LAUREA
UNIVERSITY OF APPLIED SCIENCES
Prime Mover

Selecting a Deployment Automation Tool for CRM Software in Elisa Oy

Karalar, Onur

2013 Leppävaara

Laurea University of Applied Sciences
Laurea Leppävaara

Selecting a Deployment Automation Tool for CRM Software in Elisa Oy

Onur Karalar
Business Information Technology
Bachelor's Thesis
2013

Laurea University of Applied Sciences **Abstract**
Laurea Leppävaara
Bachelor's Programme in Business Information Technology

Karalar, Onur

Selecting a Deployment Automation Tool for CRM Software in Elisa Oy

Year 2013 Pages 37

Software passes through several phases during its production process which are typically designing, developing, testing and delivering. In the software development life cycle (SDLC), the ultimate aim is to deliver the software product to its intended users with the expected functionality. Deployment is a critical step which harnesses all the work done in the previous stages in the SDLC and makes the software available to the end user. Thus, failure in the final deployment stage will waste the effort expended in earlier phases.

The responsibility for deployment usually belongs to operations teams and it is done manually or then partially automated, often using inefficient scripts. Manual deployment can be extremely difficult task, and it is easy to blunder with repetitive routines consisting of many steps such as setting up similar environments and installing software components in those environments. Some operations teams attempt to ease the manual work by writing scripts to automate the process, but ultimately this method can become complicated and burdensome.

The deployment process delays can be avoided and redundant costs eliminated in the error prone manual deployment process by changing the work culture and automating said processes. There are several products on the market to help automate this process. Achieving fully automated provisioning is the ultimate goal to produce and update services rapidly within enterprise applications in large corporations. Full automation is accomplished when the environments are set up automatically, and software installation is automatic in those environments.

The objective of this research is to introduce these automation tools to the Customer Relation Management (CRM) system at Elisa Oy and to develop a proposal for automating the deployment process as an alternative to its current manual process. CRM as a large system consists of numerous sub systems and software components. A deployment case with a selected tool will be demonstrated to show how the automation can be accomplished.

Key words automated deployment, continuous delivery, deployment activities, deployment challenges, configuration management systems, service deployment tools

Table of Contents

1 Introduction

The process of turning a new idea into a product which is ready for the final consumer consists of several steps. In terms of software production, this process is called software development life cycle (SDLC). Deployment plays a critical part in making the product available to its end user. Handling deployments manually or with inefficient scripts can become a problem in the long run for large systems since is time consuming, repetitive and technically difficult.

Although software may work well in the development environment after passing certain tests, it is still not certain that it will work in the final machine in the consumer's environment. Even if, the software is successfully installed in the production environment, the software and the environment itself will need upgrades or changes later. These changes in the system and software should not affect the users with outages, session losses or defects in functionality. Correct deployment is defined in terms of given identical inputs: the software should behave the same on an end-user machine as on the developer machine (Dolstra 2006, 3).

Changing the work culture and automating the deployment process avoid delays and redundant costs by eliminating error prone manual deployment process (Edwards, Schafer, Shortland and Honor 2009). Achieving fully automated provisioning is the ultimate goal to produce and update services rapidly within enterprise applications in large corporations.

The main aim of this report is to select the most suitable deployment automation tool by reviewing major automation products which are considered to be used in Elisa's software production department for the CRM software. Before evaluating any of these tools, we will have an informative introduction related to deployment field. Instead of going in to detail of every aspect, we will mention the most related areas to understand the purposes of these tools and how they are used. Following section presents how the research work conducted, what are the gaps in current practices and how to fill those gaps to propose new solution.

The research study consists of theoretical and practical approach. Theoretical part is informative which points out the critical areas in deployment and introduces background in Elisa Oy in a sufficient level along with a case scenario where we can work on actual problems and search solutions for them. In the practical part, a selected tool is installed in a test environment and tested to see suitability to our needs. Informative section starts with presenting the basic terminology which will help us to understand the concepts throughout the research and includes the activities involved in the deployment process. After this informative phase, there is a case scenario from Elisa and analysis of some products in the market which will help to ease the deployment process. After analysis of the current deployment process and reviewing products in the market, we test an automation tool candidate and make a sample

deployment. Finally, outcomes and suggestions are presented according to these analysis and experiments.

Automating deployment is a gradual work and takes a considerable amount of effort from planning to implementation. Every system and software are unique with varying requirements, so there is no single solution fits for all covering a full range of deployment activities. Hence, in different situation or case the needs must be identified and a solution must be selected with highest benefits encompassing new policies and automation tools.

2 Basic Terminology

In this research paper software refer to components which make up larger software and can be reused to make other software with same component pool with different combinations. Therefore, software as a service must be perceived each time dependencies between components are mentioned. The environment and provisioning of this environment for the software are also other dependency variables related to software.

Development stage in SDLC starts in the developer's machine which is called development environment. Before the software is delivered it passes through many phases and tests. Manual tests are not enough to confirm that the application is running correctly. Number of testing environments can vary but they are mainly unit tests, user acceptance tests, quality assurance tests, manual tests and performance tests. The software is deployed to these environments for testing and if it successfully passes these stages it is finally deployed to the production environment. Production environment is the final destination where the software lives and it reaches its intended users. This environment is the most critical one and its configuration is duplicated in test environments to see any change affects as early as possible in the testing stage. All the requirements of production stage are copied to test environments to create the similar feedbacks on updates or changes. Production environment consists of one or many servers and other hardware, required operating system and supporting components with configurations of these systems.

Version control systems (VCS) are used to support collaborative work in a project with many developers. Any change made to the project is traceable with version numbers of the software. First developer checks out the code from the master which mirrors the project to developer environment from a repository server. Developer creates a new branch and starts working on a new feature or just simply fixing a bug. Then he checks in these changes back to mainline. Hereby all the source code is updated and a new version of the software is saved in the repository. Distributed VCS allows many developers to work on the same project but in

different tasks. (Chacon 2009, 4)

Writing the code in high level languages (Java, C++, Python etc.) is not enough for the software to run. In order to execute the software, this high level language first must be translated to computer language by compiling. Binaries as output of compiled code also must be packaged for distribution to the environment for execution. All this process is called build process and there are many build automation tools which compile the code, package, distribute, deploy, run tests automatically and copy these files to other locations. Build automation is an important part of continuous integration and delivery.

Development team collaboration has become a necessity recently named DevOps consisting both software developers who develops the application and operations team who makes the application available and maintains the infrastructure. This collaboration eliminates frequent contacts to developers what went wrong during deployment. If the delivery process is divided between different individual groups such as development, DBA, operations, testing etc. the cost of coordination of these silos is difficult after deployment. (Humble and Farley 2011, 6) All the organization must be involved in the delivery process from build and release team to developers, testers and operations. Devops is actually a new cultural aspect how software development is combined with deployment pipeline instead of being realized as separate processes. One of the aims in Devops is that it tries to deploy as fast as development develops to ensure fast and efficient reaction to business changes. In short any code change should reflect to the production environment as soon as it passes all the stages in development and testing. This culture is governed by agile principles and appropriate automation tools selection. (Watson 2013)

3 Deployments Value in Business and in SDLC

Business requires the ability to adapt fast changing market demands and business needs. Fundamental business process in today's companies is getting the idea from inception to where it generates revenue rapidly. Shortly if someone has a good idea, the question is how it is possible to deliver that idea to users quickly. In IT dependent company's software is only ready to generate revenue until it is made available to end users. This means developing the application and checking it into source control does not complete the value stream until it is deployed to customer site. Updating the software and successfully deploying each time as often as business requires is the most favourable output a software development and operations team can produce. If an efficient deployment pipeline is not established the deployment will take many days until the software is ready for end users. This is not encouraging for a company which needs to make several changes a day and make those changes available to users. If

anything goes wrong at this stage it might induce outages and even worse for example, the customer will be charged but his action will not be recorded in the system. An unstable deployment pipeline might cause monetary and reputational losses. (Edwards 2010)

Software delivery and deployment differ from each other in the context of SDLC. Software delivery includes all the necessary work to deliver the software across all phases of SDLC although deployment is just a subroutine in SDLC implemented several times repeatedly in environments such as developer workspace, QA and UAT testing, performance test environments towards production environment. For example, you can deploy rapidly to a UAT environment but it is not delivered until it is deployed to production and made available to users. Deploying the successful builds to production environment is the actual act what makes the software available to the user. It also requires configuration of various environments for installation and execution of the software in that target environment. Since an application is deployed to production only after passing several stages, deployment is more frequent in preceding stages than the production environment. The deployment process includes complex tasks like building from the source code, packaging, distributing, installing and activating (instantiating, initializing and executing the binaries). Delivery process ensures that business is always ready to respond quickly to the market changes whereas deployments feed this process through increased efficiency in IT operations. (The International Foundation for IT 2009)

Figure 1 from IBM is a rational unified process presenting software development contents and frequency of tasks in each content. We can see deployments relation to other contents and frequency in different phases of development.

