UiO : Department of Informatics
University of Oslo

Audition: a DevOps-oriented quality control and testing framework for cloud environments

Gaute Borgenholt
master thesis spring 2013

Audition: a DevOps-oriented quality control and testing framework for cloud environments

Gaute Borgenholt

23rd May 2013

Abstract

The primary goal of this thesis is to discover a new method to quality ensure, test and select the ideal virtual machine based on performance quality and price for arbitrarily large setups, modeled into a concept that is simple to understand and convey. This thesis demonstrates how theater production processes can be used to reduce conceptual complexity in automated release management for web environments. The prototype was demonstrated on two cases, one wordpress blog comprising of a database and server and one of a multi-tiered web-based application with five separated layers of operation. A fully operational extendiable framework, with a focus on simplicity is presented as a solution.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I would like to express my sincere appreciation to my supervisor, Kyrre Begnum. Without his suggestions and criticism, in combination with dedication and enthusiasm. This thesis would not have reached it's potential. His guidance helped form this project, both the creative vision but also made the work interesting and fun. I can not express with words how much i appreciate the work you have put into this thesis.

Secondly I would like to thank Hanne-Marte Sørlie for the introduction into theater. Without her knowledge in theater, I would not have been able to obtain the knowledge needed for writing this thesis.

I would also like to extend my gratitude to my family, for the support and understanding during this thesis. Helping me in what ever way they could and doing so continuously during the whole thesis.

Chapter 1

Introduction

Today, software services are transitioning from local services to web based services. This change enables software not to be bound by any platform dependencies and the service can be used from tablets, phones and computers anywhere. This transition is in line with the users new needs, to have access everywhere, on any device, at all hours of the day.

This transition makes the services more complex to manage in addition to be more time consuming for the system administrator. With the software and information being centralized instead of locally on each device, the complexity of a system increases. In addition to system complexity, the environment expands and becoming more complex. With the services becoming centralized, the workload for the system administrators increases. Adding more task to the system administrator without adding human resources, which often are the limitation.

Challenges with web services in a cloud environment

The challenges with web services are the uptime requirements and consistency of service performance. Users expect services to have a high consistency in service quality [30]. Furthermore, the service needs to be available 24/7. By centralizing the software, any service interruptions affect all users. Furthermore, any change to the software is extremely time consuming, with planing, testing in multiple environments, test implementation, then production implementation.

Users expect frequently releases of new features

Users expect new features released regularly and frequently, with no degradation in service quality. This request for continuous delivery of new features, have lead to the creation of DevOps. A development paradigm with a focus on automation and continuous deliveries. With all the daily system administrating tasks and all the steps in maintaining a service, the system administrator in addition have to release new features with shorter

and shorter intervals to attract new users and keep the current. The expectation of new features and service quality, increases the work load on the system administrators. Humans are not able to work day and night, therefor humans do not scale along with computer systems. Thus, release management have become one of the major pains but also a solution for system administrators managing highly complex service architecture coupled with extreme uptime requirements.

Scalability through automation

Automation is a common approach [10] used to easily deploy software upgrades quickly. Anderson conclude that automation is commonly used in the field of system administration [5] [6]. Automation is scalable and heavily used in large computer environments. Through automation, one can potentially save human resources as the number of servers increases, and may help limit human errors in critical stages of system deployment. Scalability, can be a wanted side effect of automation.

Automations approaches exist today in software engineering concepts such as continuous delivery [40] and continuous integration [62]. However, the actual technical implementation or a framework are not defined, leaving the sysadmin to develop complex local tools based on only process description. Furthermore, the concepts only deal with automated component testing and deployment but do not cover other vital concepts such as fault recovery and service optimization.

This results in locally created tools which are complex, both from a technical view and human perspective. When combining a locally developed testing tool with a complex management tool, the end product will understandably be complex to understand and maintain. In addition, since all tools will be locally developed, it becomes hard to increase the common body of knowledge and for a system administrators across organization to share and compare concepts in order to improve our field. A high level and holistic approach to release management could enable system administrators to better understand and develop local solutions with a more evolved architectural design.

Cloud computing changed how we think about hardware

With the introduction of cloud computing, the playing field of system administration changed. The tradition have been to own hardware, with the introduction of cloud a new approach to system administration where released, rent hardware on an hourly basis. This opens up for not planning systems for years ahead, instead letting the system evolve with the users load on the environment. However, there is no established framework that

is embracing this opportunity.

For example, imagine a system administrator using Amazon EC2 as the base for running the web environment. Amazon EC2 provides the option to use a standard installation of any operating system or any AWS image from the Amazon AWS store. An AWS image is an image of a operating system, optimized for a service or a tool. In the Amazon AWS store there is a huge amount of images, from basic Linux or Windows images to optimized web servers.

When the system administrator is planning to upgrading of a web environment, what is the best AWS image to select? Will an AWS image optimized in the AWS store be the best selection and will the AWS image provide the necessary services to enable the server to be implemented into the environment. A system administrator does not have time to test all the available images in the AWS store because of the huge number of images available and the time the system administrator has available. There is also the possibility that a basic image will provide a better overall performance than AWS store images, for a lower cost? How can a system administrator make the bast selection without adding complexity and more work for the system administrator?

Searching for knowledge outside the field of computer science

By mimicking other fields of science, one has in the past been successful in simplifying computer processes. Examples of this is the use of biological methods to explain behavior in system administration. Cohen [27] introduced the term *virus* into computing, Finstadsveen [37] [49] used animal behavior to broaden the field of system administration thinking and Watson used the human nervous system to explain autonomic systems [51].

Using known terms from other areas is not only to simplify the explanation. System administration is a new field of research. To implement known processes from a older more evolved fields can be beneficial and explore new areas within the field of system administration. Simplification, could be a positive side effect of inheriting from more established fields of research.

Processes from the theater could help to simplify and explain the processes of automated upgrades and testing, as previous mentioned in the example. Theatrical processes have similarities with the different stages of testing and implementation in system administration, using these known terms can simplify the methods and help with understanding the technical solutions. To develop and maintain systems is expensive, therefore combining known tools, with a new interface, will shift the software updating and bug fixing to the software publisher, and make the system more sustainable for future use.

The importance of this approach is to address the problem of unnecessary complexity in upgrading large web based applications. Release management has become more important with time, based on the demand for frequent updates. Solutions existing today focus on single objects or single problems, like user friendliness, simplicity, automations, testing or deployment. All the different tools add to the complexity of the system and creates more layers to understand for the system administrators. The need for a less complex solution is apparent. Using methods and expressions from the acting industry in system administration, is a new and interesting way of thinking and could help solve this growing problem.

1.1 Problem statement

Q1 - How can automated release management simplify the administration of large scale web based applications through mimicking processes from the acting industry.

Acting industry covers the entire field of acting, from theater to movies. The definition of acting differs between fields within the acting industry. Spolin, define acting as: "Avoiding (resisting) focus by hiding behind a character; subjective manipulation of the art form; using character or emotion to avoid contact with the theater reality; mirroring oneself; a wall between players." [82] Spolin, where one of the the major contributors to modern improvisation.

A more classical definition of acting is: Acting is the visible and properly "scenic" part of the performance, through which the spectator receives the whole of the event with the for of it enunciation. [74]

In this thesis acting industry will be limited to the theatrical part of the industry. Examples of the theatrical industry can be the process of hiring actors or actresses using the casting call process or an audition to see how the actor perform. The limitation will furthermore not include people or the actors aspect of the profession, the focus is on the processes within the industry. The same processes in acting industry have been used for generations and by mimicking them, one hope to inherit the simplicity and experience into system administration.

Automated release management is a processes combining of different elements. Automated means to convert to an automatic operation, to make the process operating without the need for intervention. Release management is the process to managing software releases effectively, and is a function which are scalable and used in large system administration environ-

ments .

The initial setup is the information needed before the upgrade process start, in order for the upgrade process to be successfully completed. The system is limited to a working concept, security, reliability and performance will not be the focus of this thesis.

Large scale system consists of multiple devices for different services. Examples of this can be a layered network configuration, with multiple webservers, databases, load balancers, caching servers and storage servers. All the different elements creates a more complicated system architecture. The infrastructure have to be large enough to benefit from automation of testing and implementation processes. The limitation for this project will be an environment consisting of a few servers, this will be enough to draw a conclusion based on the results.

Web based applications is defined by Conallen as: "A web system (web server, network, HTTP, browser) where user input (navigation and data input) affects the state of the business [28]". The W3 organization defines it as: "A Web service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via internet-based protocols [89]"

Within web applications there are different technologies and programming languages, the limitation will be to focus on one technology or one basic web solution to prove if the concept is functional and can benefit from the different processes inherited from the acting industry.

Mimicking is to reuse concepts and behavior in hope of inheriting the properties of quality ensuring and optimization which are present in their approach. The limitation will be the human aspect within theater, people are more complex then computers. A human person can perform certain actions, which a computer can not. Therefore the mimicking limitation will be the computer and to what extent it can mimic processes created for a human.

Chapter 2

Background

On the surface this paper attempts to combine two completely different professions or areas of knowledge, the background chapter will further explain the reasons behind this concept. In the field of system administration, the focus will be on automation of release management in addition to quality assurance. Further, mimicking processes from the acting industry in the hope to inherit best practices from an older and more evolved profession.

2.1 Web

The World wide web or internet is today used by people of all ages, from children to seniors. In the past to get on the internet, one had to use the computer in addition to a dial-up connection. Today, users are always online, using a phone, tablet, laptop or computer. With this explosion in usage, the quality insurance and consistency in quality are becoming extremely important.

2.1.1 An Introduction to the web

World wide web were started in the late 60´s as an experimental network called arpanet, and was intended to enable faster communication using less reliable network components [31]. The initial connection were between research laboratories and schools [34]. The reason behind the network was to share what was at that time, expensive resources: processing time and storage, between the different institutions. After a couple of years, new features were designed and implemented, and within 3 years e-mail was the resource using most of the networks resources [44]. This was the first shift in usage and already then the users had power to change the direction of usage based on need.

With the growth in e-mail usage, ARPANET grew in size and in 1989, Berners-Lee from CERN came up with a proposal. This proposal developed into what today is commonly known as the world wide web or internet

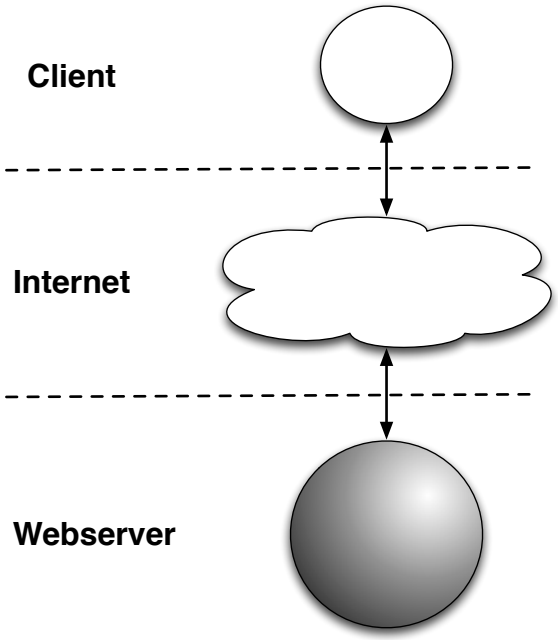

Figure 2.1: Single web server, could have an internal database.

[18]. Internet today has become a tremendous source of information, however, its usage has expanded to areas in all fields. Cline wrote in 2001 about people searching for health information online [26], Fox on online banking [41] to online gaming by Tyvand [87].

The web has become a hybrid, between communication and entertainment [67]. The web is no longer a service available for a few system administrators or researchers, but a ecosystem of services used all around the world. In addition users have different experience and knowledge in the field of computer science. The basic idea from the ARPANET has not changed in 4 decades. The foundation of the internet is still to enable faster communication over unreliable network components.

2.1.2 Web architecture for large sites

In web architecture there are differences in components needed, between small and large service architectures. When running a small site, a web server combined with a single database server can be adequate. In the past this was enough to handle the majority of the users. However, with the growth in users, the different components has more complexity and each service have become more specified.

Figure 2.1 displays how a web environment was configured in the beginning period of the internet.

Today, a web architecture can become extremely complex. The overall goal is the same as before. However, a task is split into multiple tasks or services to enhance performance. A large web architecture today consists of multiple complex components, as demonstrated by figure 2.2. The multi layer web architecture enhances the performance and enables higher numbers of users at the cost of complexity.

Figure 2.2 demonstrates the different layers in todays web environment configuration. Services have been split into layers to enable for higher performance.

2.2 Direction of usage

The usage of the web has been changing since the beginning of ARPANET 2.1.1, and is still continuous changing. The trends today moving towards cloud computing [8] and mobile devices [54], including portable computers, tablets and phones. This change in information availability is a result of the information being available from the internet and not limited to a local device. This has enabled information being available everywhere at any time. The only limitation is that the device used needs an active connection to the internet.

This shift in usage also changes the job of a system administrator. The old model where the user updates most of the software himself locally is not valid anymore. With the program or information being centralize, the sysadmin have to do all the updates for the system. The problem with this is the explosion in mobile devices and online users. The effect of system down time is critical, thus it is extremely important to limit or in theory eliminate all down time.

2.3 Saving time with Automation

Automation is a tool used to save time, resources, and eliminate errors if the same process have to be completed on multiple units. Automation is not a term limited to system administration, different areas have found the benefits of automation. Manufacturers early found out that if a process or unit have to be manufactured exactly the same each time, an automated industrial robot can be used [43] [19]. This automation revolutionized the manufacturing industry, both cost and production time went down [19]. The automations in system administration is not a new topic, its has been a topic for decades [22] [61]. During the years automation in system administration have moved from a hot topic, to become a best practice.

Automation can be used for all system solution and sizes. However the larger the system is, higher rewards or advantages are received from

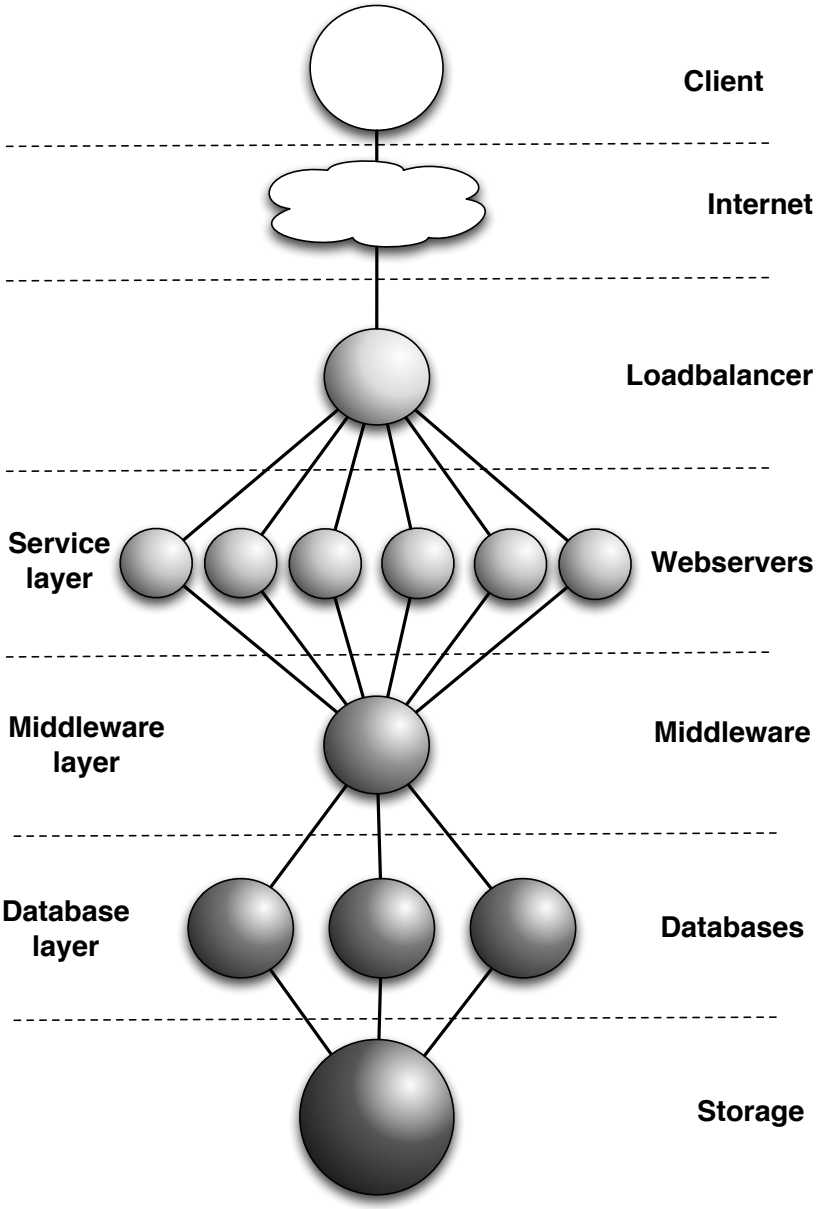

Figure 2.2: How a web environment looks now.

automation, this is because of scalability. With automation the work consumption is mainly at the first server or first system to be automated, after this there is little work with adding new servers with the same service configuration.

There is different tools or solutions that deals with automation in the field of system administration [94] [45] [73].The different tools have differences [70], but they all deal with the topic of automation within system administration. The problem with automations tools is that they provide a framework, how one use it is up to each department or each system administrator. This is why each automated solution is unique , and often complicated or undocumented. Benefits with standardization is documentation, information about errors and easier to learn. Research has shown that standardization do not destroy creativity [35].

2.4 Learning from different fields

System administration or computer science is a relatively new field of research, and because of this has changed radically over the last decades. With these changes, it is important not to be focused on one technology, but the process itself. Technology changes direction after trends, but the process overall is stable over time. An example of this could be if one write about upgrading a specific web technology, apache, it would be outdated fast. However if one talk about the general process in upgrading web services it will be long lasting. With mimicking different fields of research one hope to inherit best practices, which have evolved over a longer time period then system administration.

Examples of borrowing terms from other fields is when Cohen [27] first used the expression virus to describe a program or feature attacking a system, in the field of system administration. With this Cohen managed to simplify the explanation of a complicated computer term, using a well known biometrical term, virus. Cohen proved that combining two different fields of research can be done, and with success.

Finstadsveen used the field of ethology [37] in order to discover and learn if the animal behavior can help with autonomically systems, within system administration. Finstadsveen simulating a herd using virtual computers, and how a herd behaved in its natural environment. He manages to create a more sustainable system environment, and in the process inhered a way to express and explain the complex environment easier to both system administrator, but also people out side of the computer profession. Finstadsveen used the field of ethology to help explain autonomic processes, Watson used biometrics or the human nervesystem to explain automatic systems [51], this was not to inherit but to simplify the explanation. Two completely different fields of research have been used to explain autonomic

within system administration, and with this approach broadened the field of computer science.

All the papers have in common that the overall process is similar in both fields, and the terms used are well known. The authors describes the general purpose or process, and is therefore sustainable over time and trends within computers and system administration.

2.5 DevOps

DevOps is a software development method, designed to help organizations to rapidly produce services. Devops helps to create collaboration between development and system administration, with the collaboration help to cut down on time between development and implementation. The popularity of DevOps have grown with the recent change in technology, since users expect new features released in a frequent paste.

DevOps is a relative new method, however, there are multiple book about the usage and best practices [78] [2]. DevOps adopted the agile development method [47], which is designed to encourage for fast response to changes. Agile development is about breaking up tasks, so that each task require little planing and by that require no long time planing. The tasks are called increments, and the iteration or time frame for each tasks is between one week up to a month. The goal of agile development is to have a release after each iteration.

The success of DevOps is partly linked to a release and configuration management. With an increase in releases the success depends on having a good automated process for releases, and thus can manage the increase in releases without wast increase in time used by system administrators.

2.6 Software testing

Software testing is not a new term in system administration, Turings article in 1950 [86] started the automation of system testing. In time the topic became popular and grow over time [32] [33]. Software testing is a broad term, it deals with both black [16] and white [72] box testing, simulation testing [66] and implementation [85]. However, system testing is also divided into smaller groups based on testing methods and what it tests for [64].

2.6.1 Performance testing

In performance testing [88] the goal is to test the efficiency or performance of an object, this object can be a whole system or a single program or pro-

cess. The objective of a test can be response time, throughput, workload or general performance. The test can be set up in an order that the object will not satisfy its requirements, or performance objectives [64]. However the test can be set up to test that the object or system will have the needed performance based on a performance test, in way the test will not be configured to fail but to have the minimum performance configured and see if the object handles the workload given within the given time frame.

2.6.2 System testing

Software testing is testing the functionality of an object or objects and that the objects actions is according to the wanted result of the given action or input. This is also quality assurance, and test all features and report on errors discovered but wrong input-output result. The data used in the test phase depends on the object used for testing, and the objects functionality. The goal is to discover bugs or errors in the system, errors created by combining multiple systems, software changes, system upgrades or installation errors.

2.7 Related work

This thesis involves several different areas of computer science. The different areas are highly relevant in the direction which system administration are shifting towards. Because they are highly relevant it is much ongoing and completed research done, within these areas. However, although there have been research done in the different areas, there is little to non when combining the areas together.

2.7.1 Automated software testing

Automated software testing is a well established field, discussed in section 2.6. There are different approaches on how to test a system, and how to measure the success or failure of that system. Research has been done with the combination of cloud computing and testing, such as cloud9 [25] and $TaaS_D$ [23]. Both tools using Amazon EC2 cloud solution for the implementation and testing.

Within more tradition used tools, both httperf [63] and ApacheBench [21] can be fully automated and have been used with success. Both tools are attended for web testing, however, the tool is not aware if the web server tested is running in a cloud or on local hardware.

2.7.2 Capacity planing and resource management

Red hat have released a solution named Red Hat CloudForms [75], a solution for capacity planning and resource management for virtual infrastruc-

tures. CloudForms is an IaaS designed framework with a goal to lower costs, and increase service levels, using new, virtual-aware techniques. The solution can be used across multiple cloud technologies, including Amazon.

There are similarities with CloudForms and the intended framework created in this thesis. However, there are differences between the two solutions. The design and framework is openly available for using and further development. The tool used in the framework is opensource and is free to use, therefor no cost of using the framework. CloudForms is not available for free and is limited to certain types of hardware [76]. The framework created in the thesis is fully adoptable for all platforms and hardware types, therefore in not limited to certain standardized images.

The thesis framework uses testing to optimize the network based on price, meaning getting the lowest price for the environment which meet the minimum requirements. After selecting the environment, the framework leaves the environment to it self and the management tool. There is no monitoring or environment adjustments after the Audition.

CloudForms do not perform benchmark evaluation on the system specification in the creation phase, it monitors the network and evolves the environment live after usage. This reflects that the two solutions are not direct competitors, since they have different main tasks. Instead they can be used to complement each other. The thesis framework can automatically test and design the wanted environment, to the lowest price, before CloudForms take over, create and monitor the environment.

