TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Engineering

IAF70LT

Eduard Mubarakšin 122081IASMM

LARGE SCALE DISTRIBUTED ENTERPRISE COMPUTING SYSTEM INFRASTRUCTURE SETUP AND MANAGEMENT AUTOMATION ARCHITECTURE

Master Thesis

Jaak Kõusaar
M.Sc.
Raimund-Johannes Ubar
D.Sc.

Tallinn 2016

Author’s Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Eduard Mubarakšin

[Wednesday 25th May, 2016]

Abstract

With the move from monolithic applications to more agile microservice verticals, the number of applications in a company is on the increase. This means that infrastructure will need to keep up with the increased number of requests, hosts to manage and reduced time to market what means less time to setup each application.

In order to accelerate application landscape creation and processes involved with it, the overall application lifecycle is revised. Intention is to illustrate how configuration management tools can automate the infrastructure platform by combining different aspects of development into an orchestrated architecture compiled of existing building blocks. The key feature in this process being automation and re-usage in order to create a process that can cope with legacy systems, yet be agile enough to adapt to future technologies.

This thesis is written in English and is 77 pages long, including 12 chapters, 40 figures and 5 tables.

Annotatsioon

Suuremahulise Äriettevõte Hajusa Arvutisüsteemi Infrastruktuuri Seadistamise ja Haldamise Arhitektuur

Agiilne revolutsioon ja liikumine monoliitsetelt rakendustelt teenusorienteeritud ning mikroteenuste suunas, on veebirakenduste tarkvaraarenduse tempot ning nõudlust tunduvalt tõstnud. Suuremõõtmelistes firmades, kus iga rakendus on äri kriitilne ning informatsioon tundlik, ei saa ettevõte pilve arhitektuurile korraga üle minna. Selle tõttu pidades jääma ajutiselt vanamoelisema, virutaalmasinatel põhineva arhitektuuri juurde.

Firmasisese klastri haldamine pole agiilset revolutsiooni läbi teinud. Esimene katsetus selles suunas on Devops, mis üritab infrastruktuuri muudatusi parandada ja protsesse automatiseerimisega kiirendada. Selleks, et seda edukalt teha, on enne vaja saada põhjalik ülevaade protsessidest ja otsustest, mis leiavad aset infrastruktuuris rakenduse eluea ajal. Uuritakse mis võimalusi praegused automatiseerimise ja konfiguratsiooni-haldamise tööriistad pakuvad. Kuidas neid ärakasutades muuta olemasolevad protsessid agiilsemaks ning modernsemaks. Pakkudes kõigile osalistele automaatikaga kaasnevat mugavust, ning klientidele kindlat teenuse kättesaadavust, mida pakub kaasaegne virtualiseeritud infrastruktuur.

Töö annab ülevaate rakenduse elukäigul kaasnevatest protsessidest. Tutvustab turul olevate konfiguratsiooni manageerimis-tarkvara pakutud võimalusi. Lõpptulemusena pakutakse välja jaotatud tiimidele sobiliku raamistiku kirjelduse, mis praktilise juhendina võimaldaks parandada koostööd ja vähendada manuaalse töö osakaalu infrastruktuuri ülespanekul, manageerimisel ja haldamisel. Näidates praktiliselt juba implementeeritud näidete põhjal kuidas olemasolevate protsesse automatiseerimine aega säästab, tõstes kvaliteeti ja tagades töökindlust.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 77 leheküljel, 12 peatükki, 40 joonist, 5 tabelit.

Table of Abbreviations and Terms

API	Application Programming Interface
APZ	Application Zone
ASM	Application Security Manager
BO	Business Owner
BU	Business Unit
CI	Continuous Integration
CM	Configuration Management
DMZ	De Militarized Zone
DSL	Domain Specific Language
ES	Elasticsearch
HDD	Hard Disc Drive
HW	Hardware
OPI	Operational Instructions
OS	Operating System
PO	Product Owner
QA	Quality Assurance
SCM	Source Control Management
SOAP	Simple Object Access Protocol
STS	System Test Staging
SYS	System Test
UAT	User Acceptance Test
VM	Virtual Machine

Contents

List of Figures

List of Tables

1. Introduction

Keeping clients pleased with software development requires rapid and steady stream of new features released to the consumers. Quick time to market for new business needs. High level of Quality Assurance to avoid bugs reaching clients with quick response to bugs with hotfixes in case a bug is discovered. Achieving high quality of releases would be very difficult without releasing often and getting constant feedback from testers and users. Especially in IT businesses where the clients need new features on the market and they need the features live as soon as possible.

These new requirements for speed from the business units has caused the “agile revolution”. Making the software development more agile needs a lot of changes in the workflows of the company. Majority of the improvements are targeted at the software developers themselves and the business processes surrounding them. Some times over-looking the processes involved with infrastructure.

To achieve a quick time to market, and to please the clients, it is first essential to please the developers by offering them the best available tools. Not only having in mind the integrated development environment and agile coaching, but also providing a good infrastructure where the staging and test environments to run on. Push-button deployments, continuous integration and automated test-suits are only few key words of the possible ways to ease the life of the developers and become truly agile from start-to-end of an application life cycle.

To achieve fast pace and guarantee high quality of released software, the infrastructure needs to support the developers and work hand in hand with agile methods. Provide a smooth transition throughout the test environments while giving comprehensive and comparable test results between different stages. However that is not where agility ends for infrastructure. Automation does not end with only the development process and providing tools to get the developed artifacts tested for maturity and then to release.

The agility infrastructure needs to provide has a large role in providing the means for scalability, maintainability and high availability. Making it easy to get the application to production. Provide updates and hotfixes swiftly with minimal or zero downtime. Adapt to increases in usage and quick disaster recovery. Provide transparency for the developers and support personnel to find and fix problems quicker via monitoring and performance profiling tools.

The pipeline from zero-to-application in an infrastructure can over time turn into a complex process. Due to large segregation between functional units and people leaving the company. Having a clear high-level understanding of the whole process is vital to understand the bottle-necks in the process. An overview of currently used tools and the values they provide to the stakeholders can give good insight to further improve the overall agility of the process and thus reduce the time-to-market. Making convenient tools and approaches available to more of the teams.

In addition having an overview of solutions already used in the company is important to keep teams from re-inventing the wheel. Keeping things more similar across different distributed teams. This leads to a more uniform setup making the life of both Developers and DevOps easier. Such approach makes server setups easier to predict and maintain, while uniforming the company application landscape.

1.1. Purpose

The purpose of this thesis is to map and capture the needs of more prominent functional units within a larger-than-average software development company. Analyze the needs and the workflows involved with the individual functional units. Investigate the currently available and used solutions for process automation. Describe a functional work flow for an agile corporate application infrastructure creation architecture that provides as much value as possible for each functional unit.

Keeping in mind maintainability, usability and circumventing limitations set by the available infrastructure or business processes already in place. The resulting analysis would serve as documentation for existing and possible future improvements alongside with a comparison of benefits what moving to automation has brought. The practical examples are taken from application infrastructure of Kuehne+Nagel and as such, only a general overview is given with some details being not open for discussion.

1.2. Objectives

This master's thesis has the following objectives listed below.

- Distinguish the major roles in the application lifecycle what are affected the most.
- Capture and describe the main tasks of the major roles distinguished.
- Describe the needs of the roles to establish a common understanding of the process.
- Introduce the currently available tools that satisfy the needs of the stakeholders.
- Propose a possible solution for a smooth application lifecycle management architecture capable of providing a pipeline from zero to application.
- Illustrate the key stages of setting up a working solution.
- Practically show the advantages and benefits provided by automation, compared to manual methods.
- Investigate container technologies as a future possibility.

1.3. Organization of the Thesis

This dissertation gives an overview of the usage possibilities of the application configuration management tools in an automation-driven application landscape creation workflow. Thesis is organized as follows. Chapter 2 introduces the general application lifecycle related processes, captures the main stakeholders in the process and the needs they have to improve their work.

Chapter 3 analyzes the needs captured in Chapter 2 and maps the different requirements to the tasks and units in the organization that are responsible for carrying out the task.

Chapter 4 introduces the tools chosen for each group of tasks. In chapters 5 to 8, selected tools are introduced closer and compared to one another.

Chapter 9 highlights an automation-based documentation approach concept that is integrated into the general workflow and scripting approach.

Chapter 10 gives a practical example of the infrastructure scaffolding approach, describing the usage of the automation tool in with an easy use-case. Compares the automated approach with manual methods, proving that there is no way forward without automation.

In Chapter 11 container-based technologies are introduced that simplify the complexity of an application landscape creation by using containerization. Reasoning why this technology is promising, but needs time to mature.

Chapter 12 gives a summary of this dissertation.

2. Business Process and Needs Description

The following section will be dedicated to describing the processes that take place during the application life cycle and thus generate the requirements for infrastructure. The following topics will be covered. The release cycle process and different environments behind them. Needs that build up the continuous integration and QA framework for the company. The work flow executed at each step and the importance of each stage.

2.1. Application Lifecycle Stages

Simplified model of stages an application will go through in its lifetime will be used and analyzed to identify the main actors in the application life cycle for needs analysis. The most common steps and work flows involved with each step will be introduced. The steps are described by The Agile System Development Life Cycle [1]. The main stages are listed on Figure 1 and each stage will be detailed in the following section.

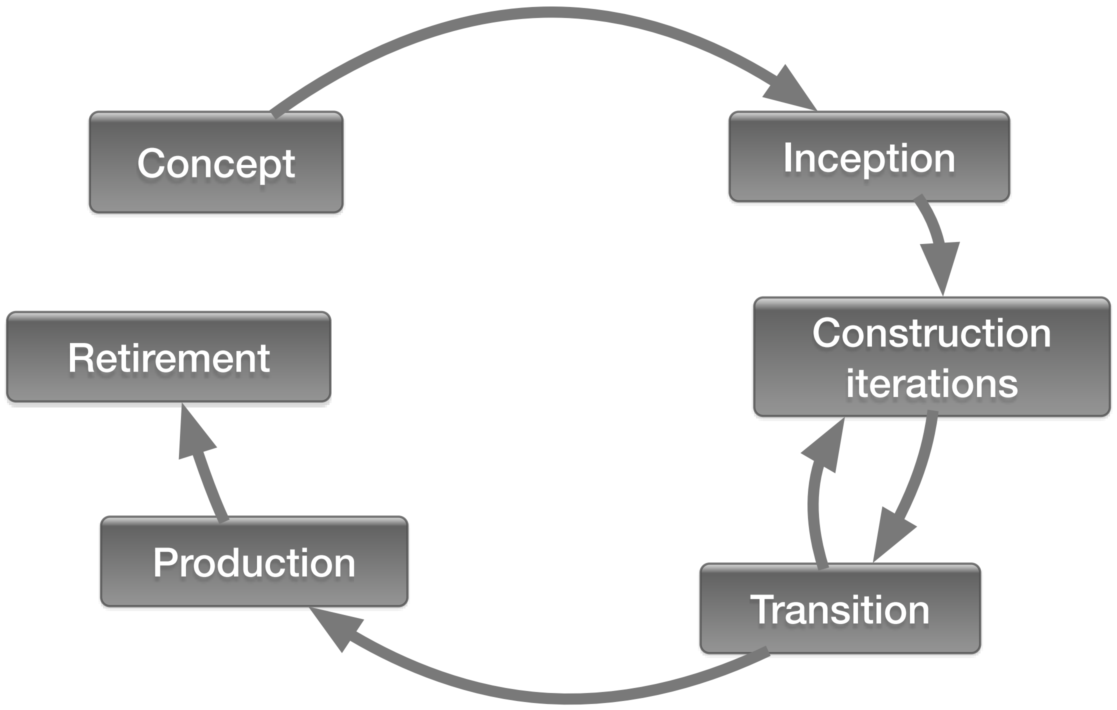


Figure 1. *Application Lifecycle Steps.*

Concept

The pre-project planning phase is the stage where the initial business needs are gathered from the clients. During this stage, the business opportunity is defined. Business owner or an in-organization consultant, with the largest knowledge of the business area is appointed. Business owner will be responsible for the well being of the business side of the project. The role will provide valuable information for the development along the life of the project.

Inception

Project initiation or inception phase is where the development team is either appointed to a project. Or a new team is created. The team lead, developers and the BO start working together on the first possible to release features. Functional requirements are created and business rules are established by architects. The development environments are set up either on existing infrastructure, or new hardware is provisioned. Continuous integration pipelines are set up and testing infrastructure is created according to the teams choice of tool set.

Construction Iterations

The main development process is the main state in what an active application would be in. The team keeps implementing requests for features in the release cycle. Developers and testers keep improving the test suits with the evolving software. Fix and maintain the production ready code base. The software base is kept up to date and updated to latest available technologies and infrastructure changes. Being that latest Java versions and the features provided by it for instance. Quality assurance keeps the code free of bugs. Provides a tactile confirmation that the system behaves correctly with the changes and fulfills both old and new business rules. The DevOps keep the infrastructure operating under constant work load and maintains the system to avoid stoppages. Maintenance is carried out in all environments.

Transition

The deployment phase is the stage where the release goes live. In this stage, all the integration is done. The release is tested across all available testing environments by the developers. QA guarantees functionality and carries out regression tests. The defects found by QA are fixed by developers and re-tested for confirmation. The users of the software are trained and user acceptance tests are carried out buy the BU.

New software is released at the end of each development cycle either by a release plan. Or urgent issues are fixed via a hotfix process initiated by the product owner. Once the developers have fixed the issue and QA confirmed the fix is production safe, the hotfix is applied. The transition phase and requirements for CI will be described closer in an upcoming section.

Production

The actual value proposition to the clients. The software developed to fulfill the business needs is released and kept in production. Customer support monitors the application and service life signs. Reacts 24/7 to incidents and provides customer facing support. Gathers the bug reports from clients and monitors application logs for common known issues that are not fixed. Act according to instructions or previous experience. The bug reports gathered from the users are reported to developers and BU. According to the importance and effect to the clients, either hotfixes are released or less critical issues are added to the backlog for the next release. Infrastructure keeps the provisioned application servers up to date and in good health, taking care of any problems that could cause downtime. DevOps keep the application up to date and make required changes as the application develops along its life cycle.

Retirement

End of life cycle. The application is either discontinued or replaced by a new vertical. The existing users are migrated to the new application if there is a successor to the discontinued application. The existing application goes to maintenance mode where no development is done. Alternatively the application is shut down completely. And the successor application starts its life cycle in production stage.

2.2. Deployment Lifecycle

Aside the general application life cycle. The deployment related processes, that focus on full delivery process[2], is considered a large part of day to day life. The deployment pipeline is one of the largest target for automation and provide the largest benefit to the stakeholders involved. In the following section an overview of application transition phase related processes and use-cases are described.

2.2.1. Weekly Release Cycle

The simplest transition stage is a single environment deployment. As an example, a weekly Systemtest environment deployment is illustrated on Figure 2, where the latest features that are planned for this release are tested daily on lower environments. If all tests pass, the issue or story is allowed to be merged to the test branch and is delivered to the corresponding test environment for intermediate testing or user acceptance testing.

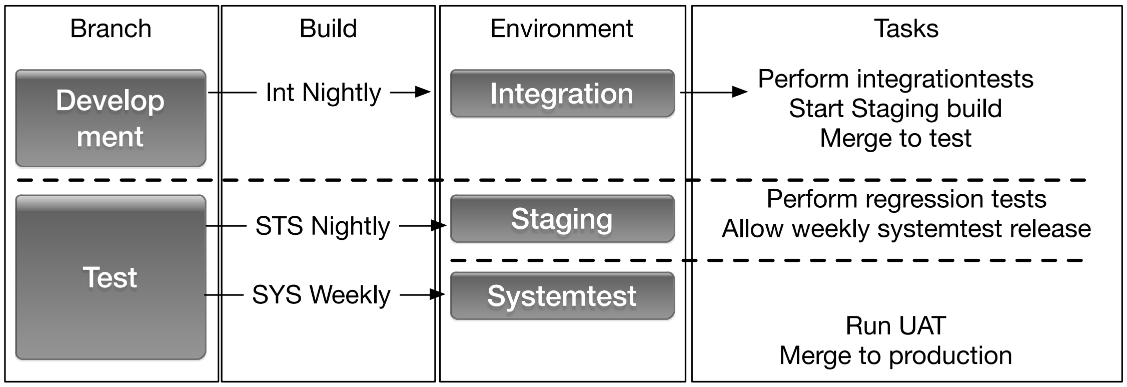


Figure 2. *Weekly Release Process.*

The weekly delivery to an environment, that is designed to be as close to production as possible, is part of the agile methodology to release and test often as possible on environments that mimic production. Systemtest, being the scaled down copy of production environment is the in house sandbox for customer demos, load tests and overall functional tests. During the whole release cycle, the latest artifacts are deployed daily to lower environments and weekly to systemtest. This goes on through out the release cycle until the deadline to release the features to clients and a stable release candidate is established.

2.2.2. Release Cycle

The main development revolves around a release cycle what involves periodical release of production ready features to clients. Figure 3 illustrates the cyclic movement of features from trunk to production branch.

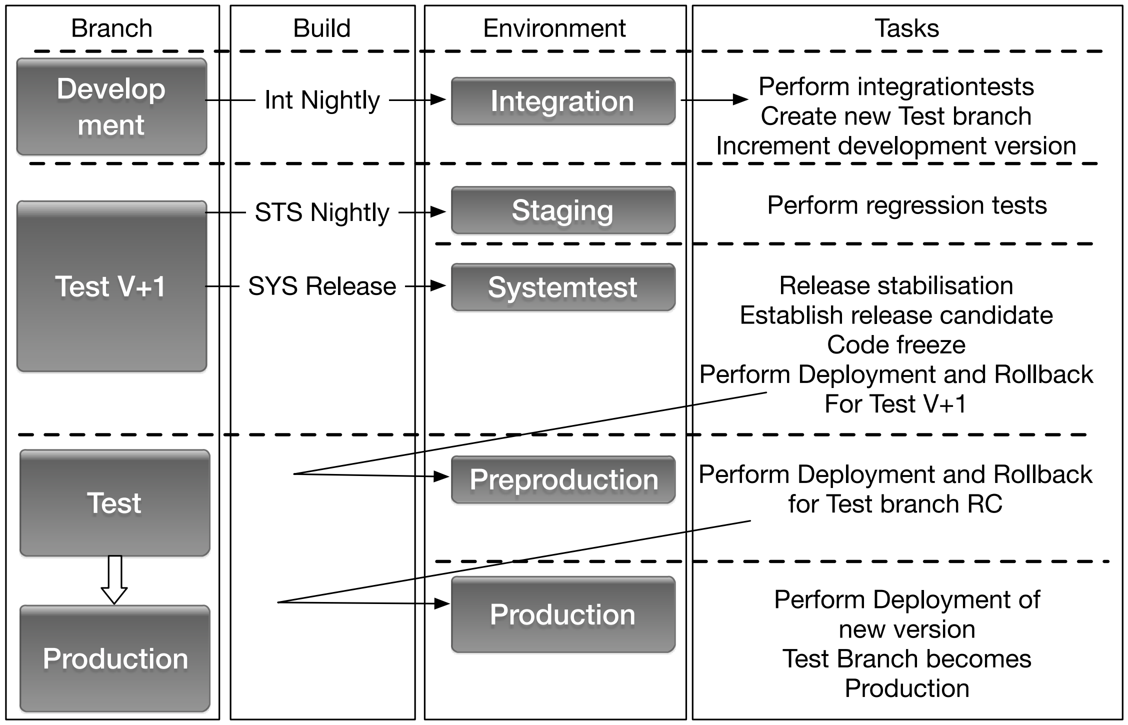


Figure 3. *Release Cycle Process.*

During a release, the version in trunk is incremented and all environments get deployed to a new release version. At each environment change, regression- and integration-tests are executed to guarantee that the release will provide the new functionality and not break the already existing features. Deployment and rollback tests are executed on Preproduction and Systemtest environments, to prepare for the worst-case-scenario on production environment to make sure no data is compromised with the rollback.