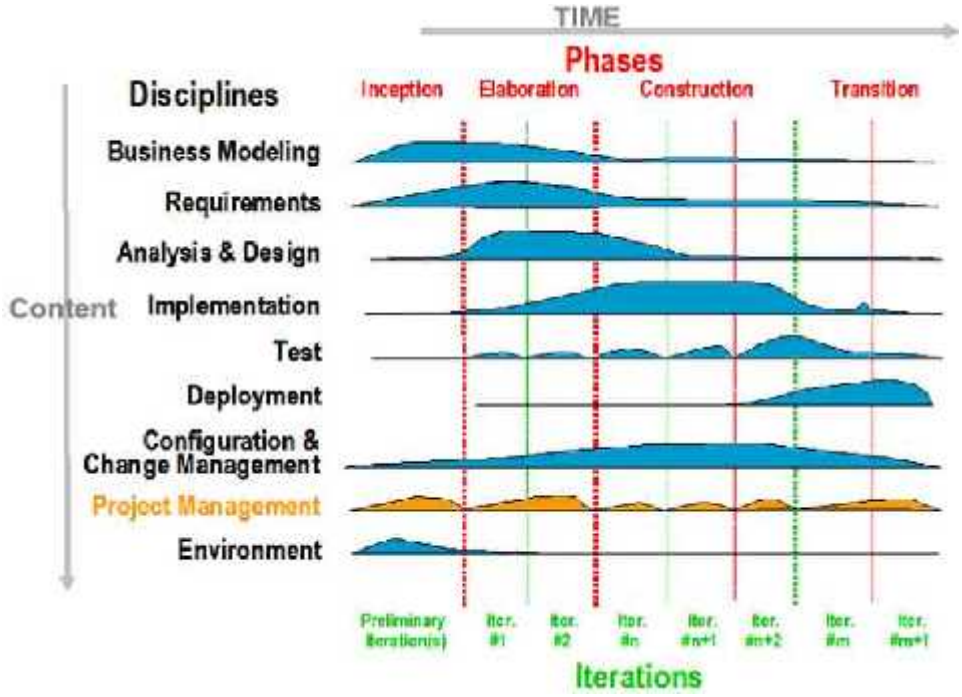

Figure 1: Deployments place in Rational Unified Process

In the diagram below the delivery process and its relation to deployment is presented. Deployment is involved from construction to delivery.

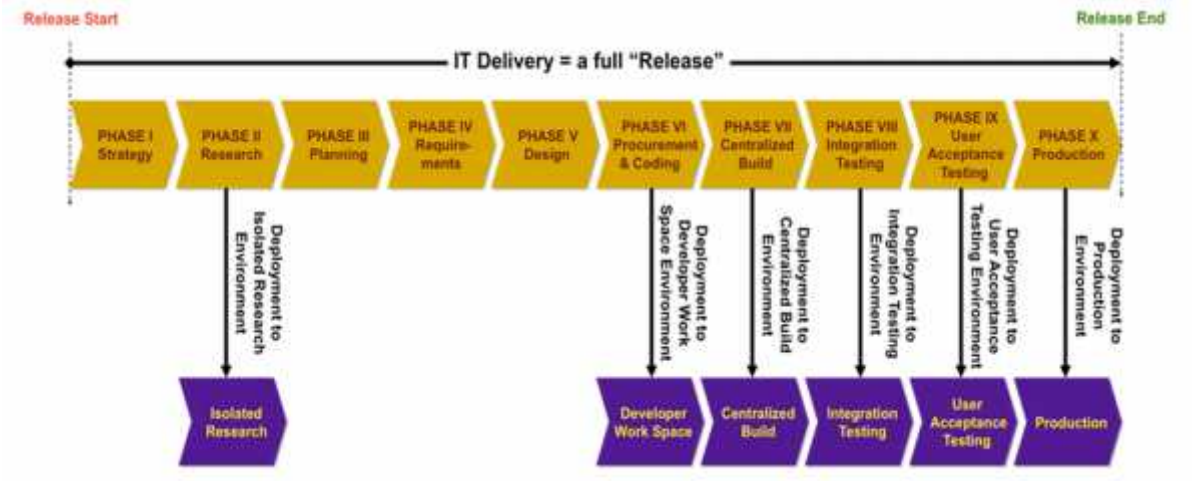

Figure 2: Delivery process and deployments.

4 Deployment Process and Activities

The deployment process encompasses certain activities and most of them are done in a specific order where as some other activities are done throughout the whole process. Applying

these activities by scripting deployment and configuration files is not expressive enough to manage the dependencies and configurations are hard to verify. But this method is practiced very often. Even if scripts are used, the best way to ensure correctness of the scripts is to use same scripts in test and production environments. By doing so, if something goes wrong during deployment it is for sure that the problem is not about scripts but instead it is environment specific configuration. If the environments are not identical, that adds up to the challenge.

The ideal deployment involves only two human tasks for deployment: picking the version of software and pressing a deploy button. This action triggers numerous activities automatically and makes debugging easier. Any change in any kind in executable code, configuration, host environment and data, the delivery system should trigger a feedback system which initiates tests. According to results feedback is delivered simultaneously and the action is taken according to feedback. Figure below shows how a change is moving through a deployment pipeline (Humble and Farley 2011, 109). Deployment activities must be automated where possible to avoid each error prone steps in activities described below.

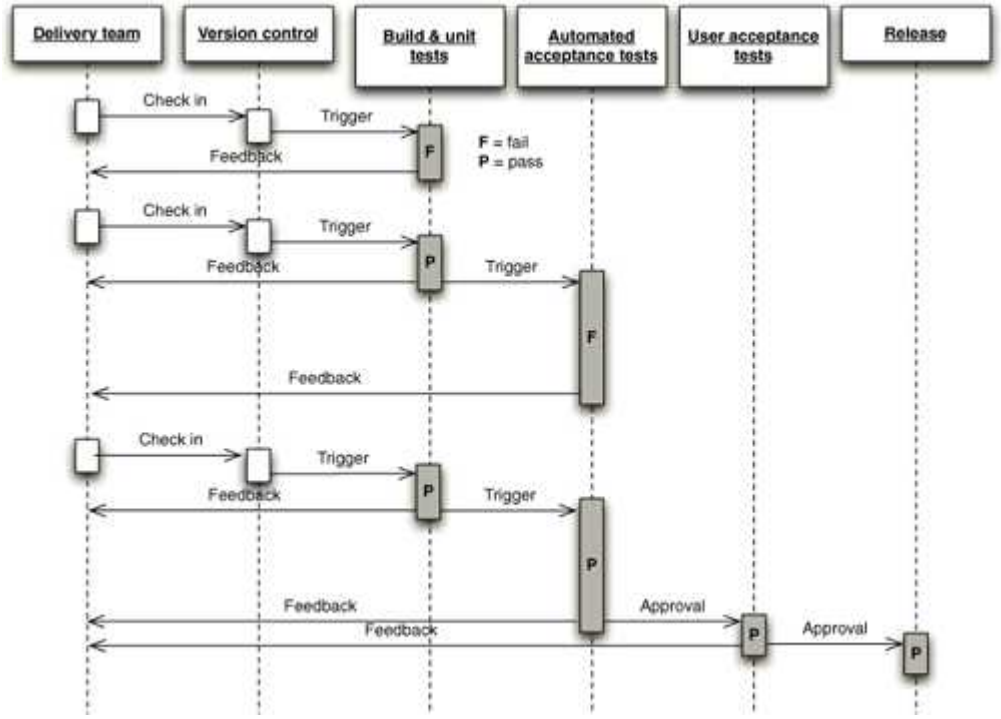

Figure 3: Changes moving through deployment pipeline

In this section, all main activities are reviewed from development to final delivery. These actions are always present whether the process is automated or not. The best way to avoid human errors of course is to automate as many of these activities as possible.

