end up causing problems possibly on multiple servers and instances. With couple of developers who were able to test the system results look promising. Instead of 8-12 steps the number of steps dropped to pushing changes to the version control system and making merge to certain branch meaning two steps. After that the code was on all the servers and instances added to the automation. The process takes several minutes with all the checks and is unsuitable for fast development process. It is more suitable for deployment process. The process speed increases dramatically when the checks that ansible does are removed and is the process suitable for the fast development cycles.

Keywords Ansible, Automation, Odoo

Pages 34 pages including appendices 0 pages

Contents

1 BASIS

1.1 Objective

As companies grow and they develop new products, they tend to acquire more infrastructure and employees. Unfortunately, byproduct of this process is deterioration of processes which the company uses to manage its growing infrastructure. (Churchill & Lewis, 1983) This is also the setting of this thesis. As new products are developed with new technology, much of the tools and guidelines needed in the process are either absent, not compatible, manual or not automated with single tool or automated in a way that transforming knowledge about it is not simple and straightforward.

The purpose of this thesis is to automate software development and deployment processes inside the client company. Client of this thesis wants to stay anonymous.

The environment in which the thesis is made is growing both with new product offerings and in employee amount. These tools and guidelines are needed so that software developers have common ground to build and develop new software on, saving time and money, reducing bugs, lowering the training needed for new employees and speeding up the deployment of bug fixes and new features into the software products.

- Will continuous integration allow less bugs in the product that clients use?
- Will automating continuous integration save developer time?
- Will automating continuous integration decrease code version desynchronization?

2 CONTINUOUS INTEGRATION AND CONTINUOUS DEPLOYMENT

2.1 Continuous integration

Continuous integration is a way of software development in which code changes from multiple developers are added together constantly as they are made. (Amazon, n.d., p. 1) Benefits of continuous integration are multiple. As new employees are hired the system scales with increasing developer amount, the barrier of entry for new employee to get their code into production is low regarding bureaucracy and communication. It improves communication among the developers as the changes are more easily seen and their effects felt when they arrive to codebase as small chunks frequently. (Rehkopf, n.d.) As the changes are made gradually this also prevents so called integration hell at the end of development cycle or other logical sprint or timeframe in which development happens. (Pettit, n.d.) Continuous integration is a good fit for agile development process as it minimizes the effort of integration and as the integration happens often it is good place to optimize. (Agile Alliance, n.d.)

2.2 Continuous deployment

Continuous deployment means deploying code changes to instances as soon as the changes are made. The emphasis is on the automation part as the changes are deployed without human interaction when all tests on it have passed. Contrary to continuous delivery where deployments are made manually by humans. (Pettit, n.d.) As the reader may notice during viewing this document the processes described may contain both methods. This is because deployment is as much as business decision as a technical one and such takes more time to be answered. It is probable that development instances will be continuously deployed, and production instances continuously delivered. The emphasis of the implementation is on the continuous deployment side and the development branches and instances follow the paradigm. (Lots. n.d.)

Continuous deployment is associated with agile software development and differs from previous methods of deployment that followed a waterfall principle of software development. Instead of developing the software from start to finish and then deploying the software deployment is done multiple times during software development cycle. This

helps to avoid problem of changing requirements which is one of the pitfalls of waterfall development. (Lots. n.d.)

3 TECHNOLOGIES AND TOOLS

3.1 Odoo

Although not written in all capital letters, Odoo is an acronym from words On Demand Open Object (Shereef, 2018).

It is often referred as ERP and/or CRM software and such descriptions are usually enough in the context they are used. However, for this document we need a little more detailed and technical explanation. Odoo can be considered as a platform for software, called modules, modules alter and/or add behavior so that in theory any use case is possible with Odoo. Module is collection of Python, JavaScript and XML files that usually alters the behavior of Odoo. (Synconics, 2012) When we need to update the Odoo instances on the servers this means that we have more than just Odoo itself to update. We need to update the modules too, and as they can be scattered around in different repositories it can be challenging.

The core installation of Odoo is usually delivered with set of modules that can help user to get going quickly. (Synconics, 2012) The behavior of these modules can be altered, or new modules installed and developed. This core is also open source software. And is same in the enterprise and community versions. Enterprise version adds additional features like for example migration from the next version of Odoo when new version is released and support from Odoo SA. (Eremeev, 2016)

Odoo is open source software, open source software refers to software that's source code is freely accessible and can be modified by anyone. Like closed source software, open source software is also under a license, but the licenses differ tremendously from each other. Open source licenses usually grant the user the freedom to use, study, modify and distribute the software as they wish. This includes for example selling the software or charging fees when offering open source software as a service. (Red Hat, 2019, p. 1)

To promote use of Odoo and support collaborative development and features there is an association called Odoo Community Association or OCA. They coordinate development of community modules and act as legal support and entity to which code can be contributed to. (Odoo Community Association, n.d.) The client has contributed to Odoo Community Associations repositories and uses multiple modules developed by the association and the community around it.

3.2 **PostgreSQL**

PostgreSQL is object-relational database system which uses SQL, with additional features, as a query language. As of writing, no SQL database supports the full SQL standard which has 179 mandatory features. PostgreSQL supports 160 of 179 mandatory features. In addition, it supports JSON datatypes and documents, borrowing ideas from NoSQL databases and allowing wider range of use cases. PostgreSQL, also known as Postgres, is open source software and got its start at the University of California at Berkeley and has been actively developed more than 30 years. It has strong reputation as being reliable, having robust feature set, extensibility. It is dedicated to the open source community to deliver performant and innovative solutions. (PostgreSQL, n.d.)

Extending PostgreSQL is possible for example with stored function and procedures, procedural programming languages such as but not limited to Perl and Python, foreign data wrappers and many extensions. (PostgreSQL, n.d.)

3.3 **GNU/Linux kernel and system tools**

Linux is open source operating system kernel capable of running on multiple processor architectures. It is well suited for server usage because of the flexibility and license it has. It is widely deployed, supported and documented. Usually Linux kernel is shipped with parts that as together form a complete operating system. A derivate work of Linux is called distribution and they are usually tailored towards certain use case. (Linux Foundation, n.d.)

Ubuntu server is a Linux distribution made to run on servers. Its offers reliable long-term update schedule which makes it good fit to use in production servers. Ubuntu Linux is based on Debian Linux. (Long, 2017)

Debian Linux is Linux distributions that combines the Linux kernel and GNU tools and other important free software. It is made from large amount of software packages which can be installed to the system with package manager. (Debian, n.d.)

In Linux when people refer to user what they mean is not only to natural persons which are regular users, but to users used by Linux system and the programs that run on it. Each user is assigned an identification number called UID (User ID) which for system users is usually under 500 and for regular users over 500. Each user belongs to at least one group. Groups are used to control access to files and folders and assigning privileges to run commands as root user. (Kuutti, 2007, p. 149)

3.3.1 APT

APT is short for Advanced Package Tool. It is used to download, install, update and remove packages on Debian-based Linux distributions such as Ubuntu. It eliminates the need from user to manually figuring out and installing package dependencies. Linux packages often are dependent on other packages to provide functionalities to the programs. APT removes the need for downloading packages from sources you might not trust as packages are distributed through trusted repositories where code you run on your machine is reviewed by developers.
(Schkn, n.d.)