2.2.3. Hotfix Release Cycle

In case of more urgent issues [3], the problem is addressed with a hotfix. If the product owner classifies the issue as an emergency or critical issue then the issue will be fixed in the current sprint with high priority and pushed to production via a hotfix procedure. Hotfix procedure involves a sped-up development cycle and quicker passing through different staging environments. In order to reach production in a timely manner. The steps in the process are illustrated on Figure 4. With less urgent issues, the bug is fixed with accordance to the sprint planning.

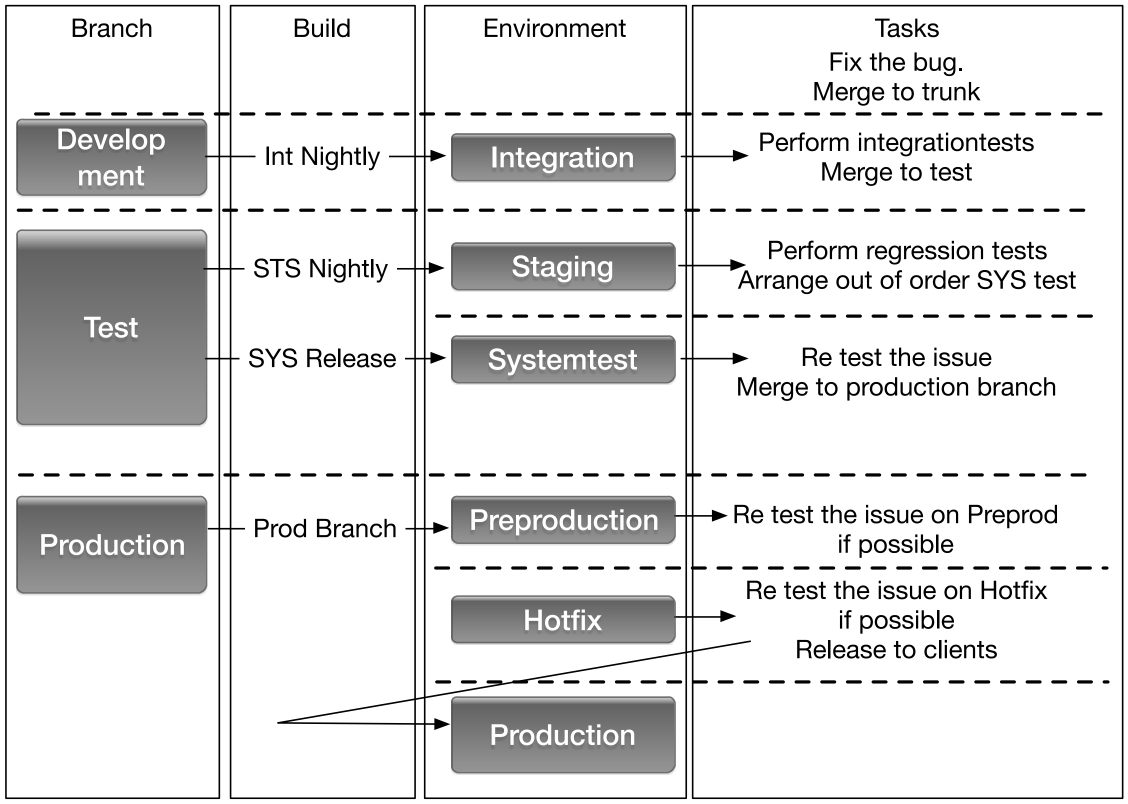


Figure 4. *Hotfix Release Process.*

2.3. Stakeholder Identification

From the short overview of the application lifecycle we can distinguish four larger groups of functional units with the largest impact on the whole project. The chosen actors are listed below.

- Product Owners
- Development and Devops
- Infrastructure
- Support

So far the whole focus has been on the agile development perspective, but for the purpose of this thesis the focus will be shifted little to the infrastructural aspect of the project life cycle. In the following paragraph a short overview of the lifecycle from the infrastructure side is given in order to understand the importance of infrastructure in the agile workflow. As a by product of shifting towards the infrastructure, the importance of developers decreases a little. Making the key role in the infrastructural changes the DevOps. As DevOps team is the main drivers in infrastructure changes and inter-department communications. Their change in importance is introduced on Figure 5.

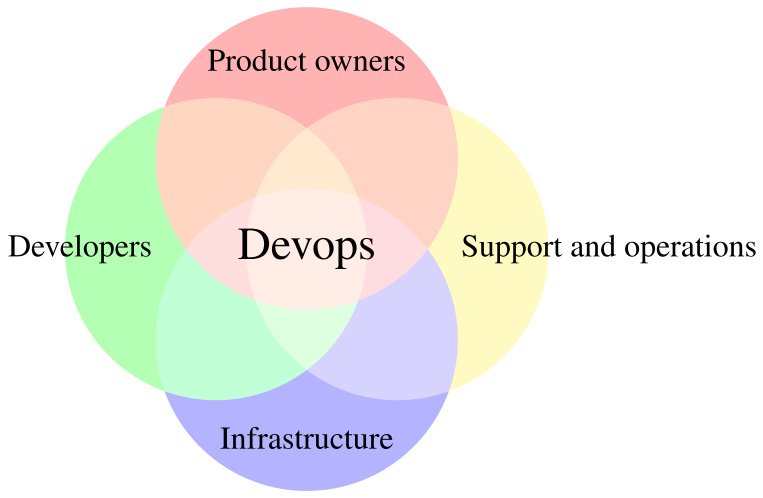


Figure 5. *Organization of Stakeholders.*

2.4. Needs Assessment

In a large software development company, during its life cycle, the application passes through a variety of different divisions within the company. Each division having its own internal workflows, needs for specific information and different outputs of their work. What all in conjunction provide the required output of keeping a modern application infrastructure up to date and running smoothly. Development at high pace and at good quality. Problems found and solved quickly without causing harm or business impact to clients.

Gaps in the information flow between different divisions within the company can lead to slowing down progress in the general work flow. That in turn causes deadlines to come up causing lack of time that causes temporary solutions, that with time become permanent solutions. In worst case scenarios they become the undocumented parts of legacy software that will make updates and migrating difficult. Lack of documentation turn maintenance more complicated and problematic while decreasing the overall observability and fault tolerance.

To avoid such miss communication caused stoppages and to ease the process in distributed teams, the needs of all parties will be described. With the goal to establish common ground and understanding of the process. To establish the requirements for the whole pipeline the architecture setup will need to go through in order to provide a smooth information flow between all parties. Locate the essential bottlenecks that would benefit the most from automation and provide the most functionality to the stakeholders.

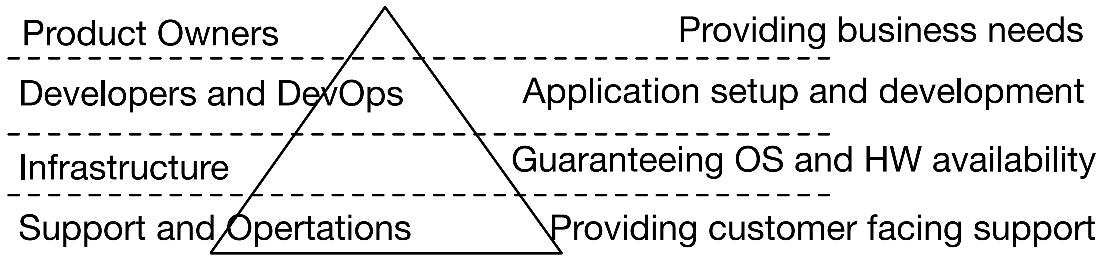

Product Owners
Providing business needs
Developers and DevOps
Application setup and development
Infrastructure
Guaranteeing OS and HW availability
Support and Opertations
Providing customer facing support

Figure 6. *Organizational hierarchy.*

The needs of larger divisions, illustrated on Figure 6 will be introduced in the following sections. The hierarchy goes from top to bottom, where the higher level has more influence over the lower level. Each division has an equally important part and value to the whole end product. The work quality at each level can prove costly to the rest.

2.4.1. Infrastructure

Larger companies can afford the expenses of owning their own data-center or to provision their own hardware in-house. Though having higher initial cost, this approach can pay off in the longer run once the enterprise scale grows or if the information is sensitive or business critical. Use of in-house HW does also introduce the need to monitor, maintain and manage the hardware. Consolidating hardware and distributing resources like HDD space, Memory, computational resources etc. Securing the HW from outside attacks is also the task at this level. Due to security consideration, all root level operations are done in this division following best security practices.

Below are listed the most important tasks performed by infrastructure.

- Provision of Hardware
- Creation of Virtual Machines in accordance to the specification
- Installation of OS
- Maintaining the OS and security updates
- Managing and maintaining root level services
- Governing host access (users and access rights)
- Guarantee uptime of application hosts and surrounding systems

The greatest values for infrastructure would be ease of use of the automation tool. A tool, that would easily allow to bring a large number of hosts into a predefined state. As an example, a group of users, or install certain packages without the need to carry out manual tasks on the provisioned hosts. It should also be capable to provide an overview of the hosts in the system without the need of an external documentation tool. The tool should be as mature and stable as possible, provide support for both current and future technologies.

2.4.2. Developers and Devops

The values provided by the previous tier create the base on what to setup and run the applications. The applications provide the main value proposition of the whole company. To provide some service or an commerce platform to the client base. Developers and DevOps set up the infrastructure on what the full development pipeline, starting from test environments, build pipeline, documentation tools and many other components that are a part of developer's day to day life run.

Developers

Developers are the ones who provide the largest value to the company by fulfilling the business needs of clients. For the developers, a well maintained work flow is of utmost importance. The developed code base would not provide any revenue without the platform to serve the software to the clients for usage. An user friendly CI pipeline that provides a smooth transition from local environment all the way to production shortens the time to release. The possibility to test the code base on an actual environment with all the test suits improves the quality of code and lessens the possibility of problems on production.

The most important tasks performed by developers are listed below.

- Development of the code base
- Creating test suits to guarantee code quality
- Fix not noticed bugs with minimal delay
- Provide new features

DevOps

The definition of “devops” varies from company to company, but the tasks “devops” do tend do be similar in most cases. Devops is the force that keeps the process flowing. Maintaining the test environments and production. Taking care of deployments and application host setups. Maximizing automation and providing simple to use, large scale application deployments for the teams. Setting up tools for the developers and supporting them in case of technical problems across all environments. Devops have usually the largest knowledge of the application infrastructure and distribution of environments.

The most important tasks performed by DevOps are listed below.

- Setup of new verticals
- Modernizing legacy applications
- Automate manual processes
- Simplify the development process
- Application landscape creation and management

2.4.3. Support and Help Desk

Support and help desk provides the customer facing support and issue management. Problem and bug issue are captured and forwarded to the developers. They are also responsible for round-the-clock application maintenance and carrying out simpler tasks to return the application to a responsive state after an incident or an issue. If there is an issue or a problem with either application performance or infrastructure, then support should be the first to find out of it and act immediately or notify the key personnel to take preliminary action.

The most important tasks performed by support are listed below.

- Incident communication and issue management
- Capturing bug reports
- Application monitoring
- Application restarts and recovery
- Monitoring dashboards and react immediately
- Provide useful information of issues to the developers
- Provide useful information to the BU

2.4.4. Product Owners

Product owners are responsible for the value provided to the clients. If some aspect of the product is not working or temporarily out of order, then they are the ones who will have to respond for it. They need to have the latest statistics available to them. Product owners need the usage statistics of users and other market segment vital info to see what to pursue and what feature requests to prioritize. On that information, they decide what aspects of the current product are doing well, and what needs to be improved. In close co operation with the client and developers, they work towards a better understanding of the client needs and how to implement them.

The most important tasks performed by product owners are listed below.

- Give feedback to the clients
- Prioritize issues and bugs, identify blocking issues
- Gather new requirements
- Organize trainings for new features
- Gather statistics and analyze quarterly results

3. Needs Analysis

From the overview in the previous section, the actual needs of the functional units will be analyzed. The needs will be approached from an infrastructural aspect. An overview will be given what each segment needs and expects from the infrastructure for a smooth and agile workflow. The initial expectations and information flow is displayed on Figure 7. The figure shows information flow between functional units. Illustrating how complex and multi-layered the dependencies between different departments are. First the needs will be listed and then analyzed in order to allocate responsibilities to departments.

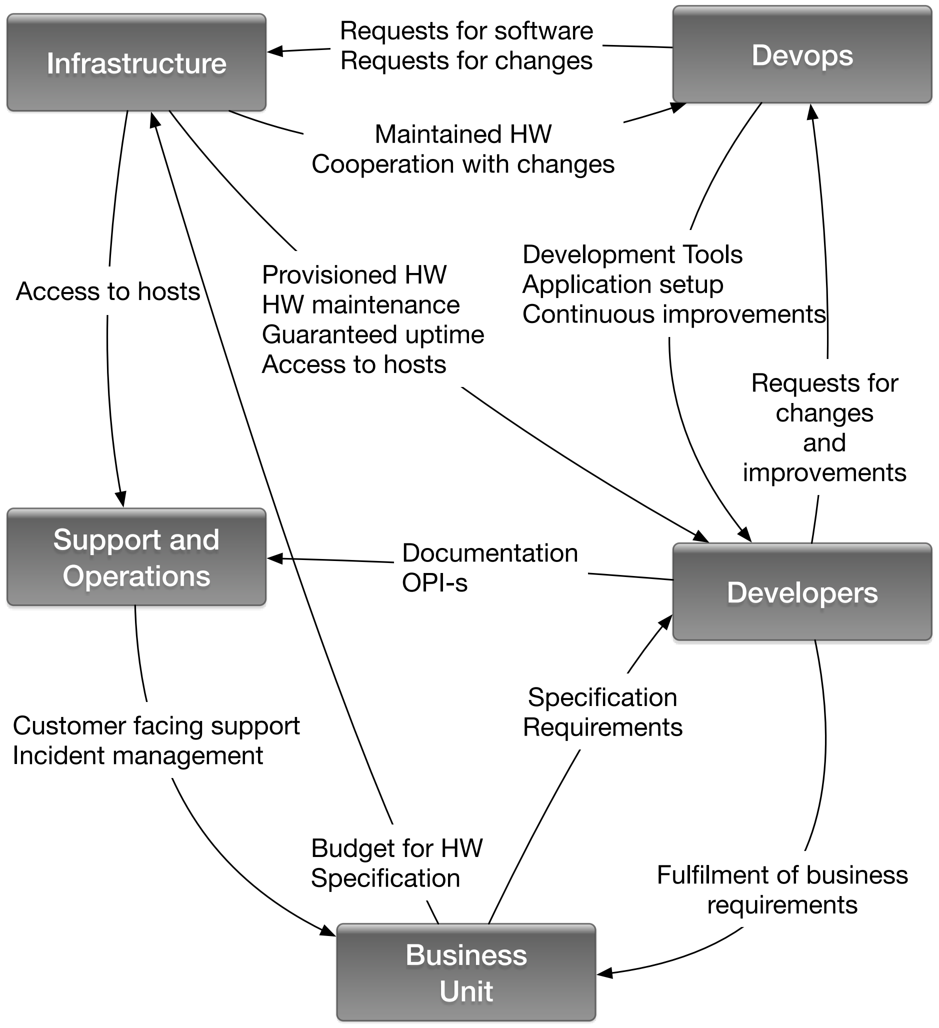


Figure 7. *Main Needs of Divisions.*

3.1. Needs listing

Infrastructure

Infrastructure is the unit whose main responsibility is to provision and guarantee high availability of the hardware. They mainly need a clear description from the team of what they need and how much of it. The provisioned hardware needs to be described and ordered ahead of time. The BU confirms the costs, if not the HW order is changed until all parties are satisfied. Once an agreement of cost and performance is reached, the infrastructure will have the task of actually providing the best possible solution for provisioning, considering the available clusters, the limitations and the needs of the team.

The workflow for HW provisioning is usually fairly well streamlined with modern clusters and enterprise-class hypervisor software. The more complicated part is the clustering, storage and networking of the provisioned HW. Provisioning is done in accordance to the requirements given by architects and developers.

Maintaining the provisioned HW is one of day to day tasks of infrastructure. Generally due to security reasons, giving out root-level access on hosts is not considered good practice in large scale enterprises. Using process control systems and user encapsulation is preferred due to security reasons.

The largest value for this tier would be an automation tool that would enable managing a large number of hosts with ease. The tool should be able to bring a system to a desired state without the need to do any special actions. The tool should also be able to enforce the state and prevent changes that are not agreed upon with the HW maintainers. Thus keeping things in check and under control, with changes being documented in change requests. The actual changes carried out by people who have experience in performing changes and thus guaranteeing quality, maintainability, security and functionality.

Developers and Devops

Developers is the tier that uses the hardware provisioned by the infrastructure. For this tier, it is not important what is done with the HW, as long as it works and the software on it provides a good and stable working environment. The developers themselves ideally develop on their local machines and not interfere with other environments. Thus developers rely on the quality of the continuous integration for their day to day work throughout the application life cycle. Having a good CI pipeline with as much tools as possible is essential. Having access to documentation on the application hosts and related resources is also of high importance as it increases the transparency for the developers and enables them to solve their problems on their own or to provide better information on the issue to the infrastructure.

Devops is the tier that provides most of the tools for CI and development support tools. Setting up testing infrastructure, application monitoring, and artifact delivery to application hosts. Having an automation tool to help this process would help to scale and speed things up, doing anything manually should be avoided. An automation tool with modularity and re-usage would be of great benefit.

Support

From the supports point-of-view, the greatest values are access, documentation and informative checks that provide insight on the actual health and state of the system. The checks and information sources are set up by the developers and devops. The better the information source, the better can support react, provide information on the issues and try to solve the issues. Providing more value to the company.

Business owners

The business owners have the least direct communication with the infrastructural side, but they have the highest impact when something is behind schedule or not working. For the BU, the greatest value is the best quality of work the infrastructure, developers and support can do. The smooth cooperation and workflow of the rest of the teams is the ultimate value a well coordinated process description can provide.

3.2. Analysis of needs listing

The free-form listing of inter-department needs and information flow from the previous sections gives the overview required making it possible to map the main building blocks in the process. The final largest responsibilities, the functional unit responsible for setting up the infrastructure in that segment and the user of the final product is illustrated on Figure 8 and explained in the following section.

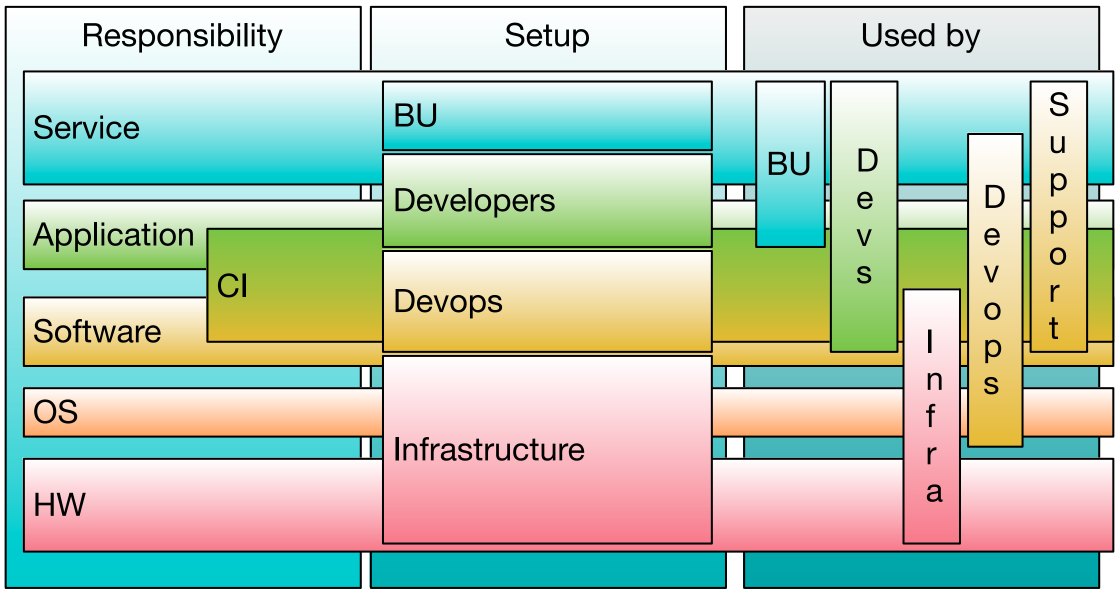


Figure 8. *Mapping of Responsibilities to Functional Units.*

The largest responsibilities and working areas illustrated on the Figure 8 emerged from the description of workflows in the previous sections. From the overlap in the setup section, it is visible, that the entire workflow can be segmented into two large fractions. The HW and OS level management being one of them and application software development and configuration management / deployments the second. The Figure above illustrates six largest layers in application landscape. HW provisioning, OS management, OS software management, CI pipeline, application setup, application runtime and service level. Shows who is responsible for setting up the services and tools related to that layer and who uses the final result. From the responsibility to functional unit mapping the tools that suit the workflow for each division can be evaluated.