2.7.3 The usage of analogies

The usage of analogies were discusses in section 2.4, however it has been used in other fields than ethology and biology. David Blank-Edelman have used different fields to explain system administration tasks or problems [**edelmanurl**]. Using topics like cooking at the keyboard, where writing cook books are compared to configuration management. Preparing and having all in place is a key element in both cooking and system administration.

Chapter 3

Approach

Every system administrator has their own approach on how to determine what hardware to use, if the upgrade will be successful and how the newly upgraded software will handle a normal user load. In order to handle this there is a need for a system or framework, that will automatically deal with the problems. In order to draw parallels or inherit best processes from the field of theater, one has to have enough understanding or information about the area. To best facilitate a successful outcome from combining theater with system administration, it is important to have a deep understanding of both.

The approach chapter will explain how the action taken will help with answering the problem statement: **How can automated release management simplify the administration of large scale web based applications through mimicking processes for the acting industry.** Within the problem statement there are three key properties: A reduction in complexity (K1), the complexity is defined from a system administrator view. A working framework(K2), based on processes or best practices from the acting industry. Find out which processes can be inherited with success from the acting industry. Service optimization (K3), how optimized is the current system when comparing cost versus performance.

The design will be a combination of theater and system administration with a focus on the key properties, based on the design, a framework will be created. A subset of the framework will be implemented, resulting prototype will help answering K2.

3.1 Learning about the art of acting

The processes around acting, hiring or production processes when setting up a play is documented and known in the field of theatric. The processes are based on best practices over centuries. To gather knowledge in the field of theatric one could read books, published papers or interview professionals within the field. Reading books or papers could lead to misunderstandings or misinterpretations, since the acting field is extremely creative and fluctuating. In order to get background information and an overall view of the acting industry, the information will be collected through interviews. Based on the information retried, books or papers will support the findings or supplement with more information.

3.2 Design phase

To better understand the complex system architecture of automating release management, a structured method for displaying the design is needed. A model can provide an overview of all functionality and in addition break down the different modules which help with the simplification in understanding the framework.

3.2.1 Modeling

To design and build the new framework, a modeling approach is a valid option based on its features and strength. The modeling approach will enable an extensive overview of all processes, modules and tasks within the design. Understanding the limitations of a modeling approach is important, more advanced decision making or choice scenarios can be modeled with success, however it could become to extensive and thereby becoming less understanding if modeled graphically.

The decision on what modeling language to use, is an important factor in both designing the architecture but furthermore the explanation and understanding of the design. A modeling language could, if used properly, help with simplifying the explanation of the software design, and help with answering K3.

A graphical model has the strength of visually displaying a context. Be that a context between modules or context between processes. This method enables people to examine and understand the design, without being dependent on a programming background, the capability to read code or understanding non visual explanations. Within graphical modeling, there are different languages. Unified modeling language (UML) is one example [39]. Unified modeling language incorporates notation and has inhered some best practices with using formalization [42]. However UML has been reported to have inconsistencies, incompleteness and being

ambiguous [77] [36].

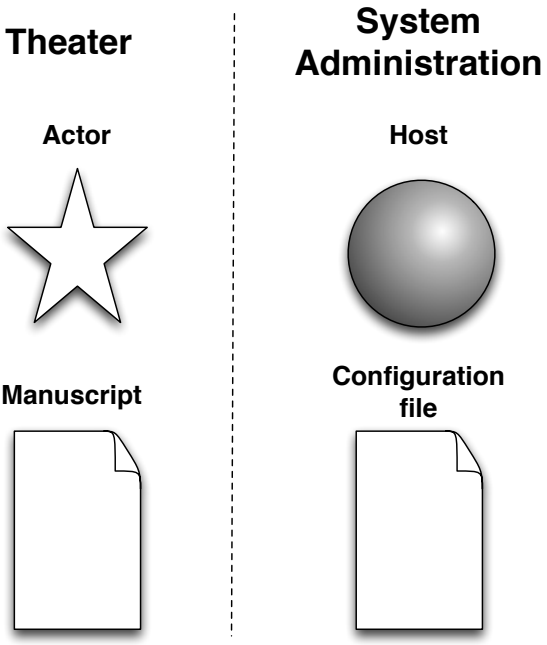

Figure 3.1: A standardized graphical design formalization.

Figure 3.1 displays the design formalization and the differences in design between system administration and theater.

Pseudocode

An example of a non graphical model is pseudocode. Pseudo-code is a high-level description of the process or principle cite . Pseudocode has inherited the conventions of programming languages. Therefore, will pseudo-code look like programming code. The difference is that code is meant for interpretation by a computer, pseudocode is meant to be read and interpreted by humans. Pseudo-code has been proven successful in explaining new approaches [38], but the limitations is that a person with the understanding of coding will read and easier understand pseudocode then a person with limited code experience.

Pseudocode can complement diagrams or graphical model languages, help with designing modules and can easily be converted into source code [56]. The following example is a pseudocode where the score in a benchmark test has to be grater than 30 to be valid and if not stop the process.

Psudocode - Display a benchmark validation example

```
IF computer benchmark score greater then 30
```

```
  THEN
    Accept
      Report success
    END
ELSE
  Stop process and report "low benchmark score"
```

The pseudocode in the example helps explaining a process. To explain the same with words will be more complicated and could leave room for different interpretations. Furthermore, a diagram would work but use more space and with more advanced processes it could be unnecessary complicated. Models will be used to display context or how different modules is combined and interact with each other, it will be used as a higher view of the system, a higher view then pseudo-code.

3.3 Implementation phase

The designed framework will be the blueprint for the implementation. The implementation will however not be a fully featured framework with all the designed features implemented. Due to the time limitation of this project the basic modules will be implemented in order to create a operational prototype, the prototype will be used for answering K2.

3.3.1 Environment

To create a solution and answer the problem statement one has the option to do a simulation or an implementation of the design in a real environment. With an implementation in a real environment, the system is vulnerable for external and internal noise. Two similar tests could have minor disruptions in the result because of noise, and for this reason a perfect recreation of result may not be possible for some scenarios. The scenarios could be with a benchmark of cpu using time pr compressed file, the result in two experiments could have a very small time difference because of internal noise. To minimize the risk of disruptions in result, the system needs to limit the possibility for noise occurring, internally and externally.

With a real-life implementation a noise free environment is almost impossible, if not impossible. It is still possible to test the framework in a noise free environment, but there will be a need for simulation. A simulation of the system will be without noise both internally and externally. This utopian system is a perfect environment, but could be considered to perfect or unnatural.

In order to answer both P1 and P2 an implementation in a real life environment will be beneficial, the noise is part of a system administrators

life. A system functional in utopia might not work as well under normal environments, and normal environment will provide the best results and answer both P1 and P3.

Local hardware or cloud

Where or on what, should the environment be tested on. To run the system on local hardware for testing reasons could be costly, if the hardware is not already available and need to be purchased. It is also time consuming to install new hardware and it is not flexible when it comes to testing with different hardware configurations. The flexibility problem can be solved with local virtualization or setting up a local cloud service on the available hardware, this will enable for creating a testing environment with multiple computer options and can be expanded if more hardware added.

The down side that is a local cloud will create more work, and potentially take time away from creating a framework to deal with problems regarding the setup and maintenance of the local cloud. With virtualization also the bandwidth between different components and the usage of component have to be considered, if the hosts uses more resources then available it will result in the system having to wait for resources. This waiting will have a degrading effect on the performance for each hosts running in the environment, therefore an increase in virtual hardware specifications for a host might not lead to a increase in performance because the system is already overloaded and therefore the hosts fight for the resources available.

A public cloud will provide the flexibility and at a low cost when testing the framework. The problem with public clouds is the unstable performance and the variety in performance even between virtual computers with the same hardware specifications. The performance problem is based on the same reasons as for a local cloud. The hosts fight for the resources and when multiple host peeks in performance there might not be enough available resources to handle the spike in performance. A public cloud will enable flexible environments, when it comes to hardware specifications but also number computers in an environment. The configuration and setup of an environment is completed after seconds and there is no time consuming maintenance of the cloud service.

Both localized cloud and public cloud is a hot topic, it is where the shift in technology are moving and have been moving for a while. Testing the design on a environment used today but also an environment for the future, will help displaying the importance of this work.

The limitations of localized hardware both running the environment on the physical hardware and in combination with a local cloud, the disadvantages are to great. The advantages for testing the framework in a public cloud are greater then the disadvantages, therefor the framework will be implemented in a public cloud based environment.

Public cloud

With cloud services being a hot topic, there are a lot of distributers of cloud services. The difference between the distributers are mainly price and SLA or terms. A short list of a few of the the larger distributers are, Amazon[91] [68], Rackspace [59] [1], Verizon [52] [29], IBM [93] [95] and Microsoft [57]. The are some differences in the computer specification for each distributer, but the overall service provided are the same. The design could be tested on the cloud provided from all distributers, however the design will only be tested on one cloud technology.

The Amazon EC2 cloud is the most used cloud service today, Amazon was one of the first to provide cloud services and has for years been the largest or most used. The EC2 have the disadvantage that the virtual host performance depends on where in the world it is located, in which data center and on what rack. However the service are stable both in downtime and in time used to create the virtual host.

3.3.2 Deployment automation

To automatically deploy software or configuration, a configurations management tool needs to be used. There are different configurations managements tools available like, CFEngine, Chef and puppet. The mentioned tools are used by companies of all sizes and have a strong community behind the software published, there are more configurations managements tools available. The differences between the tools are speed, resources used under client configuration and code language.

For this project puppet was chosen, the reason for this was that puppet have all the functions to successfully handle the automation part of the framework. In addition puppet have a strong user community [84] and being the most stable configuration management tool [48].

3.3.3 Understanding puppet

Puppet is a configuration management tool, and is used by the user to describe the wanted state for system resources.

Puppet is build up as a master and slave architecture, and the slaves will inherit the configuration from the master. The connection between the master and slave is restricted using certificates, the configuration inherited is based on host name from slave. Puppet uses modules to distribute configuration that can be used on different hosts and allows for dependencies between configuration. Meaning that a service needs to be installed, before the program is configured and the configuration need to be completed before the service is started.

The slave can be set up to contact the master one time or at scheduled time periods. This allows for the host to get a one time configuration before left alone, or continuously checked for state not wanted. If the situation is an unwanted state, the host is returned to the wanted state.

An example of a puppet manifest can be seen below. The example show how to install "apache" and then control that the service is running.

Puppet manifest

```
1  package { 'apache2':
2          provider=>'apt',
3          ensure=>'installed'
4  }
5
6  service { 'apache2':
7          ensure=>'running'
8  }
```

3.3.4 Deployment of the environment

In order to deploy Amazon EC2 servers with a focus on limiting system administrator interactions, a framework have to used. The option is to build a new framework or reuse an excising framework, with creating a new framework the life expectancy of the framework depending on the creators time available and fulfilling a need in the marked. To keep a opensource framework updated and with new releases the software is often dependent on a community, not only for the coding but also for bug and security reporting. Today there are tool out on the marked that handles the automation of deployment of computers, and for this project the benefits for creating a new tool for this usage is not great enough.

The tool chosen for the deployment automation is MLN or Manage Large Networks [12]. MLN have been used with success against Amazon EC2 in the past [13]. In a teaching environment [15] MLN have proven to reduce the management overhead when deploying virtual computers [14], one of the reasons for this is MLN support a project feature. A project is multiple or single virtual computers in one virtual environment, this project is by default isolated from other projects created however they can be joined together if wanted. This project feature can therefor with a start command build multiple virtual computers at once, and by default creating the virtual computers in a closed environment[14].

3.3.5 Brief introduction to MLN

MLN is a tool used in order to simply the deployment of environments with multiple virtual computers. The tool uses different virtualization technologies existing today and enables for the unique feature of arranging

virtual computer into projects. Each project can be controlled as one unit or the different virtual computers under each project can be individually controlled. MLN is built using perl, and the information used to create the environment is located in MLN configuration files. MLN uses the configuration file when creating the project, the project needs first to be built before it can be started or stopped. Under the build stage, two scripts are created for each host, start and stop scripts based on the information given in the configuration file. The MLN configuration file is built on keywords and blocks containing the configuration

MLN - Basic MLN configuration

```
1  global {
2         project mln
3  }
4
5  superclass basic {
6  }
7
8  host mln_example {
9     superclass basic
10 }
```

The example code here display a project called mln, the project has one host mln_example. The host will have all the configuration stated in the basic class, since the host is connected to the superclass. However the host will also be configured with the configuration specified under the host. The use of superclasses is beneficial when multiple hosts need to have similar configuration, examples of this could be a program to be installed, hostname needed to be set using a host variable. A superclass can be connected to another superclass.

MLN - Usage of superclasses

```
1
2   superclass basic {
3   }
4
5   superclass basic_and_more {
6      superclass basic
7   }
8
9   host mln_example {
10     superclass basic_and_more
11  }
12
```

In this example the host mln_example will get all the configuration defined in the superclass basic_and_more, but also from the superclass

basic. MLN has some built in features, the ability to run commands as root after a host is started.

MLN - Amazon EC2 and MLN

```
1  host mln_example {
2     ec2 {
3        use_file {
4            apt-get update
5        }
6     }
7  }
```

The use_file feature will run the command apt-get update on the virtual host after the host is successfully started. Use_file will run each line within its block, the commands will handled by the system running on the virtual host and not by the tool MLN. MLN supports the usage of plugins. The plugins can be downloaded from the web page or be created. The usage of plugins after being installed, is the same as for built in features.

MLN - Puppet and MLN

```
1  host mln_example {
2     puppet {
3       include {
4          mln_web
5       }
6     }
7  }
```

The example display the usage of the puppet plugin to MLN. This will create a host file on the puppet master with the install information regarding the mln_web puppet class. It will not connect the host to puppet, this has to be done using the use_file feature. The mln_web information has to be created in advance, as one normally would do in puppet before connecting a host to the puppet configuration.

3.3.6 Creating or reusing benchmark tools

For the different servers in the environment, there have to different solutions for benchmarking. A web server and a database server have different services running and therefor the benchmarking configuration is different between the two servers. An option is to create a new tool. Creating a new tool enables for optimizing the tool specifically for the tasks needed for the benchmarking. This could eliminate some complexity in the solution, however it could also lead to less viability for the framework. Creating a new tool is time consuming and for the system to be stable over time patches and security updates have to be released, without this the tool created will

be outdated.

Instead of creating a new tool from scratch an existing open source [58] [92] tool can be forked[53] and used as a base for further development or continue the development of the existing open source tool. This method will enable for customization of the tool and the development is less time consuming than creating a new. However time is spent on reading and understanding the code, to little understanding of features could lead to a unwanted result [55].

Using an existing tool eliminates development time. Instead the time is used to understand the tool and the command line interface. Using an existing tool eliminates future time spent on patches and security updates and therefor the system has a greater chance of survival. However the system have to be used as is, therefore no specification of the software. However since performance testing of web environments have been done for decades there are tools [63] [65] available that fulfills the requirements of performance benchmarking of web environments. Based on the elimination of development, patching, security updates and that the performance benchmarking is not the unique factor of the framework, reusing of existing software was the chosen method.

3.4 Appraising properties

To measure or determine the level of achievements of the key properties there is a need to define the terms *simpler*, *optimized* or *working*. There are several different definitions of each of the terms therefor the definitions can vary between two persons, this thesis will have the following definitions of the terms.

3.4.1 A reduction in complexity

A reduction in complexity means that the current solution (solution A) is more complex the thesis (Solution B), but how is reduction in complexity measured? Counting the steps or work from start using a solution until a working solution is created or creating a solution where less experienced system administrators can use the solution. Less technical skills are required. Solution A and B could be used and implemented by multiple system administrators and the reduction of complexity question is solved using a questionnaire based on the experience of the two solutions.

Complexity can be measured in the simplicity of the language the system administrator has to know to use the framework, however this is individual for each system administrator. How technical the framework itself is, number of features available for solution A or B. For this thesis a reduction in complexity will be based on the number of steps or words written by a system administrator, in order to have a working system. This

approach was chosen based on that a questionnaire analysis would take to long to complete, a step count will be fair for both solutions, furthermore steps are often an indicator of the work spent on configuring a solution.

3.4.2 A working framework

A working framework in this scenario is a framework that, with the use of automation completes a deployment or a deployment test of a web environment. The deployment do not have to be completed if the performance or the personal test is not fulfilled, in this scenario the environment will not be built. The framework has to be able to run benchmark tests on the chosen server, with a recommendation of system usage based the results from the performed tests.

3.4.3 Service optimization

Service optimization is a broad term, and the definition variates between system administrators. For this thesis service optimization will be the cost of a server using Amazon EC2 and the servers performance level. The cost of running a server in Amazon EC2 variates depending on computer hardware specifications, backup routines, monitoring options and using a server from AWS store. The mentioned selection are just a small sample of the addition services provided for a small increase in costs, the cost of a server is for each starting hour.

There are different methods for measuring or calculating service performance, different metrics can be collected and based on this the performance level can be calculated. The metrics could be requests handled in a second, response time, CPU or memory used and can be measured on both the client-side and server-side [24]. A different option is defining a wanted performance level, if a system perform equal or better then the wanted performance level the server can be used in the environment. However if the wanted performance level is not achieved then the server can not be used in the environment.

With a combination of cost and performance level, all servers have the wanted performance or better and therefor the selection will then fall on the server with the lowest cost to use. Based on this the servers performance level is a boolean value, if the benchmark value is equal or higher then wanted performance and the boolean value are set to success. However if the benchmark value are less then wanted performance the boolean value is set to failed.

3.5 Expected results

The objective of modeling is to identify processes within the theater, and create a model based upon the processes. Models will describe a new ap-

proach of DevOps in cloud computing in combination with release management. The models designed is without following a known standardization scheme, however, all models will follow the same design concept throughout the modeling and process explanation. When processes is identified and designed they will be linked to system administration.

When linking the two fields together, new elements will be discovered within the field of system administration. The results of modeling will be the foundation of the implementation of a prototype. Modules and features will be created, and if the element is new to system administration a combination of pseudocode and models will explain the element, furthermore, explain the benefits gained in system administration using the new element.

The prototype will be a fully operational prototype, however, with some limitation in features. The prototype will be tested using a realistic example scenario, in a real environment.

Chapter 4

Result 1 - Modeling

Theater have a long history of producing plays with a high quality in performance. They have standardized processes to ensure the best possible outcome. There are some milestones in the production of a play, creation of manuscript, casting call with a following audition, rehearsals and premiere. The different milestones, has different time length in time spent on the milestone.

Designing a complete framework that covers the problem statement is a challenge. A set of models will be created following the time line of theater production:

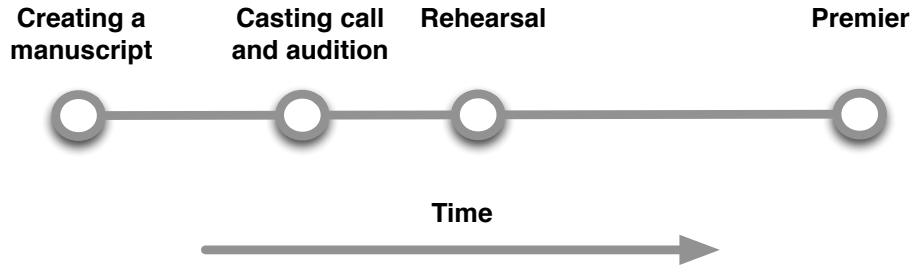

Figure 4.1: Time line of events during the production of a play

The time line displayed in figure 4.1, show the different milestones during the production of a play. The milestones are not equally spread, this is to simulate the different periods have a different time frame.

The processes inherited down from theater are designed into models usable for integration by system administrators. The models will be the foundation for the creation of a prototype, the prototype in turn, will help with answering K2.

4.1 Introduction into theater

The interview object is actor Hanne-Marte Sørlie, the interview will provide information from an actors point of view. With this the goal is to collect information about the processes regarding hiring, quality assurance, names for processes and reasons behind the processes. Collecting this information will help with answering K2, creating a working framework, based on processes or best practices from the acting industry .

Hanne-Marte Sørlie's theater background is from Bårdar academy, and a graduation from Rose Bruford College in England. She has traveled all over Norway with different plays: A Traveling Doll's House, based on texts by Henrik Ibsen. Different plays at Soria under Moria, a theater in Norway. The leading role in "Get set, ready, go", a play based on texts by Alf Prøysen. After years of theater, she shifted to making movies and movie roles. Having the lead role in "Rouquine" and played the role as Therese in the movie "Dunderland".

4.1.1 The art of theater

Theater is an art form with a long background and emphasis on quality. The western form of theater has its roots from Greece, dating back 2500 years [20]. Theater has always been a tool telling a story or delivering a message to an audience. With a cast from one or up to hundreds, delivering a quality performance to tens, thousands, perhaps millions of people at the same time using different medias.

Within the field of theater, there are different groups. The most common or most thought of when using the term theater, is traditional theater. In traditional theater there is a pre-written script, with a set of roles. The actors rehearse and the first official time they perform the whole act on a stage with an audience, is called the premiere. After the premiere, the show continuous to run until taken off the poster.

There are also free groups or alternative theater groups, one of the groups are call fringe. Fringe is often more alternative then normal theater and with this the methods or processes behind a play or performance are a little different. The script might not be pre-written, instead the actors develop the script under rehearsals using improvisation. However, they have an overall idea or concept of where or in what direction the play will go, but the steps or acts to get there are not decided. This process often is without a director and consists of a small group of actors or single individuals performing the play. Fringe plays are often located in smaller theaters or other places with an audience. Some places these small theaters are even called a fringe theater. Fringe theaters may also variate in the number of productions, from one night only to a series of performances.

Before the rehearsals can begin in normal a theater or the fringe group can start building the performance, a cast has to be hired or assembled. In traditional theater the hiring process begins by the producer or director detecting what roles will be possessed by an actor. Based on the role selection the producer or director determine specific requirements for the selected roles. The requirements could be weight, hight, language, dialect of speech or features that will fulfill the roles the director has imagined for the play. After the roles have been decided with specifications a casting call is distributed, this is a message to different agencies or job sites with information about available roles and the specification of each role.

Actors responding to the casting call will be able to perform one or multiple auditions in front of a casting panel. The casting panel may consist of the director, producer, casting director and lead actors. The auditions consists of a monologue or small part of a play given to the actors in advance. This will test the actors ability to perform a rehearsed part and the casting panel can get a feeling of the person and their personality along with acting performance.

The audition could also have a improv part, improv is shorten from improvisation. Improv is a skill highly recognized within the field of theatrics. The skill of improvisation is taught at acting schools as part of the foundation of becoming an actor. Improv will test the actors capability to be creative and act in the moment without a prepared script or monologue.

The audience may not be limited only to the casting panel, it could also be held in front of the competing actors. An audience adds pressure and is one more stressful factor within the audition process. An audition can vary in length, depending on the performance from the actor or number of actors at the audition versus time available for the casting panel. After an audition the actor can get a re-call. A re-call is the casting panel inviting the actor for a new audition. In this process a lack of feedback is a negative response, only the actors wanted for a new audition get notification.