3.3.2 Systemd

Systemd is manager for system and its services running on Linux operating system. It has features like mounting and automounting, snapshot support, tracking processes and handling system services at operating system startup and includes logging daemon. It was developed to be modern and dynamic system that could speed up system boot up times. It can manage system initialization but also provides alternatives for other system utilities like syslog and cron. (Linode, 2019, p. 1)

3.3.3 SSH

SSH is a method for secure remote login to computer from another. It offers possibility to use several different authentication methods. Its purpose is to protect the communication and ensure data integrity. It is a secure alternative to for example telnet. (Ylonen, 1996, p. 37-42)

3.4 **ANSIBLE**

Ansible is open source automation tool that simplifies server configuration and scales to multiple servers easily. It is usable on many different automation scenarios of varying sizes. It is agentless which means that there is no need to install anything Ansible related on the server which the tasks are run, only Python on Linux or PowerShell on windows. (Hummel, 2019) The main workflow with ansible are playbooks which are collection of tasks described in YAML, a human readable configuration language. (Red Hat, n.d.) YAML is short for YAML Ain't a Markup Language, it is designed to be readable by humans. (TutorialsPoint, 2018)

With simple list like structure YAML is very accessible even to non-developers if needed. The tasks are parts of what Ansible calls modules and most of them are developed by Ansible developers but there is also a community of that develops modules for Ansible. (Red Hat, n.d., p. 1).

3.4.1 Playbook

Playbook is a collection of tasks that are run on the servers appointed by the host file. A task is small, usually, action that can for example create a directory. Such task could be written as follows.

```
- name: Create a directory if it does not exist
  file:
    path: /etc/some_directory
    state: directory
    mode: '0755'
```

This would create a folder in the path /etc/some_directory.

Example of a short playbook would be as follows:

```
- hosts: odooservers
  vars:
    odoo_version: "12"
  tasks:
  - name: Install git
    apt:
      name: git
      state: latest
```

This playbook would install git to the odooservers hosts group described in the hosts file. (Red Hat, n.d.)

Ansible playbooks are run on the developer's machine or on the server that handles webhooks, later called DevOps server, and requires no extra installation on the servers that run Odoo instances. The hosts where the tasks are executed, are defined in the hosts file, used in the playbook and the user which tasks are executed as in the servers can be defined in the playbook or in the playbook command when it is run. The users are normal Linux users and work same as the developer users, with SSH-keys. (Red Hat, n.d.) This simplifies the user management process as there is no need for another user type. Devops user account, is handled along with the developer accounts using the play-book. (Red Hat, n.d.)

3.4.2 Hosts file

The hosts file is collection of addresses the playbook can connect to. As default the Ans-ble playbook will use hosts file located in the "/etc/ansible" folder, but with terminal flag we can give the playbook another hosts file to use. (Perkin, n.d) Structure of hosts file is simple, group of IP addresses can be grouped under a group name. In this example our hosts file could be like this

```
[odooservers]
123.123.123.123
321.321.321.321
```

This hosts file would define group odooservers with two IP addresses in it. As the playbook is run with this host file Ansible connects to these servers trough SSH connection and executes the tasks. In this case package name git would get installed, assuming the servers are running Debian or one of its derivatives. (Red Hat, n.d.)

3.4.3 Why Ansible

Well known and "boring" tools usually also allow resources to be used in more important aspects (McKinley, 2015). Ansible is also developed with python and can be extended with modules made with python (Red Hat, n.d.). As python is also the language used in Odoo modules it is probable that if need for custom Ansible module development would arise then there would be skilled people able to develop those employed by the client.

3.5 **Python**

"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed." (Python Software Foundation, n.d.)

3.5.1 Flask

Flask is a python library developed for to be quick and easy way to make web applications with options to scale up as the services grow. It suggests usage of additional libraries but does not require user to use any. Choosing to grow and use other libraries is left for the developer. Simple flask application can be programmed in just couple of lines of Python. (Pallets, n.d.)

Git is distributed version control system. It is fast and easy to learn. I was developed in (Git, n.d.) Version control is important part of the software development process as it allows developers to have multiple copies of the same code on which they can work on independently. It allows for centralized controlled process for controlling the flow of features and bugfixes into the software. In addition, it also works as a backup and restoration tool in the case code that does not work gets deployed into the production environment. (Chacon S. & Straub E., n.d., Pro Git. p. 1.1)

3.6 Gitlab

GitLab is, among other things, version control repository hosting service. It is open source and can be self-hosted if necessary. At the time of writing we are using the service provided by gitlab.com. (GitLab, n.d.) GitLab also has built in continuous integration tools (GitLab, n.d.). But those will be used only for testing our code, even though the tools are capable of much more. The code is stored on repositories on a host-ed platform and code changes are submitted to the repository as commits. Each commit goes to certain branch and branches are merged on certain points to development and production branches. (Chacon S. & Straub E., (n.d.). Pro Git, p. 1.1) Before the code is committed the quality is checked when git hooks are triggered, git hooks are action that are executed when something happens on the git repository at the developer's machine. (Chacon S. & Straub E., (n.d.). Pro Git p. 8.3)

User management to GitLab happens trough SSH -keys, which is very convenient as this is also one used by SHH connection, so our developers only need to know one concept and have only one key. SSH-keys are encrypted credentials used by SSH-protocol. They are generated on the user's machine and work in pairs. One key is left to the user and the other key can be shared with the server or service in which authentication is needed. The one staying in the user's machine is called the private key and the one shared with server or service is called the public key. (Bluhm, 2017) SSH is short for Secure SHell, it is encrypted protocol for having connections between machines. (Communications Security, 2019)

3.7 **Webhooks**

As the code repository receives changes from the developer, we need a way to activate necessary actions to perform. In this case GitLab offers us handy tool named webhooks which "... allow you to trigger a URL if for example new code is pushed or a new issue is created." (GitLab, n.d.) Each project is set to have webhook that, when action happens, informs our server that something has happened in the repository and server takes care of running correct Ansible playbooks.

For example, when merge request is made to 12.0-dev branch, the devops server gets notified and launches corresponding actions. Meaning in most cases that it runs Ansible playbook that in that holds all the important actions. The webhook server serves as a kind of "dumb" switch and is only necessary because webhooks cannot be directly tied to a playbook for example.

3.8 **Pre-commit**

Pre-commit is package manager for git hooks which simplifies the configuration and usage of git pre commit hooks. (Finkle, n.d) It is used in this project to check for code quality. Firstly, on the developer's machine for the committed files and then on the GitLab CI/CD pipeline for all files in the code repository. It is the same tool used by OCA and we try to follow their guidelines in addition to our own. (OCA, 2019) Some of the most important tools run by pre-commit are the following. Flake8 which is a linting tool, it checks that code syntax is correct. (Gal, 2017) Pylint, which is a static code analysis tool, it differs from simple linter in that it also tries to find programming errors and suggest refactoring. (PyCQA, n.d.) And ESLint, which is tool to check JavaScript code for syntax errors and enforce code style. (JS Foundation, n.d., p. 1) As a requirement to ESLint NPM and NodeJS are also installed. NodeJS is runtime which allows running JavaScript without a browser (OpenJS Foundation, n.d.) NPM stand for Node Package Manager and is what the name says. It's used to install ESLint and packages that it depends on. (NPM Inc., n.d.)