4. Automation tools

In the following sections, currently available and more mature tools for automation of tasks will be listed, alongside with proposed areas of usage for the tools.

4.1. Distribution and Fields of Usage for Automation Tools

Based on the needs and the largest task groups described in the previous sections and from the mapping of usage and responsibilities shown on Figure 8 . The following segmentation of tools to operations illustrated on Figure 9 is proposed. The tasks described in the illustration will be further explained on Table 1.

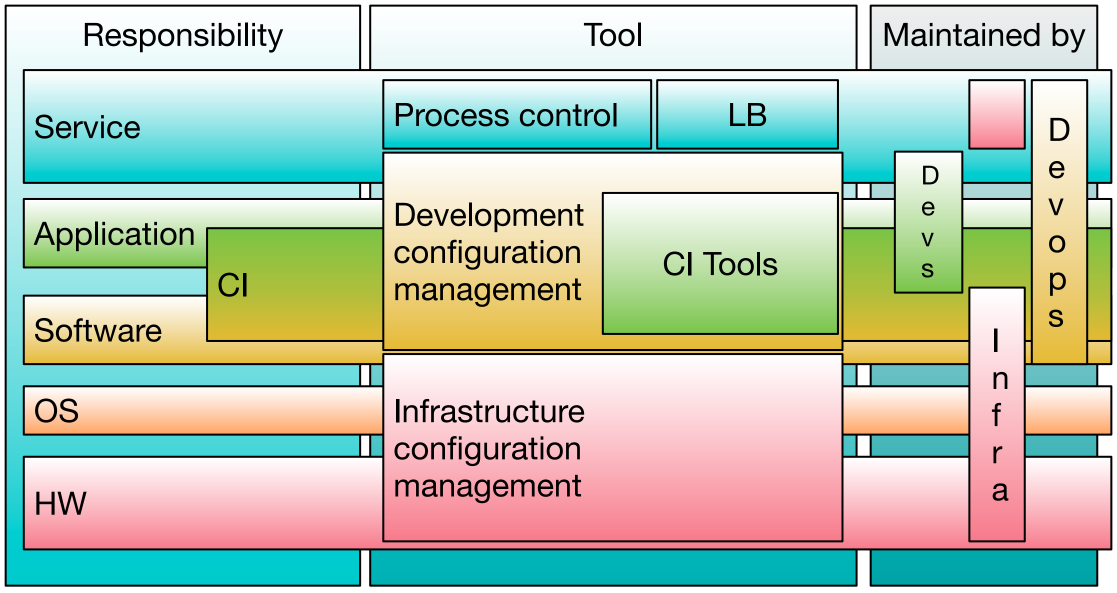


Figure 9. *Mapping of Responsibilities to Proposed Tool Sets.*

The two largest assignments and the best candidates of automation are infrastructure and configuration management. Ideally, this could be done with one tool and separate repositories. Considering the possible contextual differences in managing UNIX-based hosts versus automating application configurations. One tool might be too limiting for one task or too overwhelming for the other task. Keeping in mind the simplicity of use and fast utilization. The possibility of using separate tools is explored. On top of that supporting applications that will end up dictating the day to day life of everyone in the company could be managed by the same tools in similar fashion as production applications. Always using one tool for tasks in a certain responsibility range will provide scalability and ease of use. Especially if supporting tools are managed in a standardized manner, making it easy to handle and manage for everyone who is familiar with the setup methods used.

Description of fields of assignments	
Tool Class	Area of Automation
Process Control Systems	• Management of service runtime
Loadbalancers	• Directing traffic to the production applications and distribute load • Zero downtime deployments
Continuous integration tools	• Build control and automation
Development configuration management	• Application landscape creation and configuration
Infrastructure configuration management	• OS setup and management

Table 1. *Description of Fields of Assignments.*

The Table above gives examples of the general task the tools in Figure 9 have.

4.2. Proposed Tools Listing

The tools proposed are based on the existing solutions and already established work flows. The following tools listed below on Figure 10 will be evaluated to see what combination allows achieving the smoothest infrastructure setup and management architecture.

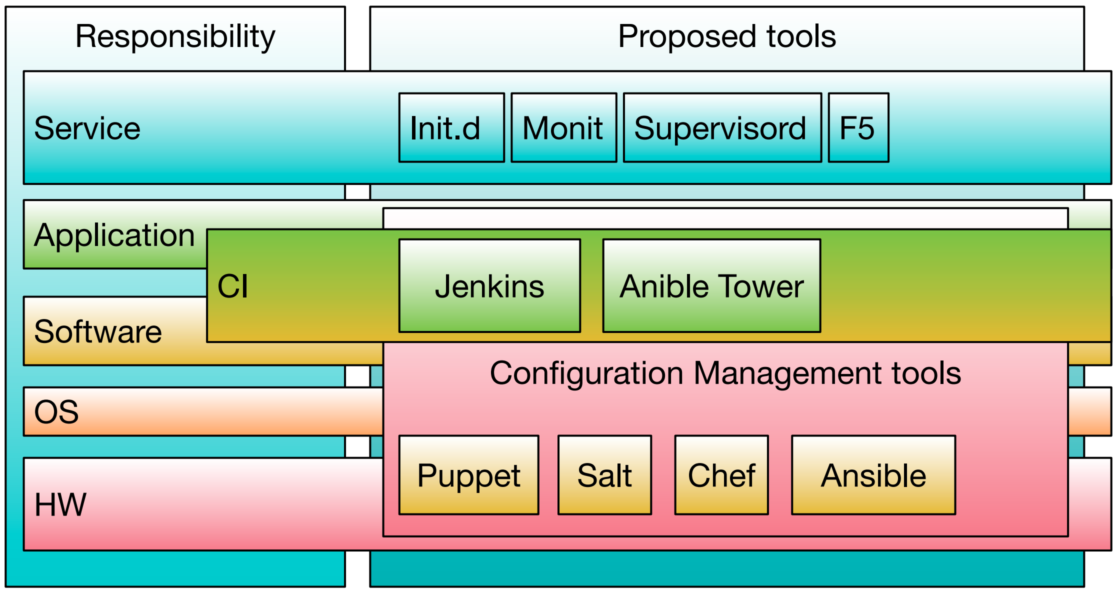


Figure 10. *Proposed Tools for Platform Tasks.*

The tools were chosen for comparison as some of them have already been experimented with within the company. Some tools have set foot in the industry and thus are rooted in the work flow of developers. The selection of more mature and widely supported tools was chosen.

The tools are used for configuration management, continuous integration work flows, application management and load balancing. Where the underlying configuration management tool would be used to set up, maintain, configure and manage all other aspects of the infrastructure.

5. Configuration Management Tools

Configuration management tools are software solutions to automate VM setup and configuration related workflow. There are two types of configuration management tools out on the market at current times. Declarative and imperative [4], both having their own positive and negative sides. The different approaches and thus the different behaviors stem from the coding language the tool is written in. The approach of the more popular automation tools is given in Table 2.

Distribution of Configuration management tools	
Declarative	Imperative
Puppet	Ansible
Salt	Chef

Table 2. *Automation Tool Distribution by Declarative or Imperative Approach.*

The difference in approach the tool uses leads to differences in coding-style and also on the determinism of the execution. In declarative approaches, the required final state is described and reaching the state is left to the underlying framework to sort out. Meanwhile, with imperative approach, the required state is reached by executing a list of commands one after another in a certain order to end up with the required outcome.

In terms of automation tool behavior it means that declarative approach can lead to some disambiguation in the execution of tasks to reach the required outcome. Meanwhile for imperative approach the sequence of execution is much more deterministic, but leaves more room for corner cases and requires defensive coding with error handling in more complex cases to guarantee success.

Depending on the underlying programming language used by the tool and overall maturity of the tool. The one large limiting factor on the selection is the semantical and syntactical express-fullness of the DSL or coding language used in the tool. For both DevOps and developers it is essential to have as low learning curve as possible. Enabling people to start using the tool with minimal delays and that the created automation would be easy to comprehend and modify. Without the need to go over long documentation and reference manuals. At the same time be flexible enough to orchestrate complex sequences of actions.

5.1. Comparison of Configuration Management Tools

For automation of infrastructure setup and configuration management, the tools listed in Table 2 will be investigated closer. A small overview of each tool will be given as well as a comparison of main features.

5.1.1. Puppet Labs Puppet

Puppet has been around the longest and thus is the most mature with the best modules and support for other tools. A basic puppet setup consists of a puppetmaster install and client agents on each orchestrated node. The clients pull periodically from the master the latest definition of the state the node needs to be in by interpreting the state from the instructions in its data stores called PuppetDB and Hiera. This data store keeps track of all the changes and facts that need to be done for the node to be in the corresponding state. The puppet agents can also work in push mode, in order not to wait for the pull to occur in case of more urgent cases.

Getting up and running with puppet can take a little longer as the syntax and basic logic needs to be understood before making any real changes. And as Puppet has an object oriented direction in mind with great emphasis on re-usage. The trail of inclusions can in time get very long and confusing to new comers.This ultimately defeating the point of easy comprehension. The other point of concern is the modules. As it is with open source a lot of different implementations can do the same thing in different ways with a slight variation. This can lead to incompatibilities between the different modules and cause non deterministic results.

5.1.2. SaltStack

Salt has been around for five years and in the last years has picked up its development process being one of the largest opensource projects on GitHub for a while. Written in Python and having a simple structure composing of masters, minions, state files, grains, pillars and modules. States, grains and pillars are basically YAML structured data, that describes the end state of the system. More dynamic things such as parametrized templates can be described via jinja markup. Or by implementing a custom module in python.

The documentation, if it is available is detailed, but troublesome to work through. Due to its rapid development, some of the features are not even properly documented at all. The initial getting started documentation and examples are good to get going, making it easy to buy into the features. Only later to discover that the complexity of orchestration in multi-tier wait and retry scenarios can turn time consuming to develop.

5.1.3. Chef

Chef is a tool with a lot of large scale clients using it. Written in Ruby and Erlang, makes it very flexible and capable. The drawback neing that the developers need to know ruby. For use with ruby on rails applications deployments it is thus very convenient and applicable. For others, due to the language base, can be considered as too much overhead. The second issue with Chef is the large segmentation of different state descriptions into different files. This makes the configuration very multi level and not transparent at the beginning.

The documentation is very structured and detailed [5]. Explaining all the basic concept from the beginning. Good documentation makes things easier to get going if the user already is familiar with Ruby DSL. Chef uses a puppet like pull feature and master to chef client hierarchy. But does not make use of the imperative approach as puppet does, making it more transparent when it comes to ordering of executions.

5.1.4. Red Hat Ansible

Ansible is the youngest of the group, lately purchased by Redhat. Utilizing a masterless push-based configuration runs, makes it one of the most light-weight, easy to manage and set up configuration management tool of the selection. The agentless design enables for every developer to have its own sandbox, or to make changes from his own local machine. Ansible has a simple, yet flexible enough structure to achieve complex behavior and orchestration. Written in Python and using a very custom DSL for orchestration files, makes it sometimes frustrating to use as the developer have to rely on examples to see how the syntax exactly goes. However, the documentation is usually adequate and has plenty of examples included. More often than not, the documentation and examples could be more detailed and have more explanations.

The learning curve of Ansible is linear and people can understand and make additions to existing play books without much effort. Moving on from there is relatively easy as there are a lot of examples and even more different modules to use and take example from. Writing custom modules is easy and is not limited to Python only. Ansible modules support basically anything that can be executed on the host and can return a YSON formed return with additional values on the call. Alternatively Ansible can simply make use of the OS level return values to decide the success or fail on its own.

5.2. Selection of Main Tools

Considering the most basic features of the configuration management tools, taking into consideration their main strengths, weaknesses and characteristics. Considering the underlaying hardware and the usage of it. Keeping in mind the existing segmentation and distribution of responsibility within the company. The following tools were chosen.

Team that manages the HW and the developers are segregated with different levels of access for each department. This sets separate limitations to the tools that can be chosen. Setting different general use cases for the tools. Since the underlaying HW is an ESX cluster based VM-s and their number will keep rising within the cluster. Then the team that will manages the hosts will have a dedicated task to keep the machines in pristine order, up to date and secure. As a dedicated task the learning curve of the tool is not that important, but the longevity of the setup is. This makes Puppet the best option being the most mature offering.

For the development teams, using Puppet would be a little too overwhelming and time consuming. Ansible as the most light weight and agile option is the best fit for application configuration related automation activities. Providing enough tools and opportunities for automation, yet being easy and logical enough for developers to manage and improve once the DevOps is done with initial setup. So the final selection of CMT is illustrated on 11.

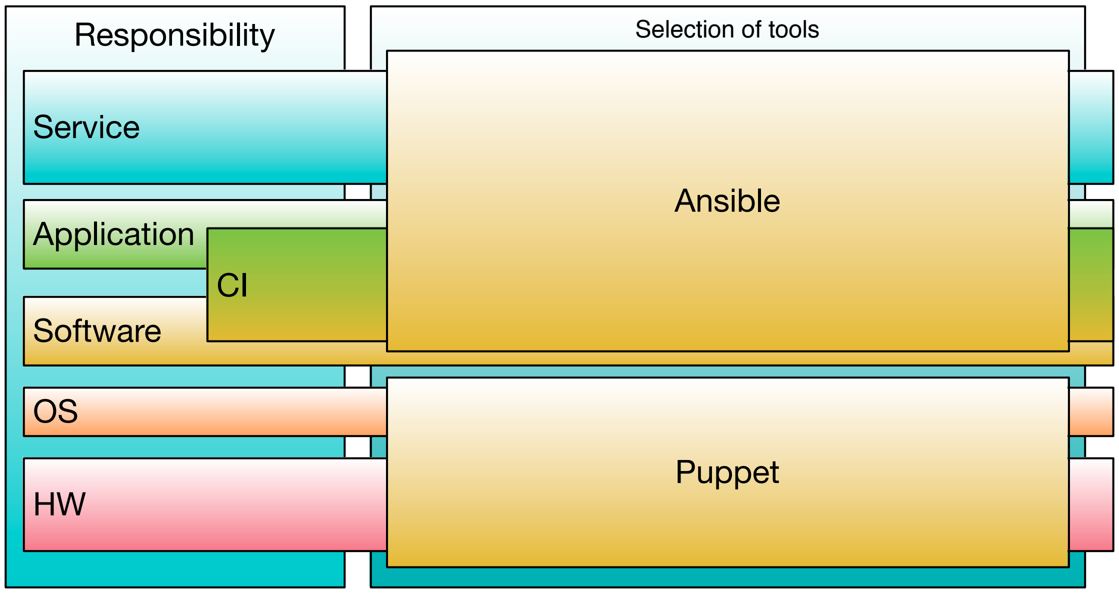


Figure 11. *Selection of Tools for Main Tasks.*

5.3. Puppet Overview

Puppet, with its very mature infrastructure as code approach and pull based master-slave setup, is more suitable for more classical VM based clusters. The puppet agents periodically keep pulling the latest manifests and guarantee, that the desired state of infrastructure is achieved. Constantly running agents make enforcing policies and ground rules easy because puppet will keep enforcing it at root level, making manual changes temporary, forcing the requirement to perform changes via puppet.

Setting up the basics for the hosts would be executed via puppet classes. Each host has its own classes included to the host, depending on the requested software on it and requirements set to the host. Shared mounts and other root-level host specifics like user management and enforced application administrator and application runtime user encapsulation. Other policies can include creation of dedicated partitions or directories with corresponding user rights. THis sets up the infrastructure for an upcoming Ansible run even on a basically stock host. Eliminating the need to worry about the lack of root rights on the host for Ansible users. The enforced user hierarchy and user encapsulation is illustrated on Figure 12.

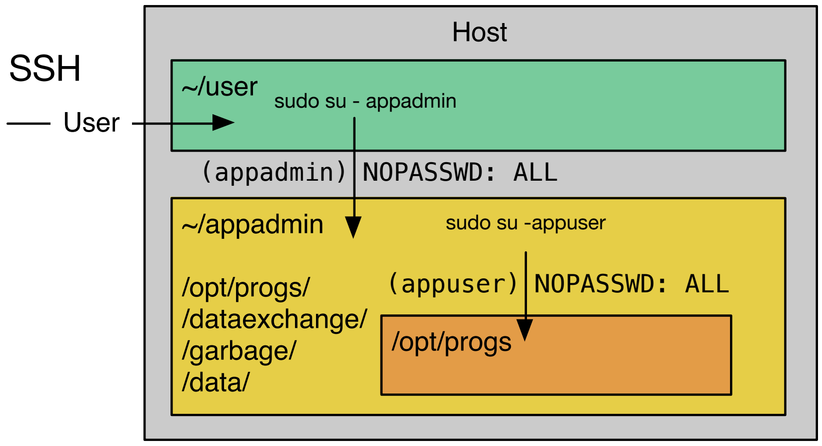


Figure 12. *User and Process Runtime Encapsulation.*

As puppet is strictly used for host preparation, maintenance and OS level software provisioning. Details of puppet will not be further explored as the Puppeteered infrastructure can be taken as a black-box that takes in requirements such as a software list, users and other OS level requests. Puppet management needs to be done ahead of application setup time and can be considered a standardized building block in the application roll out life cycle.

5.4. Ansible Introduction

Ansible is the tool responsible for the majority of application related activities. Can be considered as the main building block for infrastructure on top of the puppet managed layer. Used by both DevOps and the developers, if they choose to take the responsibility. Based on experience, the larger portion of developers, if possible would prefer to leave the infrastructure for someone else to take care of. The more active developers like to improve on top of existing solutions if they can, but refuse to take responsibility. Very few would take the responsibility of a full development and deployment pipeline.

5.4.1. Ansible Overview

In the architecture, Ansible is enclosed into a non-elevated user rights range. Meaning no root level commands are executed due to enforcing company security policy. This can be viewed as limiting factor. In reality it needs tight cooperation with the other division and this in essence leads to cross-team code reviews. As neither side would allow any half-ways working solutions or clutter in their shared responsibility range.

Ansible with its agentless design only needs to have SSH keys for the user in place, this is done via Puppet. Other prerequisite is user switching rules for enforced user encapsulation. Without the rules Ansible run can hang without timeout in some cases, as it waits for user switching password. The user switching problem is taken care of by a Puppet base user class enforced user encapsulation.

Ansible is easy to start with and get going as the majority of the information can be kept in one file that describes the general layout of the application in the infrastructure. This file is an Inventory file that holds the grouping of hosts. The inventory file can also be used for storing host specific variables and parameters. Keeping in mind the not mentioned fact that Ansible variables have global visibility scope if they are declared in inventory files. This can prove problematic when one role is ran twice on the same host with different parameters, causing only one set of the parameters apply to both runs.