4.1 Release

Release phase starts right after development stage connecting it for the processes prior to installation and the rest of the deployment covering the whole following process. It encompasses assembly of all required files, packaging, transferring them to the relevant destinations and feedback monitoring after installation. Required resources are determined here to ensure the system has suitable environment. Information related to other activities in deployment is collected for analysis and monitoring. Documentation is recorded regarding to deployments including product version, product configuration, interfaces, deployment time and delivery personnel information.

All the stakeholders involved in the software product are informed about the changes and update. If this information is not accurate or missing, it will create customer dissatisfaction. Most of the information is presented during sales for clean installations whereas less information is needed for upcoming changes. (Mantyla & Vanhanen, 5)

The users are trained about the new features through several materials like manuals, online instruction links or videos and on site meetings. After the training, there is an active customer support to further help the customer.

Deployment is scheduled with the customer whether it is a clean install or an update. Customer can do the deployment themselves but in some cases experts from third party vendors might be required to be present during deployment.

4.2 Packaging

After the assembly of the files comprising the software, they are packaged in specific folders in a certain order so that they can be transferred to the target environments such as testing and production. This package must contain the system components, deployment descriptor specifying the procedure, a system description which includes its requirements, external component dependencies and all other system management information in consumer site.

4.3 Distribution

The software is transferred several times to different environments with many servers or nodes until it is finally moved to production environment. Product is first transferred to test environments from development then to production environment when it passes all the necessary tests. Distribution of the archive files are mostly done with interconnected networks and transferring packages to the environment. In some cases, distribution to customer CDROM

is still used via deliverable CDs. Today's most deployments use the internet as a channel for distribution such as FTP and other web technologies.

4.4 Pre-installation Checks

In order to make a new installation, there are preparation activities implemented in customer site. Customer data is imported to the environments and some conversions between data types might be required. It takes long time to implement these data structures in test and production environments if data models are complex including massive databases.

Before or during installation the product is configured according to customer needs. At this stage both the customer and the vendor is present since they learn from each other about the software capabilities and customer business process respectively. Creating software which is configurable according to diverse customer needs is a technical achievement.

Software is integrated to customers other systems by using interfaces in middleware technologies with the help of vendor and customer experts. Several components talk to each other through interfaces or calling methods in other parts of the system.

Backup of databases and executable files are performed before installation. If there are periodic backups already, there isn't need for doing it before installation.

4.5 Installation

Installation is the most critical step in deployment since the actual merging of the software and environment is done in this stage by recruiting all necessary resources from the system. Pre installation checks are performed in advance regarding to environment, dependent external components, resources and integrity of the installation package. At this stage, the packages are already transferred to target environment and configuration are completed for the activation of the software. After the package is transferred to the environment it must be extracted for binary execution therefore, a package extraction program should be triggered first.(Carzaniga, Fuggetta , Hall , Heimbigner, Hoek , and Wolf 1998, 5)

4.6 Updates

Updates are easier than clean installations since installations require implementing customer specific data models and configurations in the beginning. Updates are a simple type of installation where the minimal configuration changes might be required without more comprehensive adjustments in the environment. Clean installations or certain updates are done by ex-

perts from vendors or third party agents. Simple updates can be done by the customer following instructions or vendor can help remotely. Figure below shows an update moving in a basic deployment pipeline from a commit stage to deploying into certain environments (Humble and Farley 2011, 111).

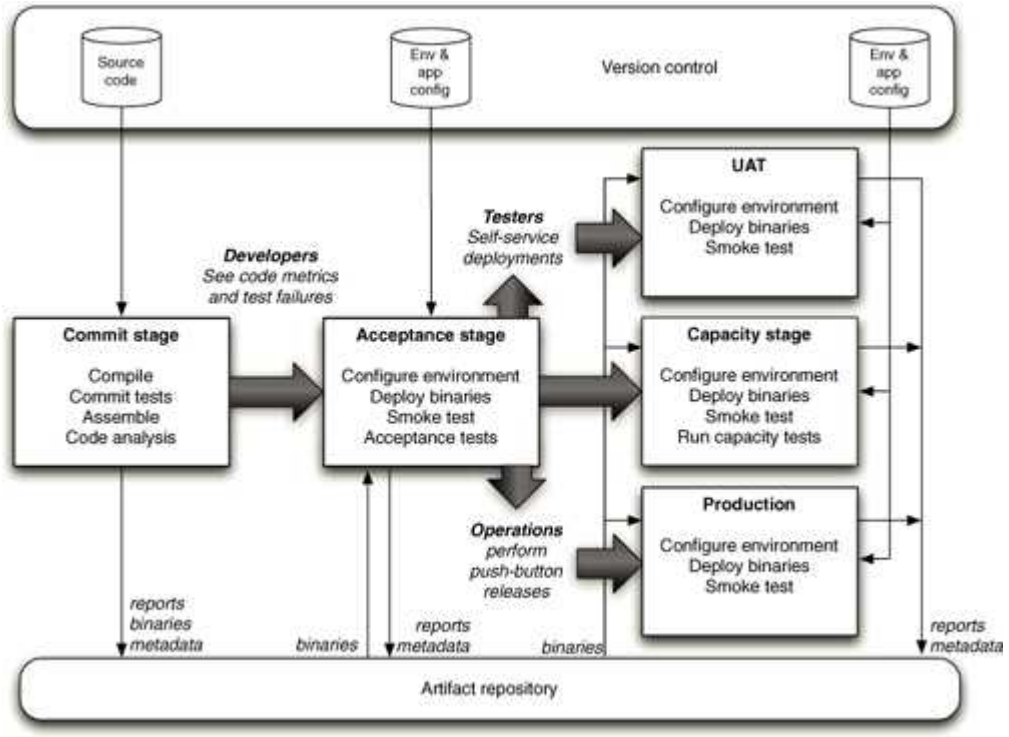

Figure 4: Basic deployment pipeline

4.7 Activation

After installation, the executable files should be run in order to start all the services which make up the whole system. Executing binaries will make the system available to the user for running the intended business processes. In clean installations, the servers are stopped and restarted before the software is activated after installation. There might be also other systems need restarting which are relied upon.

4.8 Testing

Vendor tests the software according to customer specific aspects in their premises. If a full test cannot cover all aspects due to lack of identical environment specifics in customer site, more testing is done in customer site with cooperation of the customer and vendor experts. Most ideal situation is when an environment identical to production environment can be created. But that might be challenging since customers doesn't have resources to duplicate the

production environment.

4.9 Deactivation

Deactivation is shutting down an active component of an installed system. Deactivation occurs in order to update a part of the system. After the update, the deactivated component is activated again. If the intention of the deactivation is uninstalling the software, then activating again is not needed.

4.10 Adaptation

Adaptation is required when an environment changes where the software resides. In environment updates hardware, operating systems or configurations changes are unlike software updates which the product itself changes. This requires the software to adapt these environmental changes. An example can be if we rip off one of the servers from the system, system should still adapt to the new model and serve to its users without any interruptions or operational malfunctions.

4.11 Removal

When customer business process change in a way that the system is no longer needed the software can be uninstalled after deactivation. After uninstallation other part of the system can be reconfigured to function correctly after the removed software. Dependencies are considered where other systems interacted with the software when it was part of the mechanism so that removal doesn't have side effects.

5 Deployment Challenges

Back in the days, deployment was as easy as just inserting a CD to the customers CD-ROM for updating or installing the new software. With the advance of the internet, it is done via varying sophisticated data communication channels requiring configurations in the hosting environments. First the source code is developed and tested then followed by a complex release process. The environments that host the software are often crafted individually by operations team. Third party software that the application relies on is installed. Configuration information is copied to the production host consoles of web servers, application servers, or other third party components of the system. Reference data is copied and finally the app is started, piece by piece if it is a distributed or service oriented app. Difference in ordering and timing of these steps can lead to a different outcome. So it is not simply copying binary files to cus-

tomers system. There are several issues to consider such as managing dependencies, orchestration and configurations, variables in heterogeneous environments, change management and teams collaborations. We don't aim to solve all of these issues but it is beneficial to point out as many of them as possible now for allowing future development.

Software demands correct configurations and certain resources from its environment in which it runs. These dependencies are mainly configuration files, modifications in operating systems, existence of other dependent software's and available resources like memory or network connections. Therefore, delivery team first checks if the other components exist in the customer site and the configurations are fulfilling the software requirements accompanied by adequate clusters of servers and other technical pieces. Configurations in individual machines in clusters must be identical. This kind of environmental challenges are addressed in configuration management tools which are particularly designed for easing the system side issues of deployment.