3.9 **Supervisorctl**

Since one server runs multiple instances of Odoo a tool is needed to control all processes. As normal Linux processes can be quite confusing and have high barrier of entry a simpler tool was chosen to do the job. Supervisroctl is monitoring tool for processes that uses server/client architecture and runs on UNIX-like operating systems. (Agendaless Consulting, n.d.) It uses configuration files and it manages only processes that have configuration files it also isolates our Odoo instances from the system processes on the user interface. Making it much less probable that user stops, restarts or starts wrong processes. It also enables remote controlling the processes (Agendaless Consultin, n.d., p. Features) and in the future it is probable that we integrate this remote controlling to our toolchain as developers not needing to use SSH at all would be the best case scenario from usability and security standpoints.

Supervisroctl usage to developer is simple. The developer connects via SSH to the server, switches to superuser and then runs supervisorctl command. After this they are presented text mode list of running processes and their state can be altered with commands, start, stop and restart with the instance name after the command. (Agendaless Consultin, n.d., p. Running Supervisorctl)

3.10 **Mattermost**

Mattermost is chat service that can be self-hosted. It is designed for internal communications inside the company. Other features include file sharing and search of chat logs. In Mattermost discussions are split into channels and teams and the chat history is fully searchable. Clients can be used on web browsers, desktop computers and mobile devices. Mattermost offers DevOps integration, bots, plugins and extensions, API and supports webhooks. Different authentication methods are possible, including AD/LDAP. It is used by big companies including Bungie and US Army. (Mattermost, n.d.)

4 IMPLEMENTATION

4.1 General

Before the practical part of this thesis began the integration and deployment process was manual. First the developer would write the code and test it on their local machine. Relying on manual code quality checking. Then push this code into one and only existing branch on git repository. Hoping that no code changes would conflict with other developer's code changes and create problems. If updating the code was successful. The developer would connect to one server where the change was needed trough SSH on the terminal of their machine, update the code in the code repository on the server and then update the module on the instance using the graphical user interface of Odoo.

This method had multiple points of possible failure. As performing it for the most part was left up to do by humans, simply paying no attention or being tired could have affected the process. And as no written guidelines existed the way each developer performed the update may have been different and therefore cause unexpected or different results on one of the Odoo instances.

With this method was that the code was updated to only one server, as the other instances on other servers were updated problems could arise unexpectedly and for no apparent reason. This could lead to situation where updating or simply restarting Odoo instance was stressful and uncertain operation to do, leading to developers possibly avoiding it and therefore leading to technical debt.

This update was also not visible to any other developers, meaning that there was no other way to know at which state the code was at in the servers other than manually checking it yourself. This could lead to fragmentation between installations as they were updated at different intervals. With multiple servers and growing the problems caused by this could be very severe.

The manual updating method also did not scale. It was not feasible to expect developer to update all the servers and instances every time change was made. And even if it was,

the changing number of servers would lead to some of the servers being forgotten as update happened and that way lead to even worse scenario were other developers would assume that all the servers were up to date even if they were not.

At the time the server infrastructure consisted of about a dozen servers that run Odoo instances and 3 database servers. These servers run Debian and Ubuntu operating systems.

The section concentrates on the most part on explaining the automation process and the reasons behind the actions, not listing and explaining every single task on every single playbook. The playbooks, configuration files and other needed software are all stored inside a single git repository and version controlled just like the software code. The repository has two branches, master and dev. First one is for daily usage and to be used daily in the automation and by developers and the second one is a branch for testing new features and configurations.

On the server which Odoo instances are run each instance gets its own Supervisorctl configuration file. This tells the Supervisorctl to start the process at the system startup. The configuration file is named after the instance name, let's stay for example customer123. This name is also used as a Linux username under which the Odoo process is run. This provides isolation as each user has right to only its own instance. Odoo uses the customer123 as database name, this is set in the Supervisorctl and Odoo configuration files.

4.2 **Planning**

The planning phase of the thesis began during the internship which was completed before the beginning of thesis. Work was done with the software products and as a result experience with the development process was acquired. Along the internship other employees also helped with the requirements of the automation by communicating their needs regarding the system and suggesting tools and methods in which things could be done. Much of the requirements and ideas for the implementation part of thesis was collected during the internship and no scheduled planning meetings where hold.

At the beginning containers where considered as part of the process but since Odoo is run with python and the processes can be neatly controlled and observed together, there was no need for containerization. In fact, the containerization of Odoo instances would only have complicated the setups and possibly would have added more stress on the servers in terms of computing power.

Choice between two version control repository hosting services was made, GitHub and GitLab. As the tools for code quality and other testing from Odoo community association runs on Travis, which is continuous integration tools, it would have required to switch all the repositories to GitHub as Travis is so deeply integrated and the tools and test without much of a work from our part would be provided by the association. For the workflow with software development to stay similar regarding the developer access to code repositories for example, this would have required a switch to paid version of GitHub. For each developer a monthly fee would have to be paid. The switch in addition to money would of course require that the paid accounts would need some management and changing whenever new developer arrived or old one left. For these reasons it was a priority to avoid this change and the tests of same nature were accomplished with other tools. Pre-commit and GitLab CI/CD Pipeline were used to accomplish this, and this system has some advantages compared to testing code only on the server side. As the developers write their code and code quality tests are run on their machine a large amount of time is saved. Typical Travis test on a code commit on a GitHub server took about 5 minutes when on developers machine the code quality tests run in seconds.