An inventory file is accompanied by a site file that holds the list of roles that need to be applied to the host. The roles contain tasks. Each task is basically an execution of some

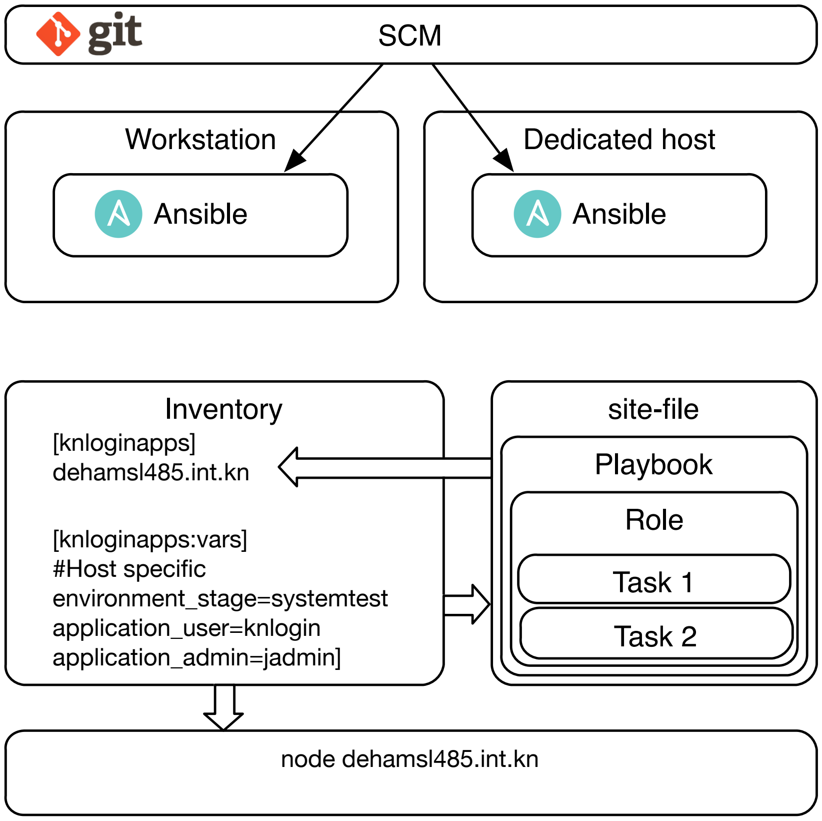


Figure 13. *Basic Terminology of Ansible.*

command. The tasks can be iterated and executed conditionally. With ansiblt 2.0 the entire structure of a role can now be even more complex and easy to create with the introduction of blocks, that also enable try catch type of approach to deal with situations.

The outcome of each task can be captured via a register call, what enables access to a lot of information returned by the task. The information is in json format and each part can be addressed individually to use as conditionals or parameters for other tasks. Accompanied with other constructs like polling and waiting until conditions are reached, gives ansible roles a great deal of flexibility and control over the orchestration carried out on the node.

The most basic terminology of ansible is illustrated on Figure 13. This illustration does not include the Ansible vault for encrypting sensitive information and custom modules are not yet introduced, the basic terminology on the figure was explained in the section above.

One possible way to structure Ansible files for a project would be as illustrated on Figure 14. The placement of inventory files can be altered and inventories moved to subdirectories, grouped by environments to enforce structuring. This structure aims to keep the setup as flat as possible avoiding separate files. The Figure also gives a short overview of the role of each file in the Ansible structure.

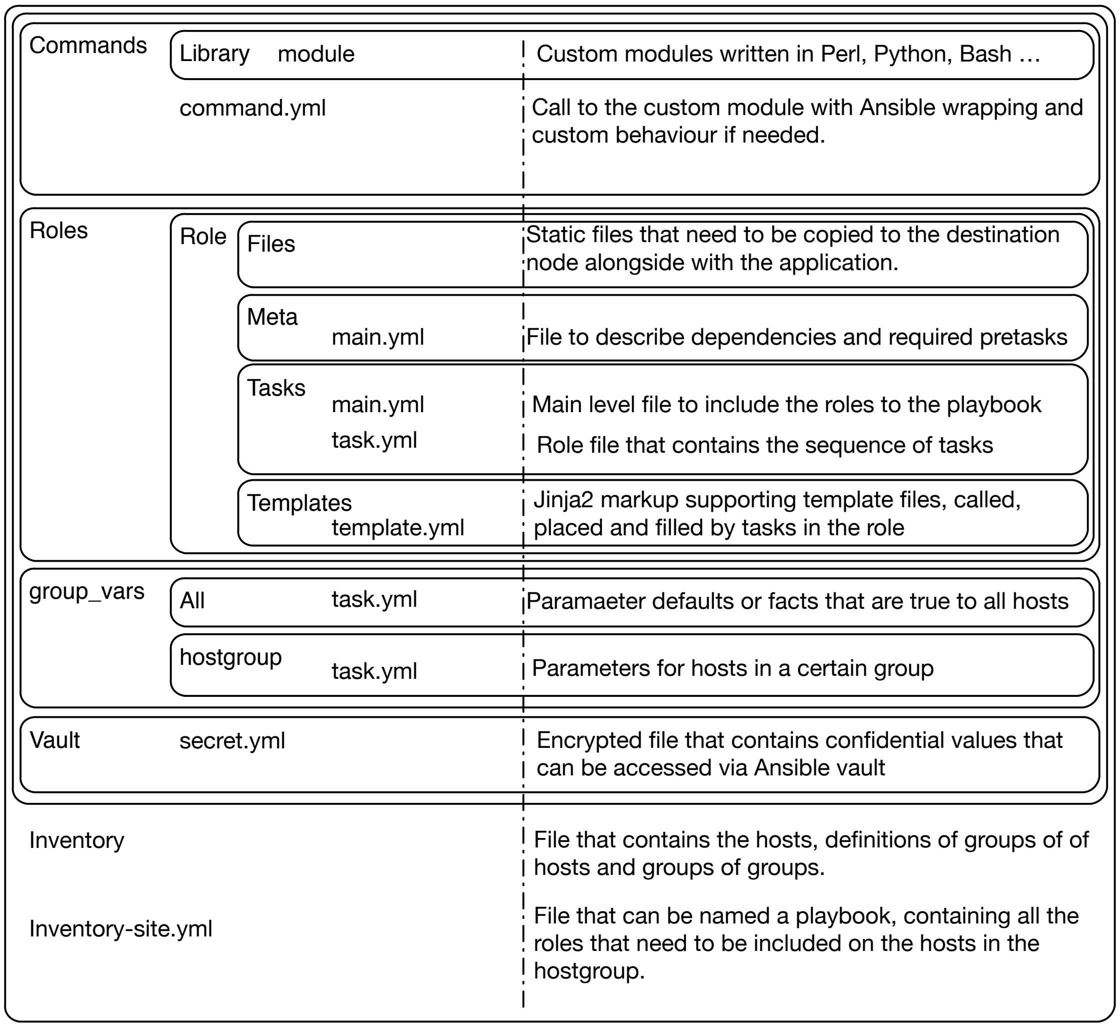

Commands
Library module
Custom modules written in Perl, Python, Bash ...
command.yml
Call to the custom module with Ansible wrapping and custom behaviour if needed.
Roles
Role
Files
Static files that need to be copied to the destination node alongside with the application.
Meta
main.yml
File to describe dependencies and required pretasks
Tasks
main.yml
Main level file to include the roles to the playbook
task.yml
Role file that contains the sequence of tasks
Templates
template.yml
Jinja2 markup supporting template files, called, placed and filled by tasks in the role
group_vars
All
task.yml
Paramaeter defaults or facts that are true to all hosts
hostgroup
task.yml
Parameters for hosts in a certain group
Vault
secret.yml
Encrypted file that contains confidential values that can be accessed via Ansible vault
Inventory
File that contains the hosts, definitions of groups of of hosts and groups of groups.
Inventory-site.yml
File that can be named a playbook, containing all the roles that need to be included on the hosts in the hostgroup.

Figure 14. *Structural Hierarchy of Ansible Files.*

A more complete example will be given in the Infrastructure Scaffolding section where the role of Anisble in the application landscape creation and deployment pipeline is introduced. That section will explain how Ansible acts as an automation glue to bring everything together and provide overall value to the personnel and the company, along with a more complete example.

6. Loadbalancers

Loadbalancers play a major role in the application infrastructure[6] by providing the required prerequisites for high availability. This is achieved by distributing the load between the nodes. In case of a node failure, LB makes the failure transparent by automatically removing the node from the pool and re-distributes the load between active nodes. Loss of session for some clients can occur in case if the sessions are not serialized due to size or other implementation related limitations. Or if the sessions are not cached in the loadbalancer to recover the session on other nodes.

With the onset of more agile development and rapid expansions, combination of using both hardware and software loadbalancers to give more flexibility to the teams has gained popularity. Both approaches are illustrated on Figure 15 where vertical 1 shows the combination of HW LB and software LB. Meanwhile vertical 2 is a pure HW loadbalancing solution.

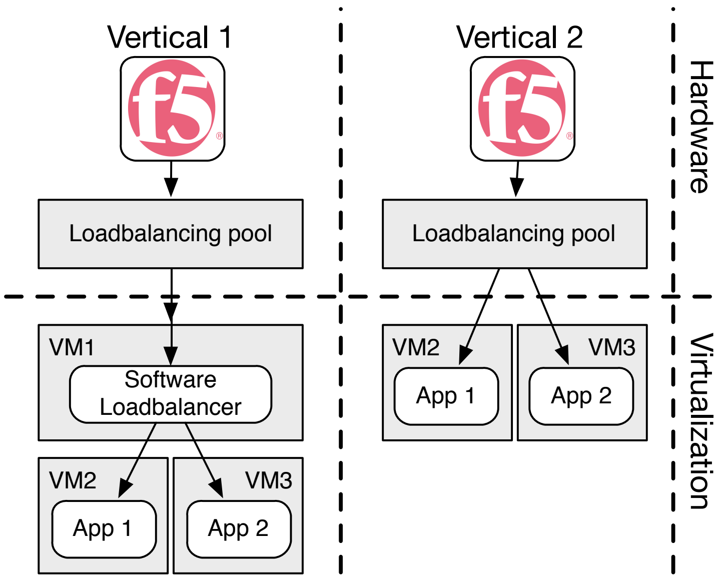


Figure 15. *Use Cases of Loadbalancers.*

Using software LB is a valid approach for infrastructure as service solutions, but for more classical VM based solutions. The usage of software loadbalancers is a concern as it introduces a new level of complexity and a single point of failure to the infrastructure. The benefit of it on the other hand is that the team has more control of their applications and deployment related processes. The draw back is that in case of problems the recovery systems need to be in place and configured ahead of time.

The way to utilize the loadbalancers in production, either a combination of HW and SW or pure HW solutions, does not define the way of doing deployments. Both solutions leave enough room for play and provide enough capabilities to do deployments in any way chosen. The latest popular deployment model is “green blue pool” deployments. The procedure is depicted on Figure 16. The pool is divided into two groups and only one of them is active. Update is carried out on the disabled group and after that the LB redirects traffic to the updated pool. This approach allows for very flexible deployment procedures. Also enables very rapid rollbacks in case of problems arise. The drawback of implementing this model on dedicated VM based infrastructure is that one of the pools is not utilized all the time and consumes resources.

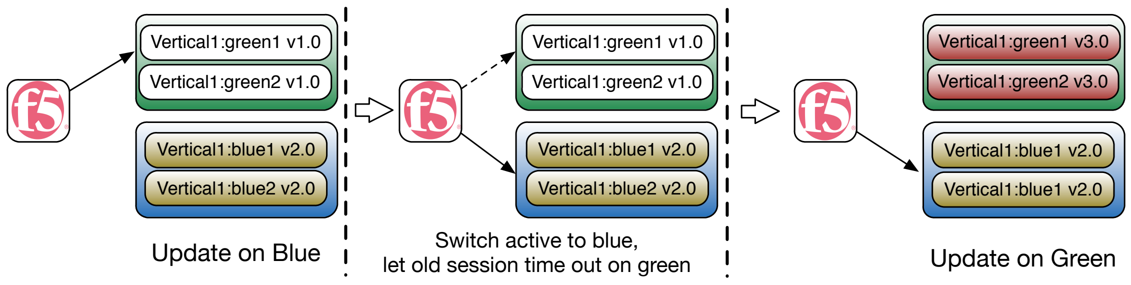


Figure 16. *Green Blue Zero Downtime Deployment Pipeline.*

A more suitable deployment model for dedicated VM based infrastructure is a cycle based rollout procedure. This approach is show on Figure 17. This involves disabling and enabling of nodes or pool members [7] and updating the applications in a predetermined order. This procedure can take longer due to being forced to wait twice for sessions to time out. The benefit of this is that all dedicated HW for the vertical is always available to the client.

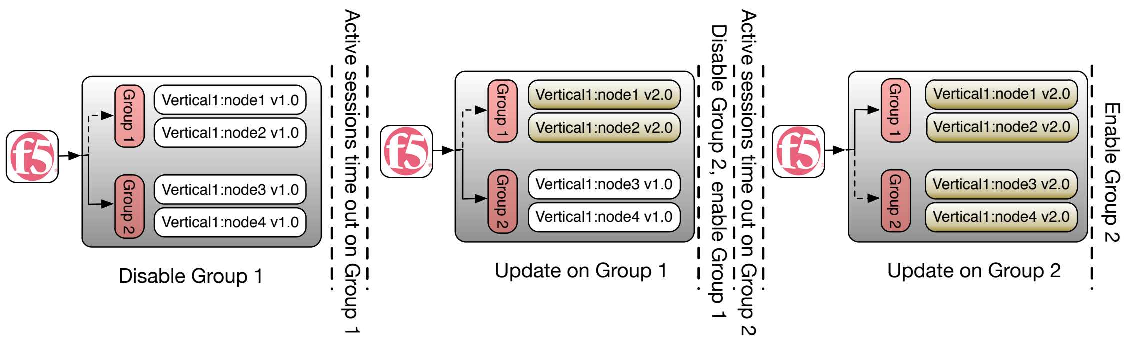


Figure 17. *Cyclic Zero Downtime Deployment Pipeline.*

6.1. F5 Networks Big IP

The cornerstone of any modern IT infrastructure is the connection to the outer world. This is in most cases some sort of a loadbalancer. One of the most widely spread provider of HW loadbalancers is F5 Networks, that up to this date has reached the “Cisco” level in the application delivery controllers solutions. Meaning it is an authority in the industry. Large client base, good documentation, support and solid performance, makes it a good choice. Most larger enterprises already having a box or two in their server park. Updating into familiar territory on business critical hardware without making any drastic changes in the infrastructure does make sense.

Other benefits Big IP products provide include dedicated HW for computation heavy tasks like SSL acceleration and encryption/decryption to provide improved performance and longevity. Dedicated HW makes the F5 provided package a very good all round solution what should make a solid statement to advise against software loadbalancing within the internal network. Using only HW LB makes the infrastructure more transparent, perform better and have less points of potential failure. Additionally providing enhanced security by the ASM[8] module that takes care of attack vectors before they ever reach the application.

The other benefit of F5 networks products, due to its large client base, it also has a lot of support by third party tools. One of them being the BIG-IP Ansible modules [9]. These modules enable delegating Application pool Node management commands via ansible roles to the F5 BIG-IP LTM pools via iControl SOAP API. Giving Ansible all the required control and information to perform all the required actions to switch nodes in a pool to disabled state and to get the current session count of desired node, making Ansible a fully autonomous fire and forget tool for deployments.

The only blocking factor here being that the features are made available via a Python library that is currently not installed by default. Since the loadbalancer is a very important and mission critical piece of hardware, changes in loadbalancers need to be thoroughly tested and verified long in advance. This task that is not done in Kuehne+Nagel F5 setup, thus making the final step of fully autonomous, zero downtime deployments at current time unavailable.

7. Process Control Systems

Process Control Systems in OS level process management context stands for means of managing a program runtime in a more convenient way than in a basic way [10] of manually executing a startup command. Most tools used to support development come with included means of executing the pre-compiled binaries. The developed software need in most cases a tomcat instance to to be started to serve the content, or in the case of integrated packages such as bootspring applications, only need the java executable and a few configuration flags to run [11].

Manually executing startup commands is out of the question in an automated environment. That is why software comes with startup scripts included and developed applications preferably have a standardized configuration and startup executable. The executables are not much use without means of conveniently managing the runtimes of the applications. There are multiple ways to interface these startup scripts with OS level services and make the applications start up automatically with the server and enable management.

The following sections will introduce a selection of basic process control systems.

7.1. Init.d

One of the oldest and basic ways of accomplishing application process management is by using the same method Linux starts (and stops) all its subsystems via init.d[12]. This modular and organized startup of subsystems is a part of the OS and thus is readily available to use on any host out of box. The drawback of it being an integrated part of the OS booting sequence is that root level access is required to make the application into a subsystem. This does guarantee the startup of the application alongside with the host and makes service calls available to the subsystem, making it easy to manage the runtime. Since init.d subsystem is basically a standardized shell script, this makes it a flexible tool for those who are competent with Bash scripting. Being a part of the OS for a long time and the specific use of the init.d does make it a very limited tool in the sense of additional features such as remote execution or integration into other tools. The other major drawback is that init.d is only a tool to guarantee the starting of processes alongside with the OS. The subsystem provides no process monitoring to make sure the process remains up and running.

7.2. Supervisord

A more modern Process Control System is Supervisord [13]. Initially used as a process watcher for monitoring long-running scripts and processes that are integrated as services and can fail silently. Supervisord provides fairly good flexibility and above all, easy configuration and usage. Once Supervisord has been installed on the host, it is very easy to configure it, even without root level access. It can be restarted and used by the owner of the socket file making it convenient to use for both automation tools and support personnel.

Configuration of supervisor is straight forward, consisting of two files. One file for supervisor specific and other for supervised application related parameters. The most outstanding lines of supervisord configuration is brought on Figure 18. The most important being on line 3 where the socket ownership is determined, allowing for non root operation of supervised applications, but not the supervisord process itself.

```
1  [unix_http_server]
2  file=/var/run/supervisor.sock   ; (the path to the socket file)
3  chmod=0700                      ; socket file mode (default 0700)
4  chown=jadmin:jadmin             ; socket file uid:gid owner
5
6  [include]
7  files = /opt/knlogin/supervisord.d/*.ini
```

Figure 18. *Supervisord.conf Snippet.*

The supervised application configuration is taken from .ini files from the directory configured in supervisord.conf file, shown on line 7 in the Figure 18. In the ini file, either the startup command or startup script is given. In case of a script, "stopasgroup" needs to be set in order to pass the stop call to the spawned process. There is no built in way for custom behavior on stopping of the application. This is a limiting factor that can be circumvented with developing a custom module.

```
[program:elasticsearch]
command=/opt/progs/elasticsearch/bin/startup.sh
directory=/opt/progs/elasticsearch/
user=knlogin
stdout_logfile=/data/log/elasticsearch/app.log
stderr_logfile=/data/log/elasticsearch/app-stderr.log
autostart=true
autorestart=true
stopasgroup=true
startsecs=5
```

Figure 19. *Supervisord Managed Application Configuration Example.*

7.3. Monit

A similar solution is provided by Monit [14]. Monit is a compact solution that utilizes already existing tools such as the init subsystem, upstart and systemd. By using existing OS level blocks, Monit manages to provide as much functionality as possible out of box and without the need for external plugins or extra software. Monit can also utilize run-level scripts to manage services that enables customized actions on both shutdown and startup of applications. The features make Monit very light weight, yet very flexible tool. The custom behavior on top of process management introduces additional options not only for reactive actions, but also proactive tools like metrics monitoring and notifications. In addition to that Monit provides basic functionality for automatic recovery and smoke-testing.

IN order to spawn processes as other user, the daemon needs to be started at root level. The calls to application management then need to be executed at root level as well. There is no built in capability to enable management by other users. This can be circumvented by adding custom sudo rules to enable calls to that service for support personnel.