After the auditions, the selected actors are contacted about the acceptance and the possibility for occupying the roles. The two parties now have to come to terms about the contract. The emphasis is on overall quality and obtaining the best actor matching the role, not on economics or overall profits.

The hiring process in a fringe performance will vary, it can be a solo performance and for that reason no hiring process. It could also be announced information about the performance, and people can respond. The differences in hiring processes between the traditional theater and fringe is based on the financial differences between the different groups. In traditional theater the actors normally get a monthly payment, from the same time as rehearsal begin. This payment do not end before the play is taken of the poster. Within fringe performances the work is often not paid, or only

paid after the premier of a performance. This is because the fringe groups are mostly independent, and without finances to support payment under rehearsals.

The difference in economics often leads to persons connected to a fringe group have no alternative motives then the love for the art, they believe in the performance and have a inner drive to express the message, and all the economic reasons are gone. Being a part of a fringe group one gain experience, and one is able to do performances whenever one have some extra time. A fringe performance can often be more idealistic or try different things, be untraditional, because there often are a less financial risk involved, and for that reason can afford to appeal to a smaller crowed when performing.

4.1.2 Improvisation within theater

Improvisation or improv, is basically a performance where there is non or little pre-planning. The field of improvisation is wast, and there are different forms of improvisation and how to perform it. Improvisation within theater is not a newly discovered area, it have been practiced for centuries. The field of improvisation started in the streets, by street performers. With time it evolved in to a genre within theater, with there own acting theory [46]. The different stream of acting focused on using the improvisation as a tool in training in order to become a better actor.

The improvisation used today in modern theater is a product the book Improvisation for the theater [79] and Impro: Improvisation and the theatre [50]. Spolin explored and developed improvisation by creating exorcises and techniques, to evolve as an actor using improvisation as a tool. Jonhnstone, invented a new field in theater, theatersports. Theatersports is a theater competition between different teams or groups, the groups compete in creating a dramatic effect [17]. Theatersports are using improvisation and because of this no competition is similar.

Improvisation groups

Evolving over time, improvisations groups or theater companies have been assembled. From small groups performing on a local stages or bars, to large groups performing in large theaters all around the world. There are different ways of organizing the groups, the groups can be inspiring actors working for the experience and therefor might not get paid, to professional groups with an economical founding.

Ways to use improvisation

Improvisation is not only used for one thing or one way, there are many ways improvisation have been partly of fully implemented within a pro-

cess. There are shows where the cast get information from the audience, and create a performance based on the information received from the audience. They have fully implemented improvisation and only uses this method. This processes have been in the later years adopted by the TV industry, and proved to be usable in other fields then theater.

In some groups improvisation is a tool used to create the plot in the play. The actors know where they want with the plot, but the way there is created using improvisation. This process is a part of the rehearsal stage, and this process is often used for smaller independent groups and one persons groups.

A different way of using improvisation is in form of a tool, a tool in order to get to know the actors and the personalities of the different actors. During an audition the actor in questioning perform a pre-rehearsed part of a play, to show the acting quality but also the full range of skill possessed. The improvisation is used with the purpose of putting the actor under pressure, and see how he or she perform. There might not be a correct respond to the improvisation challenge, more a personality test, see the uniqueness of the actor.

Learning improvisation

Improvisation is a craftsmanship hard to master, it is part on a persons personality, but can be taught. There are multiple book used for teaching and learning improvisation [80] [81] [82]. There are also different educations and courses to attend to learn about improvisation and how to be good at improvisation [71] [11] [83].

Improvisation is an attribute taught at schools with a theater or drama course [83] [71], the theater field sees improvisation as a important part of learning the theater trade.

4.2 Plan before acting

To increase the chance of a positive outcome of any action a plan is first created, based on this plan, actions are taken to achieve the wanted goal. A plan is the foundation which actions are build on, and increasing the possibility for a quality outcome. The plan can have many names. In the car service industry they have a service check list, to ensure that each car gets the same quality of service and to maintain the status of the car. The checklist is there to limit the possibility for the mechanic to forget to do a test. In cooking it is called a recipe, you follow the recipe to create a quality product and to ensure that food taste the same each time it is made. The recipe helps to maintain a stable level of quality and the same quality for all chefs.

4.2.1 Creating a manuscript

In theater this plan is a manuscript. The manuscript contains information critical for the creative performance and contains information critical to maintain a stable quality in performance. Breaking down the information contained within a manuscript, there are different forms of informations. The words spoken by the actors or the dialog between actors, in the manuscript every word the actors speak within the play is written down. The reasons behind this is to ensure the quality and reliability of performance, that it is a consistency in story told from the premiere to the last show. It also enables the possibility for understudies, in theater an understudy is a performer learning a role in a play, but will only perform the role if the actor possessing the role becomes unable to perform.

In order for actors to know what words to say, when and how, roles need to be defined. A role is a fictional person within the play or performance, played by an actor. When the director defines a role, he has a creative picture defined in mind about how the ensemble should look and what it will contain. An ensemble is the full cast of the play. The directors vision defines the features needed to fulfill the creative image, he could define a role to be a young girl with the name of Victoria which speak with a British accent. However, a role could be just a voice, almost like a presenter, heard but never seen in person within the play.

With dialog and roles as elements, a scene can be build. A scene describes actions within a single setting, often built up using three stages, a beginning, middle part and an ending [69]. The manuscript describes a scenario or environments of the scene, the surrounding elements. Furthermore, the state of the roles is described, for example if the role is upset or frightened in the scene. What roles are participating in the scene and if the roles have a dialog or not, the order of dialog and where on stage and how the dialog is also expressed.

A manuscript often have the following layout

Manuscript - The layout

```
Name of play

Describing the environment, stage, props in use.

Roles used in the scene.

First scene

First Actor:
The dialogue line
Actions, mood or other information needed for the scene
```

Table 4.1: Combining theater and System administration

Combination of theater and System administration	
Theater	**System administration**
Manuscript	Script
Role	Service
A Scene	A Scenario

```
Second Actor:
The dialogue line
Actions, mood or other information needed for the scene
```

By looking at system administration, this plan can be recognized not only in general but also the overall processes a manuscript contains. Starting with the concept of a plan. In system administration the plan is a script, this script is not a perl script or an executable script, but a collection of configuration files in combination with program information. The script has information about what services that needs to be installed, and using configuration files the service will be installed and configured after the specifications. This plan, combined with a configuration management tool, can deploy a system and maintain the system over time. The plan describe the wanted services in detail, the services it self but also the configuration of the services.

Combining theater and System administration

```
Name of play

Describing general configuration, build information.

Services needed in the scenario.

First scenario

First host:
The different communications for the host

Second host:
The different communications for the host
```

The manuscript design need to be kept plain and simple, to fulfill the problem statement, that the framework should be simpler to use than similar solutions existing today.

4.3 A casting call

The manuscript contains the creative vision of the director and play writer, and all the elements needed is located within the manuscript except the actors to fulfill the roles. The roles have some predefined character elements within the manuscript, for example if the role figure is a young girl or a young boy. This character element dictates that the actor playing the role is a young looking boy or girl. In addition, the director may have additional features added to the specification to fulfill the creative vision of the play. This could add to the character feature, the young boy will speak English with a foreign accent. When all the features of all the roles in the play are decided, a casting call is distributed. The casting call includes all the feature for each role, the reason for this is if an actor does not posses the features wanted for a specific role and can not in the time available learn the specific task, it will only waste the casting panels and actors time to try out for the part.

The casting panel would get a lot of actors trying out for each role, and this would lead to an overwhelming number of results and an extreme amount of time used by the casting panel. The director knows the elements needed to fulfill the role based on the information in the plan and the creative path, therefore if the key elements are included in the casting call, the number of actors trying out is decreased to only those who posses the wanted set of skills. The first step in eliminating the unwanted or less suitable actors, and the first step in finding the actor that is best fit for the role.

Time saved is not for just one role, but for multiple roles. Since most plays have more then one role, the number of actors trying out for a role almost multiplies for each role in the play. The time saved with some specification for each role is therefore tremendous.

In system administration, traditionally the platform and service selected have often been an unwritten law in each sysadmin department or personal preferences from the responsible sysadmin. The selection has often been done based on experience and what service type is commonly used. For example a combination of debian and apache for the configuration of a web server, based on the sysadmin configuring the system is a linux user and the system administrator normally uses debian, and apache is a web service software which have been popular for many years.

In the past, when an environment was set up, it was planned to run for months, if not for years. The system administrators had to look at the utilization of the current system, predict how much system load the users will use until the system is taken down, meaning the level of expected user load in the future. The hardware was ordered after the specification in user load for the future, high performance is economically expensive. To be more specific, to push technology forward and be at the cutting edge of both

software and hardware have a significant cost. Is it best to buy equipment meant to last 1 year, to a minimum of cost or order a system to last for 4 years to a high cost?

The 3-5 years replacement mentality is still used today among system administrator, however today there is a problem with this line of thinking. In a cloud environment like Amazon, one pay for the hardware in use. Therefore, if one continues to plan for a system to live for years, and calculate the expected usage at the end of the three years. This will lead to the company paying for unused performance for 2 or maybe even 3 years. Instead, if the system administrator calculates the expected user load for next month, then selecting the hardware that best matches the expected performance in a cloud environment, the company will minimize the unnecessary expense of hardware not being fully used.

Cloud environment enables for the usage of preconfigured images or images that have been optimized for running a service at the same time offering the option of basic an images with the operation system of choice installed. In Amazon cloud, the images are located in the AWS marketplace. The marketplace contains a huge database of images with different service running, operating systems, optimizations or tool installed. This enables a system administrator department to not select software based on old practice but on what is the best solution when taking in to consideration price versus performance.

With this huge amount of images available for use combined with frequent environment updates, one can not assess all the images with the given time frame available. This is where theater and the casting call process, can help the system administrator. Based on the manuscript created and the information specified for each role, in sysadmin term, the script with the role being a web or database server. The casting call process will eliminate all the images in AWS store that do not fit the the requirements of the service wanted, by doing that the list of images and hardware available is of a controllable size and the time used in order to test the images is within a reasonable time frame.

Figure 4.2 displays the process in a casting call. Looking at the model, the process starts with the information located in the script, it sorts out the specified requirements regarding the its role or task, if its a web server or database server. The controller then sends information to the AWS store about the requirements needed, AWS store sends back a list over all images that match the requirements.

The controller then reads the list, to control that the content received from AWS store is valid and contains at least one image. If the image list is not valid the process will halt, and the requirements list needs to be reconfigured or AWS store can not be used to build the wanted environment. If the information received from AWS store is valid, the

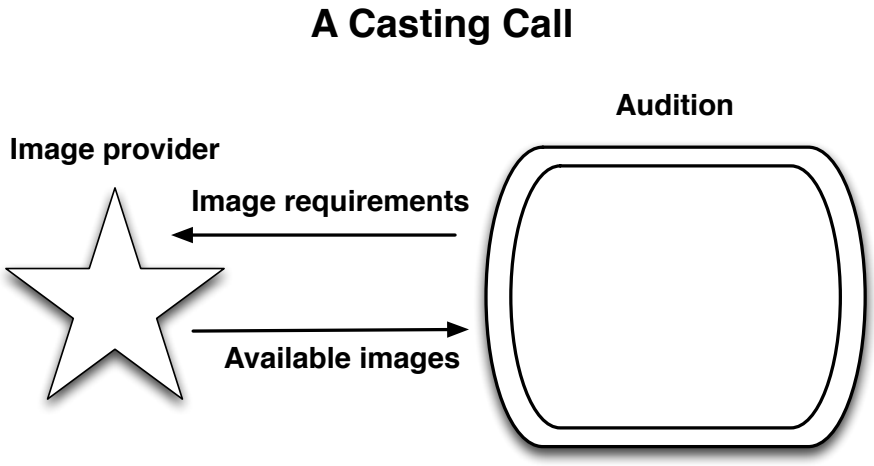

Figure 4.2: Image selection based on predefined requirements

controller send the information to the next module.

Pseudocode - A casting call

```
for each scene {
  define role
  }

send image specification for each scene
retrieve image list

while ( list is not received empty and still a new line ) {
  store image under its scene
  }
```

The pseudocode elaborates the design and show the time line between the different processes but also selections done.

4.4 Audition

When the casting call has been sent out with the requirements for each role in the play, the auditions can begin. In an audition the actors get to display their theatrical skill, using a scene or monologue they have rehearsed before the audition takes place. The scene is from the play and involve the role they are auditioning for or it could be a scene the actor have chosen. A rule known by actors, but not written down, is not to use more time then given for the audition. The casting or audition panel have limited time and therefor using more time then given, provides the panel a negative impression of the actor.

The purpose of the audition is getting to know the actor and to get a feeling of the uniqueness of the actor, however this uniqueness could also be the chemistry between two actors.

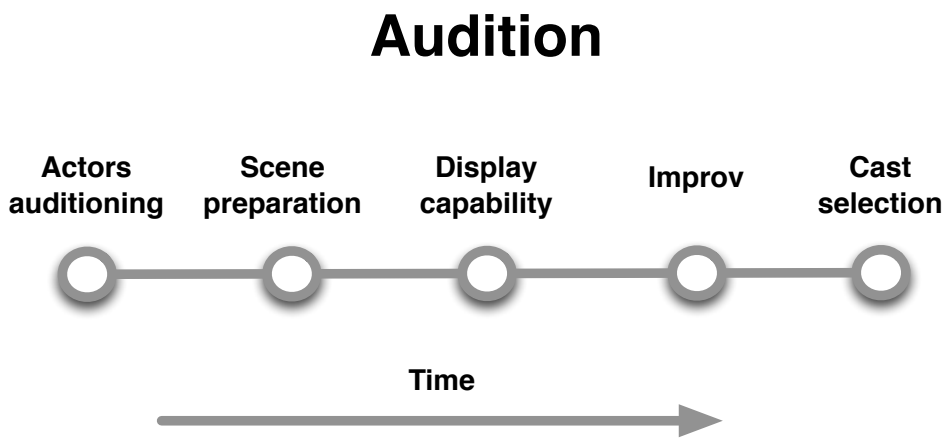

Figure 4.3: Image selection based on predefined requirements

Figure 4.3 is a time line for the audition stage. The different processes have been placed on the time line based upon execution time. With execution time, mean in what order the processes is executed, not the time the execution takes.

4.4.1 Scene preparation

The actor might need help of other actors to perform a supportive role or roles, in order for the actor to successfully display the full range of potential or to show that they posses the skills needed to perform the part. The supporting role is performed not by people auditioning, and therefor the supporting roles are prepared before the actors start the audition. To have prepared the supporting roles in advances and the same people performing the supportive roles during the whole audition process, gives all the actors auditioning the same performance from the supportive roles. If all the actors auditioning get the same quality of feedback from supportive roles, the quality of performance delivered by the different actors auditioning is easier to compare. The key for getting the real impression of an actor, is equal conditions of competition.

Not only the supporting actors are prepared, but also the environment where the audition will take place. There can be a stage in front of people or in a small room with just the casting panel. Is there a need for props, in that case the props need to be placed on stage in advance. The actors trying out have been given a number in line, and a given time frame for the audition.

In figure 4.4 the processes behind the scene preparation is explained. For every role, there is a scene to be prepared. The supporting roles are

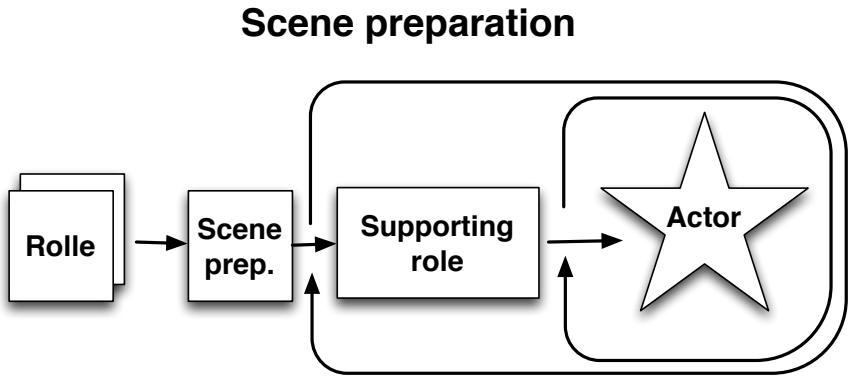

Figure 4.4: Image selection based on predefined requirements

prepared in advanced, before each role can begin the audition.

Pseudocode - Scene preparation

```
for each role {
  set the scene
  if supporting roles {
    start supporting roles
  }
  Start next part of audition
}
```

The pseudocode describes a semi translation from theater to the field system administration, or a simplification. The role is a name used to describe the different services wanted for the environment. Then the scene is set and if there is a supporting role it is started before the audition begins. After the supporting role is started, the actor is put to the test. After all the auditions for that role is done, the next role is up for audition. In a real audition an actor can audition for multiple roles at the same time, the actor does a general audition and the casting panel can hire the actor for the role they think the actor fulfill best. However, there is multiple ways of conducting an audition, so technically, non can be seen as incorrect.

In system administration the role would be looked at as a service or services needed within the environment. The scene is a description of what the role will do within the scene, its a name which are used to separate different roles and dialogue form each other. The supportive host will be hosts running services needed during the audition phase. This supporting role could be database server containing information needed by an audition for a web server, or a web server used in an audition for a load balancer role. After the supporting roles have been set up and are ready to be used, the actor can begin the audition.

4.4.2 Put to the test

The key feature of an audition is to experience or test the skills possessed by the actor, to get as much information about the actor within a short time span. To get a feeling if the actor could fit the role character and if who is the best for the role.

Performing the dialogue

The audition dialogue can have many structures or different ways of selecting the dialogue. The actor can perform a dialogue from a self chosen scene, or the casting panel could have selected a scene for the all the actors auditioning. To transfer this into system administration consider the following, the casting panel uses one specific scene to measure the skill set of the actors.

The audition can be shorten down if the casting panel understand that the actor do not posses the additional features wanted for the role, or don't possess the level of quality required. Theater life is no exception, time is limited, therefore actors auditioning will be rejected quickly, if they do not possess the wanted assets.

Figure 4.5 display the two different outcomes from an audition. The dialogue on top, display a successful dialogue. The three dialogues where performed with success, and the information about the actor is stored. The example below, display a scenario where the first dialogue is completed with success. However, the second dialogue failed. The dialogues is than aborted, and the audition is over.

In system administration one often measure the quality or a service, using benchmark tools or other testing tools. The audition dialogue would then be a sett of minimum requirements for the system, a service performance level required or minimum of service level accepted.

Pseudocode - The dialogue

```
If actor is ready {
  foreach dialogue {
    role perform dialogue
    if ( dialogue finished unsuccessfully ) {
      end dialogue
    }
  }
}
else
  wait a little before a new attempt on dialogue
  if still not ready end audition for actor
```

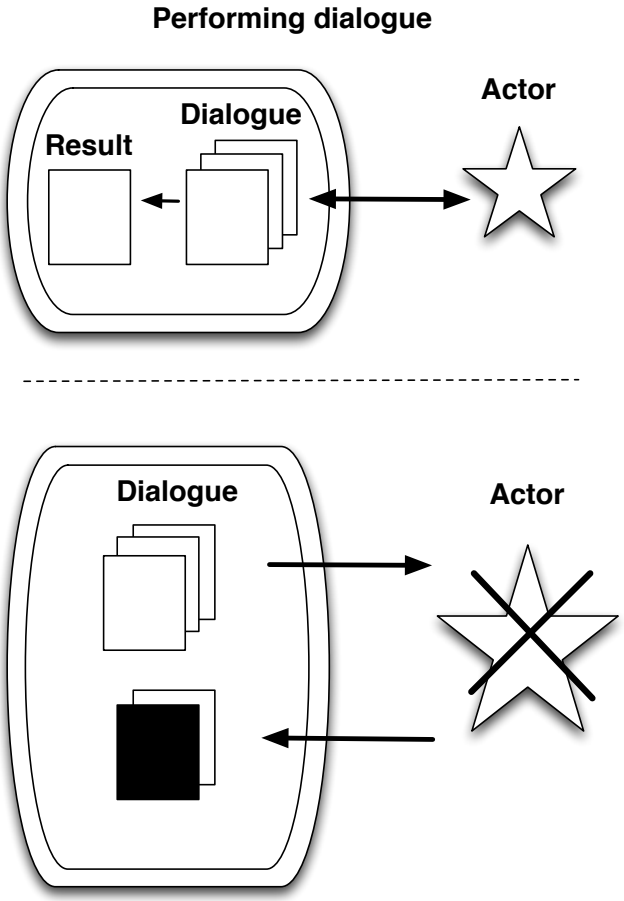

Figure 4.5: The dialogue performed during an audition

Configuring and performing the dialogue in Amazon cloud

The manuscript contain information about the minimum of accepted performance from the service in the dialogue, all AWS images performing in a adequate way is discarded from the list. However if the AWS image is not build and the testing could not begin within a certain time limit, it is also discarded. In comparison with hiring process, if the actor is not able within the time given to perform the act memorized or could not learn a act within the period from the casting call to the audition the actor is discarded, classified as not fit for the role. The same logic is inherited to system administration. If the image is not able to start and run the service within the given time period, the AWS image is discarded as not fit for the environment.

With the change in environment form local to cloud, using a server from the AWS store, the sysadmin know that the operating system and service running is compatible with each other. However the system administrator does not know if the additional services needed in the environment, is com-

patible with the operating system and service running. Today a web server is not only limited to running just one service, the server have to run a backup tool, monitoring of services, scripting for parsing of information between services. Looking at only the performance of the specific web services, for instance the connection per second. This will not be revealed if the system is compatible with the services needed to maintain the environment.

4.4.3 The uniqueness of improvisation

During the audition, if the actor have shown a high quality performance on the rehearsed part. The casting panel uses improvisation to explore the actors personalty, but also improvisation help the casting panel to experience if the actor posses features wanted for the role. To separate the good actors from the actors perfect for the role. To find the uniqueness or something special, that is wanted for the role. The improvisation is a tool used by the casting panel to get a feeling of the actor, within a short period of time. Improvisation also adds to the already existing pressure of the audition, does not allow the actor to relax during the audition process. This will relive how the actors deal with the stressing factor.

With the improvisation part of the audition, the casting panel might not have a quality range and are instead looking for how the actor responds to the task given. How the actor handle a task without a clear answer, and where the instinct of the actor is tested.

Translating this function of improvisation to the field of system administration needs to be done carefully. Think of this scenario: during the audition the host auditioning for the role as a web server suddenly is asked to be a firewall. There is no point for a web server also to be the firewall in the environment, therefor the implementation of improvisation need some boundaries or guidelines.

Improvisation can be used in order to uncover special methods and uncover hidden quality. Whenever a chef get a new pot he might have some rituals to test the quality. Another example is when you are buying a car, some people kick the tires. This action have no link to quality or performance of the car, but for the buyer it is an action he want to perform before deciding to buy a car.

The improvisation needs to be some form of communication to uncover wanted features from the system, which is not measured by a level of quality, impro have no clear result. Instead, the dialogue result needs to be interpreted by the system administrator, and based on the result get a familiarity with the system.