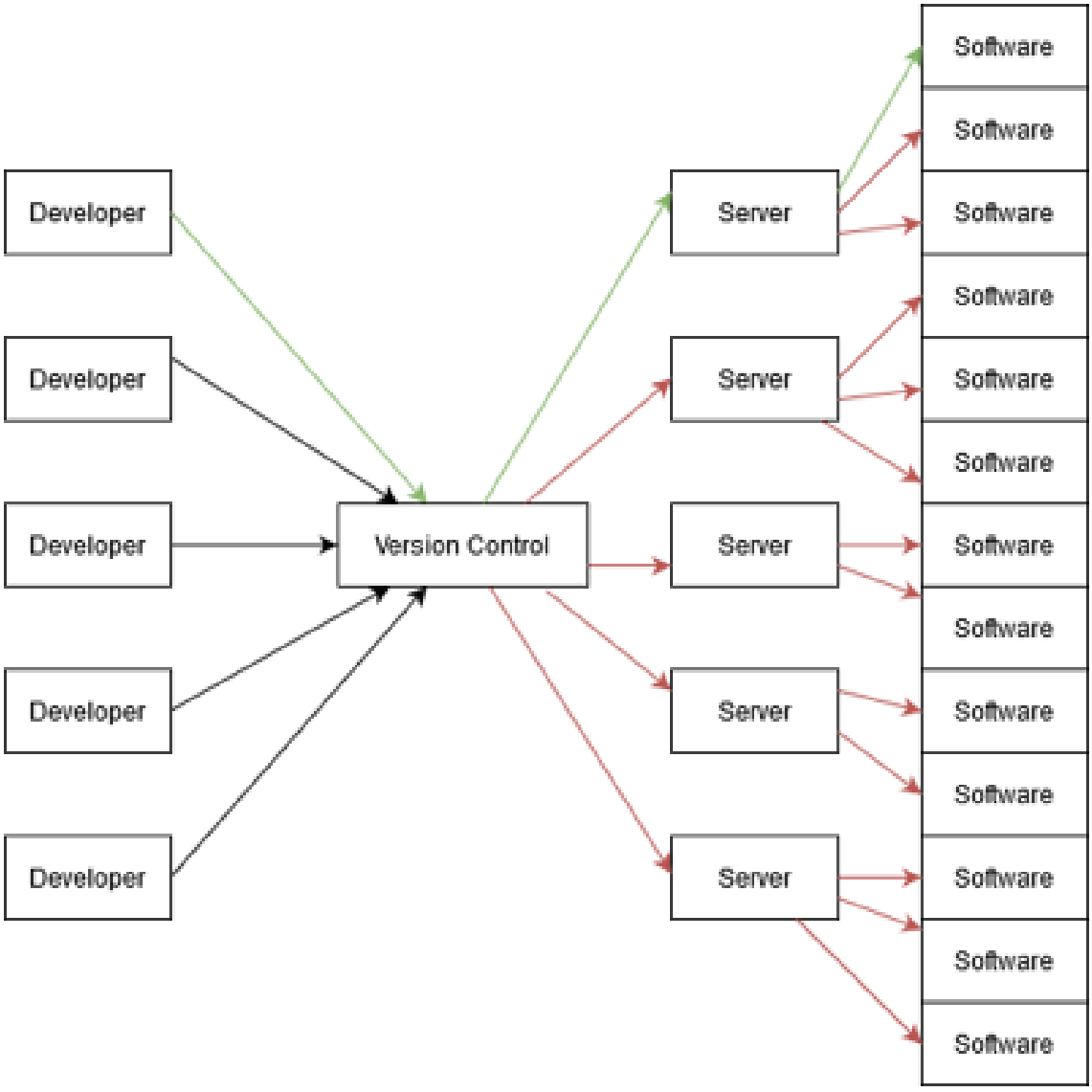
During the project, negotiations about the intervals of updating was ongoing and mostly undecided. The development instances which are only in internal use could be updated as soon as changes were made but the production instances would require timed updating. The exact interval of updating was not clear but need for it was.

As the new tools and guidelines differ much from the previous way of doing deployment a partial introduction of automation is necessary part of the project. Switching all

servers, codebases and instances to new tools instantly would cause too much chaos and waste valuable time and resources. Too rapid change can also cause people to reject it. The automation tools and guidelines were planned to support partial introduction and deployment. Ansible was chosen as the automation tool because it is simple to use. We want the automation infrastructure and configuration be easily accessible by other developers in case changes are needed and the original developer of the system is not around anymore.

4.3 **Current integration and deployment process**

The current manual integration and deployment process starts with developer writing code. Then the code is integrated in the version control system. After this developer updates the code to the server and then to the Odoo instance.



Kuva 1. Current update reach

Line represents code movement. Green is up to date. Red is out of date.

4.4 **Servers**

The servers that run Odoo instances use Ubuntu Linux operating system. Compared to Debian Linux Ubuntu receives updates at a faster pace while still having a long-term support version. New bugfixes and speedups are good in the server environment where speed and security are on the top of priority list while on the DevOps server no such speed of updates is needed, and slower updating Debian is a good choice. Both distributions receive security updates as they are released. At the time of the thesis the version of Ubuntu was 18.04 and version of Debian was 9.0.

On the servers we run three kind of Odoo instances.

4.4.1 Development instances

The development instances are first place where code is deployed. They are only used inside the company and serve as testing platform for new modules, features and bug fixes. Database in periodically overwritten with production instance database, this ensures that the environment where developers test new module, feature or bugfix is the same as the environment in which the user will use the code.

4.4.2 Testing instances

Testing instances are instances for the users to test new features and learn how our products work. Database of testing instances is periodically overwritten by the production instance database. So that the client will have their data to test and learn with.

In addition to being testing and learning platform for our users, the testing instances can also be used as new module, feature and bug fix testing platform for developers. After the code has been deployed to the development instances it is usually brought into the testing instance before the production instance. This way the users can test new modules, features or bugfixes before they are deployed into the production instance.

4.4.3 Production instances

Production instances are the most important instances of the three types of instances one installation has. It is the instance that is in daily use of the customer or similar entities. Uptime and stability of the production instance is the most important goal of continuous integration and continuous deployment, alongside with the speed of getting features to the customer.

Production instance is the last place where code is updated after it has gone through the process described in this document. Updates are done on certain intervals so when the rare occurrence of a bug happens, the developers are ready and need for fixing will not come as a surprise. Fixed instance upgrade times also ensures that features flow into the production more evenly.

4.4.4 DevOps server

At the beginning of the project a server was set up to handle the webhooks and run the ansible playbooks. Later referred as DevOps server. This was done by an Ansible playbook which performed various tasks on the DevOps server. Starting with installation of needed apt-packages, this was a list containing ansible and git packages. As this list is variable its easily extensible in the future and clearly tells the playbook user which packages are installed. Then playbooks proceed to created needed folders to hold log files. The task for folder creation in ansible playbooks is as follows.

```
- name: Make sure that log folder exists
    become: yes
    file:
     path: /var/log/devops
     state: directory
```

Name of the task can be anything and is displayed as the playbook is run. Then the playbook ads the webhook server as a cron file so it will run at the server startup. Cron is

usually used in timing actions but works here now as cron is started as a late state of system startup in which necessary services to run the webhook server will be started.

The server software that listens for webhooks was developed with Python using the Flask library. It is small piece of code that listens for HTTP request from certain port and on receiving handles the data, if necessary, runs a playbook. The purpose of the server is just to listen to webhooks and run playbooks and acts as a kind of glue between GitLab webhooks and Ansible playbooks. This server is also responsible of updating itself and the playbooks when code is pushed into the playbook repository. This will help in the future to reduce the need for human interaction on the overall process. After the files are updated the webhook catching server also restarts itself.

As Mattermost is used as the "fast" communications platform in the company, email being "slow", it is natural that the automatic integration and deployment process also informs what it is doing into Mattermost chat. In various states of operations message is sent to certain channels on the Mattermost that are reserved for this kind of usage. This allows other developers to also be up to date about what is happening to the code as all the code merges get reported to one central location.

The DevOps server is running Debian Linux version 9.0 and it is separate server from the servers that run Odoo instances. The reason for this is that own server for DevOps operations minimizes the risks of automation tools messing up other servers and operations. The computing power requirements of automation tools on the server is quite low but as the tests and tools grow in the future it is good that there is its own separate server to handle the operations.

4.5 **User management with Ansible**

Developers need access to the Odoo instance and database servers. As the software development process was constantly ongoing the logical first thing to automate was server user management. Allowing developers to access every server trough SHH was

also not planned to be prevented at any point. This was to allow the old way of updating things until the migration to full automation was done but also to have some sort of backup in case direct access was needed even when automation was in place.

Need for access included servers not covered by other automation such as those which host Joomla instances. The servers run Debian and Ubuntu Linuxes and users on the servers were normal Linux users. To be able to connect to the server trough SSH, a public SSH key must have been placed on the server. To automate adding and removing users on the server Ansible playbook which handles the operation. Developer ads their public SSH key to git repository, which is then automatically pulled into the server that runs automation playbooks and key is updated to all servers.