Application related configuration is more complicated compared to supervisord, as illustrated on Figure 20. The additional complexity on the other hand enables more control over the application execution and behavior. Introducing such benefits as custom actions on shut down. As shown on lines 2-4 on Figure 20. Configuration also enables http calls and testing for certain content with options to alert or try additional actions. Besides application related monitoring, host metrics can be monitored to discover basic problems with storage and available resources before they cause an impact.

```
1  check process elasticsearch with pidfile /opt/progs/elasticsearch2/process.pid
2      group elastic
3      start program = "/opt/progs/elasticsearch2/bin/startup.sh" as "knlogin"
4      stop program = "/opt/progs/elasticsearch2/bin/shutdown.sh" as "knlogin"
5
6  check host elasticsearch_host with address laodike.int.kn
7          if failed url http://laodike.int.kn:9200 with timeout 60 seconds then alert
8          group elastic
9
10 check host elasticsearch_cluster_health with address laodike.int.kn
11         if failed url http://laodike.int.kn:9200/_cluster/health
12                 and content == 'green'
13                 with timeout 60 seconds
14         then alert
15         group elastic
16 check filesystem rootfs with path /opt
```

Figure 20. *Monit Managed Application Configuration Example.*

7.4. Init.d, Monit and Supervisord Comparision

The main differences of the three PCS are brought out on Table 3

Comparison of Init.d, Supervisord and Monit

	Init.d	Supervisord	Monit
Root level process	Yes	Yes	Yes
Non-root level management	Yes	Yes	Yes*
Non-root level configuration	No	Yes**	Yes**
HTTP interface	No	Yes	Yes
LDAP integration	No	No	No
Custom startup scripts	Yes	Yes	Yes
Custom shutdown scripts	Yes	No	Yes
Application smoke testing	No	No	Yes
Automatic restart	No	Yes	Yes
Restart retries back off	No	Yes	Yes
Free of charge in corporate usage	Yes	Yes	No
Support for additional plugins	No	Yes***	No

*needs custom sudoers rule

**needs custom configuration

***can be developed into supervisord

Table 3. *Comparison of Process Control Systems.*

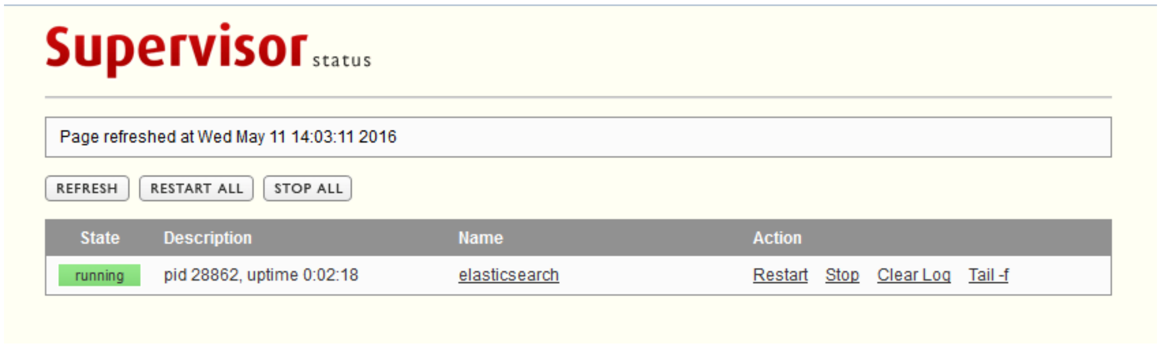
From the comparison of the three, supervisord comes out as the one best fit for the task of a basic and simple process control system. Allowing non root level configuration and management. Easy to use commandline call and very easy to template and update configuration. This makes supervisord the most suitable tool for both automation and support personnel use. All more complex cases or corner cases can be addressed either via developing more functionality directly into supervisord or can be bypassed by using Ansible in a more intrusive manner.

7.4.1. HTTP Interfaces of Process Control Systems

The two more modern process control systems, Monit and Supervisord also have a user friendly front end. This would enable developers and QA to conveniently manage the application without the need to log in to the hosts and not deal with terminal tools if they so wish. The only drawback with both is that though both have password protection enabled in the configuration. Then they both use static passwords and neither has native LDAP plugins or support.

This means that if to wish to integrate the functionality into the existing infrastructure, then this would need to be done via third party tools. The easiest option is to use Apache loopback with corresponding LDAP plugins on apache side. Using this solution would enable precise control over the access and convenience to the users.

Figures 21 and 22 illustrate the web front ends of both applications on an example of an Elasticsearch instance supervision. Both interfaces provide basic feedback of the state of supervised process. Both provide basic measures to manage the process as well. With Monit's additional functionality, the interface is also a little more informative with giving an overview of the host metrics and the custom checks configured. Supervisord web frontend on the other hand enables quick access to the stdout of the supervised application, what can be a quick information source.



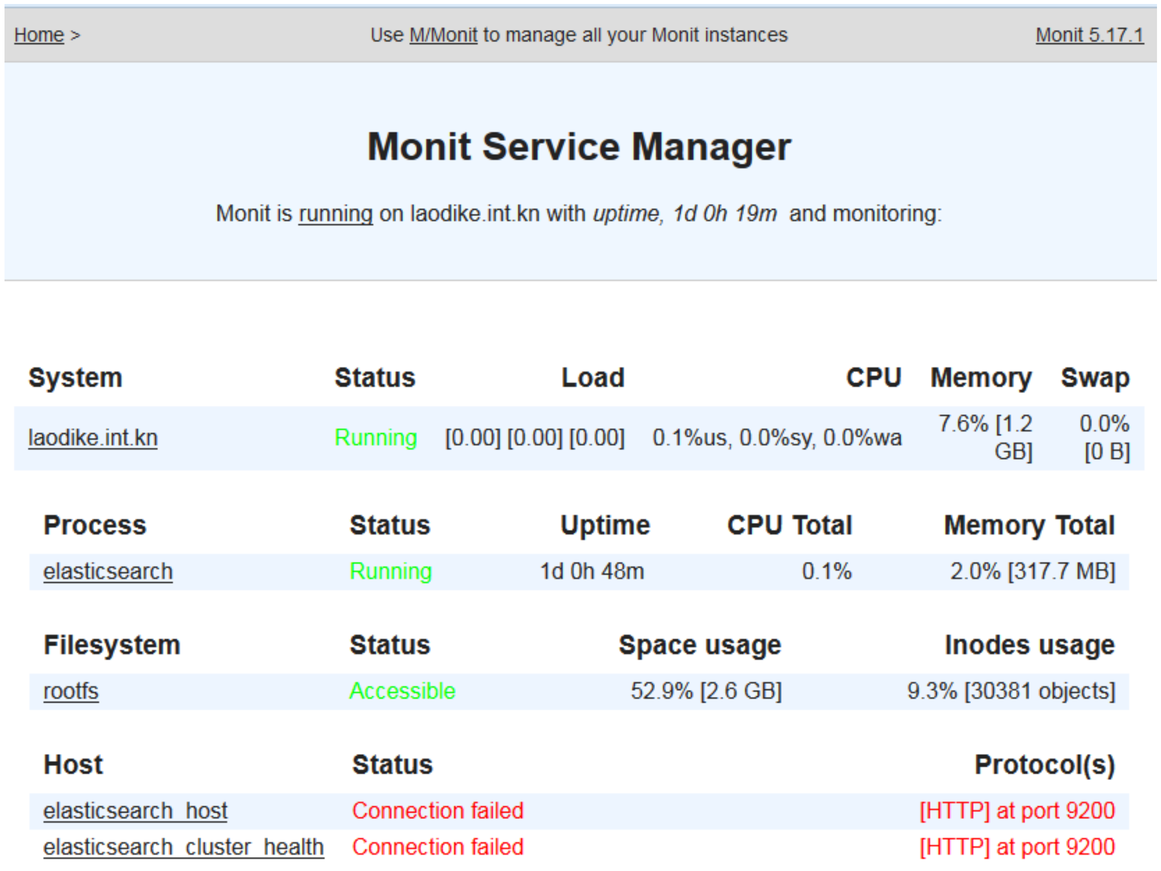
Page refreshed at Wed May 11 14:03:11 2016

State	Description	Name	Action
running	pid 28862, uptime 0:02:18	elasticsearch	Restart Stop Clear Log Tail -f

Figure 21. *Supervisord HTTP Interface.*

HTTP interfaces, thou being a convenient feature, would get too confusing and troublesome to manage with large numbers of hosts and applications. Each web interface of each host would need to be separately addressed to get access to the management layer. All the interfaces would need separate listing and documentation. This would introduce a new level of complexity that would limit scalability. The web interface can also be considered as an attach vector, thus not thinkable of applying in production. Enabling this feature could be considered only on lower level environments that are not business critical.

Considering the facts, it is more beneficial and easier for longevity to stick to the larger scale tools for visual feedback and use the PCS tools for only the basic tasks and keep all the monitoring in dedicated corporate tools.



Monit Service Manager

Monit is running on laodike.int.kn with *uptime, 1d 0h 19m* and monitoring:

System	Status	Load	CPU	Memory	Swap
laodike.int.kn	Running	[0.00] [0.00] [0.00]	0.1%us, 0.0%sy, 0.0%wa	7.6% [1.2 GB]	0.0% [0 B]

Process	Status	Uptime	CPU Total	Memory Total
elasticsearch	Running	1d 0h 48m	0.1%	2.0% [317.7 MB]

Filesystem	Status	Space usage	Inodes usage
rootfs	Accessible	52.9% [2.6 GB]	9.3% [30381 objects]

Host	Status	Protocol(s)
elasticsearch_host	Connection failed	[HTTP] at port 9200
elasticsearch_cluster_health	Connection failed	[HTTP] at port 9200

Figure 22. *Monit HTTP Interface.*

8. Automation Tool Integration

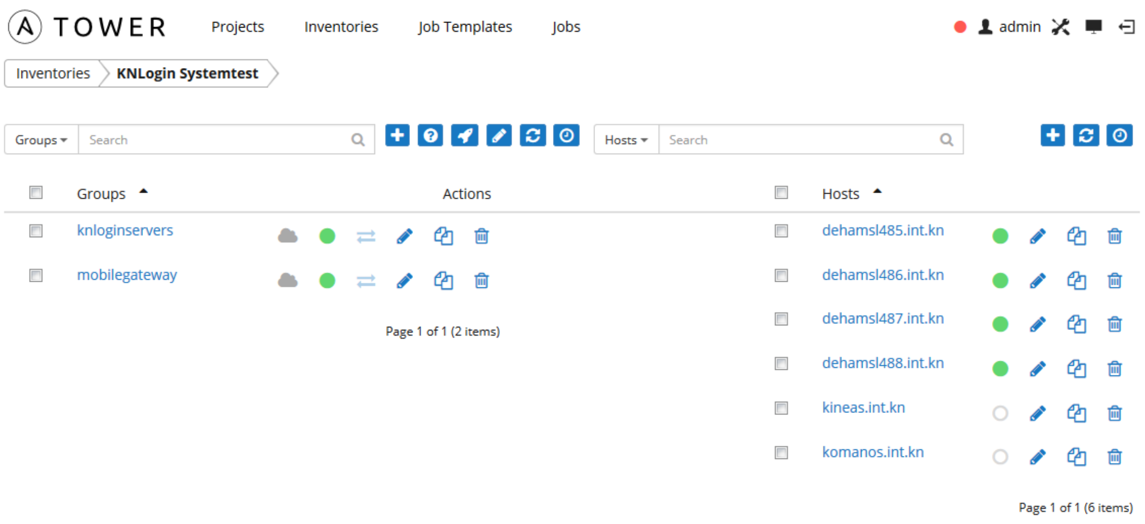
The goal of automation is to help smooth out the workflow and decrease the amount of time and effort put into tedious tasks. Freeing up time to focus on more important tasks. Full automation has two sides to it. If left without any supervision or ways of managing, can lead to problems. Having visual feedback and convenient control over the automation is almost as important as the automation itself. Having precise control over the runs, clear overview the results and have instant access to the debugging is vital to increasing the transparency. Debugging is vital to provide quick solutions when a corner case or an unforeseen exception is reached.

As puppet is used for lower levels that are developed and tested to run and reach a certain state. Developed and used by a separate team, then the tools used for that are not discussed. The much more broadly used tool Ansible, and the roles created for that, will be used cross-teams and much more frequently. In a lot of different use cases and thus need a way for scheduling and overview of Ansible runs. Get visual feedback from the run results and good overview of what happened. With means to debug and alter the job if needed.

8.1. Ansible Tower

Ansible has a very feature rich tool for centralizing and managing Ansible runs. The list of features and provided benefits is extensive[15] making it a very viable option for management and information source of Ansible runs and results. Tower being provided by the same company as Ansible, makes it very tightly integrated with good coding conventions of inventory files and host facts. Providing a good overview of the existing infrastructure out of box. Having somewhat support for LDAP[16], makes user management easier and life of users a little easier, not needing to remember yet another credential.

The dashboard view enables graphical inventory management illustrated on Figure 23 and job scheduling. In addition the built in REST API allows for centralized command execution and integration with other tools to make things easier to integrate and automate.



	Groups	Actions
	knloginservers	
	mobilegateway	

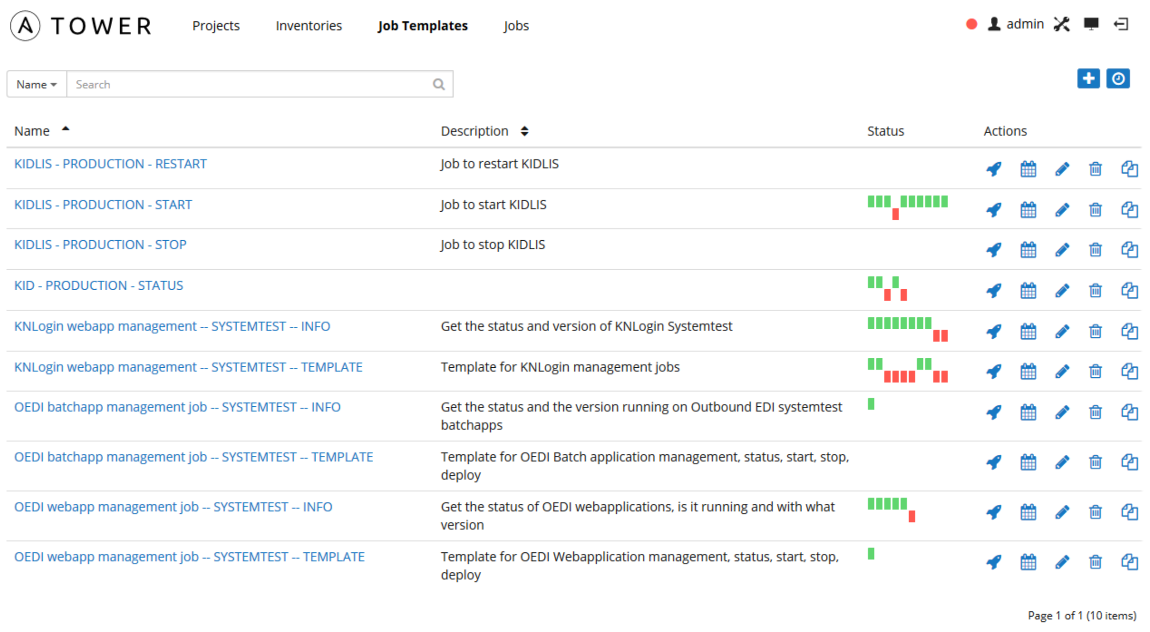
Page 1 of 1 (2 items)

	Hosts	
	dehamsl485.int.kn	●
	dehamsl486.int.kn	●
	dehamsl487.int.kn	●
	dehamsl488.int.kn	●
	kineas.int.kn	○
	komanos.int.kn	○

Page 1 of 1 (6 items)

Figure 23. *Ansible Tower User Interface - Inventories Screen.*

The drawback with the user interface at its current state is the lack of parametrized runs. Currently it is not convenient to set parameters before each run. It is required to edit the job template before the run to achieve a different run. To circumvent that, a lot of job templates for different use cases will need to be created and at that point, the UI will get cluttered and hunting or searching for the required job is not a welcome addition to the work flow. Figure 24 illustrates the situation where to have the basic functionality of starting stopping and restarting a group of applications needs 3 different jobs, even thou the task is achieved with one single role that is run with different parameters.


TOWER Projects Inventories **Job Templates** Jobs admin

Name ▾ Search

Name ▲	Description ⬍	Status	Actions
KIDLIS - PRODUCTION - RESTART	Job to restart KIDLIS		
KIDLIS - PRODUCTION - START	Job to start KIDLIS		
KIDLIS - PRODUCTION - STOP	Job to stop KIDLIS		
KID - PRODUCTION - STATUS			
KNLogin webapp management -- SYSTEMTEST -- INFO	Get the status and version of KNLogin Systemtest		
KNLogin webapp management -- SYSTEMTEST -- TEMPLATE	Template for KNLogin management jobs		
OEDI batchapp management job -- SYSTEMTEST -- INFO	Get the status and the version running on Outbound EDI systemtest batchapps		
OEDI batchapp management job -- SYSTEMTEST -- TEMPLATE	Template for OEDI Batch application management, status, start, stop, deploy		
OEDI webapp management job -- SYSTEMTEST -- INFO	Get the status of OEDI webapplications, is it running and with what version		
OEDI webapp management job -- SYSTEMTEST -- TEMPLATE	Template for OEDI Webapplication management, status, start, stop, deploy		

Page 1 of 1 (10 items)

Figure 24. *Ansible Tower User Interface - Job Templates Screen.*

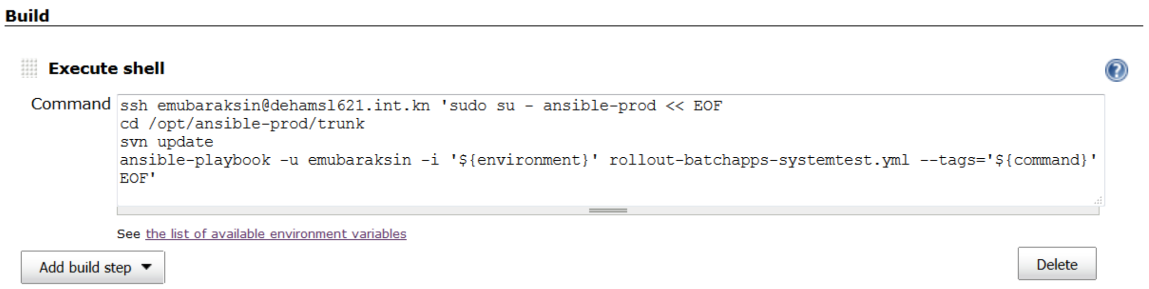
Despite Tower providing an opportunity packed with features. The developers do not find the idea of yet another extra step in the pipeline too pleasing. The LDAP integration only enables automatic user creation. Team grouping and job access still needs to be done manually. This can be considered as unnecessary managerial overhead. The jobs need to be created and configured manually even thou the command line equivalent calls that were used during creation and testing already exist. So from pure work flow related reasons Tower, thou being a promising tool, has too many prerequisites for integrating into the day to day work of a Java developer.

Tower is also a tool that is still in development and experiencing all sorts of changes and improvements. With that, also a few problems. As during our limited testing time it ran into problems with terminated jobs remaining as stray processes. Finished jobs sometimes hanging and then leading to performance issues of the tower host itself. Needing manual maintenance to keep things moving. By now the issues are probably ironed out, but the initial impression was promising, but the resulting work flow was a little disappointing. The other aspect talking against tower is the cost. At $70 a node a year, then it can get a little costly at large scale and costly to begin with if the developers see it as an extra hoop to jump through.

8.2. Ansible and Jenkins Integration

Hudson/Jenkins has been around for a while and has matured over its time. Becoming one of the go to solutions continuous integration services for software development. Having been around for this long, a lot of developers are used to it and consider it as one of the essential tools in their day to day jobs. Thou suffering from performance problems time to time, it is still very widely used and is very common in Java development pipelines. Among the multitude of its other plugins, Jenkins also has an Ansible plugin[17].

The setup of the plugin is easy and needs an ansible installation on the server to run. After the plugin is configured it enables basic inventory definition and to execute Ansible tasks as a job build step. The other way to utilize Ansible in an easy and out of the box way is to simply use the Ansible as a command line call as is illustrated on Figure 25. This call can be parametrized via Jenkins built in features as is shown on Figure 26. Creating easy to manage and use jobs that take parameters and can be called by other jobs at certain steps and with according tags is in the skill set of most of the people in the industry.