Deploying to large systems brings more challenges than simpler systems. Continuously evolving Web 2.0 companies use software-as-a-service (Saas) business models and N-tier architecture in their products such as E commerce services or social network applications. Updating these systems may require changes in thousands of machines and services in those machines might have varying dependencies in heterogeneous environments. An intermediate enterprise web application consist of at least three main tiers as the web, application, data tiers rather than having simpler two tier client-server architecture. In a most elemental form, an N-tier architecture will include a client environment where a user touches the system, an HTTP server to deliver web content to client, an application server for business logic and database server for storing the data all running in separate machines. Each of these tiers will be partitioned into sub tiers according to workload and architectural needs. The amount of tiers is directly related to the complexity of the system (Mukhar & Zelenak 2006, 6). Management of these systems encompasses complex relationship between human and computer tasks. Automation in deployment purposes should focus eliminating manual human tasks.

Today's software is far more different than their stand alone, self-contained predecessors. Heterogeneous components form inter-dependent relationships to construct multi-purpose mechanisms. A component actually will be built upon dependent pieces including data, other executable files, documentation and instructions at separate locations in wide area networks. Developing and deploying enterprise level business application software requires thorough planning and implementation techniques especially when working in large teams, long-lived services and numerous interconnected systems. These mechanisms house multiple services which encapsulate and interfaces dependent components. A change to any of these components is going to affect the other services functions which use the dependent components. An

implicit dependency in a development environment may impose a hidden risk when it is implemented in a deployment environment. Applying component upgrades or removing a component will overwrite or delete a functionality affecting part of the entity if that component is still in use by others. Therefore predefined steps must be followed safely in order to manage any changes in packaging, distributing, installing, updating and uninstalling. Deploy personnel should also be able to react to third party component updates or changes that are not under their control.

Environments are often shared between several applications which cause complications. Extra care must be taken when preparing the environment for a new deployment of an application so that it will not disturb the operation of other applications in the same environment. This is done by ensuring that changes to the configuration of the operating system or any middleware don't cause the other applications to misbehave. There shouldn't be conflicts between the chosen versions of the applications. The applications sharing the environment may depend on each other as a common practice in service-oriented architecture. Dividing an application into a collection of loosely coupled, well-encapsulated, collaborating components is not only good design. It allows for more efficient collaboration and faster feedback when working on large systems. In this situation, the integration testing environment is the first time that the applications will be talking to each other involving deploying new versions of each of the applications until they all cooperate. At the end smoke test suite which is set of acceptance tests will run against the whole application.

Advancements in data communications and increased usage of the internet made it possible to download and install third party components after reading their specifications and requirements. Even if the component is found with the functionality needed it must be still compatible with the rest of the system and able to run in target environments. These components require reliable connections to talk to others. Integration with the internet reveals another dependency variable which makes deployment a distributed problem. Third party components are usually part of other entities at the same time. Any change to these third party services will affect the system especially if there is a more tightly coupled relationship. If these challenges are not realized as nature of the modular architecture, it will encourage monolithic application development which is an inefficient way of production causing software duplications. However, solutions must be thought modular architecture in mind.

The hardware consists of mainframes, workstations, servers, personal computers and mobile devices, all possibly running different operating systems construct a heterogeneous environment. All these platforms work together and challenges exist for software development considering interoperability. Platform types are one of the variables which involve configurations and dependencies. (Carzaniga, Fuggetta , Hall , Heimbigner, Hoek , and Wolf 1998, 7) The

automation software therefore must comply in all these environments and act as a platform and language neutral agent.

Another issue is that updating software in heterogeneous networks is not as easy as updating single technology family. This issue might create security risks as software need to be updated frequently and it should be always up to date. After determining whether updates are available, the update decision is made considering behaviour of the updates software and its dependencies.

Development team is responsible for making the changes and developing the software whereas operations team is responsible for the maintenance and availability of the environment where that software resides. Agile and lean development techniques allowing release ready software has grown faster than the operation teams capabilities which caused a bottleneck in deploying the applications. This resulted in delays, long outages and lack of confidence in deployments. There is also a necessity for experts' presence like a database or system administrators during deployments since they might be the only ones who have the environment and product domain knowledge in case a complication occurs. Experts deal with repetitive and trivial tasks rather than more valuable creative tasks. These tasks are not repeatable for the future use since it is done manually. Using a common automation tool for both developers and system operators will reduce the dependency to experts and related troubles in their absence while increasing confidence in deployments by avoiding unpredictable outcomes. This practice will increase the knowledge transfer between developers and operations personnel and create more available manpower. Most of the critical deployment tasks occur in customers' site which requires authenticated access by only proficient personnel. A proper deployment automation tool will give the appropriate admin rights in customers' site for a flexible and effective activity management.

A flexible deployment process allows migration between environments or adapts changes quickly which is always a present possibility in business and technology. Otherwise, migration can become a major issue causing delays in project schedules. The system should not depend on any machines in the network. When a server is down or it is upgraded the deployment pipeline must be able to react these changes and still keep the functionalities as expected. (Nayak, Sudarsan, Venkataramappa, Wang and Williamson 2005. 2)

Manual deployment requires extensive documentation which describes detailed information related to steps to be taken. This documentation has to be updated frequently which is time consuming involving many people. It is hard to verify who followed the documentation and implemented it accordingly. Automated process is more auditable than the manual one. (Humble and Farley 2011, 5)

6 Case Scenario

We have identified the activities in general and issues related to deployment. In this section, we introduce the complexity of the system used in Elisa and current software development process. This section will emphasize the practical challenges and reasons why automation is required including in what spectrum of activities that it can be utilized. A sample deployment with a selected automation tool is going to be presented afterwards.

The background information about the system and the process is acquired by interviews with software development manager and dedicated CI team which consist of a DBA, CI architect and a CI consultant. These personnel took part actively during the research with discussions, technical support and providing necessary information. Internal documentation included complete information which is an active wiki continuously being updated and improved by all developers. Another information resource was accessing the scripts themselves in Git repositories of projects which were used in deployments.

6.1 Background of the System

Elisa is a large IT company and it manages its internal and external operations with advanced and complex technologies. The company has been in the market for many years and eventually its system and software is aged. Businesses have merged with other companies acquiring their systems with different technologies and sizes over time. The legacy systems are combined with today's latest technologies making it complex heterogeneous structure. There are different projects and products managed separately in Elisa. Therefore, the tools and processes used in each one of them vary. One of systems like the CRM is spread over a wide area network in Finland and abroad with over 2.5 million users. High number of developers contributing in this system makes team collaboration and orchestration a challenge. Many departments in the company utilize CRM along with other systems for their internal business process and managing customer interactions. Numerous subsystems and software components make up to the bigger CRM system. One of the small applications is chosen as an input for the automation tool candidate to see how the candidate handles the process.

CRM is used by customers, resellers and the company representatives. Customers can see and manage their account information. They can view information and make purchases of various products and services. Resellers have all required system functions such as reaching detailed

product and services information, communication and support. Company representatives use this system through its advanced interface to search and manipulate any customer, product and service related information.

The architecture used for enterprise level CRM application development is N-tier architecture and components have dependencies among each other across the tiers. Depending on the business requirements the system uses a variety of software and hardware technologies. Several operating systems are used such as most of the UNIX systems like Solaris, Linux versions and others such as Windows operating systems. Software languages including Python, Java, Perl, Scala, Groovy, JavaScript and shell scripts are employed throughout development, coordination and scripting purposes. Different products from varying vendors are part of the system such as Tomcat, Resin and Jetty for application servers, MySQL, Nosql etc. for databases and Linux machines for web servers.

In CRM system, the dependencies between components are complicated due to interdependent 200 components in addition to around 240 third party components and libraries. Currently there are 136 software projects in different developer environments maintained simultaneously. Dependency figure from IVY dependency manager below shows dependencies in a part of the system.