Giving access to the servers to developers also allows them to access logfiles and in do code deployment and testing without automation, like before, as the process of moving to automation can then be done partially over time. Logging automation and easy access from developer machine is outside of the scope of this thesis.

The user management playbook has a list of usernames that need to be added to the server. As it is connected to the hosts it loops trough the usernames and adds them to the Linux system as system users. Adding to them the needed groups. Then it goes through the list of public keys that are in the same git repository as the playbook and loops trough them, adding them as trusted keys in the system.

4.6 **The development environment**

During the development process it's important that all developers have the same development environment, with each other and with testing and production Odoo instances. As a result of this code changes behave the same on all stages of development and deployment. (Shevat, n.d.) It is set up with ansible playbook, to ensure identical environments. The playbook is acquired from git repository which ensures that all developers can easily keep in sync with the latest changes. The process of setting up and updating development environment is a process that each developer must and will do multiple

times as the code in the repositories and in the development environment playbooks change.

First the developer clones the repository using git with a command:

```
git clone <repository address>
```

After the developer has cloned the repository that holds the playbook, they need to run specific command to run the playbook.

```
ansible-playbook playbooks/local-dev-env.yml -i hosts/development-hosts --ask-become-pass --connection=local
```

Playbook installs specific Odoo version to the developer's machine along with needed python package dependencies from python package index. At the time of writing the Odoo version was 12.0. Then it creates Odoo user to the developer's machine and generates an SSH-key pair for the user. Odoo users SSH-key needs to be added to GitLab so the odoo -user has access to the code and can clone it. It can be done by navigating to address https://gitlab.com/profile/keys and issuing command "cat ~/.ssh/id_rsa.pub" and copying the output to the GitLab user interface.

The fact that same playbooks can be used on the server and the developer's machine also allows to have a single list of repositories to use. This reduces repetition and servers for the developers as a single source of truth when figuring out which repositories are in use. Reason for using odoo -user is that the servers use it, so same playbooks can be used setting up and updating the development environment and the server environment. Then it copies Odoo development configuration files into right place. Then it clones the company's own source code repositories from multiple different repositories. The repository that holds the playbooks also contains configuration files for Odoo so there is no need to create ones. These are cloned to the same folder path as in the server. Module directories on the repositories are linked into one directory so the Odoo configuration files can be kept small. The linking is handled automatically by the

playbook as developer is setting up the environment but can also be done manually if needed.

Previous actions deal with programs, code and tools that are common to development environment and servers on which Odoo is run. In addition to those the developer will benefit from playbook as it installs tools that are needed or might be needed during the Odoo module development. For example, pgAdmin3, graphical user interface to PostgreSQL database and SQLite browser which is powerful tool when dealing with Odoo import and output files which are in CSV (Comma Separated Value) files.

Code quality testing tools along with the needed configuration is also installed. For the time being this means just the pre-commit tool described earlier. The pre-commit configuration files are placed in each repository folder, but originals are always in the same repository as the playbooks so that when changes are made, they will flow into the code repositories from the playbook repository as the developers update their environments. The code quality check tool, pre-commit, checks by default only the files the developer has added to each commit. This way the flow of developing is not disturbed too much when the checks first arrive in the repository in which they previously where absent. Code quality checks to all files in repository are made in the GitLab CI/CD pipeline.

After playbook has set up the environment it is identical to other developer's environments and developer can now start Odoo with for example changing to directory /opt/odoo/<odoo-version>/server and running command:

```
./odoo-bin -c ../conf/odoo-<odoo-version>-all-dev-modules.conf
```

The folder structure is same as in the servers. The odoo binary itself is installed at path /opt/odoo/<odoo-version>/server. And the modules to path /opt/odoo/<odoo-version>/addons. Odoo configuration file needs to have all the paths to modules parents' folders listed. As the module amount grows the configuration files can get very big and as each repository is usually its own line the developers may remove, add or forget to remove or add paths. For this reason, both on the server and the developers machine

the playbook links the modules into one folder using Linux file links. This allows the development configuration file to be small as the only paths it needs are. /opt/odoo/<odoo-version>/addons – the odoo core addons path, /opt/odoo/<-odoo-version>/addons/mpg-linked-all and /opt/odoo/addons/mpg-dev-linked-all.

After this the Odoo instance is accessible at localhost at port 8069.

4.7 **Branching strategy**

Module repositories are divided to at least two branches named after Odoo version and their code state. First branch was at the time of the thesis called 12.0 and is referred as production branch and 12.0-dev, referred as development branch.

As the developer starts developing new module, or feature to module, first thing they do is branch of from the development branch. Which, according to Atlassian (Atlassian, n.d.) ensures that development branch will not hold broken code at any point as the changes are made in this branch first. It also helps in making the developer coordinate when to merge. Coordinated merging means that conflicts in the code changes can then be predicted to happen at these events, which means the developer can anticipate them and make necessary evaluation about the time they need for the merge. This branch is often called a feature branch and is usually named after the feature, bug fix or new module it holds inside with developer branch as prefix. For example, developing new module which adds favorite color for the partner model name could be 12.0-dev-partner-favorite-color.

4.8 **Pre-commit**

As the developer commits their code into the feature branch their code is tested locally with different kind of code quality tools. Tool called pre-commit handles the different kind of tests. Most of the testing tools and code quality rules are advised to be used by Odoo community association. As the community has shared guidelines about code quality tools and we use them this eases our burden and ensures that our developers are able to commit to the associations code repositories if necessary.

The code quality testing on the developer machine is required when new commit is made. This means that only code that passes the tests can be pushed into the repository. This is the case with all the branches. Testing code from the beginning even in feature branch means that there will not be a large amount of changes to be made when code is merged as all the code must follow the quality tools configurations.

4.9 **Gitlab CI/CD Pipeline**

When commit is made to git repository, GitLab CI/CD pipeline will run tests on the code. First it establishes Docker container running Linux with latest Python and Odoo installed. Then it installs Odoo and things needed to run it. The Odoo is then run with commands to install the modules to make sure they install. After this all files in the repository are checked with pre-commit. This testing environment is very versatile and can be used to test almost anything and offers large variety of ways to expand the testing.

4.10 **Merging**

Instead of developers merging all changes to one branch, a feature branching system was introduced in which the code changes move through branches as merge requests. There is no tool to enforce this but as a crucial part of continuous integration it is important to have rules by which code is moved. This is part of the bigger picture in automation that spans to the company culture. When everything can't be automated and unfortunately something must be left to be done by humans, it is important that it is something that is easy to see and follow. In this case any developer familiar with GitLab user interface can see the branches and guess what we are doing. Furthermore, the culture of continuous integration is easier to pass on to new developers because it has a name and is widely known concept of which documentation and similar material exists.

After developer has finished programming new module, feature or bugfix and the commits pipeline tests have passed a merge request is created. After another developer has looked at the code and approved it, the pull request can be merged to the development branch. This triggers a webhook which informs the DevOps server to run playbooks to update the development code repositories and restart the Odoo development instances.

After the module, feature or bugfix has been deployed to development server and tested a pull request can be made from development branch to production branch and code will be deployed to the development servers on the next production update.