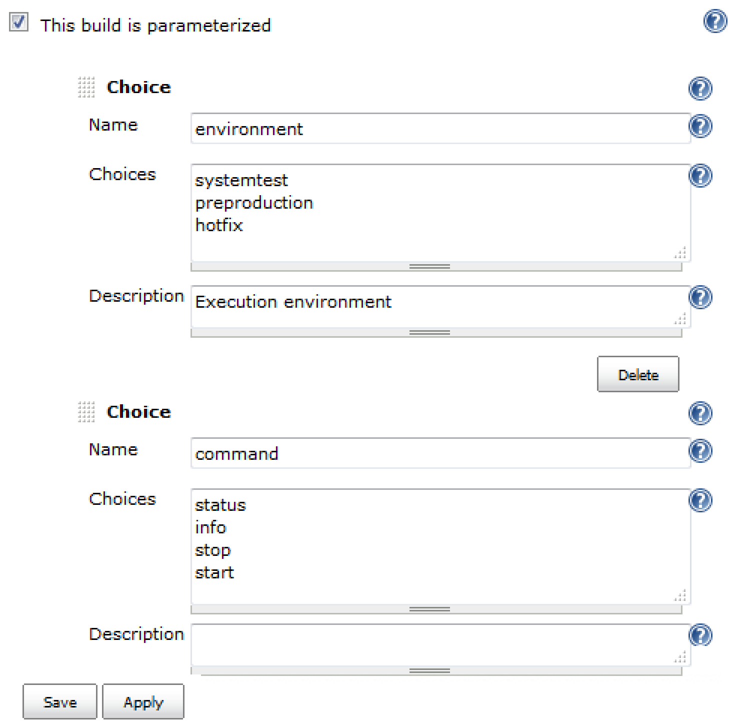


```
ssh emubaraksin@dehamsl621.int.kn 'sudo su - ansible-prod << EOF
cd /opt/ansible-prod/trunk
svn update
ansible-playbook -u emubaraksin -i '${environment}' rollout-batchapps-systemtest.yml --tags='${command}'
EOF'
```

Figure 25. *Jenkins Job Configuration - Execute Shell Build Step.*

This all enables easy to manage parametrized jobs that can be called by other jobs or by developers to achieve a certain task that is needed. As previously stated, Figure 26 illustrates the Jenkins built in parametrization support. By using this feature achieving a user friendly front end for an Ansible role is very straight forward. A resulting Jenkins job that interfaces an Ansible role with all the parameters is shown on Figure 27.

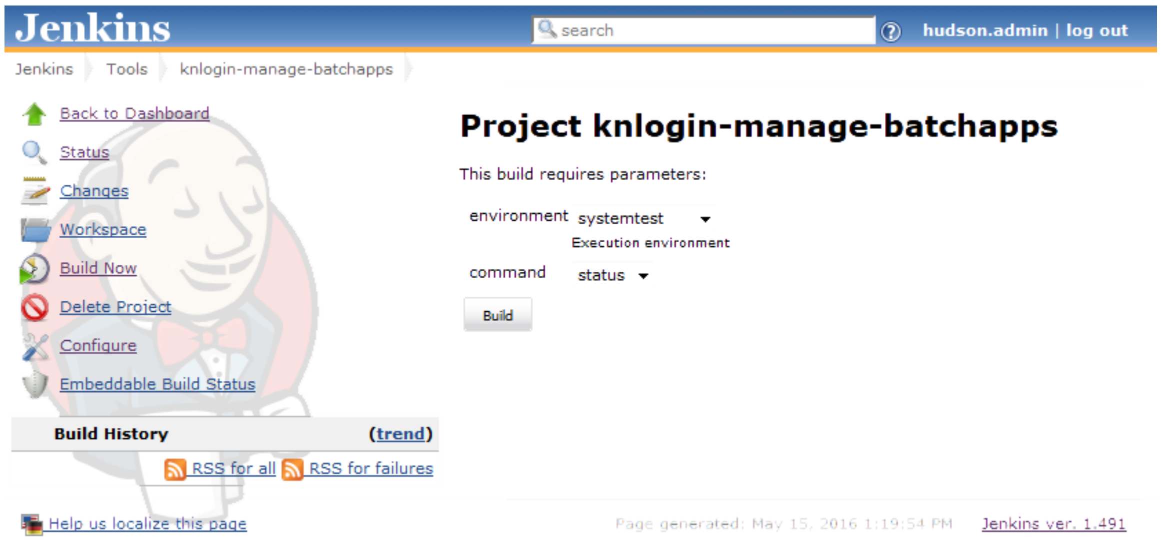
The benefit of this is that it makes the push button automation seamless and invisible for the developer. Integrates it to the existing work flow. And with ansible role run based return values, that interface with Jenkins' default shell return value interpretation also enables failed states and visual feedback of the run. As Ansible is called as a command line call by Jenkins, then this also means that the build log automatically captures the stdout of the Ansible role run. Making the results and debugging info readily available within Jenkins. The readily available capabilities of Jenkins and the familiarity the developers have with Jenkins leads this to be the closest fit for current times. Until a more beneficial tool comes along that is enough to make the developers open for a change.


This build is parameterized

Choice
Name: environment
Choices:
systemtest
preproduction
hotfix
Description: Execution environment

Choice
Name: command
Choices:
status
info
stop
start
Description:

Figure 26. *Jenkins Job Configuration - Parametrized Build.*


Project knlogin-manage-batchapps

This build requires parameters:

environment: systemtest
Execution environment
command: status

Page generated: May 15, 2016 1:19:54 PM

Figure 27. *Jenkins Jobs Screen - Ansible Role Job.*

9. Automated Documentation

One large part of any development process is documentation. With automation the important role of documentation more often than not takes a back seat. Due to the large-scale and urgent needs of requesters. Documentation is sometimes left lagging behind. This leads to outdated information. Unknown and undocumented processes and sometimes even false understandings and miss communications.

Automation tools such as Puppet or Ansible can be viewed as code. The best way to document code has been within the code itself. Comments should be a normal and everyday part of any Puppet manifest or Ansible role, task or module. The coding documentation and effort should be coordinated within the team itself in accordance to the available resources and time. To provide the best possible solution for code documentation.

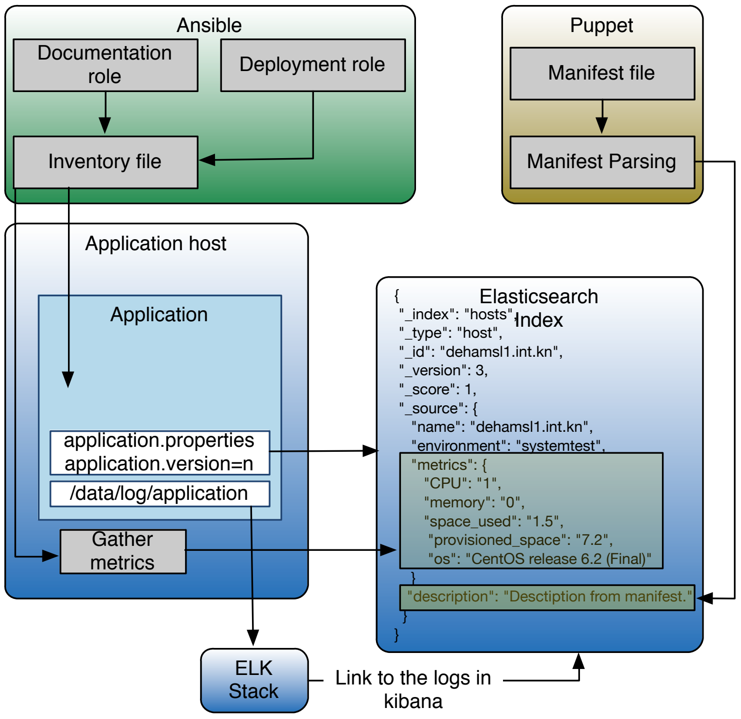
Automation tools allow to go beyond code documentation. Automation tools and the descriptions in them are a great information source on the actual state of the infrastructure. One such example is network visualizers that self explore and mark down the network to provide useful information. Using that information in documentation can provide up to date information [18]. More importantly, fee up people. In similar fashion, with a little help, Ansible inventories can provide an excellently comprehensive overview of projects and their machines. The same goes for Puppet manifests. This information is readily available for those who are familiar with either of the tool.

Making this information easily accessible from one place that centralizes all the information available on the node. Can make things more transparent and easy to follow for everyone concerned. Especially if the useful information already exists somewhere in an abstract way and can automatically gathered. Especially in the case of automation tool DSL having certain constructs how to define something. Or has a certain way of returning information after an automation run.

Ansible is a great opportunity of gathering facts on hosts. It has a lot of useful constructs to gather information that can be captured and saved during a playbook run. So taking a step from documentation via code comments. Towards active facts gathering and centralizing as a coding habit can not only simplify the documentation effort, but also provide a lot of useful information readily available for other parties improving the infrastructure transparency.

9.1. Documentation Centralization Automation

In modern IT infrastructure the ways of how to represent and store data has evolved. There is numerous methods of how to gather information. One new way to index, search and retrieve data is Elasticsearch[19]. Elasticsearch is an abstraction layer implemented in java to utilize the full-featured text search engine benefits provided by Apache Lucene[20]. The high performance indexing and search capabilities provided by Lucene. Combined with Elasticsearch's widely adapted and easy to use API that has implementations in most wide-spread programming languages. Among them being Python[21], Ruby[22] and somewhat support for indexing directly in bash[23].



```
{
 "_index": "hosts",
 "_type": "host",
 "_id": "dehamsl1.int.kn",
 "_version": 3,
 "_score": 1,
 "_source": {
   "name": "dehamsl1.int.kn",
   "environment": "systemtest",
   "metrics": {
     "CPU": "1",
     "memory": "0",
     "space_used": "1.5",
      "provisioned_space": "7.2",
      "os": "CentOS release 6.2 (Final)"
   }
  "description": "Desctiption from manifest."
  }
}
```

Figure 28. *Self Documentation Framework Conception Illustration.*

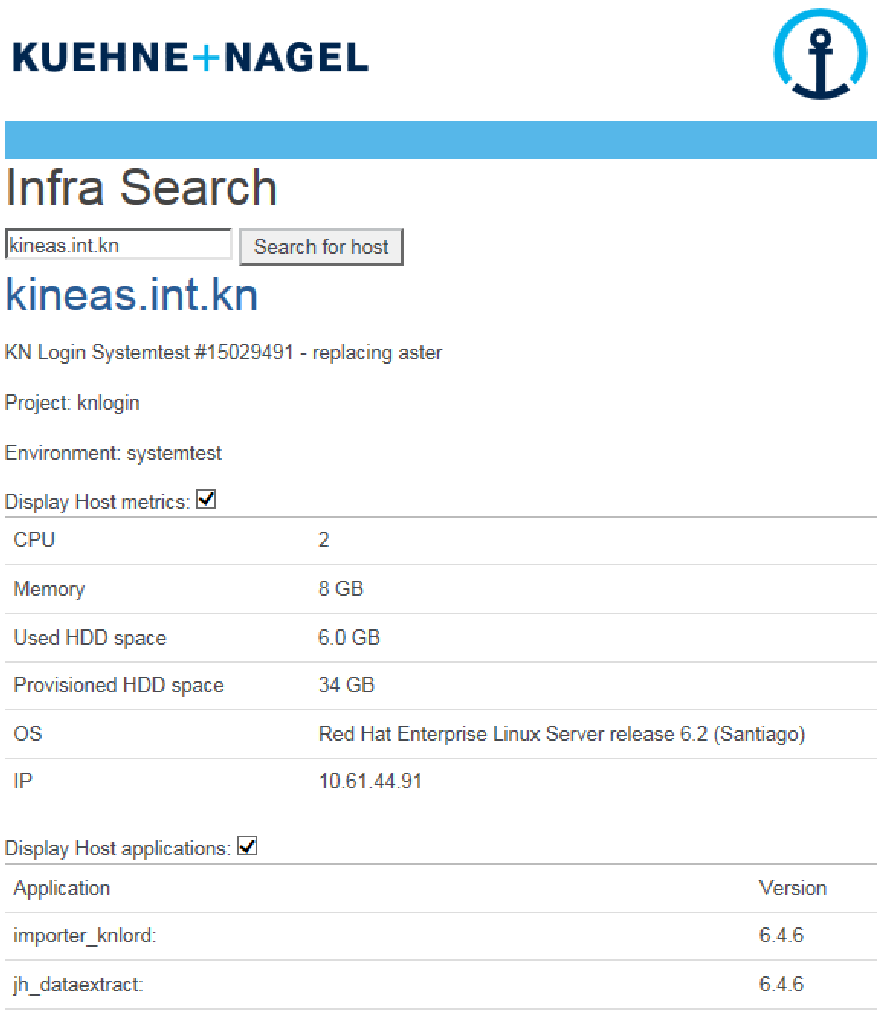
This easy to use and widely supported way to index data provides the perfect opportunity to create a sort of a multi agent based data acquisition system to help ease the documentation effort. Figure 28 illustrates the conception where multiple different sources, and programming language based components offload helpful information to a central ES index.

The following Ansible role on Figure 29 illustrates the basics of registering values and then using a bash call to create the index for host related information. And since the key for the index will be the hostname itself, then there is no need to check if the index is already created or not and makes data gathering easy. This index can then be updated and by other roles or by that matter any other function or build job that can provide useful information. For instance the playbook that installs and sets up logstash shipper for an application has enough information in the roles to add a link to the Kibana dashboard that contains the relevant information for this application.

```
---
- hosts: all
  gather_facts: no

  tasks:
    - name: Get host name
      command: uname -n
      register: host_name

    - name: Get CPU count
      shell: 'cat /proc/cpuinfo | grep "processor" | wc -l'
      register: host_cpu

    - name: Get Total memory
      shell: free -g | grep Mem | sed -r 's|^([^.]+).*$|\1|; s|^[^0-9]*([0-9]+).*$|\1|'
      register: host_memory
        .
        .
        .
    - name: Create initial index for host with basic metrics
      shell: 'curl -XPUT http://antiochos.int.kn:9200/hosts/host/{{host_name.stdout}} -d
        ''{"name": "{{host_name.stdout}}", "environment": "{{environment_stage}}",
        "metrics": { "CPU": "{{host_cpu.stdout}}", "memory": "{{host_memory.stdout}}",
        "space_used": "{{host_used_space.stdout}}", "provisioned_space":
        "{{host_provisioned_space.stdout}}", "os": "{{host_os.stdout}}",
        "host_ip": "{{host_ip.stdout}}"}, version: ""}'''
      delegate_to: 127.0.0.1
```

Figure 29. *Excerpt From Metrics Gathering Role.*

Once the information is indexed. Representing it can be done in a lot of ways as again ES has a lot of compatibility with a lot of other tools and programming languages. And creating an implementation for visualizing the front end can be done in many ways. The way I went for the proof of concept was to use the ES javascript support[24] with a simple Angular based site to display the index search results. The resulting search page is illustrated on Figure 30. Thou similar results can be achieved using other methods.



Infra Search

kineas.int.kn

kineas.int.kn

KN Login Systemtest #15029491 - replacing aster

Project: knlogin

Environment: systemtest

Display Host metrics: ☑

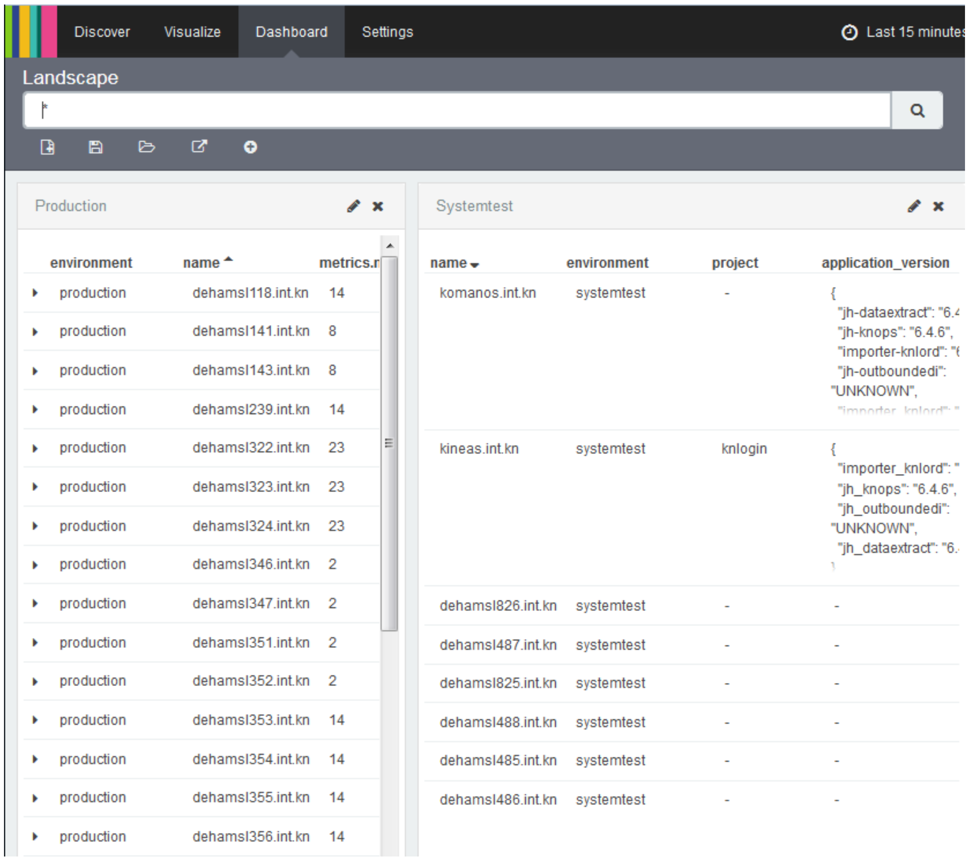
CPU	2
Memory	8 GB
Used HDD space	6.0 GB
Provisioned HDD space	34 GB
OS	Red Hat Enterprise Linux Server release 6.2 (Santiago)
IP	10.61.44.91

Display Host applications: ☑

Application	Version
importer_knlord:	6.4.6
jh_dataextract:	6.4.6

Figure 30. *Infrastructure Search Page Screenshot Displaying Search Result of One Server.*

A much easier way to display and search for indexed data is Kibana[25]. Kibana is typically used for data analytics and visualizing data. But also for full text search in application logs. It is already very widespread and thus it means that people are used to the work flow related to the introduction of Kibana. Making it easy to provide additional value via a dashboard that provides an overview of information available on the hosts in a project, or the whole infrastructure. A dashboard implementation that simply filters application hosts by their environments is shown on Figure 31. To someone who is familiar or has previous experience with working with Kibana, finding and filtering information would cause no problems.



Production

environment	name	metrics.r
production	dehamsl118.int.kn	14
production	dehamsl141.int.kn	8
production	dehamsl143.int.kn	8
production	dehamsl239.int.kn	14
production	dehamsl322.int.kn	23
production	dehamsl323.int.kn	23
production	dehamsl324.int.kn	23
production	dehamsl346.int.kn	2
production	dehamsl347.int.kn	2
production	dehamsl351.int.kn	2
production	dehamsl352.int.kn	2
production	dehamsl353.int.kn	14
production	dehamsl354.int.kn	14
production	dehamsl355.int.kn	14
production	dehamsl356.int.kn	14

Systemtest

name	environment	project	application_version
komanos.int.kn	systemtest	-	{ "jh-dataextract": "6.4 "jh-knops": "6.4.6", "importer-knlord": "("jh-outboundedi": "UNKNOWN", "importer_knlord": "
kineas.int.kn	systemtest	knlogin	{ "importer_knlord": " "jh_knops": "6.4.6", "jh_outboundedi": "UNKNOWN", "jh_dataextract": "6. }
dehamsl826.int.kn	systemtest	-	-
dehamsl487.int.kn	systemtest	-	-
dehamsl825.int.kn	systemtest	-	-
dehamsl488.int.kn	systemtest	-	-
dehamsl485.int.kn	systemtest	-	-
dehamsl486.int.kn	systemtest	-	-

Figure 31. *Kibana Dashboard Displaying Indexed Data, Filtered by Environments.*

By changing the way we consider documentation and use the building blocks provided by automation tools. Changing our mindset and scripting/coding style to keep in mind the additional benefits of indexing aspects of information bits about the infrastructure. We can improve the amount of documentation like information accessible to people with one simple search and spare the frustration of hunting in outdated wiki pages or long forgotten documents about orders of machines. Providing a clearer view on the infrastructure and improving communication.