Figure 4.6 explain how the improvisation part of the audition is handled. The actor receives some key words and based on the key words

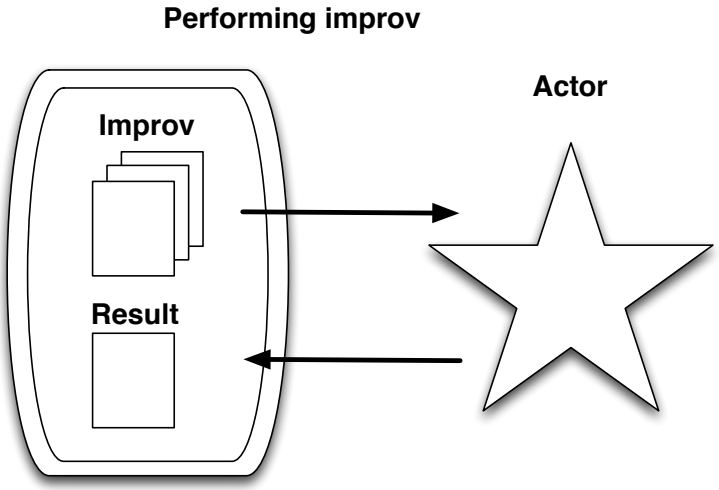

Figure 4.6: The improv dialogue performed during an audition

the actors have to perform. The overall expression of the performance is stored, and can be used in the selection phase later in the audition process.

Pseudocode - Improvisation

```
if ( all dialogues completed successfully ) {
  while ( impro to run ) {
    run improv
    store result for improv
  }
}
```

By inheriting the process, improvisation from the field of theater. The system administrator can specify any task the server need to fulfill in order to be usable for the environment. The plan will test the performance of the system, therefore all AWS images performing less then the stated requirements have all ready been eliminated. Therefore the improvisation is one more step closer to find the server best suited for the wanted environment, but also the server that will blend in with the excising support services already in use.

Audition evaluation and cast selection

In an audition the casting call makes notes on all actors auditioning. When the audition is over the casting panel evaluates the actors. The casting panel discuss and rank the actors before deciding who is allowed to do one more audition. After further audition the best qualified actor is hired for the role. In this process it is the purpose of the casting panel to find the best actor

for the role, often regardless of price.

The actors completing the audition do not normally get feedback on the audition, if the casting panel want an actor to do a new audition or they will offer the actor the role, they will contact the actor. If an actor is not contacted by the casting panel, it means that the actor did not get the role or is not wanted for any additional auditions.

When judging the result from a system administrator point of view, the best is always preferable. If looked at from a economical point of view the cost is often the most important element. For this project a combination of the two options are wanted. The goal is to get the cheapest system that exceeds the minimum system requirements.

Pseudocode - Evaluation and cast selection

```
foreach scene {
  foreach role {
    foreach actor auditioning {
      if price of actor is cheaper the now lowest {
        save price and actor information
      }
    }
  }
  display information about cheapest host for role
}
```

Figure 4.7 demostrates the process behind fulfilling a role after the audition for all roles is completed. The actors completing the audition is viewed, and the cost of the actor is assessed. The actor with the lowest cost is selected, and given an ofter to fulfill the role.

Figure 4.7 demostrates the fulfilling of a role, if multiple actors have the lowest cost. The actors with the same cost is then grouped together and a selection will be based on the outcome of the improvisation part of the audition.

After the audition process is completed, with one or more auditions used to evaluate the actors, the actor chosen for the role and the crew of the play have to reach an agreement on the hiring terms. The terms can range from employment period, number of days traveling to perform the actor money, normal contract terms for any contract related to work hiring. If the two parts come to an understanding and the contract is signed, the audition process ends and the rehearsal phase begins.

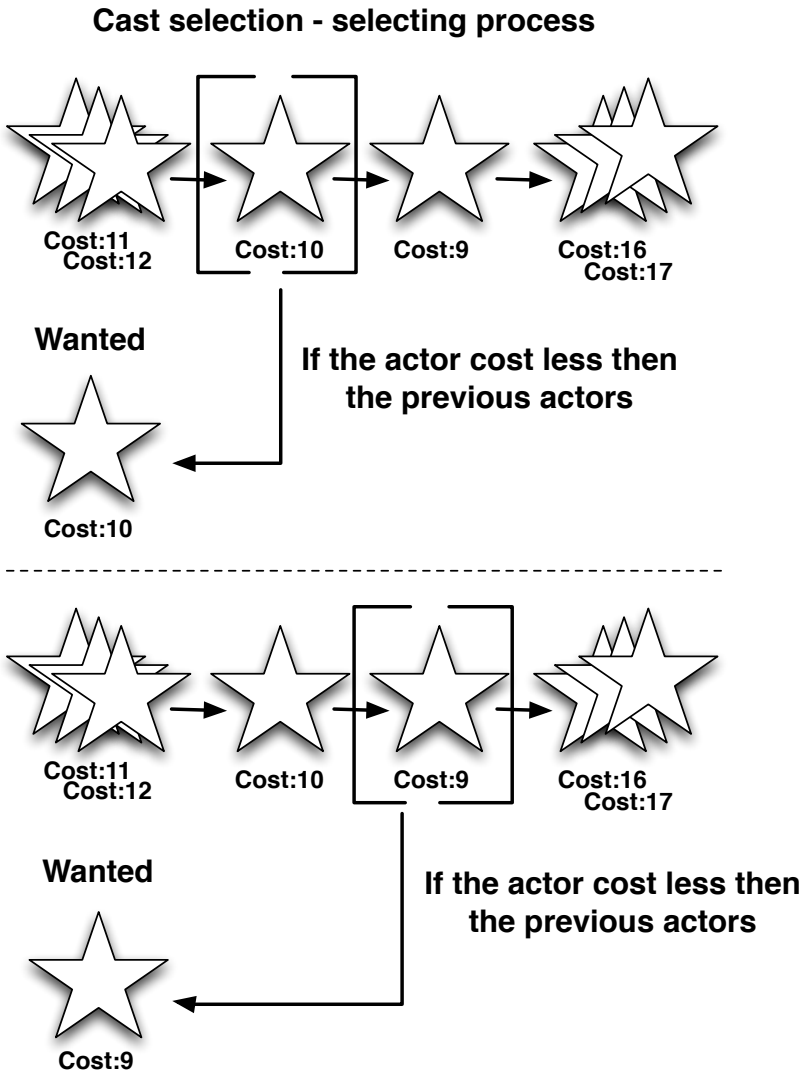

Figure 4.7: Role fulfillment with an actor

4.5 The grand premiere

After the manuscript is written, the casting call is finished, the auditions phase are completed and the role where filled. The actors rehearsed and rehearsed until the directors creative picture for the play, but also quality is meet, its time for the grand premier. The grand premier its where all the hard work gets paid for. The actors on stage performing the play in front of an audience. Where the director hopes his or hers creation will contain a higher quality of performance then what the people expect, that the play will be a success. The premiere is just all the different parts, after rehearsal, comes together and perform the whole play at once. All the roles, dialogs and scenes, all at once.

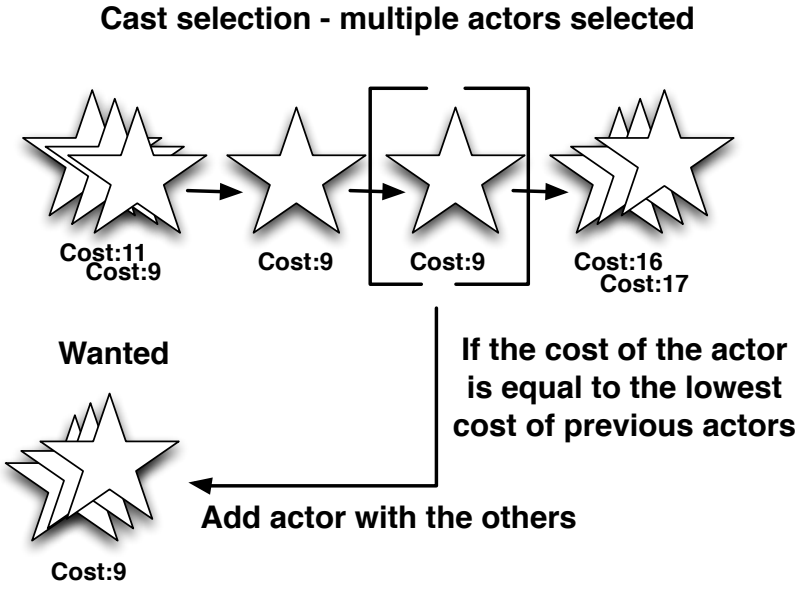

Figure 4.8: Role fulfillment, were multiple actors can be selected

In system administration one can relate this process to the release stage, where the system is set into production. The production stage or the production implementation is normally in the night time, or after work. In oder to minimize the impact for the users, and to have time to roll back if there is an error so the production environment do not work according to plan. The processes is normally started at the testing phase, have a copy of the production environment and do all the implementation on the test environment. After the installation the system is tested, and if no errors are discovered the process is repeated at the production environment. The process is done by the system administrator and the process is time consuming, and with the human interaction the error rate need to be considered.

People make mistakes, one have to take in to consideration that errors will happen. The error rate will increase in certain situations, for instance: if the system administrator is working to much and is tired, nervous and with time differences. With shifting the work hours to allow for night work, the body will be a little tired so the possibility for an error to be made is a little higher. With combining the process of theater into system administrator, it enables for minimal human interaction.

Using the information in the audition module, the services are rated based on a overall score in the selection phase. The different services have been build in different settings and have been tested to work with other services according to the plan. Using the AWS images that where rated as best for the environment, for all the services in the environment. No human interactions is needed, since each role or service where individually

build and tested in the audition phase. The selected cast can now be build and set in production.

The creation of the environment and cast needs no additional information, since each member of the selected cast have been build and tested within the different audition phases. The information regarding the hosts and services within the manuscript is reused from the audition creation, therefore the creation of a fully working environment can be created automatically.

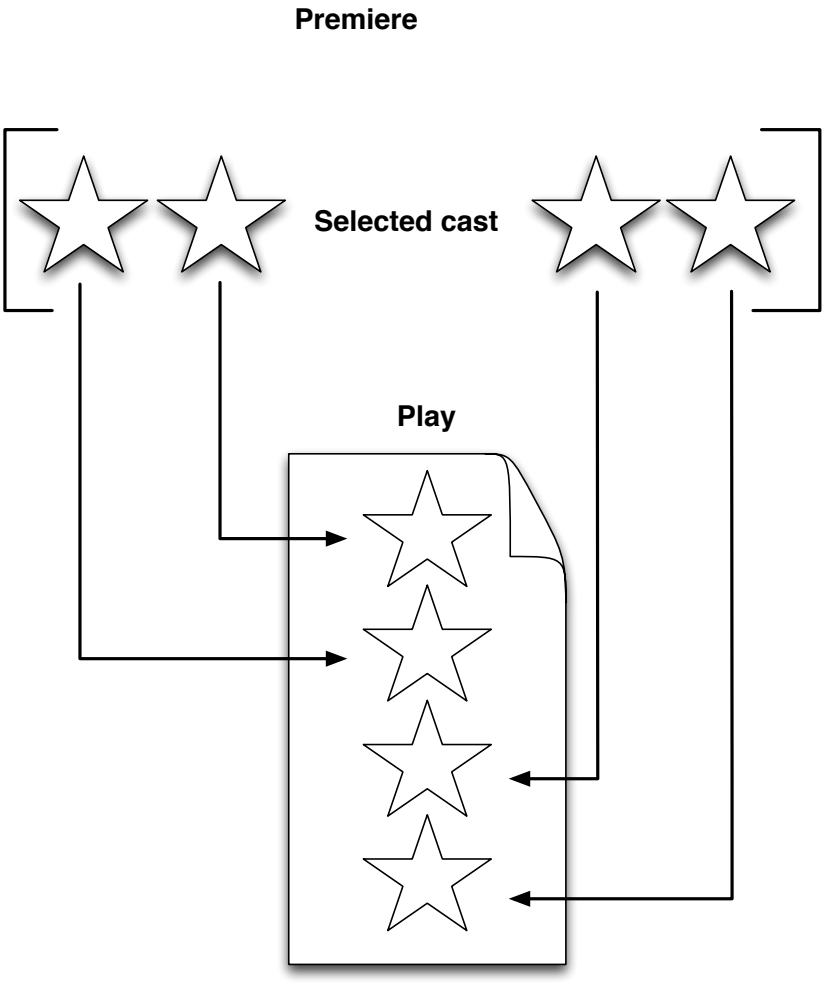

Figure 4.9: The assembling of a cast

In figure 4.9 the different actors selected become part of a cast, the selected cast will rehearse and perform the play.

4.6 Summarization - audition architecture

When going back to the timeline in figure 4.1, all processes have some level of design inheritance from theater and in most cases a strong inheritance.

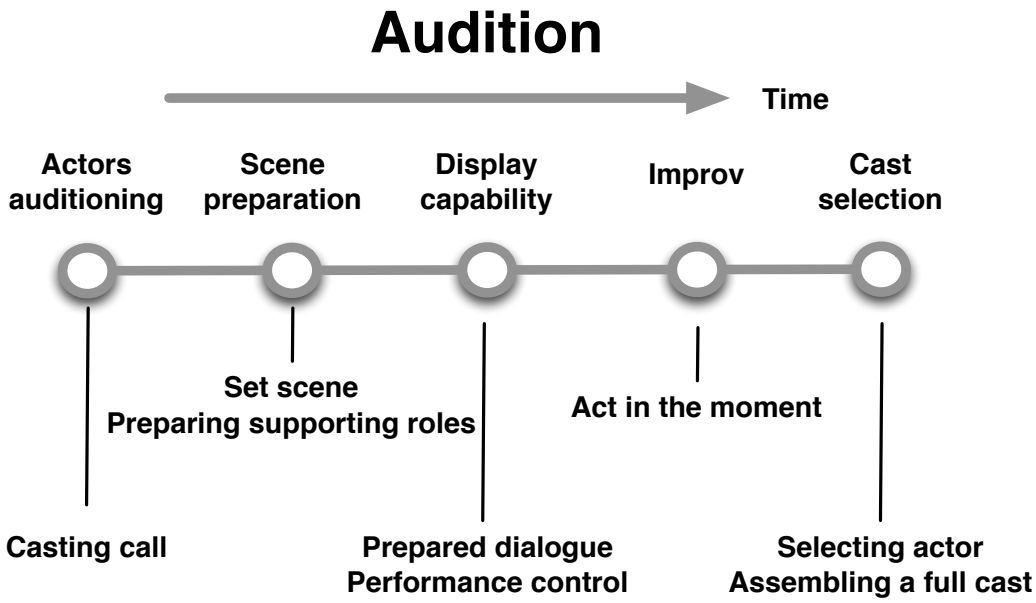

Figure 4.10: Image selection based on predefined requirements

Figure 4.10 show the different milestones in an audition process. The theater processes have been dissected and a vital process in the field of system administration have been created.

Figure 4.11 show all the different modules merged together, and displays how all the different modules is placed in the design and how they work together. The manuscript dictates the different boundaries for the audition and the play. Actors responding to the casting call, is evaluated based on an dialogue and an improvisation part. The actors with the wanted features and quality is available for cast selection. The actors selected for a role is then given an offer to be part of the permanent cast.

The manuscript is created in advanced and contains all the information needed for the whole audition process. When the casting call is completed, the actors comes to the audition and one by one audition for the role they signed up for. The actor auditioning will perform a dialogue and if the quality or performance meet quality wanted, the casting panel will give the actor an improv challenge. When the audition is over, the casting panel review all the actors and match each role to the most suited actor, and the actors selected will be a part of the permanent cast.

Table 4.2: Models to implement in the prototype

Models to implement in the prototype	
Manuscript	No
Casting call	Yes
Dialogue	A Yes
Improv	A Yes
Cast selection	Yes
Play	no

4.7 Models to implement

The module casting call will not be implemented, according to table 4.2. Since the selection of Amazon as a cloud technology, they do not provide an API for the Amazon AWS store. For that reason it is at the current time not possible to create a function which will get a list of images or do an image selection automatically.

The play module will not be implemented, because the goal is to create a prototype and the play module will use all winning hosts and create one MLN code for whole play. The code will already have been created in the audition by the Audition script, however, there is not need for this feature in order to help answering the problem statement. Therefor the play module is not implemented.

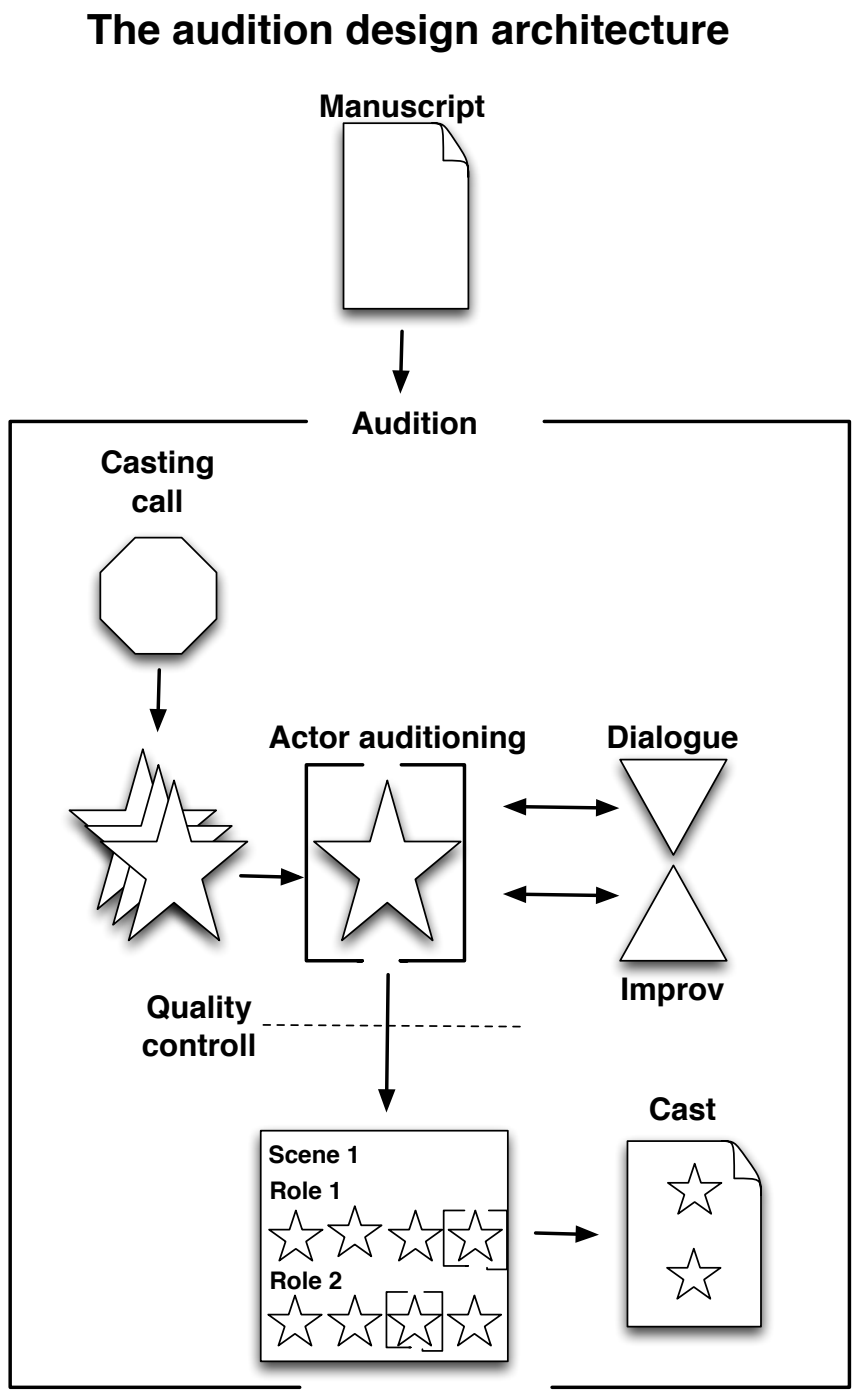

Figure 4.11: The overall view of the selected processes within an audition

Chapter 5

Result 2 - Prototype

Throughout the prototype chapter one example will be used to explain the prototype and its features and how it is built. The example scenario will involve a webserver using wordpress with a database connected. In the scenario, it is the webserver that will be tested to figure out what kind of image or hardware type will fulfill the systems requirements, at the lowest cost.

5.1 System platform

The system will be built and run in Amazon EC2. The reason for this is both cost, access to a variety of hardware and the amount of AWS images in the Amazon store. However, the framework is not bound by Amazon and can be changed to a different cloud service with little effort.

Figure 5.1 show how the platform and how the environment will be built. Amazon EC2 will be the virtualization framework used to cerate and run the virtual computers. To create, manage and configure the hosts, the tools MLN and puppet will be used.

To fully automate the framework MLN is used in combination with puppet, MLN has a built in plugin for Amazon EC2 that is tested with success. MLN can be used on different environments (XEN [9], VMware [12], Amazon [13]), and with few commands set up a virtual computer or multiple computers because of this unique feature of setting up multiple computers simultaneously, MLN is the perfect tool to be used by the framework.

5.1.1 A controlling host

Figure 5.2 display the usage of a controller. The controller is the host running the audition, this host contains the MLN and puppet configuration in order to hold a successful Audition. The Audition is a script modeled after an audition process, and the script is run on the controller.

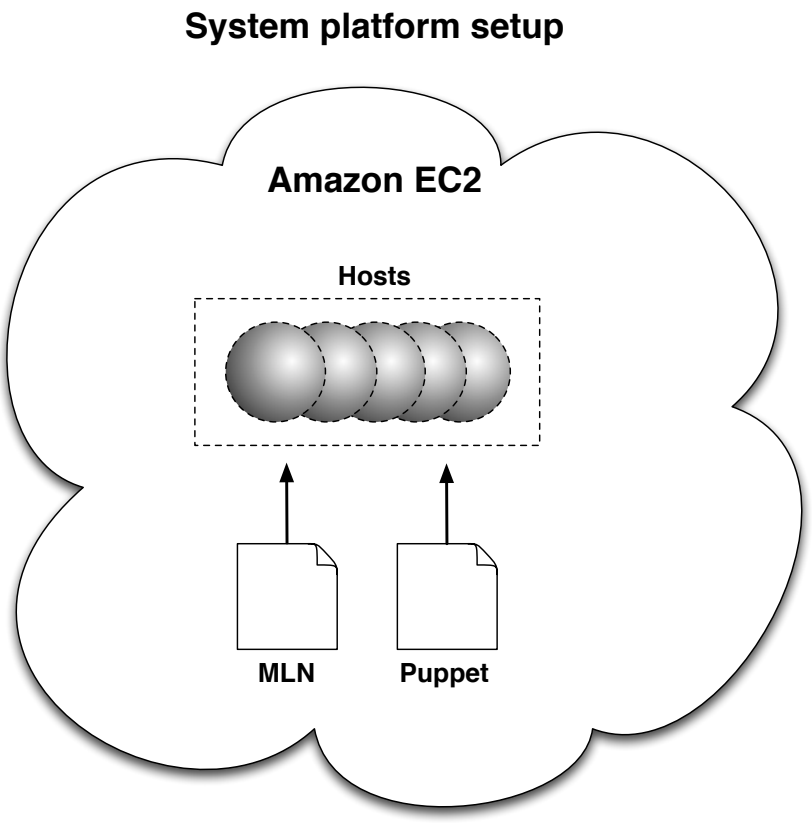

Figure 5.1: The system platform setup

5.1.2 Amazon storage

In Amazon EC2 when the virtual hosts instance is shutdown the local hard drive is removed, but any EBS volume connected will continue to exist. A shutdown could come from the shutdown command, however it could also come from a system failure or another error within the system. Therefore, the system could shut down without human intent, and with the system shutting down the files and data collected is lost. This means that if the controller instance goes down, not only will the framework stop working but all the data collected until the shut down will be lost.