4.11 **Development branch**

As the merge happens into development branch, a webhook is sent to DevOps server. This triggers automation to deploy changes into servers. When merge request is opened, the webhook server leaves a comment on it letting the developer know that merging this pull request will trigger automation. In the development's branch case, the automation is updating the development repositories on the Odoo instance servers. What exactly is to happen cannot be revealed on the comment as they might be public sometimes, but some information that the developer action will spring actions elsewhere is mandatory.

4.12 **Updating the repositories**

At this project there is tens of repositories hosted on the GitLab service. Updating all the repositories on all the servers is too error prone and complicated to do by hand. This has been the previous method of keeping all up to date. The new method is addition to the old method and updating repositories by hand is still possible, but in the future no repository will be forgotten or in the wrong branch. The automatic update overrides repositories on the server that are in the wrong branch and sets the branch into the development branch. In addition to this the automatic update also removes any old repositories or so called "backups", which are usually just copies of code repository folders that are named differently, that developers might be tempted to do if updating is left to humans only.

During the process the Ansible playbook sends messages to company's Mattermost chat so the developer doing the merge and other developers can be up to date bout at what part of the automation process is going on now. Ansible playbook is run that clones' changes of development repositories into the servers. It does this by looping over the

list of repositories and committing git clone task on each of them. Ansible git module is used which is developed by Ansible community.

The playbook is also place with a list of all git repositories we use. Alongside our git repositories it also updates OCA git repositories that host the community made modules.

4.13 **Updating the modules to the development instances**

After the code repositories are up to date, the development Odoo instances are restarted using Supervisorctl. Restart is made with its own playbook, presented here.

```
- hosts: odoo
  tasks:
    - name: Restart development instances with supervisorctl
      become: yes
      command: supervisorctl restart dev:*
```

Defined as the hosts is the Odoo instance server group and the restart itself is made with a command task. This task can run any Linux command. In this case the command is

```
supervisorctl restart dev:*
```

which restarts all the instances in the dev process group. Become means that the playbooks uses superuser privileges on the serve it connects to.

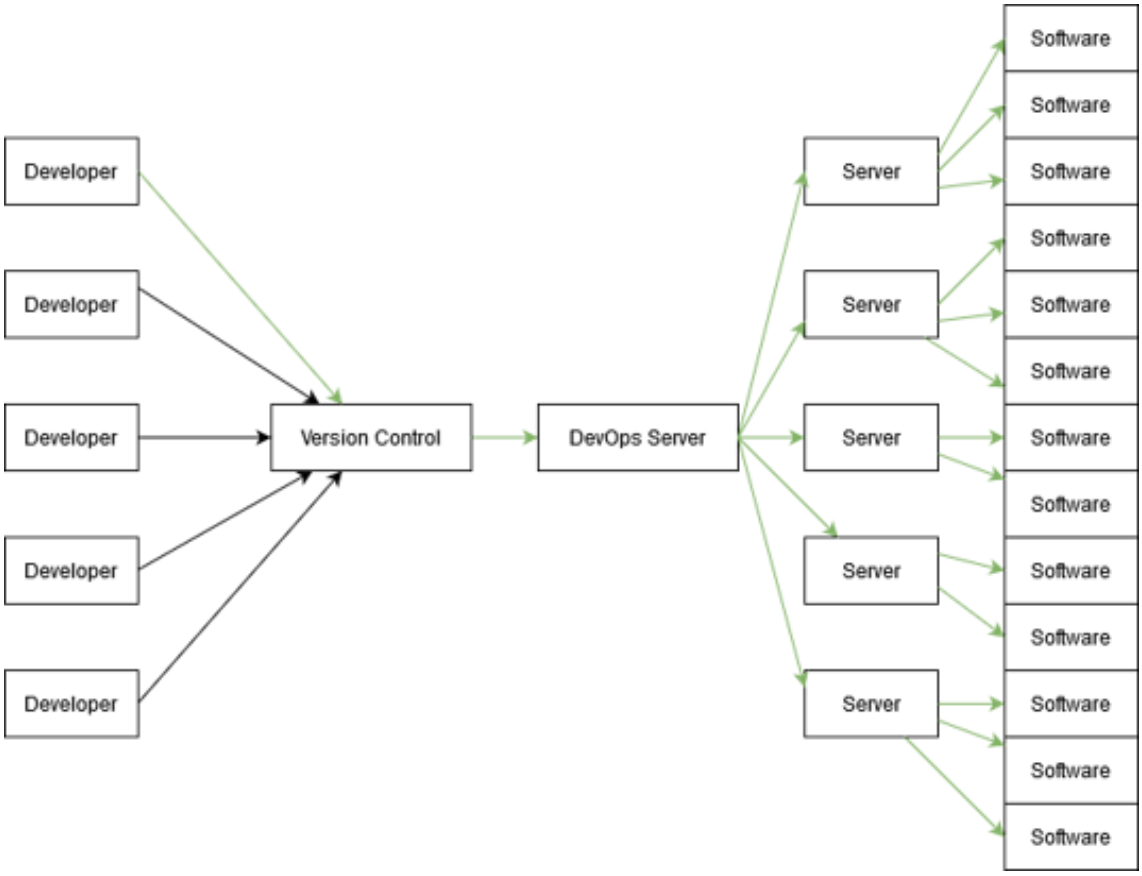
As the development instances restart, a module is updated which starts auto upgrade of all modules. This module update is set in the instances Supervisorctl configuration file with -u flag. Update of this module launches module auto upgrade -module which checks a hash checksum of modules and compares it to see which modules to upgrade. Odoo has terminal flag "-u all" which updates all the modules but most of the time not all modules are upgraded and since the update of all modules tries to update all modules it takes a very long time. This would not necessarily be problem on the development instances of Odoo but on production instances the possible downtime must be minimized so this more precise method of updating modules was devised.

4.14 **Updating the code to the production instances**

Updating of the code to the production instances cannot be done at the same time as the changes are made to the code. Because the updating process requires restarting the instances this would mean downtime on the services, sometimes multiple times a day. These kinds of interruptions would affect the users negatively. Instead the productions server updates would be timed using Cron timing. Cron is a Linux tool used to run commands at certain intervals. One possible update interval would be one week. The servers would update and restart on Sunday to Monday night and that way if problems occur, they appear at expected time and leave the rest of the week for fixing them if necessary.

4.15 **Automated integration and deployment process**

After the automation process is in place the process now looks like in the following image.



Kuva 2. Planned update reach

The code never goes to the DevOps server as the image suggests but the main point is that the updated “path” now covers all the servers and instances.

5 RESULTS

During the thesis process all servers were switched to use the new system. The server addition was done gradually and after each step the automation was tested. As expected, some problems occurred during the process but most of them were the kind of bugs that the automation was designed to prevent in the first place. Some modules were not up to date on one server, for example. These kind of mismatches of versions should not happen in the future as the process is now programmatically forced to happen in certain way and update all the code to the newest version. This should in theory help catch more bugs before they get to the client as the promise that the code is always at same state helps reduce the area of which problems can occur.

Using ansible the gradual deployment of automation was simple. Server addresses were added to the list of addresses on which ansible would operate. Necessary users were set up on the servers. Then automation was run, and checks were made to see if anything was not working. It was to be expected that at least some code was out of sync and some servers even had changes made directly to just that server. In the future these problems should appear in very small amounts or not at all. As the automation always overrides the server state regarding code, there is no point for developer to make anything other changes than very short lived.