10. Infrastructure Scaffolding

In the previous sections we have gone over the largest steps an application will go through in its life cycle. The reasons behind the need to be able to release often and without effort cyclically. The main needs people in a team have, the expectations they have from other people and surrounding teams. The building blocks of the possible solution to the cross-team effort known as application landscape.

Building blocks like loadbalancers, Process control systems, configuration management tools. All with having a goal in mind to create a pipeline of orchestration. One that would enable to create a smooth information flow from inception to production. From zero to application with the help of a common understanding of the process. The decisions that need to be done and communicated to other teams ahead of time.

10.1. Application Platform Implementation Process

The process to a successful application vertical kick off is down to smooth information flow between the requesters and executers. In large scale companies that have multiple projects running in its enterprise the process can over time become poorly documented and the issue only gets worse with distributed teams that do not have the convenience of going straight to the management or the corresponding team to discuss the problems directly.

This can be a real problem to new teams that can lack the insider information to manage on their own. To avoid such situations and to improve the efficiency of existing teams. An infrastructure scaffolding is given. As a reference guide to align the efforts to common base blocks on what to start from. Even thou the steps are familiar to everyone in the chain of implementation. It often seems that there is a lack of high level understanding of the whole process outside their own scope.

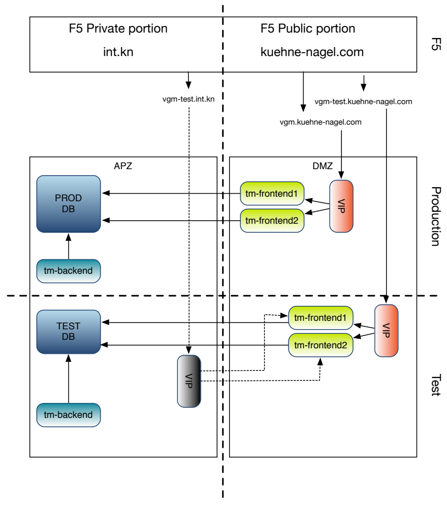
Table 4 illustrates at high level the process from inception to implementation in a large distributed enterprise with teams across the world.

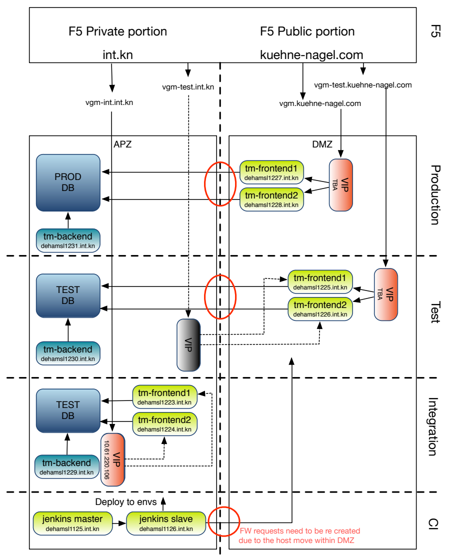
Project Vertical Scaffolding Flow	
Decision	Executors
Requirements	BU and Developers
Cost	BU and Cost center
Hardware and storage	Infra Team
Clustering/Provisioning	Infra Team
Network zones	Infra Team, based on BU Requirements
Firewall Rules and Loadbalancing	Network Team, DevOps, Developers
Application rollout	DevOps and Developers

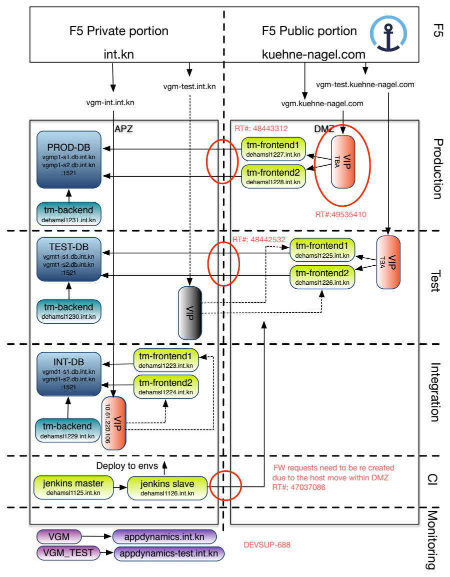
Table 4. *Decision and Decider Listing.*

Requirements are captured in the Inception phase, setting the time frame for implementation and the basic information for the rest of the process to go on with. Setting the needs for all the infrastructure. As infrastructure reflects directly the business needs. The predicted load and frontend/backend distribution in the Infrastructure. The storage requirements and the distribution of storage, that sets limitations to clustering of the VM-s. Clustering in a ESX cluster based infrastructure can decide the performance and data latency of the provisioned VM-s. The clustering also depends on the network zone of the host. Changing this by migrating the host from one zone to the other is easy, thou it will change the host IP, rendering the existing firewall rules useless. Once all the infrastructure related tasks are done, application landscape related processes can start. Heavy co operation between Infrastructure team DevOps and Developers ensures rapid implementation.

Having a good overview of the overall state of current events is essential to eliminate miss communications and bottlenecks. Also having the information readily available can enable escalation of tasks even if the original requester or implementer is not available at the moment. Visualizing the existing information in easy to comprehend block diagrams can be an convenient way to convey the progress. This will introduce some effort in manual documentation. Compared to the alternative, what is sending countless mails to people in different teams. The investment is worth the effort. Figure 32 is a real life example how documentation effort helped re-align inter department communication. How the initial information forwarded from management was not complete. And as our team works on the information provided to us, this could have had set back a time sensitive project. With proper grouping of Ansible inventories, combined with a firewall rule checking role. A dependency graph could be generated automatically from the available information that is hidden in Ansible. Making it theoretically possible to automate the reporting.


(a) Initial missing information on integrationtest


(b) Update, Int and build server information provided


(c) Progress and critical tickets

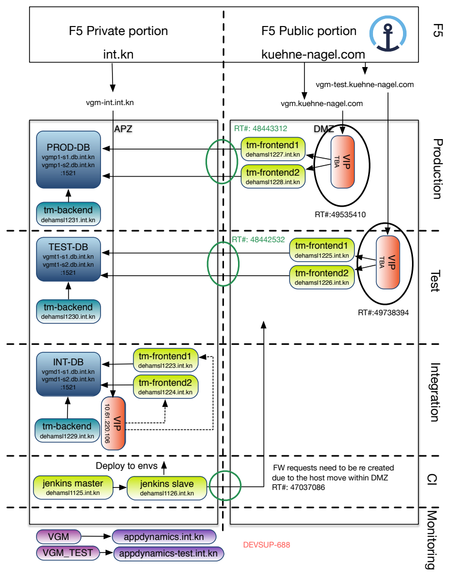

(d) Final state, all functionality available

Figure 32. *Progress Visualization Documentation.*

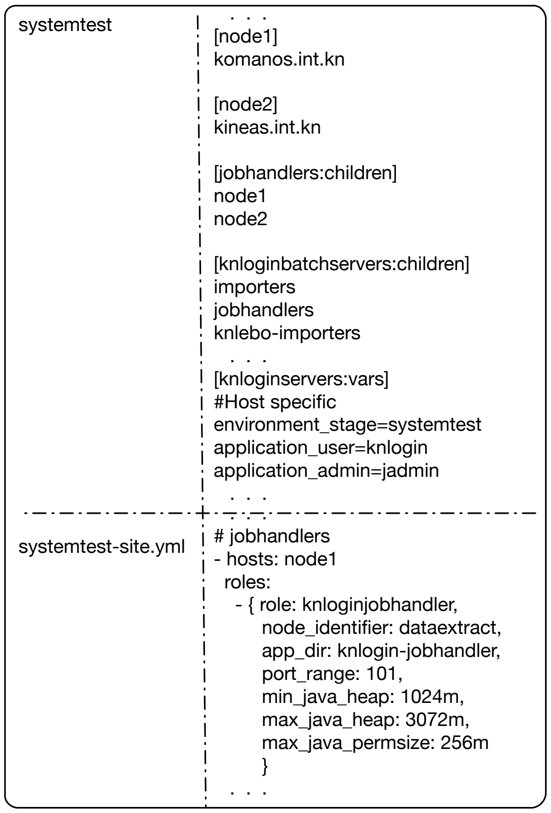
10.2. Application Roll-out on the Platform via Ansible

Once the application platform is established by the Puppeteers, then the platform can be populated via an Ansible playbook run. Also referred to as site run or simply application landscape roll out. This is achieved via a command line call shown on line 1 on Figure 33. The command calls ansible playbook with the corresponding user followed after the -u tag. Utilizing the inventory specified after -i. What in this example is Systemtest inventory. The roles in the playbook or also referred to as the site file are ran on the hosts in the inventory. Figure 34 gives an overview of the content of the files and the information they contain.

```
1  ansible-playbook -u emubaraksin -k -i systemtest systemtest-syte.yml
2  ansible-playbook -u emubaraksin -k -i systemtest rollout-batchapps-systemtest.yml
   -e version=6.3.12
```

Figure 33. *Ansible Command Line Calls.*

The snippets in on Figure 34 illustrate the bare minimum for the easiest use case of rolling out a back end java application. The parameters for the roles are defined in the role call if they are individual for a group. Alternatively the same parameters can be over written in host variables directly in the inventory files. Doing this introduces clutter into the inventory. With the other option being creating a separate file for variables. Such an approach increases the level of layers and can lead to loss of overview. Ultimately it is up to the coder to decide how to approach this. In our implementations, the verticals are light weight. Keeping the relevant information in the inventory file while keeping the hierarchy as flat as possible thus has more benefits than drawbacks. As this is mostly a cosmetic decision and has no functional impact, then the current decision is to keep things as flat as possible.


systemtest

```
 . . .
[node1]
komanos.int.kn

[node2]
kineas.int.kn

[jobhandlers:children]
node1
node2

[knloginbatchservers:children]
importers
jobhandlers
knlebo-importers
  . . .
[knloginservers:vars]
#Host specific
environment_stage=systemtest
application_user=knlogin
application_admin=jadmin
  . . .
```

systemtest-site.yml

```
# jobhandlers
- hosts: node1
  roles:
    - { role: knloginjobhandler,
        node_identifier: dataextract,
        app_dir: knlogin-jobhandler,
        port_range: 101,
        min_java_heap: 1024m,
        max_java_heap: 3072m,
        max_java_permsize: 256m
        }
  . . .
```

Figure 34. *Application Site Roll Out.*

The roles ran against the inventory that are required to set up a back end application landscape is illustrated on Figure 35. The one role called in the play book has a dependency on a role. The role included has sub dependencies. This all enables precise orchestration and ensures that all required tasks are included and executed. Ansible self enforces defensive coding of play books. Firstly by providing good blocks to build checks on and secondly by automatically evaluating each return value of every task. If the return value signals an OS level failure or just the return value is not 0 Ansible will react to that, mark the host as failed. Stops executing on the node and on parallel nodes. Simply failing the job can have consequences in case of more important roles. So defensive coding and error handling is advised. In role tasks that check if all parameters are defined and targets of the role are present. But on application roll out runs, a clean run without any errors is a guarantee of a functional result.

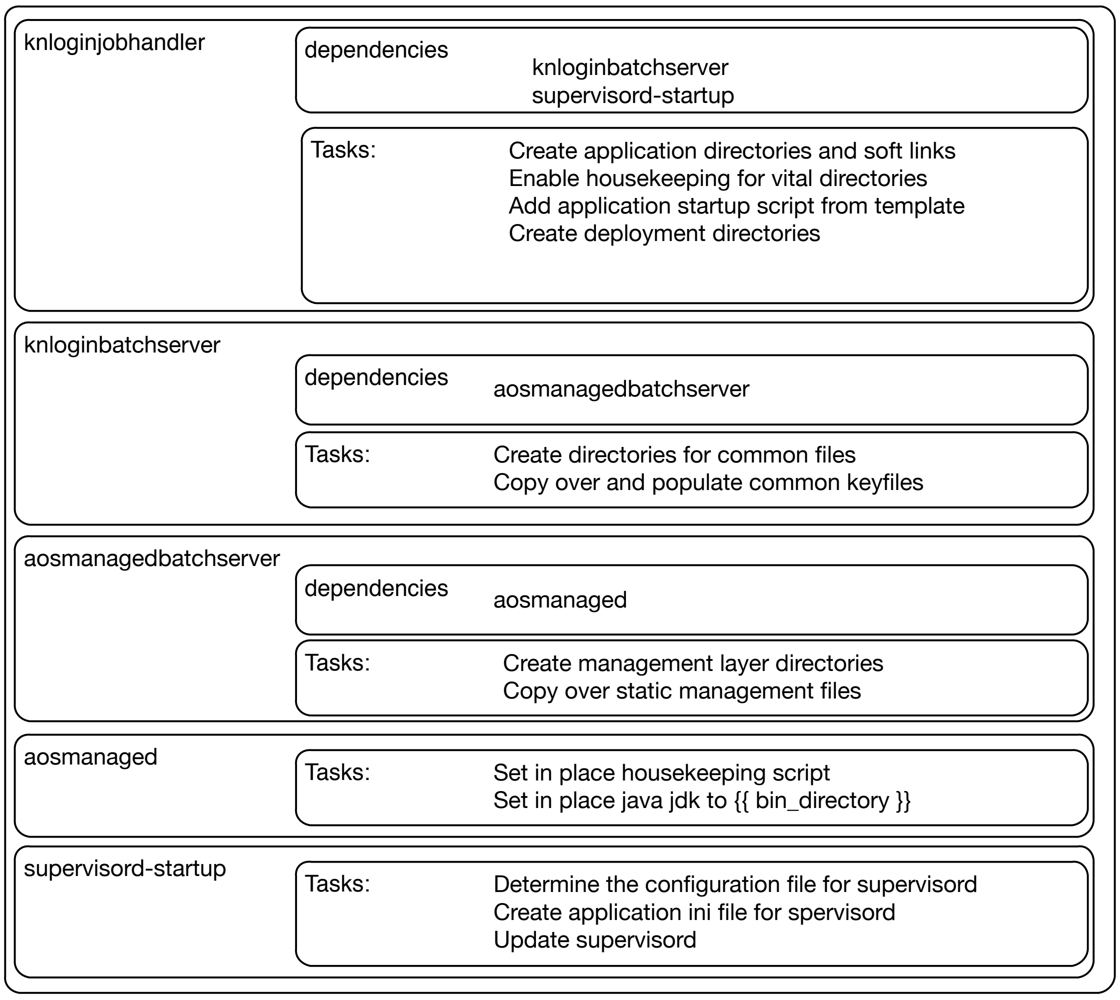

knloginjobhandler
dependencies
knloginbatchserver
supervisord-startup
Tasks:
Create application directories and soft links
Enable housekeeping for vital directories
Add application startup script from template
Create deployment directories
knloginbatchserver
dependencies
aosmanagedbatchserver
Tasks:
Create directories for common files
Copy over and populate common keyfiles
aosmanagedbatchserver
dependencies
aosmanaged
Tasks:
Create management layer directories
Copy over static management files
aosmanaged
Tasks:
Set in place housekeeping script
Set in place java jdk to {{ bin_directory }}
supervisord-startup
Tasks:
Determine the configuration file for supervisord
Create application ini file for spervisord
Update supervisord

Figure 35. *Application Rollout.*

The command line call on line 2 on Figure 33 is a separate play book for application management and deployment. This is a simplified example not including the F5 modules as currently the Ansible modules for F5 are still to be introduced to the production load balancer. The play book in the command is illustrated on Figure 36 and calls the roll out play that does not make any changes to the landscape. Thus it does not include any roles. Instead it includes commands that contain tasks that are needed for orchestration.

```
# make custom modules available
- hosts: knloginbatchservers
  roles:
    - custom_modules
# stop all the batch apps
- include: commands/knlogin_ng/stop-batchapps.yml
# deploy all the batch apps
- include: commands/knlogin_ng/deploy-batchapps.yml
# start up the batch applications
- include: commands/knlogin_ng/start-batchapps.yml
# check status of applications
- include: commands/common/checkstatus-batchapps.yml hostgroup=knloginbatchservers
# Get the version of applications
- include: commands/common/get-versions.yml hostgroup=knloginbatchservers
```

Figure 36. *Ansible Playbook - rollout-batchapps-systemtest.yml.*

The main commands called in the figure above and their overview is given on the Figure 37. In this case only the basic behavior is needed as back end applications are asynchronous and their momentary downtime is invisible for clients.

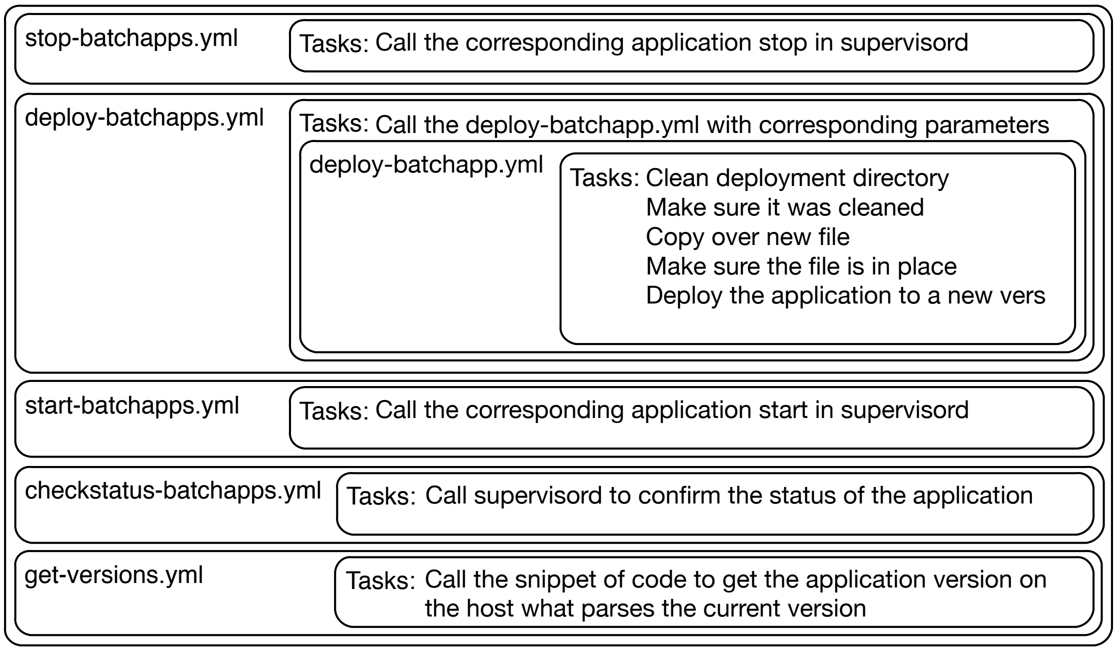

stop-batchapps.yml
Tasks: Call the corresponding application stop in supervisord
deploy-batchapps.yml
Tasks: Call the deploy-batchapp.yml with corresponding parameters
deploy-batchapp.yml
Tasks: Clean deployment directory
Make sure it was cleaned
Copy over new file
Make sure the file is in place
Deploy the application to a new vers
start-batchapps.yml
Tasks: Call the corresponding application start in supervisord
checkstatus-batchapps.yml
Tasks: Call supervisord to confirm the status of the application
get-versions.yml
Tasks: Call the snippet of code to get the application version on the host what parses the current version

Figure 37. *Ansible Playbook - Task Included in Backend Deployment.*

Putting all the available information to good use is where automation makes the difference. Using all the available building blocks to quickly and reliably go from zero to application. With a lot of automation implemented via Ansible, making it the heart of the application rollout process. Figure 38 illustrates a more complete deployment and management pipeline that serves the needs of all stakeholders.