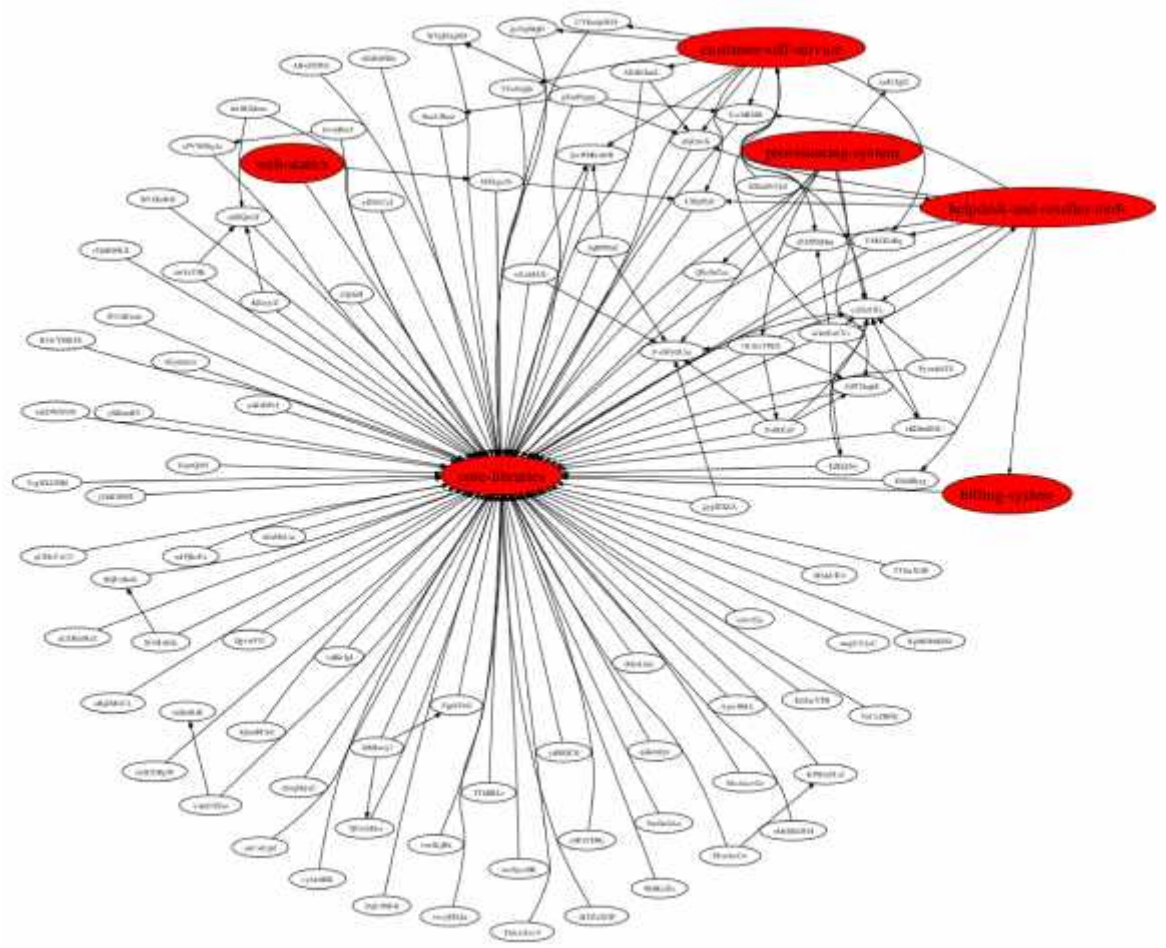

Figure 5: CRM dependency graphic

The introduction above gives a clear picture that this system requires sophisticated techniques for development, management and delivery. Current methods used in deployment are not efficient enough and we will be looking for better solutions with automation. In the later sections, the deployment automation tools are analyzed for this purpose.

6.2 Background of the Development and Deployment Process

Continuous integration practices are applied currently by a CI team in the software development department and it is improving towards continuous delivery. Automated deployments comprise important part of the continuous delivery in addition to automated builds and tests.

This section explains the stages from development to deployment currently in practice in Elisa OY for the CRM software. First developers in the production team check out and create a new branch for a new feature from VCS. The development process starts in the developers' local machine mostly using Eclipse IDE. The new code is built and tested locally using ANT

build automation tool. If tests are successful, the build artifacts are saved in local Git repository then these changes are pushed to the version control system which triggers Jenkins continuous integration (CI) tool for integration and dependency tests. First, It compiles the changed module and its dependencies followed by testing all modules. After passing this stage, the artifacts are saved in Artifactory repository manager. Finally, all the code is merged to the main trunk in VCS and tested several times such as automated acceptance tests and manual tests. At this point, we have the code for a new feature or a bug fix, next the deployment of this new version to the production environment is done manually and partially by scripts. Production environment consists of application servers and databases. The technologies used in this environment are same as afore mentioned background of the system part.

Developers and other participants in projects use wiki for collaboration, documentation and communication. Some automation tools send feedback right after developer actions such as in CI environments but communication needed in the deployment process is managed with manual communication means. If there was an automation tool, it could be able to report and log deployment activities and their results automatically.

Current deployment is done by scripts additional to manual configurations which are hard to maintain and requires on-duty experts during deployments. Users bear the risk to lose sessions during updates and changes. It has a possibility to cause outages therefore a specific time in a specific day is chosen for deployment to affect minimum amount of users. Any changes in the system are not managed in a flexible and confident way. If a problem reveals after deployment to production, a manual rollback and undoing changes are also a time consuming task. In the situation where the problem cannot be fixed quickly, the change must be reverted to the previous change in VCS. The release ready product is not tested thoroughly due to different testing and production environments therefore unexpected failures are likely in deployments. Currently a team is assigned to create an identical test environment to production. There are 20 deployments in average per month and improvements in deployment pipeline can increase this frequency and its quality. Automation tools and changes in delivery culture can be a solution to many issues in deployments.

6.3 Defining a Sample Deployment Process

A case scenario for deployment is the most basic form of deployment for the beginning. A service among many other services which consists of CRM system is deployed to a certain server with certain configurations. In order to limit scope, other services in Elisa Oy is not mentioned other than CRM. Sessions of the users should be preserved during deployments. Any outages should be avoided if deployment goes wrong by a rollback reaction from the sys-

tem. Deployment history should output the deployment time, version number and deploy personnel. Initially this is an ideal deployment but in the future number of requirements can increase.

7 Deployment Approaches and Architectures

Approaches for deployment are mainly manual, script, language and model based methods and each one has some advantages for special cases. A small scale system can benefit from a manual or script based method with its fast start-up and easy to learn feature benefits. But larger scale systems require language or model based methods. These more advanced approaches come with the cost of complexity and learning curve (Figure 3). They are not as fast and easy to adapt as script based approaches. Below graph gives a clear picture of costs and benefits of each approach. (Talwar & Milojicic 2005, 71).

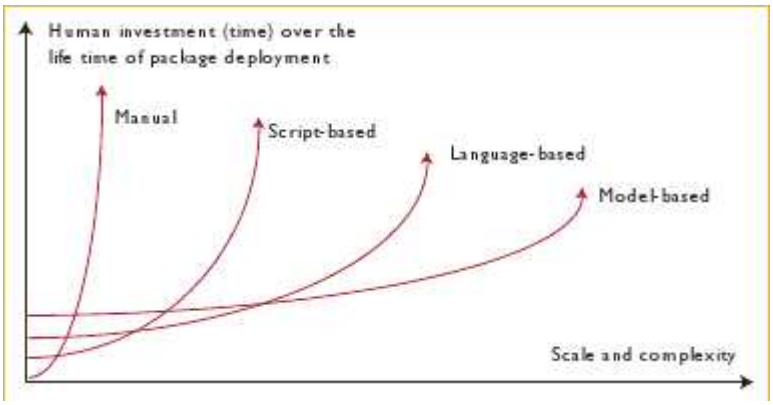

Figure 6: Deployment Approaches

Depending on the complexity of the system and needs of the organisation a single deployment tool or a tool chain might be required. The most common solution for managing automated deployments in large systems is separating the process in two main tiers such as configuration management(operations side) and application orchestration(developers side) (Figure 4). There are tools which can manage both of the tiers in small scale systems with some extent but none of them are comprehensive enough to cover all tasks in each tier within the whole spectrum. Therefore, an application tool chain is recommended as configuration management systems (CMS) and application service orchestration tools. Advanced CMS tools are usually model based whereas application deployment tool can be sufficiently script based. Combining these two tools will give more flexibility for large systems.