There are different methods to solve the accidental shutdown scenario, since the framework is automated all the files and data needed already exist and can be initiated again with much effort. With just recreating an amazon instance for the controller the framework could start again, and when finished the only thing lost is time and the cost of running benchmarking on different AWS images twice. A different method is to connect the controller instance to a EBS volume, in case of a shutdown the information saved would still exist after powering the instance again. Due to the fact that the implementation is a prototype, the system will handle the case of a non

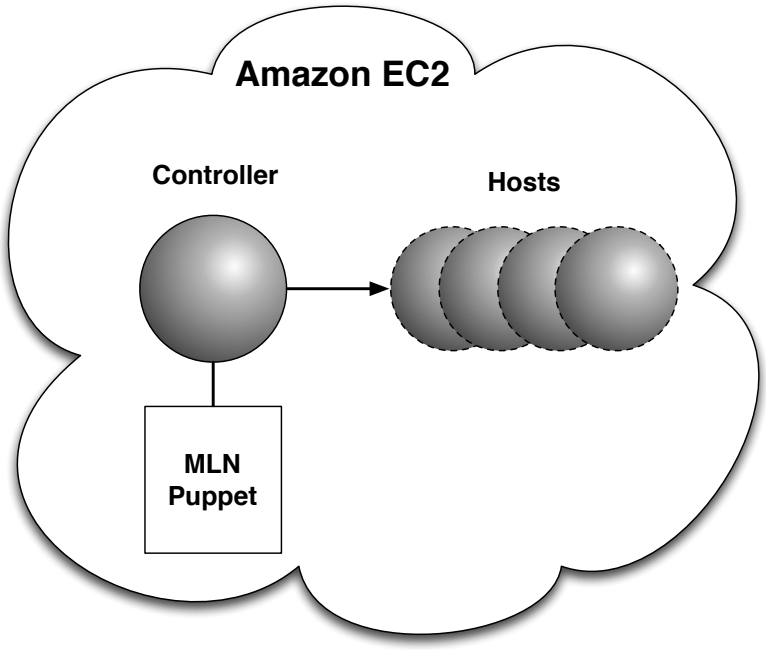

Figure 5.2: The host controlling the Audition

planed shutdown of the controller after the kiss method. Since all the data to perform a new run will be easily available, it will be simpler to restart the test rather then to resume the previous run.

5.2 The layout of the manuscript

The manuscript is where the system administrator can define the entire environment. In the manuscript the different roles, previously mentioned in chapter 4 defines both the environment, but also all features which will conduct the audition or the performance wanted from the environment. The manuscript is built up using a block format. The block format is commonly used by system administrators, therefore the syntax of this approach should be familiar and easy to understand. The manuscript is dependent on the usage of MLN, therefore the MLN template is used for this performance needs to be stated at the beginning of the manuscript. Further explanation of the MLN template will be conducted later in this chapter 5.4.

Manuscript - An example manuscript

```
mln_file mln-template-wordpress.mln

scene frontpage {
  role web
```

```
  support_roles database

  dialogue {
    [web]: connect.db [database]
    [web]: content.web /index.php Lorem Ipsum
    [web]: benchmark.web 50 /index.php
  }
  improv {
    ./perl.pl
  }
}

scene backend_performance {
  role database

  dialogue {
    [database]: restore.db wordpress_base.sql
    [database]: transactions.db 300
  }
}
```

The manuscript above shows a manuscript with two scenes, the first scene, scene frontpage, has only one role. However, there is a need for a supporting role, to help the role in question with the audition. The role will be tested with three different tests, and there is an improvisation part within the audition.

The second scene, scene backend_performance, has one role with no supporting roles. The hosts trying out for the audition will be tested using two different test dialogues.

5.2.1 Scenes

Each scene is built up by the key word *scene* and the scene name, then the block to hold the information regarding the current scene.

Manuscript - Create a scene

```
scene frontpage {

}
```

The scene created here is named frontpage. In order to create a scene some information is required within it. The scene name has no other function then to separate the different scenes and acts like a container for all the information regarding the current scene.

The leading role

The design dictated that there had to be certain information mentioned for each scene, in order to have a dialogue there had to be a role, and by that dictates that there has to be a role. The role in this case is called web.

Manuscript - Create a role

```
scene frontpage {
  role web

}
```

The role is not a specific server or a hardware type, it is a service or combination of services. Web could in this case be apache [7] or lighttpd [60], a services which provide a web service, hence the name web. However, the specific service is not specified in the manuscript, the service specification is located in MLN and will be explained later in this chapter. Therefore "web" is just a name for the role.

Numbers of one role

In a theater the goal of an audition is to find one actor to fulfill a role, an understudy can also be hired. Two actors can not have the same role, performing at the same time. Two Romeo's in Romeo and Juliet, does not make the play two times better.

In system administration scalability is important, especially when dealing with release management. In system administration there could be the need for two webservers or more, in an environment. In system administration two webservers, can be two time better then one. Therefore, there is a need for multiple equal roles. This option is enabled using the roles_available command, and the specify the number of roles. However, if the line is not part of the manuscript created, Audition will assume the roles wanted are one.

Manuscript - Numbers of roles wanted

```
scene frontpage {
  role web
  roles_available 2

}
```

The supporting roles

A role might need supporting roles in order to complete its dialogue. There is no limitation to how many supporting roles the system can use. Different

supporting roles are separated with the use of "," in between the names, instead of blocks. The key-word used to inform about any supporting roles within the scene is *supporting_roles*.

Manuscript - Adding a supporting role

```
scene frontpage {
  role web
  supporting_roles database,loadbalancer

}
```

The dialogue

After the wanted environment has been described, the desired dialogue is defined. The dialogue is built up stating: who is telling the dialogue, what plugin is used and what information is sent to the plugin. The dialogue is using blocks to hold all the dialogue for each scene.

Manuscript - Create a dialogue

```
scene frontpage {
  role web
  supporting_roles database

  dialogue {
    [web]: connect.db [database]
    [web]: content.web /index.php Macbeth
    [web]: benchmark.web 50 /index.php
  }
}
```

The dialogue in the example shows the possibility of this feature, the first dialogue (from top), let the web role connect to the role database. It enables the web server to connect to the database, which has the content stored. The next dialogue is also to the web role, and it will check the index page on the web role for the word "Lorem". This is a quality check, it is a check that the web role did connect to the database supporting role and that the database is connected and displays the correct information. The last line is the benchmarking, the performance check. It is the web role that is under testing and the quality required to be available for the environment is 50.

In order for the dialogue to be completed successfully all the lines need to be returned with success and by that pass the requirements needed for the environment. In the case of a dialog returning failed and therefore not meeting the requirements, the dialogue is aborted and no more dialogues or improvisation will be conducted on the host. The host will be turned

off and a new host will be up for audition. In the case of all dialogues returning success, the host will continue with the improvisation and add the host information to a hash containing hostname for every host for each scene, that successfully completed the dialogue.

The improvisation

The improv is not used in the automated selection process, but is used in the situation where multiple hosts are tied for the first place in the selection phase. The improvisation is not a pass or fail for the dialogue within the improvisation, and therefore the system administrator has to assess which AMI and hardware combination is wanted for deployment after looking at the output from the improvisation.

The improv dialogue in the example below show how to set up the improvisation. The improv part is usable using the keyword improv, the audition will run all the information contained within the brackets. Improv is only for the host in questining, therefore there is no information about whom the improv is for. Furthermore, the improv is not relaying on any plugins. The manuscript syntax is commandline commands, that will be executed on the host.

Manuscript - How to improv

```
scene frontpage {
  role web
  supporting_roles database

 dialogue {
    [web]: connect.db [database]
    [web]: content.web /index.php Macbeth
    [web]: benchmark.web 50 /index.php
  }

  improv {
    ./perl.pl
    ps aux
  }
}
```

The improvisation commands run on the host during an audition, will store the output in plain text in different text document. The document has the same name as the host undergoing the audition, and located under the folder "improv", where the audition script is run from. The improvisation part is not a fail or pass in order to create output, the process will sent the raw output to file. Therefore the content of the file could be nothing, if the commands run is not compatible with the system running on the host.

5.3 How Audition works

Audition is a script written in perl, and run on the controller unit. The controller unit in this case is a virtual computer running in Amazon cloud environment.

Each role or service will have multiple different hardware configurations and different AMIs, therefore each combination of the two needs to be tested. In order to start and stop multiple computers in Amazon cloud, the tool MLN was used. MLN also enables for the usage of excising EC2 commands distributed by Amazon. The roles are started when being used and stopped when all the dialogue is completed or if a dialogue failed to complete successfully. However the supporting roles are started at the beginning of the scene, this can be further explained viewing the overall processing of the audition script.

Pseudocode - An Audition

```
foreach role {
  boot support roles
  apply support roles configuration
  foreach candidate {
    boot the candidate
    apply role configuration or finish
    foreach scene {
      run dialogue or finish
      run improvisation
      store results
    }
    shut down candidate
  }
  shut down support roles
}
display results
```

5.3.1 The supporting host

The supporting hosts are started and prepared before the host to be tested is booted up, to have the supporting role up during the whole scene saves time. Since the configuration only has to take place ones.

Amazon has elected to use a cost system which charges not for actual run time but for commenced hours. Running a virtual host in Amazon for 1 minute or 59 minutes has the exact same cost, therefore to keep the virtual machine used by the supporting role the same virtual machine, will lower the cost of the audition.

The reason for using the supporting host for the whole scene is more clear using some examples. If one host uses 19 minutes to complete the audition, and there are 6 hosts auditioning. To get the supporting host up and running with the configuration takes 3 minutes, and all hosts and supporting hosts cost 1 dollar each commenced hour.

By creating a supporting host for each candidate, it will result in 6 hosts times 1 dollar (6*1), and 6 supporting hosts times 1 dollar (6*1). The total cost for an audition will be 12 dollars, six hosts hours plus six supporting host hours.

If using the same supporting host for each host the cost would be: 6 hosts times 1 dollar (6*1), and 2 hours for one supporting role (19 minutes * 6 hosts = 114 / 60 = 1.9 = 2). The total cost for an audition is then 8 dollars. The money saved using the same supporting role for the scene showed in the examples is 1/3 of the initial price.

Not only will it lower the cost but it will also save time. Since the configuration will only have to be run once, the time used to set up the supporting role will be a one time time expense, and not a time expense multiplied by the combination of AMIs and hardware configurations.

5.3.2 Candidate names and host names

The candidates are defined in the manuscript under each scene. The service and the different testing and commands for the candidate to perform under the Audition phase. The information about the candidate is build up using information from several different sources. The name scheme for the candidate is a combination of service and information about the images and hardware combination. This will create a unique name for all scenes, services, candidates and any combination of AMI's and hardware types in Amazon EC2. See figure 5.3.

Figure 5.3 display how the name is created for the different hosts auditioning. The candidatename consists of the role the candidate is audition for, the AMI and hardware information for the candidate auditioning.

5.4 The MLN template configuration

The MLN template stated in the manuscript using the command "mln_file"" connects the name of the role, to an actual configuration, be that a service or specific software packages. The MLN template file consists of different superclasses, the different superclasses is a definition for each of the roles mentioned in the manuscript. The format for the template is dictated by the tool, since the file is fully operational MLN code.

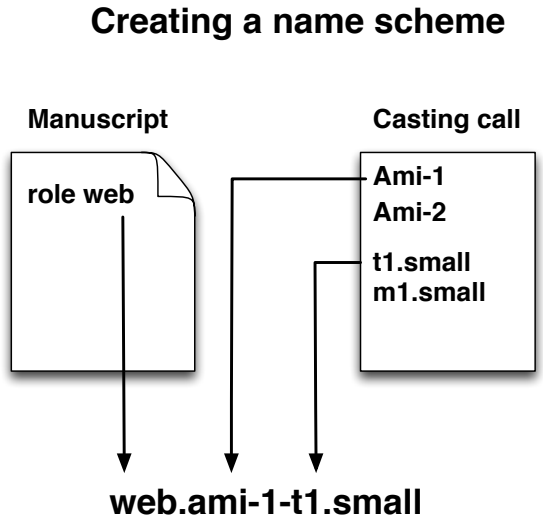

Figure 5.3: Creating a name scheme for the hosts auditioning

```
1
2   global {
3      project macbeth
4   }
5
6   superclass basic {
7       puppet {
8           nodename $hostname.$project
9           include {
10                ssh
11               }
12      }
13      ec2 {
14          key Enterprise
15          user_file {
16             apt-get update
17             apt-get -y install puppet
18             }
19      }
20  }
21
22  superclass wordpress_db {
23      superclass basic
24      puppet {
25         include {
26         wordpress_db
27         }
```

```
28      }
29  }
30
```

In the mln template file there is no information about any host except superclass definitions. All the host information regarding MLN is created by the Audition. The creation of hosts is based on the manuscript and casting call and created automatically.

5.5 A complete Audition

The complete Audition will consist of an environment with a web server and a database server. The web server is the role the test will be conducted on, and the database server will be the supporting role. The role of a web server will consist of the service wordpress, a popular web service.

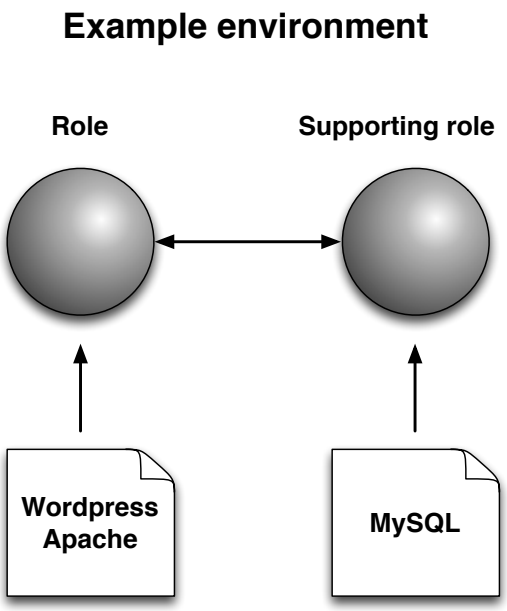

Figure 5.4: Graphical display of the wordpress example environment

Figure 5.4 show the environment wanted in the wordpress scenario. The role auditioning for, is for a wordpress web sever. The wordpress server will be connected to a supporting host. The supporting host will run a MySQl database with the content displayed by the webserver.

5.5.1 Host AMI's and hardware types

Due to Amazon not providing an API or any way to automatically retrieve a list over usable AMI's, the AMI and hardware information needs to collected in advance. Different AMI's have a small differences in price,

however, using different hardware configurations the price for the different hosts will not be the same.

For this Audition one AMI i selected, this AMI is used throughout the Audition. The AMI is an Ubuntu 64-bit operating system. In Amazon EC2, using this operating system adds no extra cost pr hour for the different host.

Different hardware types

The different hardware types selected for the Audition concerning a wordpress environment is:

The different hardware types selected

```
m1.small
m1.medium
m1.large
c1.medium
```

Based on the initial tests, the images chosen is the four images with a performance closest to the minimum requirement for the environment.

M1.large instance specification

The M family in Amazon EC2 meaning the instances starting with m1 or m3, are general purpose instances. The instances have some compute power, but the instances have a balanced performance between all resources.

C1.medium instance specification

The C family or the instances starting with C1 or CC2, is instances optimized for compute power. The instances have a higher CPU performance, and is recommended my Amazon for high-traffic web applications [4].

Cost

Due to the limitation in automation surrounding AMI's and instances in Amazon EC2, there is at the current time not possible to automatically get hourly cost for each instance from Amazon. Therefore, the instance cost had to be collected before Audition could begin. The AMI chosen for this thesis, is a linux distro free to use. Therefore, there is no additional cost other then the instance cost.

The on-demand hardware types or instances in table 5.1 are priced according to Amazon EC2 instance prices May 2013 [3].

Table 5.1: EC2 price list

Amazon EC2 prices	
Hardware type	**Cost per hour**
m1.small	0.060 $
m1.medium	0.120 $
m1.large	0.240 $
c1.medium	0.145 $

5.5.2 Failing to learn the role

In certain scenarios the host auditioning did not send the correct information to the Audition, and because of this did not receive any instruction about the installation of service or host configuration. In the event of this scenario, Audition will after some time terminate the host and continue with a new host.

Due to time constraints it was not possible to further optimize the processes running within an Audition. The Audition will treat the scenario as if the host failed to meet the wanted quality in performance, and therefore the host is not available in the upcoming cast selection phase.

Such a scenario is taken into consideration within the field of theater. If the host is not able to show up for an audition, the next actor is sent in. If the actor is not ready to start the audition or has not been able to learn the scene the casting panel expect too see, the casting panel will stop the actor and ask him leave. That way, the casting panel can use time more efficiently.

5.5.3 Creation of plugin

The plugin is design and implemented as modules, each plugin file consist of one or more plugins. All the plugins follow the same format.

Manuscript - How the plugin is built

```
[webserver]: connect.middleware [middleware]
```

First comes the role performing the line, in the example above it is the webserver performing the line. Following the plugin used to in order to execute the dialogue. After the plugin there are different information, that can be used. In the example above the "[middleware]" will be changed by Audition to the name or ip dependent on the supporting role. However, there is also the option to send the whole line to the plugin.

Manuscript - How the plugin operates

```
[webserver]: performance.middleware 192.168.1.12 20
```

In the example above, the plugin performance.middleware is sent "192.168.1.12 20". Then it is up to the plugin to interpret the information and use it correctly. By allowing the plugin to handle the different information, all the dialogues can be standardized. This also allows for system administrators to simply create new plugins, since the plugin itself executes the different action.

The plugin handles the validation of the result, based on the plugin execution. The return value sent from the plugin back to Audition, is fail (0) or success (1). This enables for a system administrator to crate plugins without limitations, the Audition do not need information about what the plugin does or how. Audition only want to know if the plugin ran successfully or failed.

5.5.4 The controller

After starting testing the framework using the wordpress example environment with more hosts auditioning for the web server part, a new problem was discovered. Audition used a lot more time than expected. Looking at the different processes behind the Audition reveled that, the time used were because of EC2 commands and host configuration time.

During the testing stage, the controller host started to experience some problems. The problems were not consistent, therefore hard to investigate. The controller was run on a host with limited hardware resources. Therefore, the controller was migrated to a new host, with better hardware available to the host.

After the controller was migrated to a host with hardware type c1.medium, the time used for an Audition decreased. The new hardware resulted in faster response from EC2 commands and host configuration.

5.5.5 Benchmark and thresholds

Since lowering the expectation to match the performance of the environment, the minimum requirements regarding performance for the part of web server was set to 12 connections each second. The different testing stages in creating Audition, with the usage of environment data based on the wordpress example, displayed that using 12 connections pr second would make some hardware types fail the audition. This is in itself not a bad thing. A option were to go for massive VM's, but for the test it was important to let several candidates successfully finish the dialogue and some candidates failing the dialogue.

Requesting a performance higher than some selected hardware types performs, helps to display the selection stage. It also ensure that the Audition prototype, working according to the model designed.

In this prototype implementation the controller is the host for conducting the dialogue and improvisation. Creating a situation where the controller has no available resources left is unwise. The resource limitation could interfere with the Audition and the outcome or result.

5.5.6 The complete manuscript

To get a more complete picture over all the different modules and how the act together, a graphical display of the Audtion architecture has been created.

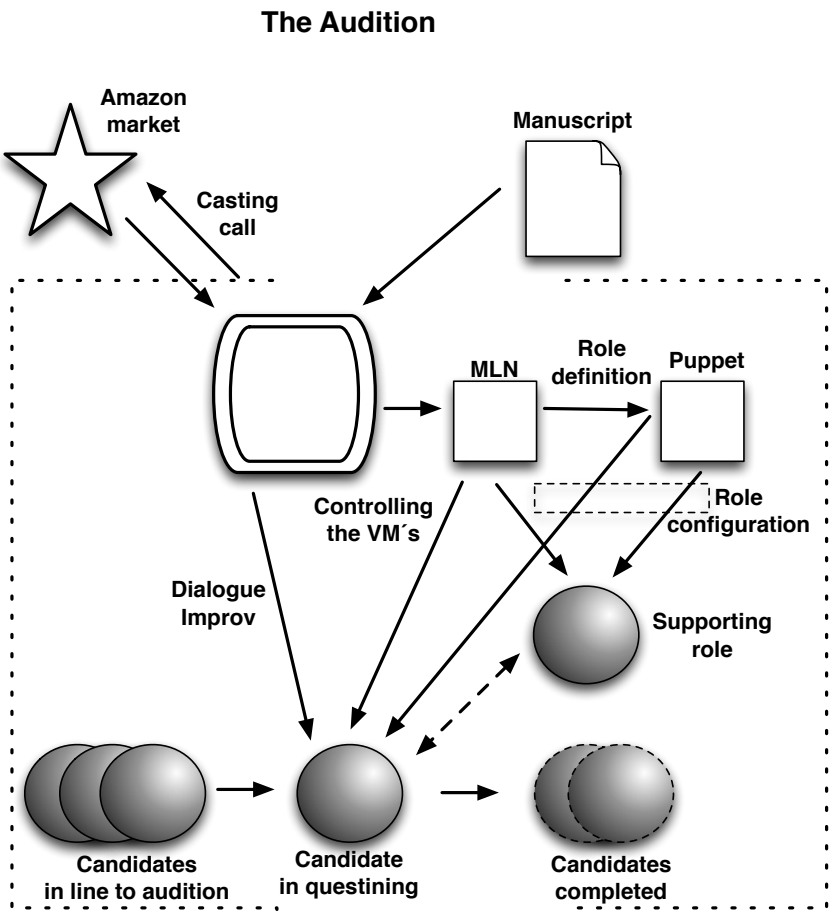

Figure 5.5: Graphical display of a complete Audition

Figure 5.5 show how all the different modules works together in an Audition. Audition will begin with reading the manuscript, before proceeding with a casting call. Audition will use MLN and Puppet to help with the establishing of candidates and supporting roles. Once the candidate are up, Audition will run the dialogue and improvisation on the candidate.

5.5.7 The complete manuscript

The manuscript uses the mln-config.mln file, in order to define the different roles. The scene frontpage contains the wordpress host, with database as supporting host. There are three different dialogues, first dialogue connecting the database to the web server. Then the dialogue validate the content displayed by wordpress, then the performance testing.