At first the process of updating all the code into the servers took something in range of 5-15 minutes. This was obviously too slow for fast paced continuous integration where the changed made to the code would usually be updated to the server right after they were made. This slow speed is completely because of ansible. On the other hand, ansible makes sure that everything you want to happen in a playbook also happens. For speedups these checks can be circumvented. Since we are running actions that will happen many times, sometimes tens of times in a day, and will rarely fail we can trust that the code will be up to date on the servers even without all checks that Ansible does.

The speed of ansible is something that must be taken into consideration if you are planning to do automation with it. Since the main use case for Ansible is to automate the handling of the infrastructure, the speed is not the most important aspect of the project. Which is understandable. The checks that ansible does can be circumvented with pure shell calls, this adds complexity but not very much since many benefits of Ansible will still be usable. Like the added structure dictated by using playbooks. The shell calls are made by ansible and are in the same place in the playbook as the ansible module command which previously did the code cloning. With the checks circumvented the speed in which code was updated dropped to 0.5 – 2 minutes which is very tolerable speed for this kind of automation.

Before the automation the process to get the code to the server was cumbersome. Connect with SSH to server, update the code, hope nothing will break, and you did not make any human errors. As time goes by and the developer does this again and again the process gets faster and some automation by the developer can be added. Nothing still prevents the developer from making human errors and variations in the manual updating process can vary developer to developer. It is also time consuming to teach the process to new developers. As the process is automated and the code is updated as soon as it is pushed into repository there cannot be any variation on the process and no matter how new the developer is the time it takes to update is constant. This saves developers times.

Before the automation process was in place code could be several versions behind depending on the server. As the developer usually only updated one server while developing the code, but the code could also be used on other servers, if there was need to know what version of the code was on the server it had to be updated manually. Now there is a guarantee that the code in all servers is always the newest code.

However, the part of automation that updates the code to the software instances was not finished during working on this thesis. The idea was to run a checker on software reboot that would see if any module code was changed and then update only the changed modules. It would have been simpler to just update all the modules, but this

can take a long time and updating can happen multiple times a day or even an hour. So, the added complexity of updating only certain modules is unfortunately a must.

During the update process messages will be sent to different locations to inform the developer about the state of the update. When the merge is completed in GitLab the automation sends a message to the pull request telling the developer that automation is taking place. At the same time message is also sent into a specific Mattermost channel. This channel also receives information about which servers were updated and was planned to also inform which instances updated which modules. Providing the developers with this kind of information is very important and in addition to going into the Mattermost channel it should also be logged somewhere with timestamps. In the case some server or software instance breaks the logs can be checked to see what changes have happened near, in place and time, the breakage.

The DevOps server ties all this together. It runs server software made with Python using the Flask library. As the request come from the GitLab automation pipeline they are handled, and necessary Ansible playbooks are then run. It is made in a way that when adding servers, the python code does not need to be changed, only the Ansible configuration and playbook files. It could probably be made even more error resistant (see 7.1.4) but as it is now it should be adequate.

Error resistance and small changes were big part of rolling out the automation process since automation and tools are meant to decrease the workflow of the developers and too radical and/or fast changes can have opposite effects. During the whole process. from design to end, many technologies were considered and tested. For example, containers. Finding the perfect combination of complexity and features is of course ongoing process that has no end but starting with idea that it easier to add than take away parts of the process I do believe the direction of the project is on correct path.

“Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.” - Antoine de Saint-Exupéry, Airman's Odyssey

6 SUMMARY

6.1 Future maintenance

The tools used in the thesis work were all chosen because either they are easy to learn or most of the developers were already familiar with them. In cases where some other tool could have done the job, but existing tool could do the same job the existing tool was chosen to not add complexity into the process. These choices were made to ensure the system is maintainable in the future even if the writer leaves or is otherwise unable to maintain the system.

6.2 Challenges

As the tools and other software like Linux operating system and Python are familiar to me through hobbies and internship the thesis faced no major technical challenges. Some are to be expected as the project is not finished yet. Usually the last 10% of the project is 90% of the work.

One of the challenges of the thesis was scheduling. As I was planning my thesis during the internship on summer and autumn 2019, I was expecting to work on the thesis through January to March full time, meaning about 40 hours a week. My planning and feature set were made using this estimate. Changes to plans were made after my internship and I started working 3 days a week and working on my thesis 2 days a week. This meant that I had to downscale the project to bare minimum.

Ansibles speed when cloning and updating git repositories was slower than expected. As these procedures are simple and usually very fast this came as a surprise. On the other hand, the thing that is slowing Ansible down on these are the checking it does when cloning and updating. In the cases where speed is not important the tradeoff is good, because Ansible is more reliable than many other methods of doing the cloning and updating.

7 POSSIBLE FUTURE ADDITIONS

7.1.1 New Odoo instance setup

In addition to updating Odoo instances on the servers, functionality for setting up new Odoo instances is planned. As the development environment is already being set up with Ansible playbooks the next logical step is to use playbooks on the server too. The process of setting up new instance should be so easy that non-developer could do it. This would allow sales team to spin up new instances of Odoo for customers when needed. This would require graphical user interface and could be integrated on the server that handles webhooks. A simple webpage with new instance name, that would also be used as database and Linux username on the server, and selection of which product would be installed on the server. This automation would require generating configurations for Supervisorctl and Odoo instances and saving the generated Odoo database password to somewhere automatically.

7.1.2 Supervisorctl remote control

Supervisorctl allows remote control of running instances. As the configuration files are now scattered on the servers, each containing configuration files for instances run on that server, the remote control would allow the configuration files to be on the DevOps server and all Odoo instances running on the servers to be handled manually.

In addition, this would allow the developers to control the instances from their own machines, further reducing the need to use SSH connection to the instance servers.

7.1.3 Automatic module update

If same module is used in multiple Odoo instances, problems can also occur if the module is not updated on the instance. This problem was considered during the making of the thesis and some attempts were made to make it happen but given the timeframe they were not successful. There is not any technical reason why updating the modules on the instance would be impossible.

7.1.4 Simplify the DevOps server software

Currently on the DevOps server a Python server is run which handles the HTTP request coming from GitLab. This could be switched to use Apache web server and python scripts to CGI. It should increase the reliability and decrease the complexity on this tool. Reliability increase would be because Apache starts more reliable than custom made server and CGI runs each request in its own process meaning that error will only close that process and not the whole server.