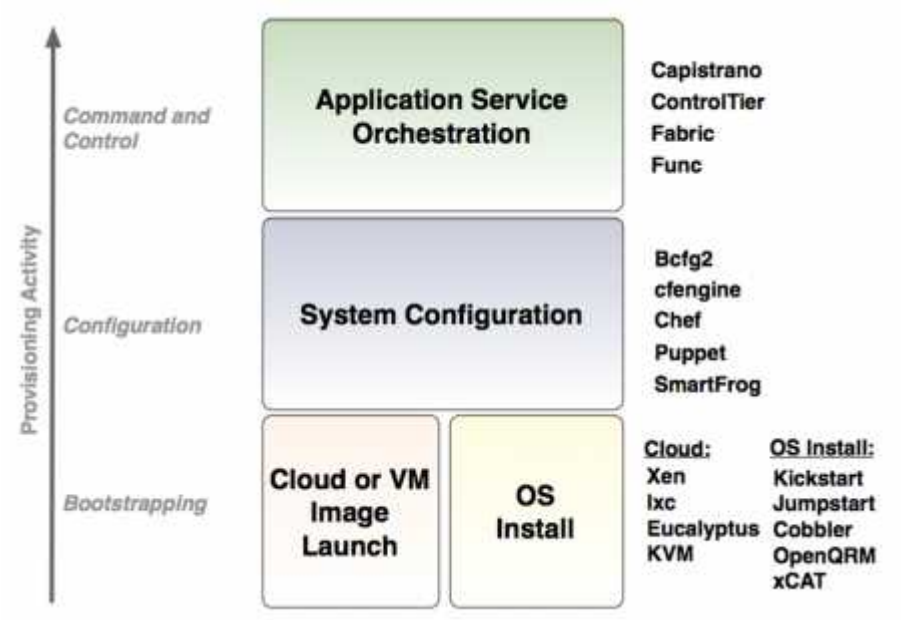

Figure 7: System configuration tier is for CMS and Service orchestration tier is for deployment tools.

Fully automated provisioning ensures the ability to deploy, update, and repair the application infrastructure. All deployments, updates and fixes must be done through this tool using only pre-defined automated procedures rather than accessing to the individual machines. These activities are done through a provided specification to the tool and the rest is handled automatically by starting up, configuring and deploying the entire mechanism from bare hardware(or virtual machines).(Edwards, Schafer, Shortland and Honor 2009)

Below we will look more in detail what are the typical features offered by these technologies and products. In general, they include system and configuration description, packaging, distribution, installation, security, update and network management.

8 Major Deployment Tools

As we described before, there are two main tiers considering deployments automation which are configuration management and software deployment tools. CMSs are beneficial for describing how the infrastructure should look and maintain it that way. Since CMSs are not designed for remote scripting, software deployment tools complements this task. They are reviewed in two different sections and tables are drawn to compare fundamental features and properties.

There are large amount of products in the market and it is impossible to review all of them. While selecting these products criterias should comply certain conditions such as popularity, large community support, active development, ease of use and special features. The review list is not limited to open source products since a commercial tool might provide a superior feature which is worth the costs. Nevertheless, if a commercial product is not distinctive enough, we preferred to review open source tools.

Popularity and broad community support ensure to find help quickly when there is an issue with the set up or implementation. Active development shows that the product is not abandoned and is still evolving by trying to answer market needs. Features and attributes of the products what makes them special if they can offer a solution to known problems. User friendliness is a matter since there will be many users using this tool including managers, junior developers, software developers and system administrators. These were the most fundamental guidelines to select possible candidates for further review.

There are three options for deploying into remote machines. The most powerful one consists of a tool chain which packages the application up using platforms packaging technology and has an infrastructure management or deployment tool to initialize the middleware. Deployment tools like Rundeck and infrastructure management tools like CFEngine and Puppet are declarative and idempotent, ensuring that the right version of the packages is installed on all necessary boxes. So this option employs two separate tools for CMS and deployment purposes.

If this option is not feasible, the second option is to write a script that runs locally, and have agents that run the script on each of the remote machines. Continuous integration servers like Jenkins run pre-written scripts in remote machines as if they are running locally. These scripts are saved in VCS. It also provides jobs which runs during a failure, displays console output and provides a dashboard where monitoring deployment including deployment status, software version and environment information.

The third option is to write a script that logs into each server and runs the appropriate commands. Tools like Fabric, Func and Capistrano helps to script own deployments and SSH to execute commands on remote machines (Humble & Farley 2011, 161).

The following review section is going to mainly focus on deployment tools but it will also introduce CMS tools since they are important complementary part of the tool chain.

8.1 Configuration Management Systems

System management and configuration is a crucial part of the deployment and it prepares the environment and resources for software. Configuration management refers to the process by which all artefacts relevant to project, and the relationships between them, are stored, retrieved, uniquely identified, and modified. It sets the OS into a correct state with a desired patch level for the intended application so that the application can be deployed there. All the environments and configuration of third party elements should be applied from version control through an automated process. Automated configuration management allows to repeatedly recreate every piece of infrastructure used by the application. Infrastructure includes the OS configuration, application stack, its configuration, infrastructure configuration and so forth. The changes in the production environment must be recorded and audible.

Correct state of the environment involves certain update packages and configurations. Most of these CMS tools also have the ability to do these configurations as well as maintaining them. They can also deploy the software but we are going to review some other more specific tools for deployments and have an overview on CMS tool for pure configuration. The table below (figure 5.) shows major CMSs currently in the market. The attributes of the table lists the specifics about these tools. The language is the programming language used to construct the product. Mutual authentications manage and identify users. Verify mode ensures if the configuration works before actually installing configurations.

	Language	License	Mutual auth	Encrypts	Verify mode	First release	Latest stable release	number of platforms supported*
Ansible	Python	GPL	Yes	Yes	Yes	2012-03-08	2013-02-01 1.0	6
CFEngine	C	GPL, COSL	Yes	Yes	Yes	1993	2012-07-20 3.3.5	8
Bcfg2	Python	BSD	Yes	Yes	Yes	2004-08-11	2012-07-03 1.2.3	5
Chef	Ruby	Apache	Yes	Yes	Yes	2009-01-15 0.5.0	2013-02-13 11.4.0, 2013-02-15 11.0.6 (server)	8

Puppet	Ruby	Apache from 2.7.0, GPL before then	Yes	Yes	Yes	2005-08-30	2012-10-18 3.0.1	8
Quattor	Perl, Python	EDG, Apache 2.0	Yes	Yes		2005-04-01	2012-02-22	2
Salt	Python	Apache	Yes	Yes	Yes	2011-03-17 0.6.0	2012-07-30 0.10.2	6
SmartFrog	Java	LGPL	Yes	Yes		2004-02-11	2009-01-26 3.16.004	5
Spacewalk	Java (C, Perl, Python,PL/SQL)	GPLv2	Yes	Yes		2008-06	2012-03-07 1.7	2
STAF	C++	CPL	No	Partial		1998-02-16	2012-06-29 3.4.10	8

*Platform support for AIX. BSD, HP UX, LINUX, WINDOWS, MACOS X, SOLARIS. OTHERS
Figure 8: Comparison of major open source CMS tools.

8.2 Software Deployment Tools

Systems consist of many services and servers while each of these specific services must be deployed into certain servers. Software deployment tools coordinate the deployment of each service into correct servers and configure environment specific components. Each time a service is deployed other related components need to stop and start in a specific order. This orchestration is also ensured by deployment automation tools. The coordination occurs between components in all tiers in N tier architecture. When all services are orchestrated and started successfully, these services run all together to make up the business application. (Edwards, Schafer, Shortland, Honor, Thompson 2009) In figure 6 the features of the major deployment automation tools is listed. The commercial tools like Nolio and Deployit is not included in our table since these products can be overkill in our initial transition period to deployment automation. These products are more than simple deployment tools offering solutions in other layers such as provisioning and configuration management. Some of the commercial products

have proved themselves over time and they have big customer base. In order to find simpler tools, such commercial products are excluded.

	Lang.	Roll-back	Ses-sion	Histo-ry	UI	Comm	latest ver-sion	Architec-ture
Rundeck	Java	Yes	Yes	Yes	Web+CLI	Ssh	2/21/2013 1. ver.	stand alone
Fabric	Py-thon	scripts	No	Yes	CLI	Ssh	1/3/2013	stand alone
Func	Py-thon	scripts	No	Yes	CLI	Xmlrpc	7/4/2011 and still active(see git repo)	server + agent
Capistrano	Ruby	yes	Yes	Yes	CLI	Ssh	still ac-tive(see git repo)	server + agent
Mcollec-tive	Ruby	Yes	Yes	Yes	CLI	Msg que	14/02/2013	server + agent
Liverebel	Java	Yes	Yes	Yes	Web+CLI	ssh	3/18/2013	server + agent

Figure 9: Comparison of software deployment tools

Language: Language is the programming language which this software relies on in order function and coordinates the activities. The user of the tool should understand the corresponding languages to benefit more from these products.