The improvisation part contains two different commands, there is a need to check the folder structure on the host computer. The other improvisation test is to get the timezone used by the host.

A manuscript for wordpress environment

```
mln_file mln-config.mln

scene frontpage {
    role wordpress-web
    support_roles wordpress-db

    dialogue {
       [webserver]: connect.db [database]
       [webserver]: content.web /index.php Lorem
       [webserver]: benchmark.web 12 /index.php
    }

    improv {
       ls -l /
       date
    }
}
```

5.5.8 Execution

The Audition is executed from the controller host, the execution is command based using the command line.

Execution of Audition

```
\$ audition wordpress.manuscript
```

The Audition need to be run with root access, the Audition will write files used by MLN and Puppet.

5.5.9 Audition output

The output produced in the selection phase in the audition of a wordpress environment, reports two hosts completing all the dialogue with a better

performance than the minimum requirements.

Output from Audition

```
Role: wordpress_web
2nd place: m1.large with ami-def89fb7
\$0.240 per instance-hour
1st place: c1.medium with ami-def89fb7
\$0.145 per instance-hour
```

The output shows that there where two hosts selected, m1.large and c1.medium, Audition selected c1.medium as the best solution for the wordpress environment.

5.5.10 Improvisation output

Under the impro folder, for the wordpress environment, two files are found under the Audition. There is one file for each host completing the dialogue with success.

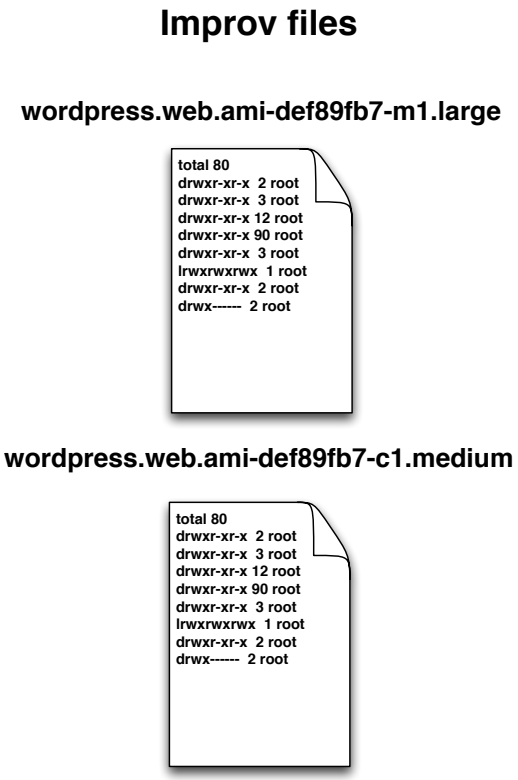

Figure 5.6: The result for an improvisation

Figure 5.6 displays the file nameing scheme used in the prototype, and

the result from the improvisation for selected hosts.

The files can be located under the folder "improv", in the same path as Audition. The content for the files are almost the same, with a small difference in time the "date" command where run at. The reason for this is they use the same AMI, and therefore they will give the same response for the commands given in the improvisation stage.

The content from the Audition improv

```
total 80
drwxr-xr-x  2 root root  4096 Jan 24 07:07 bin
drwxr-xr-x  3 root root  4096 Apr 30 16:44 boot
drwxr-xr-x 12 root root  3880 Apr 30 16:43 dev
drwxr-xr-x 90 root root  4096 Apr 30 16:46 etc
drwxr-xr-x  3 root root  4096 Jan 24 07:07 home
lrwxrwxrwx  1 root root    33 Jan 24 07:06 initrd.img
                -> /boot/initrd.img-3.2.0-36-virtual
drwxr-xr-x 18 root root  4096 Jan 24 07:06 lib
drwxr-xr-x  2 root root  4096 Jan 24 07:05 lib64
drwx------  2 root root 16384 Jan 24 07:07 lost+found
drwxr-xr-x  2 root root  4096 Jan 24 07:03 media
drwxr-xr-x  3 root root  4096 Feb  9 02:53 mnt
drwxr-xr-x  2 root root  4096 Jan 24 07:03 opt
dr-xr-xr-x 89 root root     0 Apr 30 16:43 proc
drwx------  4 root root  4096 Apr 30 16:46 root
drwxr-xr-x 16 root root   620 Apr 30 16:47 run
drwxr-xr-x  2 root root  4096 Jan 24 07:07 sbin
drwxr-xr-x  2 root root  4096 Mar  5  2012 selinux
drwxr-xr-x  2 root root  4096 Jan 24 07:03 srv
drwxr-xr-x 13 root root     0 Apr 30 16:43 sys
drwxrwxrwt  3 root root  4096 Apr 30 16:46 tmp
drwxr-xr-x 10 root root  4096 Jan 24 07:03 usr
drwxr-xr-x 13 root root  4096 Apr 30 16:46 var
lrwxrwxrwx  1 root root    29 Jan 24 07:06 vmlinuz
                  -> boot/vmlinuz-3.2.0-36-virtual
Tue Apr 30 16:47:10 UTC 2013
```

5.5.11 Scripts created

A perl script and plugin files where created for this thesis. The name and description for the different files is located in table 5.2.

5.6 A more complex example

The wordpress example used for testing the implementation, is too small to show of all features provided by the framework. One of the benefits with

Table 5.2: Different files created

Scripts used		
Type	Name	Description
Audition	audition.pl	Main script, runs the audition process
Plugin	connect.db	Connects host to database
	content.web	Confirms a valid displayed
	httperf	benchmarking a webserver using httperf

Audition, is the scalability. With the change of one line for each scene, one can specify how many servers of that role is wanted.

In figure 2.2, the new way of separating the environment into layers was shown. The figure displayed an environment consisting of: loadlancer, six webservers, one middleware server, three databases and a storage server. The complete manuscript can be found in section 5.6.6.

Based on the manuscript, each scene will be tested with a combination of all different AMS's and instances. For all scenes the improvisation is the same, to test if the system is compatible with the monitoring tool used in the current environment. All other scenes are then specially designed to test the role, with the usage of supporting hosts.

The first step in all scenes is to first start up and configure all the different supporting roles for that scene, before starting the audition for the role in questioning.

5.6.1 Auditioning for the role as loadbalancer

After the configuration of the webservers, the audition for a loadbalancer can begin. The host auditioning first run the connect.load dialogue, this will connect the loadbalancer to the webservers. If successfully connected the perform.load is started. This dialogue will test if the loadbalancer can send 50 or more requests, to the webservers.

After the dialogue is completed, the improvisation check the hosts compatibility to the existing monitoring tool. The result is stored for usage in the cast selection in a later stage of the Audition.

5.6.2 Auditioning for the role as webserver

The webserver auditioning will connect to the database using connect.db plugin, the content of the page delivered from the database is controlled. Before a benchmark test of the webserver is performed.

5.6.3 Auditioning for the role as middleware

After completion of the configuration regarding the host, the middleware server connect to the database using the connect.db plugin, before checking

the correctness of the information in the database. Then the webserver is tested, tp ensure the middleware can communicate with the webserver and the database. The last dialogue ensures that all services are running and no compatibility problem on the middleware server.

5.6.4 Auditioning for the role as database

The database first connects to a storage unit, the storage unit in this case could be a SAN. After connecting to the storage server, the database is tested on the transactions rate, using the transaction.db dialogue. The minimum requirement for the database is equal or more then 100 transactions pr second. There are three roles to fulfill.

5.6.5 Auditioning for the role as storage

The storage server first connect to the database, before testing the database. This will test to make sure the database is compatible with the storage solution. Then the storage server is tested writing and reading to the drive, with a minimum requirement in performance.

5.6.6 The complete manuscript

A more complex manuscript

```
scene frontpage_load {
    role loadbalancer
    support_roles webserver1,webserver2
    dialogue {
       [loadbalander]: connect.load [webserver1][webserver2]
       [loadbalander]: perform.load [webserver] /home.php 50
    }
    improv {
      monitoring.pl
    }
}

scene frontpage_web {
    role webserver
    roles_available 6
    support_roles middleware, database, storage
    dialogue {
       [webserver]: connect.middleware [middleware]
       [webserver]: benchmark.web 50 /index.php
    }
    improv {
      monitoring.pl
    }
}
```

```
scene frontpage_middle {
    role middleware
    support_roles webserver, database, storage
    dialogue {
       [webserver]: connect.db [database]
       [webserver]: content.web /index.php Lorem
       [webserver]: benchmark.web 12 /index.php
       [middleware]: services.middle
    }
    improv {
      monitoring.pl
    }
}

scene frontpage_database {
    role database
    roles_available 3
    support_roles storage
    dialogue {
    [database]: connect.db [storage]
    [database]: transactions.db 100
    }
    improv {
      monitoring.pl
    }
}

scene frontpage_storage {
    role database
    support_roles database
    dialogue {
    [database]: connect.db [storage
    [database]: transactions.db 100
    [storage]: read.storage 200MB
    [storage]: write.storage 100MB
    }
    improv {
      monitoring.pl
    }
}
```

5.6.7 After the Audition

At the completion of Audition, one will ha a fully deployed, quality and compatibility controlled, multi layered environment, consisting of six different layers. The environment have been tested for compatibility for

services running on the different layers, but also the additional services already integrated into the environment. The network will have the exact configuration as the shown in figure 2.2

5.7 Analysis of execution

Using the wordpress example, an Audition where successfully executed. The example did not show the all the feature of Audition, however, it proved that the prototype works according to the models created.

The Audition used to test the wordpress environment, with four different hosts and one supporting host, used 26 minutes to complete. In the wordpress scenario there were three dialogues and two improvisations performed by the different hosts.

5.7.1 She simplicity in Audition

The more complex manuscript created and deployed 12 hosts, with five different roles. Each combination between selected AMI's and instances was tested, for every role. The different roles where tested, on both performance but also compatibility with the layer over and under the role and with services existing in the environment.

The manuscript contains 61 lines, including bracket ending and space between different scenes. With the more complex example, the Audition used in average 5 lines for each host in the environment. To fully understand the framework and to see how the system scale, think of an example: You want the same environment, but you need more hosts. Three loadbalancers, 15 webservers, five middleware severs, 10 databases and 10 storage server. To create this using Audition would require a manuscript consisting of 64 lines, because three more scenes needed the roles_available option. From there to the next scaling of the environment would require no more added lines.

5.7.2 Cost a more complex example Audition

If one uses the same four instances, introduced in the wordpress example, however this time with three different AMI's, the cost of an full Audition using the manuscript created in section 5.6.6, would be:

$$((5 * 0.060) + (5 * 0.120) + (5 * 0.240) + (5 * 0.145)) * 3 = \$8.475.$$

However, there are also some supporting roles needed:

$$8 * 0.240 * 2 = \$3.84.$$

The assumptions is that each supporting role is up for two hours to complete the scene.

The total cost for the audition is $12.315. For this price, one gets a fully deployed, multi layered environment. The environment consists of 5 layers, and the environment is performance and compatibility tested between the different layers and with the existing solutions. All this for less than $13 and with minimal effort from the system administrator, is not costly.

Chapter 6

Discussion

This chapter will discuss all aspects regarding this project, approach, design, prototype, result and the overall process. An answer to the problem statement will be proposed, based on the processes leading up to this chapter.

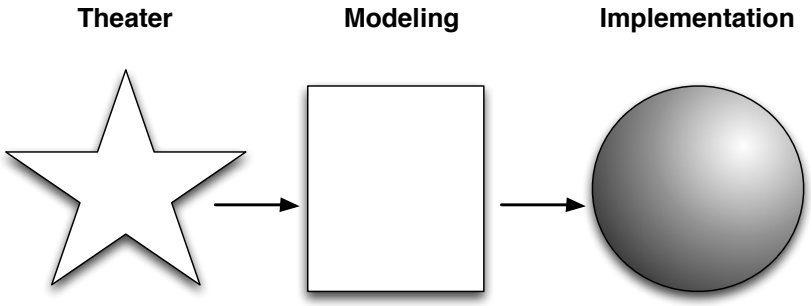

Figure 6.1: The process of inheritance between two different research areas

Figure 6.1 show the inheritance processes. The process resulting in R1 and R2.

6.1 System administration based on theater

Inheriting processes from a completely different field has different strengths and weaknesses, and merging the processes has different obstacles.

6.1.1 A system administrator approaching theater

Being a system administrator with little knowledge about theater, the field of theater was harder to learn than previously thought. Theater is a vast area of knowledge and experience. To get an understanding of the area, different methods were used. An interview with Hanne-Marte Sørlie, books and other written materials recommended by professionals.

Peoples assumptions

At first contact with various professionals, most though the request was initiated by an actor after job information. Therefore, establishing a dialogue was harder then expected. However, by having to explain the project repetitively to non-professional system administrator, helped to get a better understanding of the thesis.

6.1.2 The purpose of using theater

The goal of using the field of theater and the processes within, was to help simplify release management in system administration. To explore the possibility for an inheritance between the two fields, and would the outcome be simpler to use then existing solution? Are there processes that will explain and design a perfect framework.

6.1.3 Strength of theater

Theater productions have been using their processes for centuries, therefore the processes have a long track record of success. Each play is different, each character is different, each scene is different, the monologues are different. Considering all this, the process behind a play is still the same. Small modifications to the process have been done over the years, however the overall processes have kept their general form over time.

The processes have been adopted by similar fields of work like, the movie and television industry. They adopted the processes behind theater, and the adopted work is still used today.

Scalability

The processes from theater work on plays of all sizes. They work for professionals, amateurs and even for school plays. This proves that the processes are scalable and will suit any project, no matter size, large or small. Some adjustments to the processes are needed, for example, a school play involving small children.

Simplicity

System administration analogies are for most people without special knowledge in the field hard to understand, because system administration is a large and technical field. Non-professional do not possess enough knowledge in system administration to understand and relate technical terms and solutions. By relating system administration terms to other similar terms in fields more widely known, overall information and the relation between processes is easier to understand. This method has be used and proven functional in the past [37] [27].

Theater is culture, an cultural activity to perform and to watch. Theater is also used in schools and after school activities, this means that humans get first hand experience about theater from a young age. By using theater, most people will have some understanding of the processes. Combining the two fields of knowledge a technical and complex solution in system administration, can be explained to a person without education in computing. More precise, a technical and complex solution can be explained to persons in a company without a technical background.

6.1.4 Weakness of theater

The two different fields are completely different. This means that the fields do not often cross, in a processional way. This results in that people with a high level of knowledge in both fields are rare.

The field of theater is based on creativity and the human drive to evolve and make it better, the people in the industry are playful and spontaneous. Setting up a play is a creative process, and the final project evolves during each step. In computer science the human part can be creative, a computer, however, is not creative and can no evolve without human interactions.

Learning the field of theater, one discovers there is more to learn then first though. There are terms, methods and processes. There are also the reasons behind a process, and the wanted outcome of a process. The theater field has different areas, same as for the field of computing. There are high levels of education in acting, directing, writing, scenography and set design. Therefore to gain an understanding behind and about the processes in theater, requires a lot reading and learning from people with a understanding of the field. Knowledge within theater can be found by reading and doing research, but the methods are not one way or the other. In computer science there are one or zero, yes or no, right or wrong. In theater there are different levels, there is most of the time many ways of doing things.

Humans have personalities, and all persons are a little different. This also applies to actors, or even applies extra to actors. An actor might be or become a prima donna or diva, with resulting in conflicts between the cast. This conflict between the cast, might delay the premier, the final product is influenced by conflict, or result in actors or other personnel quiting.

To implement the diva into system administration, could as an example be: Programing a webserver to suddenly and without warning stop talking to the middleware. If testing monitoring tools or disaster recovering, this feature could be beneficial. However, for the Audition this feature would not bring any wanted side effects with implementing a diva into Audition.

Limitations

Many theaters have a high focus on creativity and delivering the best performance, the economical part is often overlooked. Resulting in that many theaters depend on extra funding to not be closed down because of negative balance.

6.1.5 Inheritance challenges

In theater the advantage and weakens is the human aspect. A person will make a mistake at some point, it is only to be expected. One attempt to limit the mistakes using rehearsals and trial performances. The advantages with interaction between people is if the person makes a mistake, the other person can react and help to minimize the impact. To transfer this feature into computer science is to wast for this thesis, this feature is a separate research area. Since the mistake can happen in all stages, the computer would need to know all errors possible to happen and a solution for that.

6.1.6 Expectations for theater

Start working on this thesis one had some expectation about the theater, these expectations were meet. However, improvisation proved to be much more used and the area was deeper then expected. Discovering the different usage helped with designing the framework. Not only allowing for performance testing, but discovering the uniqueness of the system. Improvisation helps the system administrator get a feeling for the server.

6.2 Defining "simpler"

Defining the term simpler is hard. To define simple, is something that is easy. However, a hard solution to a solution almost impossible can also be defined as simpler. By that statement, simpler is something that is easier then something else.

Then the question is, how to measure if something is simpler then something else? There are many different ways of this, time spent on the process, lines of code, how understandable something is, reusability or usability. There are different ways of collecting the information, survey, line count, time testing or implementation time.

The processes in theater are not used to make something simpler, it is too ensure quality and the best performance. However, by adopting processes from theater, one hope in the process to inherit simplicity through peoples common knowledge about theater. By using a common area of knowledge, will help to explain complex solutions simpler.

6.3 Adopting theater in system administration

The interview resulted in a set of models which was created in R1. Based on the models, a framework where implemented in R2. The different stages designed, were implemented in different phases. This allowed for each stage could be tested, before a new stage was added.

After the implementation of all stages were completed, a full scale test was executed. The test was a simulation of a real environment. In the test the wanted role was a webserver, and the webserver needed help from a database to serve the webpage. Different hosts auditioned for the role, if a host did not execute the dialogue correctly the host was eliminated. The hosts not eliminated went through an improvisation part, were the system administrator can get a better understanding of the host. One of the surviving hosts, would then be selected for the role. The selection was based on the overall cost of different hosts.

The implementation although simplistic, show that the models designed worked according to plan. Demonstrating the Audition can work using a script. The system administrator is still a vital part of the process. The manuscript have to be written and puppet classes need to be created if not already existing.

After the Audition is completed, the system administrator have to maintain the environment until the next Audition. The day-to-day tasks for the system administrators has not been eliminated, and is still a vital part of the environment.

6.4 Usage of plugins

During the Audition the dialogue used the plugin with success. However, the plugins are not confined to only the ones created for this thesis. System administrators can create their own plugins, with the wanted content. The plugin wrap in features, usable in the dialogue. By standardizing how the dialogue is written in Audition, it is easy for system administrators to reuse plugins written by other.

6.5 Performance expectation

When conducted a full scale test of the prototype using the wordpress environment, no AMI-images or hardware types completed the audition with success. The problem occurred when shifting from a low scale Audition, including only one web server and no supporting host. After giving the supporting host more and better CPU cores and more memory, still none

of the hosts auditioning did complete the audition with success.

At that stage the benchmark limit where lowered, and after a new Audition all hosts completed with success.The problem was that the expectations to the performance of the environment was such that they where not fulfilled buy the actual performance. However, performance is not one of the key elements of this thesis. Based on this the expected environment performance were lowered to match the performance of the environment.

6.6 Variations in results

When collecting the results from the full scale wordpress environment and from initial prototype testing, there were occasionally a different result in the cast selection. After bug testing the prototype for any errors within the selection phase of Audition, the focused shifted towards interferences or differences outside of Audition.

The problem was the instances provided by Amazon EC2, they had some variation in performance. This fluctuation in performance may be enough to decrease the performance of the host, so that the minimum requirements is no longer meet. The fluctuation in performance using Amazon EC2 have been documented in other research papers [90].

The change in the selected cast is therefore a sign that the framework is working according to the model created, and not an error within the selection process of an audition.

6.7 The cost of Audition

The economical cost of Audition, I would argue is inexpensive. As stated in section 5.7.2, the cost of Audition using the complex example manuscript cost a little over 12 dollars. The return for 12$ is: A environment consisting of 12 host, which have been tested on performance and compatible with all the new hosts in the environment. The whole environment is optimized on price, however, the system performs better than the minimum performance wanted from the network.

However, it is important to know that the 12$ is for the Audition. The running cost of the network after the Audition is completed, will be in addition according to the Amazon EC2 pricing.

6.8 The thesis experience

My initial thought was "How can theater help system administration, and how can I explain it?". The project started with learning about the field of

theater, and the field proved to be a larger then expected. Luckily i met a person with great knowledge about the theater field, and this started the creative phase of creating models.

Implementing theater processes into a framework, is time consuming. Not all ideas could be used, and I learned that it is never wrong to go back and try again. The thesis required thinking outside the box, to be creative and see solutions. To see how Audition evolved and the potential within it, was one of the best experience during this thesis.

6.9 The fields affected by this work

Using visualization technology the well establish 3-5 year hardware lifetime is obsolete. With cloud computing one can lease a server on a hourly basis, instead of buying the hardware. Therefore, the environment can change frequently without any large economical cost. The system administrator wants to maintain a stable quality of service, while at the same time considering the economical situation. Thus, the goal is to have an environment which meets the performance generated by users, to a lowest price possible. This is a win-win situation for both sides.

A way to enable for these frequently changes in environment and software, without adding more work pressure to the system administrator, is using an automated release management tool. The usage of automated release management has increased after system administrator discovered the benefits. Automated release management enables for scaling, the number of system administrator is stable as the number of servers and services increases.

6.9.1 Broaden the understanding of system administration

During the thesis, it has been shown that using terminology form a different field of research has been used with success. The key is using a well established term, to explain a complex system within system administration. This enables for non technical personnel to understand complex systems, and based on this can better understand the reasons behind, how system administration works.

As an example, think of the following scenario: In a meeting the system administrator is told about a completely new design that is ready to be implemented as soon as possible. The system administrator explains, that in order for the implementation to be deployed in production, different EC2 instances need to be tested. Then software validation, to ensure the software versions for the new page need to work with the environment. Next a test implementation, performance validation, quality assurance, then the

final deployment.

At that time, the non-technical person would understand little, and have little basis for setting a deadline. Instead, by referring to theater one could explain the process in a more understanding way: First there needs to be an audition to assess the quality of the actor, the audition will detect any compatibility problems regarding the excising actors. After the audition, the play need to be rehearsed, before the grand premiere.

6.9.2 Continuous software releases

Automating the deployment of large scale environments in combination with quality control, and reducing the complexity of the system. This is not an attempt to reduce the need for system administrators. It is more an attempt to release the system administrator from static and time consuming tasks. Furthermore, trying to embrace DevOps thinking of constantly new releases, combining this thinking with system administration. Instead of an software release, why not a complete environment release?

6.10 Future work

The framework is designed with a focus on extensibility, this enables for the system administrator to add features wanted with little effort.

A test implementation was presented in thesis, however to use Audition in a real working environment would result in more information about how the framework works. Generating more data about the Audition, as the environments evolves. This data could help the framework to evolve and design and implement new features.