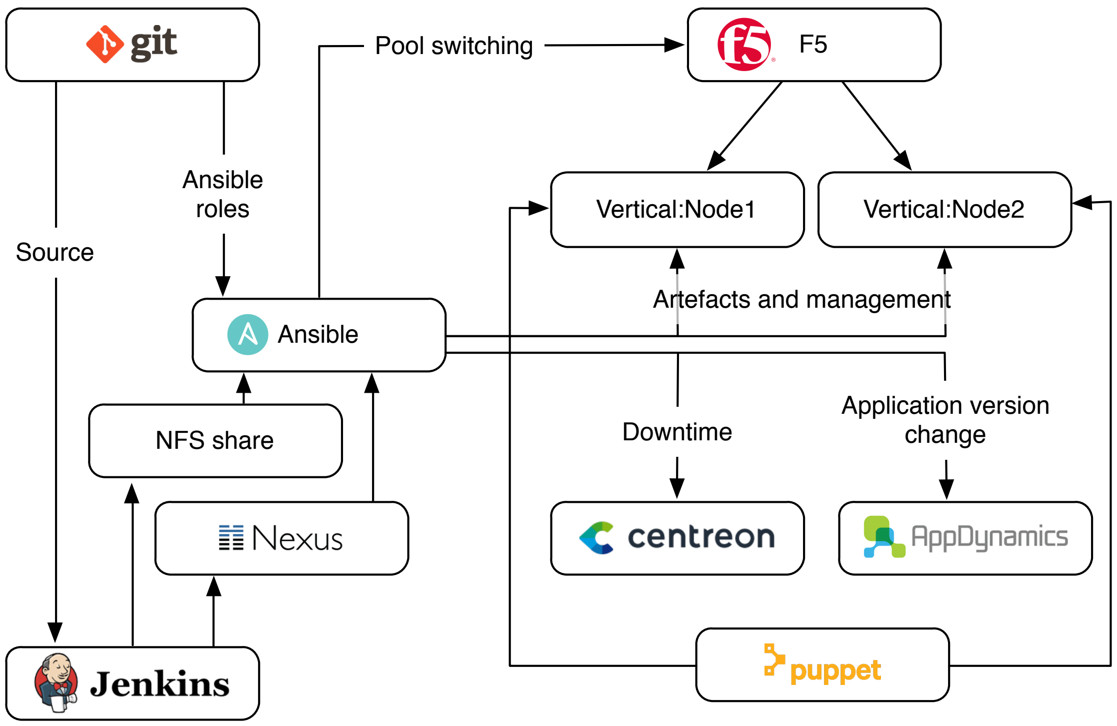


Figure 38. *Application Infrastructure Automation Structure*

Puppet with enforcing security updates and user credentials. Managing root level services and guaranteeing that the host is always ready for usage. Followed by Ansible in tight symbiosis. Making all other pieces of the puzzle available to use in a seamless yet transparent manner. Ansible modules taking care of loadbalancer management, application version updating in a rolling manner. Making deployments seamless and not noticeable for clients. Setting downtimes for hosts in Centreon to avoid unnecessary alerting of support personnel. Sending Application version change related information to Appdynamics toolset for business reporting. Freeing up Developers and DevOps engineers from having to manually carry out updates on production.

10.3. Real Life Benefits of Automation

Real life benefits of automation can be substantial. As an example, the time spent on System test environment weekly deployment will be illustrated over the progress from legacy manual methods to scripted orchestration. Finishing up with a full application management pipeline via Ansible. The initial landscape consisted of 3 hosts and 12 applications, with virtually no automation. Consisting of manually transferring artifacts to a distribution server. Distributing the artifacts from that host to the nodes and carrying out the deployment related processes and housekeeping. The downtime window for this process was 1 hour and with all the tasks included. usually took around 50 minutes of terminal usage in best case scenario.

The first step towards automation was writing shell wrappers to get rid of manual execution to save time spent on mind numbing tasks. This enabled to cut time on the simpler use cases and invest more time into further improvements. Figure 39 illustrates the process with the acceleration scripts. This basically reduced the effort to running 3 scripts on 3 hosts. Taking the time from 1 hour down to 15 min of executing and maintenance.

With the introduction of new infrastructure and visualization, Ansible was introduced. Once the orchestration was completed it takes only one parameter change for an autonomous deployment job to run. Minimizing the deployment effort even further, despite the increase in number of managed servers. As the number of nodes increases Ansible will still deliver a consistent execution times per node.

Comparison of manual process, scripted and Ansible

	Manual	Scripted	Ansible
Number of hosts	3	3	7
Number of applications	15	15	15
Terminal usage time	50	15	5
Time per node	$16\,\frac{2}{3}$	5	$\frac{5}{7}$

Table 5. *Estimated Time Consumption Comparison.*

As can be seen from the Table 5 the efficiency has increased dramatically even in the easiest use case and shows the benefits of automation. The simple roles used to orchestrate the basic use case are also used in more complex use scenarios. Meaning that re-use of roles will further save the development time too. Not only the benefit of push button automation.

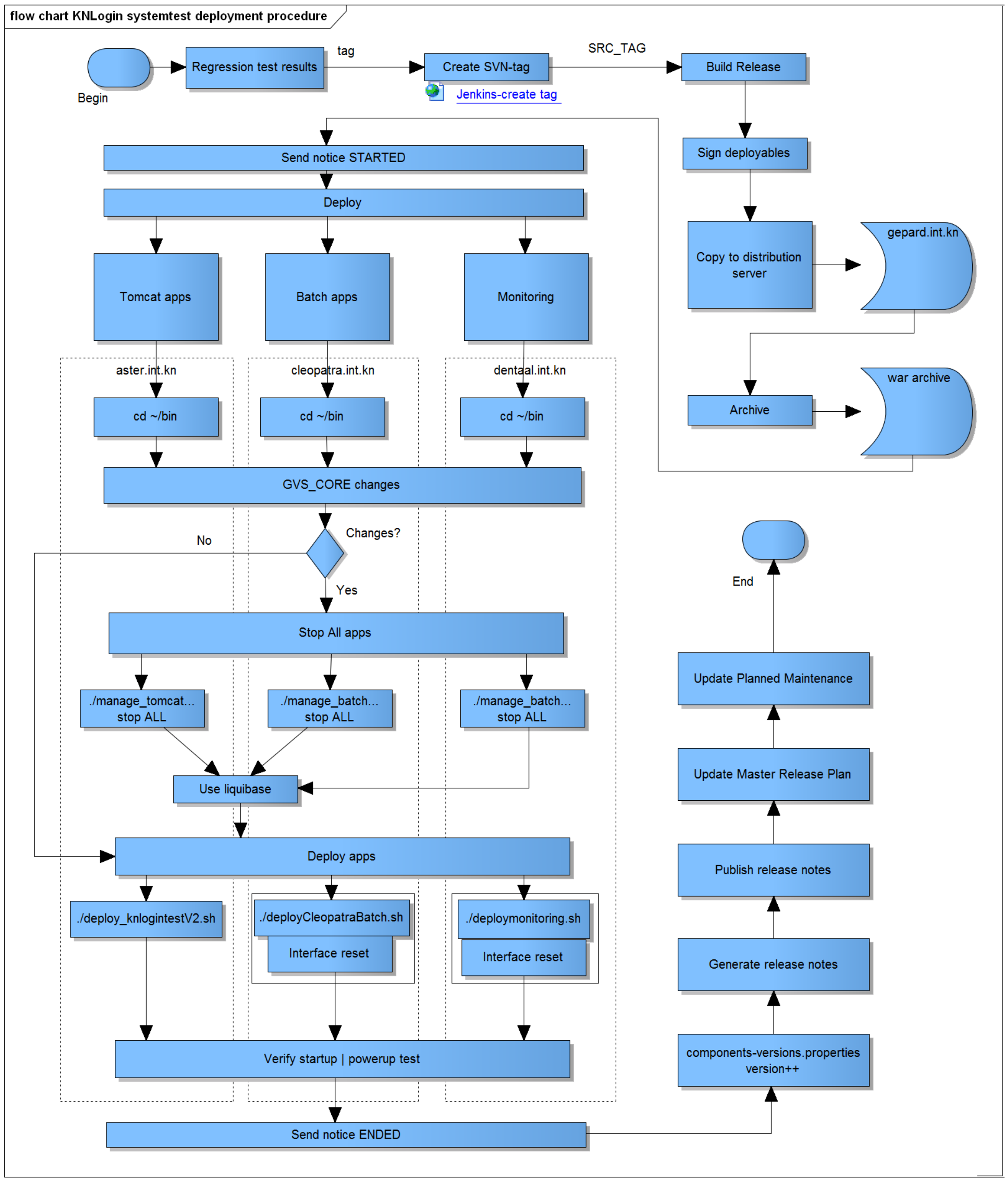


Figure 39. *Manual Deployment Procedure.*

11. Container Technology

Introducing automation tools and streamlining the processes involved with infrastructure provided a massive step forward in manageability, maintainability and decreased time spent on the tasks involved. Leading to virtually no manual operations on the hosts themselves. Providing major improvements compared to manual solutions. Moving on from automation, the next big improvement and major trend for the future can be considered containerization[26].

Before this can happen a different trend that is already taking the hearts and minds of agile developers and architects in major companies. The move from monolithic applications and application verticals towards more light weight microservices [26]. What is in its essence the next step forward from Service oriented architecture solutions.

Microservices are considered easier to limit their boundaries, and thus making them easier to localize and apply some form of container technology on them. This is especially true to applications that follow the twelve factor application principals [27]. The guidelines try to steer application developers towards good principals and common grounds to develop easy to containerize applications.

There are some very promising technologies out there on the market that such as Docker and distributed system kernel such as Apache Mesos along with DCOS running on top of it. Providing a lot of simplifications and optimizations compared to VM and Virtualized OS approach [28]. This being an emerging technology. Trying to promise almost as much Java did in its time with write once, run anywhere. This promise has still not really been delivered to this date by Oracle [29].

Container technologies provide benefits compared to more classical VM based approach. They enable simpler application management and basically automatic application setup with the “just run my app” principal. All the user has to do to get a container up and running is to pull the image and starting it up. As a container is a self sustained setup, then there is nothing to configure on the application host. The second benefit is the easier management as containerisation is, in effect, OS-level virtualisation as opposed to VMs, which run on hypervisors, each with a full embedded OS. This difference between VM-s and containers is illustrated on Figure40.

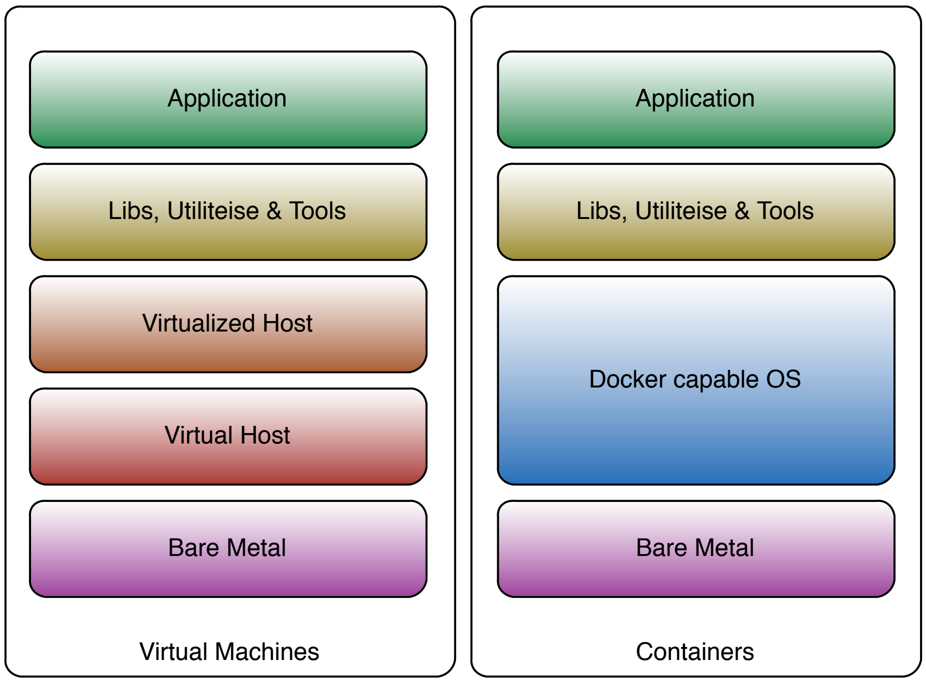


Figure 40. *Virtual Machines and Containers.*

On the other side containerization does introduce new problems and uncertainty[30]. One example of an old problem in a new shape is sprawl[31]. Since it is easy to pull a container, set it up, do some modifications. Re upload it and set it up again. In no time the benefit of easy use can turn into a sea of unmanaged containers as developers do not usually have the habits of administrators. The second concern is security. As Docker daemon runs at root level and no one is ever sure how trusted the current software solution is and how trusted the available containers on docker hubs really are.

The third limiting factor against container technology is that to avoid VM sprawl, a unified image is currently used and maintained all the VM-s in the ESX cluster to keep things easy to manage and up to date. The problem with that is the version of kernel is patched up to all the security threats, but it still does not change the fact that the kernel is not the latest on the market. Thus not supporting container technologies. An updated image is in the working but as the update is a large task, it will take time in a large scale enterprise where each application is business critical.

As the investment in current solution is already done and is working reliably, providing all the required benefits and keeping all the stakeholders content. Then there is no plan with current OS to move to container technologies.

12. Conclusions

To modernize a process that is a common effort of a large group of people in the company, starting from non technical people in the business unit, all the way to developers and hardware administrators. We first need to get an overview of the whole flow of information in the process of creating an application infrastructure. Only understanding and improving agile methods will not give maximum benefits unless infrastructure is modernized too. Modernized not in the sense of switching to virtualization, but in the sense of improving the overall communication and automation.

With more classical VM based infrastructure that has additional complexity of maintaining dozens of VM-s and Operating systems on top of them. With wide selection of applications deployed on them. This is a large effort unimaginable without automation. Proper usage of automation is and will continue to save countless hours of manual labor. With modern large scale enterprises where one application can span dozens of hosts, managing this would be impossible manually.

It is not an option to go on without automation, but if done light heartedly, without having the bigger picture in mind. Blind automation can lead to large segregation, unstandardized scripts and generally an unmaintainable sequence of hacks that keep introducing more hacks to address the corner cases. To avoid this, getting familiar with the general processes surrounding application life cycle, release cycle and application infrastructure building blocks is vital. This has led me to a better understanding of the whole process and the pieces to the automation puzzle.

The multi level shared responsibility infrastructure management model described in the thesis is already giving benefits with more relaxed deployments. Easier to manage and more transparent configuration management with less time spent on individual nodes, doing manual changes. It is far from final and will continue to be developed and improved, but it is certain that automation and configuration management tools are here to stay and the DevOps culture will continue to bring agile methods to infrastructure, with a goal to establish a full framework of easy to follow guidelines from zero to application.

References

[1] The agile system development life cycle. Accessed: 2016-04-25. [Online]. Available: http://www.ambysoft.com/essays/agileLifecycle.html

[2] Going beyond scrum. disciplined agile delivery. Accessed: 2016-05-02. [Online]. Available: https://www.disciplinedagileconsortium.org/resources/documents/beyondscrum.pdf

[3] The bradley bug chart. Accessed: 2016-05-02. [Online]. Available: http://www.scrumcrazy.com/file/view/ScrumBug-613.pdf/347475346/ScrumBug-613.pdf

[4] Declarative vs. imperative models for configuration management: Which is really better? Accessed: 2016-05-02. [Online]. Available: https://www.upguard.com/blog/articles/declarative-vs.-imperative-models-for-configuration-managementf

[5] An overview of chef. Accessed: 2016-05-20. [Online]. Available: https://docs.chef.io/chef_overview.html

[6] Load balancing 101: Nuts and bolts. Accessed: 2016-05-08. [Online]. Available: https://f5.com/Portals/1/Cache/Pdfs/2421/load-balancing-101-nuts-and-bolts-.pdf

[7] Disabling nodes or pool members for maintenance (11.x - 12.x). Accessed: 2016-05-08. [Online]. Available: https://support.f5.com/kb/en-us/solutions/public/13000/300/sol13310.html

[8] Big-ip application security manager. Accessed: 2016-05-19. [Online]. Available: https://f5.com/products/modules/application-security-manager

[9] Existing ansible big-ip modules. Accessed: 2016-05-19. [Online]. Available: https://devcentral.f5.com/Portals/0/Cache/Pdfs/2807/existing-ansible-big-ip-modules.pdf

[10] Job control basics. Accessed: 2016-05-09. [Online]. Available: http://www.gnu.org/software/bash/manual/html_node/Job-Control-Basics.html#Job-Control-Basics

[11] Installation as an init.d service (system v). Accessed: 2016-05-09. [Online]. Available: http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#deployment-service

[12] Designing integrated high quality linux applications. Accessed: 2016-05-09. [Online]. Available: http://www.tldp.org/HOWTO/HighQuality-Apps-HOWTO/boot.html#9.4

[13] Supervisor: A process control system. Accessed: 2016-05-09. [Online]. Available: http://supervisord.org/#supervisor-a-process-control-system

[14] Easy, proactive monitoring of processes, programs, files, directories, filesystems and hosts. Accessed: 2016-05-11. [Online]. Available: https://mmonit.com/monit/

[15] Compare tower editions. Accessed: 2016-05-15. [Online]. Available: https://www.ansible.com/tower-features

[16] Using ldap with tower. Accessed: 2016-05-15. [Online]. Available: http://docs.ansible.com/ansible-tower/2.2.0/html/administration/using_ldap.html

[17] Ansible plugin. Accessed: 2016-05-15. [Online]. Available: https://wiki.jenkins-ci.org/display/JENKINS/Ansible+Plugin

[18] Documenting jello: How we automated our infrastructure documentation. Accessed: 2016-05-17. [Online]. Available: http://thecache.trov.com/documenting-jello-how-we-automated-our-infrastructure-documentation/

[19] Elasticsearch architectural overview. Accessed: 2016-05-17. [Online]. Available: https://buildingvts.com/elasticsearch-architectural-overview-a35d3910e515#.od18ycj2y

[20] Apache lucene core. Accessed: 2016-05-17. [Online]. Available: https://lucene.apache.org/core/

[21] Python elasticsearch client. Accessed: 2016-05-17. [Online]. Available: https://elasticsearch-py.readthedocs.io/en/master/

[22] Elasticsearch::api. Accessed: 2016-05-17. [Online]. Available: https://github.com/elastic/elasticsearch-ruby/tree/master/elasticsearch-api

[23] An elasticsearch development workflow with curl and bash. Accessed: 2016-05-17. [Online]. Available: http://asquera.de/development/2013/07/10/an-elasticsearch-workflow/

[24] Elasticsearch.js - javascript api. Accessed: 2016-05-18. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/client/javascript-api/current/index.html

[25] How to use kibana dashboards and visualizations. Accessed: 2016-05-18. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-use-kibana-dashboards-and-visualizations

[26] A computer weekly buyer's guide to containers and microservices. Accessed: 2016-05-24. [Online]. Available: http://www.computerweekly.com/ehandbook/A-Computer-Weekly-buyers-guide-to-containers-microservices

[27] The twelve-factor app. Accessed: 2016-05-24. [Online]. Available: http://12factor.net

[28] Easy docker deployments with mesosphere dcos on azure. Accessed: 2016-05-24. [Online]. Available: http://www.slideshare.net/mesosphere/easy-docker-deployments-with-mesosphere-dcos-on-azure-59961329

[29] Write once, debug everywhere. Accessed: 2016-05-24. [Online]. Available: http://electronicdesign.com/embedded/write-once-debug-everywhere

[30] The ugly side of container technology. Accessed: 2016-05-24. [Online]. Available: http://searchsoa.techtarget.com/blog/SOA-Talk/The-ugly-side-of-container-technology

[31] Virtualization sprawl (vm sprawl). Accessed: 2016-05-24. [Online]. Available: http://whatis.techtarget.com/definition/virtualization-sprawl-virtual-server-sprawl