Rollback capability: In case a deployment goes wrong this ability will allow to go back to last working version. Therefore, there is no need to for manual work and doing all the work again for deploying the last working version.

Preserving Sessions: Users loose sessions during deployments due to restarting many services and servers. Preserving sessions will avoid any interruptions and allow users to use the application without realising there was an update.

Execution History: Tracking changes becomes useful when details regarding to deployment needed to be reviewed. Most basic attributes of the history are the personnel who made the deployment, version number and time of deployment.

UI: Graphical user interfaces are more user-friendly and allows deployments to anyone who is not familiar with the system. Instead of writing command line entries deployment is done by only push of a button.

Communication: The deployment tool eventually will need to communicate with all servers in the system in order to manage deployments. These products use technologies like ssh, xmlrpc or others for most secure way for communication. While ssh is useful for a single machine manipulation remotely it is not designed for multi-system remote scripting. (DeHaan 2008)

Latest version: Latest version ensures an active development of the tool if the date is recent. This is a proof that the product is not abandoned or reached its end of life cycle and it is still in development.

Architecture: Most of the systems have server-agent architecture which requires a central management console and agents installed on remote machines. But there are simple stand-alone products which are basically installed in one location and they send commands to remote machines.

9 Selecting a Tool Chain

One of the biggest challenges in CMS tools is adaptation and learning curve of the tool and domain specific language. Puppet is chosen for CMS purposes, since it was already in use in other projects even though it was never used in the team which is responsible for CRM. Since Puppet is used in other project it is beneficial to inherit the knowledge form those projects and experts. An initial interview was held with an expert to see the Puppet scripts from a git repository. There will not be any integrity and compatibility issues in the future if we merge those projects which use Puppet. Puppet also has support for most of the platforms and the community behind it is still active. Verify mode allows Puppet to try a change before actually implementing it which is a valuable feature. Closest candidates to Puppet could be CFEngine, Bcfg2 and Chef due to their active development, large community and advanced features.

Puppet is one of the most popular open source systems currently available along with CfEngine and Chef. The underlying principles of this kind of tools are the same. Puppet manages configuration through a declarative, external domain-specific language (DSL) tailored to configuration information. This allows for complex enterprise-wide configurations with common patterns extracted into modules that can be shared. Thus, configuration information duplication can be avoided. Puppet configuration is managed by a central master server. This server runs the Puppet master daemon which has a list of machines that it controls. Each of the con-

trolled machines runs the Puppet agent. It communicates with the server to ensure that the servers under Puppet's control are synchronized with the latest version of the configuration. When a configuration changes, the Puppetmaster will propagate that change to all the clients that need to be updated, install and configure the new software, and restart the servers where necessary. Agents will be aware of these changes and pull the configurations. The configuration is declarative, and describes the desired end state of each server. This means they can be configured from any starting state, including a fresh copy of a VM or a newly provisioned machine. The user doesn't need to know ruby to use puppet even though the product is written in ruby language. But instead it uses a meta language and operator needs to understand this DSL in order to make configurations and extensions. This is usually the case in this kind of tools. Mode of operation is that deamon pulls the configuration. But in deployment tools user pushes the changes. (Humble and Farley 2011, 258)

When it comes to deployment tools, we chose three products for further review and these are Liverebel, Fabric, and Rundeck. These three tools got separated from other with different reasons which are listed below. There was some time reserved to try how these software work and their working mechanisms were analysed. This review allows estimating what kind of software is needed and how it is going to be used.

9.1 Fabric

Fabric is one of the tools we reviewed due to its simplicity. It was also used in another project other than CRM so it was easy to get some sample scripts and interview with the systems team. Another reason we included this tool for a further review is a ready knowledge base is an advantage while choosing these kinds of products. Fabric is just a Python library with predefined functions which can run from the command line directly. The operator should know python in order to make use of python scripts, combine them and extend functions when necessary. A rollback option will be available if scripted. User pushes changes to remote servers with ssh. In a sample scenario, a web application is managed via Git on a remote host vcshost. On localhost, we have a local clone of said Web application. When we push changes back to vcshost, we want to be able to install these changes immediately on a remote host server in an automated fashion. We will do this by automating the local and remote Git commands.

Fabric consists of 12 modules which can be used in a custom made fab file to run local and remote commands. Operations module includes many useful functions and they are as follows:

fabric.operations.get, downloads one or more files from a remote host.

fabric.operations.local, runs a local command
fabric.operations.open_shell, Invoke a fully interactive shell on the remote end.
fabric.operations.prompt, Prompt user with text and return the input (like raw_input).
fabric.operations.put, Upload one or more files to a remote host.
fabric.operations.reboot, Reboot the remote system.
fabric.operations.require, Check for given keys in the shared environment dict and abort if not found.
fabric.operations.run, Run a shell command on a remote host.
fabric.operations.sudo, Run a shell command on a remote host, with superuser privileges.

There are other modules including tasks, utils, network, docs and so on. The classes and functions can be extended and new ones can be created using the Python programming language. Therefore, Fabric becomes easy to implement and extensible choice. It doesn't have a GUI but can be run from command line. It is well suited for basic tasks since it is easy to adopt and flexible.

9.2 Liverebel

Liverebel made it to the final candidate list because it tries to do both configuration management and deployment in an efficient way. This tool was the only commercial one and we installed a trial version in our test environment for testing. The product offers two options to manage configuration by either including configuration specifics in the jar file or as an additional separate file. It has a user-friendly GUI additional to its command line interface. Users can be defined to operate the deployment process with a username-password authentication. However, it is designed mostly for Java world and this is a minus for the current environment since there are other technologies as well. But the product recently started to support PHP and Perl which is an improvement. During the trial period, the product was locked on a single page after an upgrade which made us hesitant to recruit this tool.

Some of the key features of Liverebel are presented as instant live application updates, preserving user sessions during updates and executing custom scripts. Deployment is made easy with one click if the configurations are done correctly. During an online meeting with Liverebel team they presented how the product works. There are two servers running the application therefore there are not session losses when each server is stopped, updated and restarted in turns. If something goes wrong after deployment, system automatically rolls back to its last healthy state. Task log allows user to follow what is happening in the background during updates.

Below is a snapshot from the GUI. After selecting a server and a software version clicking “deploy now” button will deploy that software to the intended serer.