Allow for integration to Amazon AWS marketplace, this would allow for third-parties to register an image which could be used by the Audition. This feature can not be implemented before Amazon provide an API to the marketplace. However, when the API is released, the images could be assigned a general role-description. A general role-description like web server, which the Audition can use in the casting call.

The implementation is limited to using Amazon EC2 virtualization. Implementing the possibility for choosing different technologies, could lead to cost saving using a local cloud technology, for instance OpenStack or Xen. Implementing new virtualization technologies would create a larger marked for third-parties images.

Chapter 7

Conclusion

Cloud technology and DevOps have and will continue to influence the field of system administration in the future, due to the change technology and how users interact with technology. Adopting processes from theater has been a small step in the right direction regarding automated system deployment in a cloud environment.

Inheriting elements from the theater enabled for complex technical solution to be simplified, and in the process opened up for a new thinking regarding cloud computing. By not owning the hardware, enables for a continuous evolving environment. The processes inherited have resulted in designing a prototype to demonstrate the possibility within the design.

The thesis discovered a new method to quality ensure, test and select the ideal virtual machine based on performance quality and price for arbitrarily large setups, modeled into a concept that is simple to understand and convey. The prototype was demonstrated on two cases, one wordpress blog comprising of a database and server and one of a multi-tiered web-based application with five separated layers of operation. A fully operational extendiable framework, with a focus on simplicity is presented as a solution for the problems with release management in cloud computing.

The processes behind the theater have survived for centuries, the processes focus on quality. Which have resulted in millions of people laughing, crying and smiling, over the last millennium. Using a different field to share resources, can produce a positive or negative outcome. So far theater has not been a part of system administration, but now that door has been opened, different thoughts, strengths and weaknesses have been detected. This thesis demonstrates how theater production processes can be used to reduce conceptual complexity in automated release management for web environments.

Bibliography

[1] Hussam Abu-Libdeh, Lonnie Princehouse and Hakim Weatherspoon. 'RACS: a case for cloud storage diversity'. In: *Proceedings of the 1st ACM symposium on Cloud computing*. ACM. 2010, pp. 229–240.

[2] John Allspaw and Jesse Robbins. *Web Operations: Keeping the Data on Time*. O'Reilly Media, 2010.

[3] Amazon. *Amazon EC2 Pricing*. http://aws.amazon.com/ec2/pricing/. [Online; accessed May 2013]. May 2013.

[4] *Amazon EC2 Instances*. http://aws.amazon.com/ec2/instance-types/. [Online; accessed May 2013]. May 2013.

[5] Paul Anderson. 'System configuration.' In: *Short Topics in System Administration* (2006).

[6] Paul Anderson. 'Towards a high-level machine configuration system.' In: *Proceedings of the 8th USENIX conference on System administration* (1993), pp. 19–26.

[7] *Apache*. http://www.apache.org/. [Online; accessed April 2013]. Apr. 2013.

[8] M. Armbrust et al. 'A view of cloud computing'. In: *Communications of the ACM* 53.4 (2010), pp. 50–58.

[9] William D Armitage, Alessio Gaspar and Matthew Rideout. 'A UML and MLN based approach to implementing a networking laboratory on a scalable Linux cluster'. In: *Journal of Computing Sciences in Colleges* 23.2 (2007), pp. 112–119.

[10] R. Barrett et al. 'Field studies of computer system administrators: analysis of system management tools and practices'. In: *Proceedings of the 2004 ACM conference on Computer supported cooperative work*. ACM. 2004, pp. 388–395.

[11] *BATS School of Improvisational Theatre*. http://www.improv.org. [Online; accessed May 2013]. May 2013.

[12] Kyrre Begnum. 'Managing large networks of virtual machines'. In: *Proceedings of the 20th Large Installation System Administration Conference*. 2006, pp. 205–214.

[13] Kyrre Begnum, Nii Apleh Lartey and Lu Xing. 'Cloud-oriented virtual machine management with mln'. In: *Cloud Computing*. Springer, 2009, pp. 266–277.

[14] Kyrre Begnum, John Sechrest and Steven Jenkins. 'Getting more from your virtual machine'. In: *Journal of Computing Sciences in Colleges* 22.2 (2006), pp. 66–73.

[15] Kyrre Begnum et al. 'Using virtual machines in system administration education'. In: *Proceedings of 4th International System Administration and Network Engineering Conference. System and Network Engineering*. 2004.

[16] B. Beizer. *Black-box testing: techniques for functional testing of software and systems*. John Wiley & Sons, Inc., 1995.

[17] Lynda Belt and Rebecca Stockley. *Improvisation through theatre sports: a curriculum to improve acting skills*. Thespis Productions, 1991.

[18] T. Berners-Lee. 'Information management: A proposal'. In: (1989).

[19] T. Brogårdh. 'Present and future robot control development—An industrial perspective'. In: *Annual Reviews in Control* 31.1 (2007), pp. 69–79.

[20] John Brown. *What is Theatre?: An Introduction and Exploration*. Focal press, 1998.

[21] Justin F Brunelle and Michael L Nelson. 'Evaluating the SiteStory Transactional Web Archive With the ApacheBench Tool'. In: *arXiv preprint arXiv:1209.1811* (2012).

[22] M. Burgess and R. Ralston. 'Distributed resource administration using cfengine'. In: *Software: practice and experience* 27.9 (1997), pp. 1083–1101.

[23] George Candea, Stefan Bucur and Cristian Zamfir. 'Automated software testing as a service'. In: *Proceedings of the 1st ACM symposium on Cloud computing*. ACM. 2010, pp. 155–160.

[24] Y Diao Chess et al. 'Managing Web server performance with AutoTune agents'. In: *IBM Systems Journal* 42.1 (2003), pp. 136–149.

[25] Liviu Ciortea et al. 'Cloud9: A software testing service'. In: *ACM SIGOPS Operating Systems Review* 43.4 (2010), pp. 5–10.

[26] R.J.W. Cline and KM Haynes. 'Consumer health information seeking on the Internet: the state of the art'. In: *Health education research* 16.6 (2001), pp. 671–692.

[27] Fred Cohen. 'Computer viruses: theory and experiments'. In: *Computers and Security* (1987), pp. 22–35.

[28] J. Conallen. 'Modeling Web application architectures with UML'. In: *Communications of the ACM* 42.10 (1999), pp. 63–70.

[29] Andrew Conry-Murray. *The Public Cloud: Infrastructure As A Service*. TechWeb, 2009.

[30] H.M. Deitel et al. 'Web services: a technical introduction'. In: (2003).

[31] Y. Deswarte and D. Powell. 'Internet security: an intrusion-tolerance approach'. In: *Proceedings of the IEEE* 94.2 (2006), pp. 432–441.

[32] M.S. Deutsch. *Software verification and validation: Realistic project approaches*. Prentice Hall PTR, 1981.

[33] R.H. Dunn and RH Dunn. *Software defect removal*. McGraw-hill New York et al., 1984.

[34] P.N. Edwards. 'Infrastructure and modernity: Force, time, and social organization in the history of sociotechnical systems'. In: *Modernity and technology* (2003), pp. 185–225.

[35] J. Farrell and G. Saloner. 'Standardization, compatibility, and innovation'. In: *The RAND Journal of Economics* (1985), pp. 70–83.

[36] Harald Fecher et al. '29 new unclarities in the semantics of uml 2.0 state machines'. In: *Formal Methods and Software Engineering* (2005), pp. 52–65.

[37] Johan Finstadsveen. 'If your Webserver was an Animal, how would you describe it?' In: (2011).

[38] David A Forsyth and Jean Ponce. *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference, 2002.

[39] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[40] M. Fowler and J. Highsmith. 'The agile manifesto'. In: *Software Development* 9.8 (2001), pp. 28–35.

[41] S. Fox and J. Beier. *Online banking 2006: surfing to the bank*. Pew Internet & American Life Project, 2006.

[42] Robert France et al. 'The UML as a formal modeling notation'. In: *Computer Standards & Interfaces* 19.7 (1998), pp. 325–334.

[43] Mikell P. Groover. *Automation, Production Systems, and Computer-Integrated Manufacturing*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007. ISBN: 0132393212.

[44] K. Hafner and M. Lyon. *Where wizards stay up late: The origins of the Internet*. Simon and Schuster, 1998.

[45] V. Hendrix, D. Benjamin and Y. Yao. 'Scientific Cluster Deployment and Recovery–Using puppet to simplify cluster management'. In: *Journal of Physics: Conference Series*. Vol. 396. 4. IOP Publishing. 2012, p. 042027.

[46] Alison Hodge. *Twentieth-Century Actor Training*. Taylor and Francis, 2000.

[47] Ming Huo et al. 'Software quality and agile methods'. In: *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*. IEEE. 2004, pp. 520–525.

[48] Rahman Md Jahidur. 'Investigating Configuration Management Tools Usage in Large Infrastructure'. In: (2012).

[49] Kyrre Begnum Johan Finstadsveen. 'What a webserver can learn from a zebra and what we learned in the process'. In: *CHIMIT '11 Proceedings of the 5th ACM Symposium on Computer Human Interaction for Management of Information Technology* (2011).

[50] Keith Johnstone. *Impro: Improvisation and the theatre*. Routledge, 1981.

[51] O. Kephart and D. M. Chess. 'The vision of autonomic computing.' In: *Computer* (2003).

[52] Eric Knorr and Galen Gruman. 'What cloud computing really means'. In: *Infoworld, April* 7 (2008).

[53] Bruce Kogut and Anca Metiu. 'Open-source software development and distributed innovation'. In: *Oxford Review of Economic Policy* 17.2 (2001), pp. 248–264.

[54] D. Kotz and R.S. Gray. 'Mobile Agents and the Future of the Internet'. In: *Operating systems review* 33.3 (1999), pp. 7–13.

[55] Charles W Krueger. 'Software reuse'. In: *ACM Computing Surveys (CSUR)* 24.2 (1992), pp. 131–183.

[56] Anthony A. Lekkos and Carl M. Peters. 'How to develop module logic using pseudo-code and stepwise refinement'. In: *Proceedings of the 15th Design Automation Conference*. DAC '78. Las Vegas, Nevada, USA: IEEE Press, 1978, pp. 366–370. URL: http://dl.acm.org/citation.cfm?id=800095.803116.

[57] Alexander Lenk et al. 'What's inside the Cloud? An architectural map of the Cloud landscape'. In: *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. IEEE Computer Society. 2009, pp. 23–31.

[58] Josh Lerner and Jean Tirole. 'Some simple economics of open source'. In: *The journal of industrial economics* 50.2 (2002), pp. 197–234.

[59] Ang Li et al. 'CloudCmp: comparing public cloud providers'. In: *Proceedings of the 10th annual conference on Internet measurement*. ACM. 2010, pp. 1–14.

[60] *Lighttpd*. http://www.lighttpd.net/. [Online; accessed April 2013]. Apr. 2013.

[61] J. Loope. *Managing Infrastructure with Puppet*. O'Reilly Media, 2011.

[62] C. Mann and F. Maurer. 'A case study on the impact of scrum on overtime and customer satisfaction'. In: *Agile Conference, 2005. Proceedings*. IEEE. 2005, pp. 70–79.

[63] David Mosberger and Tai Jin. 'httperf—a tool for measuring web server performance'. In: *ACM SIGMETRICS Performance Evaluation Review* 26.3 (1998), pp. 31–37.

[64] G.J. Myers, C. Sandler and T. Badgett. *The art of software testing*. Wiley, 2011.

[65] AB MySQL. *MySQL*. 2001.

[66] Thomas H Naylor and Joseph Michael Finger. 'Verification of computer simulation models'. In: *Management Science* 14.2 (1967), B–92.

[67] J. Nielsen. *Hypertext and hypermedia*. Vol. 263. Academic Press San Diego, CA, 1990.

[68] Daniel Nurmi et al. 'The eucalyptus open-source cloud-computing system'. In: *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*. IEEE. 2009, pp. 124–131.

[69] Raymond Obstfeld. *Novelist's Essential Guide to Crafting Scenes*. Writers Digest Books, 2000.

[70] F. Önnberg. 'Software Configuration Management: A comparison of Chef, CFEngine and Puppet'. PhD thesis. University of Skövde, 2012.

[71] *Oslo National Academy of the Arts*. http://www.khio.no/Engelsk/. [Online; accessed May 2013]. May 2013.

[72] T. Ostrand. 'White-Box Testing'. In: *Encyclopedia of Software Engineering* ().

[73] S. Pandey. 'Investigating Community, Reliability and Usability of CFEngine, Chef and Puppet'. In: (2012).

[74] Patrice Pavis and Christine Shantz. *Dictionary of the theatre: Terms, concepts, and analysis*. University of Toronto Press, 1998.

[75] *Red Hat CloudForms*. http://www.redhat.com. [Online; accessed May 2013]. May 2013.

[76] *Red Hat Unveils Hybrid Cloud, PaaS Plans*. http://www.pcworld.com. [Online; accessed May 2013]. May 2013.

[77] Gianna Reggio and Roel Wieringa. 'Thirty one Problems in the Semantics of UML 1.3 Dynamics'. In: *OOPSLA*. Vol. 99. Citeseer. 1999.

[78] Matthew Sacks. 'DevOps Principles for Successful Web Sites'. In: *Pro Website Development and Operations*. Springer, 2012.

[79] Viola Spolin. *Improvisation for the theater: A handbook of teaching and directing techniques*. Northwestern University Press Evanston, IL, 1983.

[80] Viola Spolin. *Theater games for the classroom: A teacher's handbook*. TriQuarterly Books, 1986.

[81] Viola Spolin, Carol Bleackley Sills and Rob Reiner. *Theater games for rehearsal: A director's handbook*. Northwestern University Press, 2011.

[82] Viola Spolin, Paul Sills and Carol Sills. *Theater games for the lone actor*. Northwestern University Press, 2001.

[83] *The Julilliard school*. http://www.juilliard.edu. [Online; accessed May 2013]. May 2013.

[84] Aleksey Tsalolikhin. *Summary, Configuration Management Summit*. 2010.

[85] Tugkan Tuglular. 'Test case generation for firewall implementation testing using software testing techniques'. In: *Proceedings of the International Conference on Security of Inform. and Networks*. 2008, pp. 196–203.

[86] A.M. Turing. 'Computing machinery and intelligence'. In: *Mind* 59.236 (1950), pp. 433–460.

[87] J.E. Tyvand and Universitetet i Oslo Institutt for informatikk. *On the Predictability of Server Resources in Online Games: An Investigative Approach*. J.E. Tyvand, 2011. URL: http://books.google.no/books?id=U5NLMwEACAAJ.

[88] F.I. Vokolos and E.J. Weyuker. 'Performance testing of software systems'. In: *Workshop on Software and Performance: Proceedings of the 1 st international workshop on Software and performance*. Vol. 12. 16. 1998, pp. 80–87.

[89] W3C. *Web services architecture requirements, W3C Working Draft*. http://www.w3c.org/TR/2002/WD-wsa-reqs-20020429. (Visited Jan. 2013). 2002.

[90] Guohui Wang and TS Eugene Ng. 'The impact of virtualization on network performance of amazon ec2 data center'. In: *INFOCOM, 2010 Proceedings IEEE*. IEEE. 2010, pp. 1–9.

[91] Lizhe Wang et al. 'Scientific cloud computing: Early definition and experience'. In: *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*. IEEE. 2008, pp. 825–830.

[92] Steve Weber. *The success of open source*. Vol. 368. Cambridge Univ Press, 2004.

[93] Aaron Weiss. 'Computing in the clouds'. In: *Computing* 16 (2007).

[94] D. Zamboni. *Learning CFEngine 3: Automated System Administration for Sites of Any Size*. O'Reilly Media, 2012.

[95] Jinzy Zhu et al. 'IBM cloud computing powering a smarter planet'. In: *Cloud Computing* (2009), pp. 621–625.