SOURCES

Atlassian. (n.d). Feature Branch Workflow. Obtained 10.02.2020
https://web.archive.org/web/20200102164748/https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow

Agendaless Consulting. (n.d.) Supervisord. Obtained at 15.02.2020
https://web.archive.org/web/20200106124925/http://supervisord.org/

Agile Alliance. (n.d.). Continuous Integration. Obtained at 16.04.2020
https://web.archive.org/web/20191025110818/https://www.agilealliance.org/glossary/continuous-integration

Amazon. (n.d.). What is Continuous Integration? Obtained 14.02.2020
https://web.archive.org/web/20200211125333/https://aws.amazon.com/devops/continuous-integration/

Bluhm. (2017). What Are SSH Keys? Obtained at 16.04.2020
https://web.archive.org/web/20180703225941/https://jumpcloud.com/blog/what-are-ssh-keys

Chacon S. & Straub E., (n.d.). Pro Git Obtained 15.02.2020
https://web.archive.org/web/20200210130045/https://git-scm.com/book/en/v2

Churchill & Lewis. (1983). The Five Stages of Small Business Growth. Obtained 14.02.2020
https://web.archive.org/web/20190225042146/https://hbr.org/1983/05/the-five-stages-of-small-business-growth

Communications Security Inc. (2019). SSH Protocol. Obtained at 16.04.2020
https://web.achive.org/web/20191221051037/http://www.ssh.com/ssh/protocol

Debian. (n.d.). Debian. Obtained at 16.02.2020 https://web.achive.org/web/20191225142713/https://www.debian.org/releases/etch/s390/ch01s03.html.fi

Eremeev. (2016). Odoo Community VS. Odoo Enterprise. Obtained at 15.02.2020 https://web.achive.org/web/20170328155012/https://xpansa.com/odoo/odoo-community-vs-odoo-enterprise/

Finkle. (n.d.) Pre-Commit. Obtained 14.02.2020 https://web.archive.org/web/20200207213449/https://pre-commit.com/

Gal. (2017). What is Flake8 and Why We Should Use It? Obtained 14.02.2020 https://web.archive.org/web/20190603123433/https://medium.com/python-pandemonium/what-is-flake8-and-why-we-should-use-it-b89bd78073f2

Git. (n.d.). Git SCM. Obtained 14.02.2020 https://web.archive.org/web/20200208200307/https://git-scm.com/

GitLab. (n.d.) Source Code Management. Obtained 15.02.2020 https://web.achive.org/web/20200212214151/https://about.gitlab.com/stages-devops-lifecycle/source-code-management/

GitLab. (n.d.) Webhooks. Obtained at 15.02.2020 https://web.archive.org/web/20200207033351/https://docs.gitlab.com/ee/user/project/integrations/webhooks.html

GitLab. (n.d.). Continuous Integration. Obtained at 15.02.2020 https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/

Hummel. (2019). What is Ansible. Obtained 14.02.2020 https://web.archive.org/web/20191230101207/https://cloudacademy.com/blog/what-is-ansible/

JS Foundation. (n.d.). ESLint. Obtained 15.02.2020
https://web.archive.org/web/20200203211119/https://eslint.org/

Kuutti, W. (2007). Linux. Jyväskylä: WSOY.

Linode. (2019). What Is Systemd? Obtained at 16.02.2020
https://web.achive.org/web/20200124190104/https://www.linode.com/docs/quick-answers/linux-essentials/what-is-systemd

Linux Foundation. (n.d.) What Is Linux? Obtained at 16.04.2020
https://web.achive.org/web/20200119040427/https://www.linux.com/what-is-linux

Long. (2017). Difference Between Ubuntu Desktop and Ubuntu Server. Obtained at 16.04.2020
https://web.achive.org/web/20190713043217/https://www.makeuseof.com/tag/difference-ubuntu-desktop-ubuntu-server/

Lotz. (n.d.). Waterfall vs. Agile: Which is the Right Development Methodology for Your Project? Obtained at 16.04.2020
https://web.archive.org/web/20190724182715/https://www.seguetech.com/waterfall-vs-agile-methodology/

Mattermost. (n.d.) Product. Obtained at 15.02.2020
https://web.archive.org/web/20200204190718/https://mattermost.com/product/

McKinely. (2013). Choose Boring Technology. Obtained 14.02.2020
https://web.archive.org/web/20200203125856/https://mcfunley.com/choose-boring-technology

Npm Inc. (n.d.) About NPM. Obtained at 15.02.2020
https://web.archive.org/web/20190213203059/https://docs.npmjs.com/about-npm/

OCA. (2019). Black Isort Pre Commit. Obtained 15.02.2020
https://odoo-community.org/blog/the-oca-blog-1/post/black-isort-pre-commit-97

Odoo Community Association. (n.d.). Odoo Community. Obtained 15.04.2020
https://web.archive.org/web/20191001151016/https://odoo-community.org/

OpenJS Foundation. (n.d.). About. Obtained at 15.02.2020
https://web.achive.org/web/20200104054853/https://nodejs.org/en/about/

Pallets. (n.d.). Flask. Obtained at 16.02.2020
https://web.achive.org/web/20200215101424/https://www.palletspro-
jects.com/p/flask/

Perkin. (n.d). Ansible Inventory File. Obtained at 16.04.2020
https://www.rogerperkin.co.uk/network-automation/ansible/ansible-inventory-file

Pittet. (n.d.). Continuous Integration vs. Delivery vs. Deployment. Obtained 14.02.2020
https://web.achive.org/web/20200101115136/https://www.atlassian.com/continu-
ous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

PostgreSQL. (n.d.). About. Obtained at 16.02.2020
https://web.archive.org/web/20200206023748/https://www.postgresql.org/about/

Proceedings of the 6th USENIX Security Symposium, pp. 37-42,

PyCQA. (n.d.) Pylint. Obtained 15.02.2020
https://web.archive.org/web/20200101231241/https://github.com/PyCQA/pylint

Python Software Foundation. (n.d.) Blurb. Obtained 14.02.2020
https://web.archive.org/web/20191220065315/https://www.python.org/doc/es-
says/blurb/

Red Hat. (2019). How Ansible Works. Obtained 14.02.2020
https://www.ansible.com/overview/how-ansible-works

Red Hat. (n.d.) Modules. Obtained 14.02.2020
https://web.archive.org/web/20200213192931/https://docs.ansible.com/ansible/latest/user_guide/modules.html

Red Hat. (n.d.) What Is Open Source. Obtained at 16.04.2020
https://web.archive.org/web/20200213061534/https://opensource.com/resources/what-open-source

Red Hat. (n.d.). Playbooks. Obtained 14.02.2020
https://web.archive.org/web/20200213192929/https://docs.ansible.com/ansible/latest/user_guide/playbooks.html

Rehkopf.(n.d.). Continuous Integration. Obtained 14.02.2020
https://web.archive.org/web/20200101115136/https://www.atlassian.com/continuous-delivery/continuous-integration

Schkn. (n.d.). APT Package Manager on Linux Explained. Obtained at 16.02.2020
https://web.achive.org/web/20191216160555/https://devconnected.com/apt-package-manager-on-linux-explained/

Shereef. (2018). What Is Odoo? Obtained at 15.02.2020
https://web.archive.org/web/20180723051510/https://www.cybrosys.com/blog/what-is-odoo-open-erp

Shevat. (n.d.). Effective Development Environments. Obtained 14.02.2020
https://web.archive.org/web/20190717212443/http://spacebug.com/effective_development_environments/

Synconics. (2012). What Is Odoo Features. Obtained at 15.02.2020
https://web.archive.org/web/20190303040929/https://www.synconics.com/what-is-odoo-features/

TutorialsPoint. (2018). YAML Introduction. Obtained at 16.04.2020
https://web.archive.org/web/20181127110729/https://www.tutorialspoint.com/yaml/yaml_introduction.htm

USENIX, 1996
Ylonen. T. SSH - Secure Login Connections over the Internet.