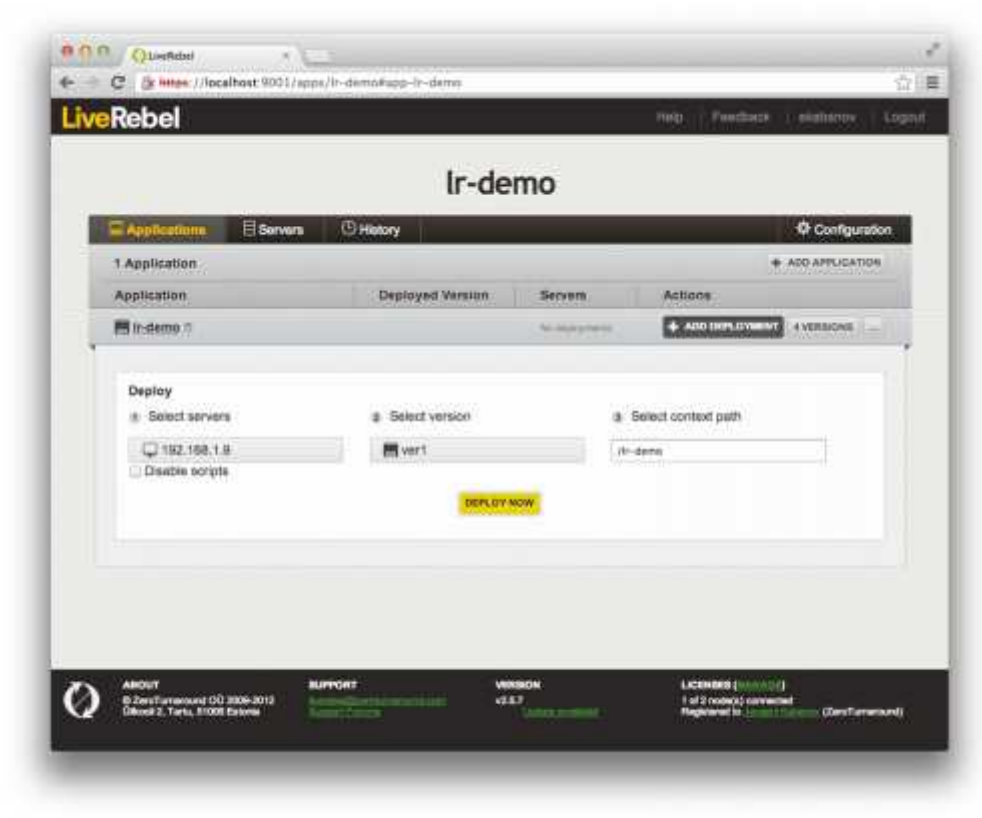

Figure 10: Liverebel GUI

9.3 Rundeck

Rundeck is the last product we have reviewed more in detail and experimented by installing in a testing environment. It is a spin off project from a successful predecessor called ControlTier. Rundeck offers user-friendly GUI additional to CLI. It is written in java but the users don’t need to know Java due to easy user interface. It is possible to create workflows by combining scripts and jobs. Role based access control with LDAP support allows access to users with different background. Architecture involves a server and communicating remote agents. It has rollback capability and many more modules for common tasks. Rundeck can

save execution history in a relational database or file-based data storage. It can also import node sources from Puppet or others which makes it easy to cooperate with CMS tools.

Jobs and ad-hoc execution are some of the main features of Rundeck. Jobs keep a sequence of steps, node filter and job option. Then a Job with a unique ID can be run to execute all these steps. Jobs can be composed out of many other jobs which will make it a step in the sequence. On the other hand ad-hoc commands can execute individual shell commands as a user would run it at an interactive terminal. GUI includes three main tabs such as run, jobs and history(Figure 11). Run and jobs are for managing command execution and history is to review the changes. Run screen allows seeing live execution revealing all events in the background. Jobs can run on a schedule or on demand and people can be notified when they are completed.

Figure 11: Rundeck main tabs.

10 Solution to the Case Scenario

It is beneficial to use two separate solutions for system configuration and service deployment. These two separate solutions will replace different manual or scripted steps in the deploy-

ment process which was counted in the section “Deployment process and activities” and it will automate the whole process. Puppet is the strongest candidate in among CMS tools due to its support for most of the platforms and large community support. There are several examples in the market which shows how to combine Puppet with other service deployment tools. Puppet is used in some other project for purely configuration purposes but not in CRM. The ready knowledge base in the company is going to decrease one of the biggest costs related to learning curve and adaptation phase. It gives the opportunity to interview with the experts in that project and review their sample configurations and scripts. There is no other superior alternative to Puppet which can make us ignore the advantages of accessing ready knowledge in close premises. Puppet is responsible for ensuring consistent system configuration and deployment tool will handle more dynamic aspects like deploying new releases of the custom applications or ad-hoc server monitoring.

Service deployment tools can be combined with CMSs and there are varieties of choices. The most suitable software which can company Puppet is Fabric and Rundeck. Liverebel was eliminated due to its Java specific requirements which don’t suit well in our heterogeneous environment. Fabric can be used to automate some simpler tasks but managing whole process will get difficult to maintain. Rundeck on the other hand proved to be the most suitable tool due to its rich feature list and easy to use GUI.

One of the most fundamental features of Rundeck is the Jobs and workflows which can organize commands, group them and run them in target environments or clusters. Saving commands as Jobs or Ad-hoc commands helps to centralize system manipulation and makes it more auditable. There will be only a single place to run commands which prohibits accessing the individual machines.

User friendly GUI allows easy control for users with different background such as managers, junior or senior developers and system administrators. It can authenticate these users with different profiles through its role based access control mechanism. Making this product usable for several type of users will accomplish a change in work culture and move it into Devops as mentioned earlier. Therefore it will eliminate or decrease the need for on-site experts significantly.

Rollback ability will allow fast recovery from a failed installation. This will increase the confidence in deployments and the installation schedules will not be limited to certain hours where there is least amount of users. Sessions of the users also could be preserved when configurations are done correctly. Increasing confidence in deployments will ensure better safety and in return it will increase frequency of deployments which is a desired outcome.

Execution history reports the user who deployed the application, time and project details which assists for trouble shooting. These history reports can be used to improve the deployment pipeline in later stages.

It can import CMS node resources from Puppet which makes Rundeck a suitable product chain for a final solution. It is designed to work with many other CMS tools as well such as Chef and Amazon EC2. It also supports many platforms and programming languages even though it is written in Java. These properties gives this product a flexible quality in design.

Rundeck has an active community with many participants in the market. They have active discussions in forums where they meet other users and developers to solve common problems or make improvements. Large community is a good sign to see how popular is the product in the market and makes users more confident when they can discuss with their common issues.

The product is a spin-off project from its successful predecessor ControlTier. Rundeck is still active in development with the recent version 1.5 released on 21.2.2013. Active development means that this automation tool will react market needs and improve over time.

In conclusion Rundeck becomes a strong candidate for deployment automation purposes for the reasons mentioned above. Elisa Oy is planning to implement concrete tasks to see the product on the job. By the time of writing, the product is installed in two servers on a test environment and reviews are still being made by the CI team. A fully working implementation can be reached by gradual steps and it will take considerable amount of time and effort. A fully automated deployment process can be reached with the help of other tools on CMS side such as Puppet. Puppet is suggested to accompany Rundeck in tool chaining for automation purposes

References

Carzaniga, A., Fuggetta, A., Hall, R., Heimbigner, D., Hoek, A. & Wolf, A. 1998. A characterization framework for software deployment technologies. University of Colorado. Department of Computer Sciences. Technical Report. Accessed 23 February 2013 http://www.ics.uci.edu/~andre/papers/T3.pdf

Chacon, S. 2009. Pro git. New York: Apress.

DeHaan, M. 2008. Open source project: Func the fedora unified network controller. Redhat Magazine. Accessed 17 March 2013 http://magazine.redhat.com/2008/02/21/open-source-project-func-the-fedora-unified-network-controller/

Edwards, D. 2010. DevOps is not a technology problem. DevOps is a business problem. Accessed 3 February 2013. http://dev2ops.dtosolutions.com/2010/11/devops-is-not-a-technology-problem-devops-is-a-business-problem/

Edwards, D., Schafer, A., Shortland, A., Honor, A., & Thompson, L. 2009. Achieving fully automated provisioning. Accessed 17 March 2013 http://api.ning.com/files/aVcfIQKPor*gYXVVIm1Wiix6FnVIDx3NgNc5PrbhcQRE5t6CHBDFz6LvJLBeZCfa0Og6A93E6DJyhK70c45C3bTOZc7qCKXN/FullyAutomatedProvisioning_Whitepaper.pdf

Humble & J. Farley, D. 2011. Continuous delivery. Boston: Pearson.

Mäntylä, M. & Vanhanen, J. Software deployment activities and challenges – a case study of four software product companies. Aalto University. Accessed 17 March 2013 http://www.soberit.hut.fi/~mmantyla/CSMR_final_mmantyla_install.pdf

Nayak, R., Sudarsan, S., Venkataramappa, V., Wang, Q. & Williamson, L. 2005. Automated deployment of an application. United States Patent Application Publication. Accessed 11 March 2013 http://www.google.fi/patents?hl=en&lr=&vid=USPATAPP10874495&id=F5iRAAAAEBAJ&oi=fnd&dq=automated+deployment&printsec=abstract#v=onepage&q=automated%20deployment&f=false

The international foundation for IT. 2009. IT deployment framework. Accessed 12 March 2013 http://www.if4it.com/SYNTHESIZED/FRAMEWORKS/DEPLOYMENT/deployment_framework.html

Talwar, V. & Milojicic, D. 2005. Approaches for service deployment. Hewlett-Packard Laboratories. Georgia Tech Accessed. 13 February 2013
http://www.cc.gatech.edu/~calton/publications/IC-05-03-SF-eval.pdf

Watson, M. 2013. Defining the dev and ops in devops. Accessed 1 March 2013.
http://devops.com/

List of Figures