Appendix A

The different plugin scripts

```
1  sub connect_db {
2
3      my $hostname = $_[0];
4      my $params = $_[1];
5
6      open(SSH,"ssh␣$hostname␣/root/connect-db.sh␣↪
           ↩ $params␣|");
7      while ( my $line = <SSH> ){
8
9              if ( $line =~ /OK/ ){
10                 close(SSH);
11                 return 1;
12             }
13         }
14     close(SSH);
15     return 0;
16 }
17
18 1;
```

```
1  sub content_web {
2
3      my $hostname = $_[0];
4      my $string = $_[1];
5      ( my $url, my $content ) = split /\s+/,$string;
6
7      my $checkurl = "http://${hostname}$url";
8      print "Url: '$checkurl' content: '$content'\n";
9
10     open(WGET,"wget -O - -q $checkurl |");
11     while ( my $line = <WGET> ){
12           if ( $line =~ /$content/ ){
13              close(WGET);
14              return 1;
15           }
16     }
17     close(WGET);
18     return 0;
19 }
20
21 1;
```

```
1  sub benchmark_web {
2
3    my $webip = $_[0];
4    my $arguments = $_[1];
5    my $num_call = 1;
6    my $num_conn;
7    my $limit;
8
9    ( my $rate, my $page ) = split /\s+/,$arguments;
10
11   $limit = ($rate-2);
12   $num_conn = ($rate*10);
13
14   open(HTTP,"httperf␣--server=$webip␣--uri=$page␣↪
       ↩ --port␣80␣--rate␣$rate␣--num-conn␣↪
       ↩ $num_conn␣--num-call␣$num_call␣|");
15   while ( my $line = <HTTP> ){
16     print "LINE:␣$line";
17     if ( $line =~ ↪
         ↩ /Request\s+rate:\s+(\d+.?\d?)\s.*/ ){
18       if ($1 >= $limit) {
19         close(WGET);
20         return 1;
21       }
22     }
23   }
24   close(HTTP);
25   return 0;
26 }
27 1;
```

```
#!/usr/bin/perl

use Data::Dumper;
use Cwd qw();

my $project = "macbeth";

our %PLUGIN_LIST = ();

findPlugins();

foreach (keys %PLUGIN_LIST ){
  my $file = $PLUGIN_LIST{$_};
   print "$file\n";
   require "$file";
}

open(FILE,$ARGV[0]) or die("AAAAHHH $!\n");

my %SCENES;
my %RESULT;
my %RANK;

my @supportip;
my @supportdns;
my @supportr;


my $mln_file;
my $importantrole = 0;
my $importantinprov = 0;
my $nrlinesdialog=0;

while ( my $line = <FILE> ){
   if ( $line =~ /mln_file\s(\S+)/ ){
     $mln_file=$1;
     print "File is named: $mln_file\n";
     }
   if ( $line =~ /scene\s+(\S+)\s+{/ ){
     my $scene = $1;
     print "found scene $scene\n";
     $line = <FILE>;
     while ( not $line =~ /}/ ){
       print "parsed line $line";

       if ( $line =~ /dialogue\s{/ ){
         my $dialogue = "dialogue";
          $line = <FILE>;
```

```
49          my $dialognr = 1;
50            while ( not $line =~ /}/ ){
51              $SCENES{$scene}{$dialogue}{$dialognr} ↪
                    ↩ = $line ;
52              $dialognr++;
53              $line = <FILE>;
54             }
55             $nrlinesdialog=($dialognr-1);
56           }
57
58        if ( $line =~ /improv\s{/ ){
59          my $improv = "improv";
60          $line = <FILE>;
61          my $improvnr = 1;
62          $importantinprov = 1;
63          while ( not $line =~ /}/ ){
64            $SCENES{$scene}{$improv}{$improvnr} = ↪
                  ↩ $line ;
65            $improvnr++;
66            $line = <FILE>;
67            }
68         }
69
70
71         if ( $line =~ /\s+(support_roles)\s+(.*)/ ){
72           $line =~ s/\r\n//;
73           $line =~  /\s+(support_roles)\s+(.*)/;
74           my @support = split(',', $2);
75           $m=0;
76           while ($m <= $#support) {
77             $srole=$support[$m];
78             $SCENES{$scene}{$1}{$srole}="1";
79             $m++;
80             }
81           $importantrole = 1;
82         }
83
84
85         elsif ( $line =~ /(\S+)\s+(.*)/ ){
86           $SCENES{$scene}{$1} = $2;
87         }
88       $line = <FILE>;
89       }
90    }
91 }
92
93 close(FILE);
94
```

```
95 open FILE, "<", "casting" or die("AAAAHHH␣$!\n");
96 my @IMAGES;
97 while ( my $line = <FILE> ){
98     push(@IMAGES, $line);
99 }
100 close(FILE);
101
102 chomp(@IMAGES);
103
104 my $filetowrite="MLN_CONFIG";
105 my $filetoread="MLN_TEMPLATE";
106 my $mlnhosts=$#IMAGES;
107
108 open MLN_TEMPLATE, "<", $mln_file or die("Where␣↪
    ↩ is␣the␣file,␣arrrr␣$!\n");
109 open MLN_CONFIG, ">", "mln-config.mln" or ↪
    ↩ die("The␣file␣could␣not␣be␣created,␣arrrr␣↪
    ↩ $!\n");
110 while (<$filetoread>) {
111    print $filetowrite $_;
112 } # close while template
113
114 foreach my $scene ( keys %SCENES ) {
115   $i=0;
116   while ($i <= $mlnhosts) {
117     $hostinfo = "$SCENES{$scene}{role}";
118     $hostinfo =~ /(\w*)/;
119     print $filetowrite "host␣$IMAGES[$i]-$1␣\{\n";
120     print $filetowrite "␣␣superclass␣" ↪
        ↩ .$SCENES{$scene}{role}. "\n" ;
121     print $filetowrite "␣␣ec2␣\{\n";
122     print $filetowrite "␣␣␣␣type␣$IMAGES[$i]\n";
123     print $filetowrite "␣␣␣␣ami␣ami-3fec7956\n";
124     print $filetowrite "␣␣␣␣␣␣\}\n";
125     print $filetowrite "\}\n";
126     print $filetowrite "\n";
127
128     $i++;
129   }  #end mlnhosts while
130
131
132  if ( $importantrole == "1" ){
133   while (($key) = ↪
       ↩ each($SCENES{$scene}{support_roles})){
134     print $filetowrite "host␣" .$key. "␣\{\n";
135     print $filetowrite "␣␣superclass␣" .$key. ↪
        ↩ "\n" ;
136     print $filetowrite "␣␣ec2␣\{\n";
```

```
137       print $filetowrite "␣␣␣␣type␣m1.medium\n";
138       print $filetowrite "␣␣␣␣ami␣ami-def89fb7\n";
139       print $filetowrite "␣␣␣␣␣␣\}\n";
140       print $filetowrite "\}\n";
141       print $filetowrite "\n";
142   }
143   }
144
145   } #end scene forwach
146
147   close MLN_CONFIG;
148   close MLN_TEMPLATE;
149
150   system("mln␣build␣-r␣-f␣mln-config.mln");
151
152   $mlnhosts=$#IMAGES;
153   $testdatabase;
154
155   foreach my $scene ( keys %SCENES ){
156     print "Scene\n";
157     $p=0;
158     if ($importantrole == 1) {
159     while (($key) = ↪
          ↩ each($SCENES{$scene}{support_roles})) {
160       $output = `mln start -p $project -h $key`;
161       sleep(60);
162       $output =~ /.*\s(i-\S*)\s.*/;
163       my $ec2id = $1;
164       $ec2din = `ec2din $ec2id`;
165       $ec2din =~ /.*(ec2-.*.amazonaws.com).*/;
166       $sdns = $1;
167       $supportdns[$p] = $1;
168       $outputsupport = `ssh -i ↪
             ↩ /home/ubuntu/.ssh/enterprise -o ↪
             ↩ "StrictHostKeyChecking␣no" ↪
             ↩ ubuntu\@$sdns "ifconfig" `;
169       $outputsupport =~ ↪
             ↩ /\s+inet\s+addr:(10.\d+.\d+.\d+)\s+/;
170       $supportip[$p] = $1;
171       $testdatabase = $1;
172       $supportr[$p] = $key;
173       $p++;
174       sleep(60);
175
176
177     }
178    }
179
```

```
180   $j=0;
181   while ($j <= $mlnhosts) {
182     $hostinfo = "$SCENES{$scene}{role}";
183     $hostinfo =~ /(\w*)/;
184     $hostrole = $1;
185     $imagename = "$IMAGES[$j]-$1";
186     print "imagename: $imagename\n";
187     print "projectname: $project\n";
188     $output = `mln start -p $project -h $imagename`;
189     $output =~ /.*\s(i-\S*)\s.*/;
190     my $ec2id = $1;
191     $ec2din = `ec2din $ec2id`;
192     $ec2din =~ /.*(ec2-.*.amazonaws.com).*/;
193     my $hostdns = $1;
194     print "$hostdns\n";
195     sleep(60);
196     $puppetsuccsess = 0;
197     $hostrun=0;
198     while ($puppetsuccsess != 1) {
199      $outputhost = `ssh -i
          ↪ /home/ubuntu/.ssh/enterprise -o
          ↪ "StrictHostKeyChecking no"
          ↪ ubuntu\@$hostdns "cat /tmp/puppet.txt
          ↪ 2>/dev/null" `;
200       if ( $outputhost and $outputhost != 4 &&
            ↪ $outputhost != 6 ) {
201         $puppetsuccsess = 1;
202         print "puppet succsessfully installed\n";
203       }
204       else {
205           $hostrun++;
206           sleep(60);
207           print "cant connect to host, run nr
                ↪ $hostrun\n";
208           if ($hostrun == 10) {last;}
209       }
210      }
211
212      $wholeline;
213      $restemp;
214      $dropimprov=0;
215      $nrimgdialog=0;
216      $dialognr=1;
217      if($puppetsuccsess == 1 ) {
218        print "starting bacnmark testing\n";
219        @dialog = keys %{$SCENES{$scene}{dialogue}};
220        $dialoguelines = scalar (@dialog);
221        $d=1;
```

```
222         $dialoguehost=0;
223         $dialogueplugin=0;
224         $dialoguecontent=0;
225         while ($dialoguelines >= $d) {
226           $wholeline = $SCENES{$scene}{dialogue}{$d};
227           $wholeline =~ ↪
                ↩ /\s+\[(\w+)\]:\s+(\w+\.+\w+)\s(.*)/;
228           $dialoguehost = $1;
229           $dialogueplugin = $2;
230           $dialoguecontent = $3;
231           $w=0;
232           my $inshost;
233           while (  $dialoguecontent =~ ↪
                ↩ /\[(database)\]/ ){
234             my $temphost = $1;
235             $u = $#supportr;
236             while ($w <= $u) {
237               if ($1 eq $supportr[$w]) {
238               $inshost = $supportip[$w] ; # find ↪
                    ↩ the IP
239             }
240             $w++;
241           }
242         $dialoguecontent =~ ↪
              ↩ s/\[$temphost\]/$testdatabase/;
243       } # end dialogue content while
244       $dialogueplugin =~ s/\./_/g;
245       $restemp = &$dialogueplugin("$hostdns" , ↪
            ↩ "$dialoguecontent");
246       if ($restemp == 0) {
247         print "done␣with:␣$imagename\n";
248         print "The␣host␣failed␣the␣dialog\n";
249         $dropimprov=1;
250         last;
251       }
252       else {
253         $nrimgdialog++;
254         print "done␣with:␣$imagename\n";
255         print "The␣host␣compelted␣the␣dialog␣with␣↪
              ↩ succsess\n"
256       }
257       $d++;
258     }
259
260     if ($nrimgdialog == $nrlinesdialog) {
261       $RESULT{$scene}{$hostrole}{$imagename}{$wholeline} ↪
            ↩ = $restemp;
262     }
```

```
263
264     if ($importantinprov == 1 and $dropimprov == ↪
            ↩ 0) {
265       my $improtemp = Cwd::cwd();
266       $path = "$improtemp/improv/";
267       print "starting improv testing\n";
268       $improfile = "improv-$imagename";
269       @idialog = keys %{$SCENES{$scene}{improv}};
270       $idialoguelines = scalar (@idialog);
271       $d=1;
272         while ($idialoguelines >= $d) {
273           $wholeline = $SCENES{$scene}{improv}{$d};
274           $wholeline =~ s/\r\n//;
275           $iwholeline = $wholeline;
276           $improvout= ↪
                ↩ &improv("$hostdns","$iwholeline");
277           $fpath = "$path$improfile";
278           open(MYFILE, ">>" , $fpath) or ↪
                ↩ die("AAAAHHHasd $!\n");
279           print MYFILE $improvout;
280           close (MYFILE);
281           $d++
282         }
283       } #end inprov
284
285      `ssh-keygen -f "/root/.ssh/known_hosts" -R ↪
           ↩ ec2-54-235-248-54.compute-1.amazonaws.com`;
286     }
287
288    $j++;
289    } #end mln hosts
290
291 } # end scene
292
293 @pricename;
294 @price;
295
296 open(PRICE, "pricelist.txt") or die("AAAAHHH $!\n");
297 while ( my $line = <PRICE> ){
298  ( my $name, my $price1 ) = split( /\s+/,$line);
299    print "name: $name and the price is: $price1\n";
300    push(@pricename, $name);
301    push(@price, $price1);
302 }
303 close(PRICE);
304
305 my $imagelist = $#pricename;
306 $imagelist = $imagelist+1;
```

```
307
308 foreach my $scene ( keys %RESULT ){
309
310     my %ROLEHASH = %{$RESULT{$scene}};
311
312     foreach my $resultrole ( keys %ROLEHASH ){
313       my %HNAMEHASH = %{$ROLEHASH{$resultrole}};
314       print "rolle: $resultrole\n";
315
316       foreach my $hname ( keys %HNAMEHASH ){
317           $n=0;
318           my %HBENCH = %{$HNAMEHASH{$hname}};
319           print "hname: $hname\n";
320           $hname =~ /(.*)-.*/;
321           $fimagetype = $1;
322           while ($imagelist >= $n) {
323             if ($fimagetype eq $pricename[$n]) {
324               $RANK{$resultrole}{$hname}=$price[$n];
325               }
326           $n++;
327           }
328             foreach my $hresult ( keys %HBENCH ){
329     }
330    }
331  }
332
333 }
334
335
336 foreach my $frank ( keys %RANK ){
337     $imgprice=999999999999;
338     $cheapestimg="";
339     my %RANKROLE = %{$RANK{$frank}};
340     foreach my $resultrole ( keys %RANKROLE ){
341       print "Image: $resultrole and price:" . ↪
          ↩ $RANKROLE{$resultrole} . "\n";
342       $tempvalue = $RANKROLE{$resultrole};
343       if ( $imgprice >= $tempvalue ) {
344         $imgprice = $tempvalue;
345         $cheapestimg = $resultrole;
346       }
347     }
348  print "Image: $cheapestimg and price: $imgprice ↪
     ↩ \n";
349 }
350
351
352 sub improv {
```

```
353   my $hostname = $_[0];
354   my $command = $_[1];
355   $result = `ssh -i /home/ubuntu/.ssh/enterprise
          ↩ -o "StrictHostKeyChecking no"
          ↩ ubuntu\@$hostname " $command " `;
356   return $result;
357 }
358
359
360 sub findPlugins {
361
362 my $temp = Cwd::cwd();
363 $path = "$temp/plugins";
364 if (-d $path) {
365   my $plugin;
366   my @dirlist = `ls $path`;
367   foreach $plugin (@dirlist){
368     chomp($plugin);
369     if (not ($plugin =~ /^\./ or $plugin =~ /~/) ){
370       my $key = $plugin;
371       $PLUGIN_LIST{$key} = "$path/$plugin";
372     }
373   }
374   closedir(DIR);
375 }
```

Appendix B

Article written based on thesis work

Audition: a DevOps-oriented service optimization and testing framework for cloud environments

Gaute Borgenholt
University of Oslo
Institute of Informatics
gautebo@ifi.uio.no

Kyrre Begnum, Paal E. Engelstad
Oslo and Akershus University College
of applied sciences
Department of Computer Science
{*kyrre.begnum—paal.engelstad*}*@hioa.no*

Abstract

This paper demonstrates an approach to automated testing and quality assurance in cloud environments, which also takes deployment cost into consideration. With a distributed service architecture and some given performance goals, the end result will be a suggestion of the optimal resource type and filesystem with the lowest price point for each function of the architecture. Our solution is modeled after the auditioning process in the theatre industry, which provides a process that fits well into our context and is easy to understand and follow. The resulting tool, Audition, is a working implementation of our model and is extendable in several ways, allowing for integration with local technologies.

Keywords

DevOps, Automation, Configuration Management, Virtualization, Cloud, Tools

1 Introduction

Deploying a site in an IaaS cloud environment provides incredible flexibility and streamlined operations logic. Sysadmins can boot new virtual machines in seconds and, when combined with modern configuration management tools, they can start being productive in minutes. The Cloud is proposed as a universal *panacea* for all our operations logic and DevOps ailments, but with it comes new challenges that are not yet addressed:

- Price. The overall cost picture has become more difficult for the individual sysadmin to control properly. There is a wealth of cost-related variables tied to usage, time and resource allocations.
- Complexity. Modern services consist of many building blocks. Finding the cheapest option that still meets performance goals is a cumbersome process and leads to old-fashioned over-provisioning, which in turn drives costs up.

Overall, the current cloud APIs and consoles do a good job at presenting the cost and parameters relative to single virtual machines. However, most large-scale sites operate with a multitude of virtual machines, which are assigned different roles and configured in very specific ways. Doing the selection of the right set of virtual machines running in the right constellation, can be a work-intensive process, given the complexity in terms of the building blocks of a service and the often strict cost requirements associated with it. To address this problem, we propose Audition, which is a tool to automate this complex and work-intensive analysis process. Audition will find a constellation that meets the given performance requirements of the service at the minimum cost. The cloud paradigm already provides cost benefits in terms of the ability to scale resources based on demand. Audition may augment the scaling and provide additional cost benefits by minimizing the cost for the current constellation of virtual machines given the current performance requirements.

Imagine a company who's primary business is running a website. Using agile development and streamlined releases in the DevOps spirit, they anticipate releases often, sometimes only a week apart between rollouts. Every time they deploy a new build, they also create a new version of their environment in a cloud, which then starts to receive new traffic as the old site winds down.

For every new release, the operations engineer is faced with the same decision: What instance type should be used for each role in the site? Roles, such as web-server, database, loadbalancer and middleware server, all have their specific performance requirements. The easy answer would be to go for a powerful type, but that would drive the price up. Furthermore, since the current version of the site only lives for a relative short time (weeks),

there is no use over-provisioning it for years to come.

Once the type is decided upon, there is the question of what appliance image to use for each role. Since configuration is centralized using tools like Puppet, Cfengine or Chef, the actual Linux version and distribution can be changed at every release. This allows for a great freedom, and one can avoid "lockdown" to a single distribution or family of them. Moreover, cloud providers, such as Amazon AWS [1], have their own marketplace for virtual machine images, which enables sysadmins to virtually cherry-pick the right, specialized image for the job. In practice, however, it is a problem of navigating through a wealth of available images, and the amount of testing required would be impractical. Remember, since the site changes very often, these changes can impact the requirements regarding hardware type (performance) and Linux distribution (libraries and versions) every time.

With the Audition tool presented in this paper, we demonstrate a way to streamline site optimization for complex virtualized environments, which are tightly integrated with configuration management. We build a process of automated testing and benchmarking along with cost analysis, which takes into account the different roles of servers and presents a recommended constellation for each new deployment. Even though Audition sounds familiar to ordinary test frameworks, the software developed is not the only focus of Audition. Instead, the uniqueness of Audition is that it is the machine image combined with the hardware type that is tested when running the software.

Our project modeled the process after the well-established auditioning process of the acting industry. Figure 1 shows a high-level diagram of the concept we adopt. More details can be found in Figure 2 when the architecture and implementation is described. We decided on mimicking their process based on the following arguments:

- We find a group of actors auditioning for a play to have many similarities to our case. The entire deployment can be seen as a play, where each server plays a distinct role with a very clear description from a manuscript. The sysadmin resembles the director, who manages the manuscript, but needs to find the right actor for each role.

- The auditioning process establishes two important properties with each actor:

  - Quality assurance. Can the actor perform from the manuscript to the level of quality as demanded by the director?

  - Price optimization. From a business perspective, the best actor for each role is the one who performs within acceptable limits and demands the least pay. (Actors who drive ticket sales by being famous are not part of our model.)

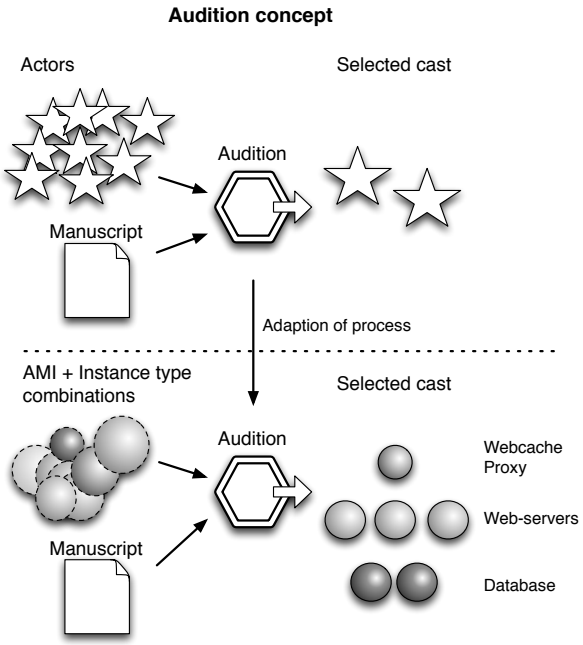

Figure 1: Concept illustration. Just like actors auditioning for a role, so do virtual machines in a cloud environment. The resulting *cast* will be the chosen virtual machine types and images for the architecture in question.

- Easy to understand. In our field, using analogies and loaded terminology is a common way to convey basic functioning of a tool. Good examples of such are Puppet [12], Chef [2] and Autopsy [5]. Blank-Edelmann and Lee have with an entertaining and comical perspective demonstrated that there are similarities between our profession and other, more established ones [8]. The deeper and inspiring message is that they face similar challenges and often have developed strategies within their own domain in order to cope. This may be transferred to our field.

We find that the need to assess the behavior of a virtual machine is especially important for cloud environments. A virtual instance may not only vary in its resources, but one is also free to pick images from different operating system distributions and third parties. Furthermore, these images may be featured with certain specializations in order to work smoothly in a cloud setting. Some of these modifications might not cause problems during regular

testing but may pose problems later while integrating other infrastructure services. For example, non-standard device naming may confuse monitoring toolkits or missing libraries for the backup system to work properly.

In the next section we present the auditioning model and how we have applied it to our case. Section 3 describes the architecture and implementation while Section 4 explains the intended workflow. Section 5 will discuss our solution.

2 The casting model applied on release management

In this section we provide a brief introduction to the auditioning process in the world of theatre. For every part described, we link it to the domain of system administration and describe how it is incorporated into the model.

2.1 The manuscript

At the heart of every major theatre and movie production is the manuscript, or script. The manuscript contains details about each role along with the dialogue and other descriptions of the environment and context. The dialogue is organized into scenes and only a subset of roles may be present at each scene. Every role is occupied by an actor, who interprets the role. The director is responsible for managing the actors in such a way that their interpretations match with the manuscript and the overall theme of the play.

The manuscript and its components translate well into our context. Our systems are assigned *roles* too, such as database, web-server and monitoring host. Together, they function as one large ensemble performing together for a single purpose: a service. The roles are defined as manifests in a configuration management system which clearly defines what each role should do. The basic elements of configuration management are specifying which software packages to install, setting the content of configuration files and ensuring that desired services are running. Our model expands that list with ensuring correct functioning of the software as well as desired performance levels for each role in relation to the SLA.

The interpretation is still left with each system, as they may vary in version, distribution type and performance.

2.2 The casting call

Before actors can become part of a play, they need to audition for a role. The audition is advertised through a casting call, which is sent to agencies and job boards. The casting call provides details about what roles are available and the traits and skills the role requires. Such traits can be physical appearance or the ability to speak with an accent. This is an early sorting mechanism in order to limit the number of applicants for each role. The actors who fit the bill will sign up for the audition.

In our field, this phase would normally be handled by the operation or solution architects before a system is deployed. There would not be a call per se. One would, on the other hand, identify key hardware traits which would follow each role. Examples are the minimum number of CPU cores or number of network interfaces.

Our model assumes that most cloud environments have a wide range of machine images, which can used for as a base for a virtual instance. In some cases, like Amazon AWS, there is even a marketplace where third-parties can sell specialized images for a premium. We consider all of these images as potential candidates for a role in our service. However, in order to create an instance, it has to be coupled with a hardware type. In Amazon, these types range from small single-core hardware resources to high-memory, high-CPU configurations. In OpenStack[3]they are called *flavors* and serve the same purpose. Therefore, any image/type combination essentially constitute a potential candidate for a role. We imagine a process where a casting call can be sent to a cloud environment, and a list of candidate image/type combinations would emerge from it. The resulting audition would be the opportunity for each candidate to display their conformance with the manuscript.

2.3 The audition

The success of each actor is decided based on their audition. There are some unwritten rules, for example, one should never waste time during the audition. Sometimes, every actor has a fixed time-slot in order to help organize the days of audition.

Every actor may be handed a part of the manuscript beforehand for preparation. This part could be a monologue or a dialogue where a supporting actor will be present on the stage. The first part of the audition is obviously to assess the quality of the performance of the provided piece from the manuscript. Next, the director or casting director may ask the actor to engage in improvisation (or improv) in order to gauge how well the actor responds to direction as well as to get a sense of the overall skill set. The improv session may involve situations that are not directly relevant to the manuscript at hand, but which may reveal other skills and traits of the actor. The director may have prepared a set of improv directions that are not known to the actor. The instructions may challenge the flexibility and imagination of the actor, such as ”You have a banana and a bowling ball, now rob a bank!” Improvisation skills are often highly regarded amongst actors. There are classes as well as several books devoted to the subject [15, 16]. One can

understand the value for the director to assess the ability of the actor to follow instructions. It will reveal to the director how well they are able to work together during rehearsal and later during the actual performance.

For sysadmins and test managers, an audition does sound familiar to ordinary test frameworks. The difference here is that the software developed is not the only focus of the test. Rather, it is the machine image combined with the hardware type that is tested when running the software. It will be tested based on the instructions in the manuscript. The audition follows these steps in order:

- **Role characteristics**
  Configurations belonging to the role that are applied by a configuration management system.
- **Dialogue**
  Automated interaction to check for correct functioning
- **Improvisation**
  Assessment of general qualities

For the three first items, success can be determined automatically. The configuration has to apply successfully before automated interaction can begin. The Dialogue can be anything from simple checks to comprehensive performance tests. For performance tests will then be parameterized with goals and will only be tested if the preceding dialogue was correct. Success in a performance benchmark means that the described performance goals are met.

One might ask how a computer system can possibly improvise anything. However, we believe that the introduction of the improv concept fills a gap in our profession. Improv corresponds to a process we often do, but which does not have a common name, as will be explained in the next paragraphs.

Whenever seasoned technicians, be that car mechanics or system administrators, works with their hardware and systems, they have their own rituals in order to gauge some sort of quality or confidence. They are often unrelated to the actual purpose of the system, but can loosely be described as "taking the car for a spin around the block". This might be certain benchmarks or commands that may stress one or two aspects of the system. For sysadmins, we argue that this phase is important because one becomes familiar with the system and its performance by comparing it to how others have performed in similar tasks before. Even if the direct output from the commands are not actually saying anything important, the fact that the familiar tools can be installed and that the favorite editor has the correct version are of interest to the sysadmin. Overall, this process of familiarization will give the sysadmin a sense of "how will it be for me to work with you?". Using the theater analogy, this is a similar question as faced by the director during an audition. The way to measure co-operability and flexibility of an actor is to engage the actor in improvisation (improv) exercises. We argue that the sysadmin subconsciously does the same by running some "good old commands".

For the improvisation part there is no clear success. The result of the improvisation will be recorded and presented to the sysadmin for review only if the candidate had success in the three preceding steps.

2.4 Cast selection and contract negotiations

The actor will not get an offering at the audition, but will instead wait for a call-back in the time after the auditions. The call-back may ask for another audition or offer the role. Normally, no news means the actor was not selected.

Once the offer has been extended to the selected actor for a role, contract negotiations begin. Even though a director is in charge of the quality of the performance, there is an obvious business dimension. Bluntly put, if two actors perform equal, the one with the most reasonable price may get the role.

In our model, the selection is also done when all candidates have finished their audition. For each role, there will be a remaining set of candidates who had success on the first elements of their audition. They are ranked based on their cost, which will be determined based on the price of the instance type combined with the possible premium of the machine image. This means that it is not the fastest performing candidate who wins, but the cheapest who performed within the desired performance thresholds. In the case of a tie between several candidates, the decision will have to be made by the administrator based on the output from the improv section.

The end result is a suggested cast of machine image/instance type combinations for each role along with a total price point for the entire ensemble. This list will form the blueprint of the actual deployment of the new production environment.

3 Audition prototype

Audition aims to re-use as many established and trusted tools from system administration as possible. It also needs to be extendable and configureable so that it can be adapted to local practices. Below is a short description of the building blocks which Audition uses.

3.1 Chosen technologies

- **Cloud platform**
  The implementation of Audition presented in this paper focuses on Amazon Web-Services (AWS), because the cost aspects are easy to define and clearly visible. However, Audition may apply equally well to other platforms, such as local OpenStack, VMware, Xen or KVM.

- **Virtual machine management**
  MLN is an extendable management tool for virtual machines in local or cloud environments [6, 7]. It provides a powerful descriptive language which Audition utilizes.

- **Configuration management**
  Our implementation of Audition relies on Puppet to manage internal configuration of each virtual machine. However, Audition can easily be expanded through plugins to support other tools, such as Cfengine [7].

3.2 Implementation

The tool is implemented as a command-line tool, written in the Perl programming language. The supplied input is the manuscript, which is written in a block-based configuration file format. It provides the roles, scenes and dialogue. The roles correspond to the roles needed in the project, like webserver or database. The role has a corresponding puppet class representing the technical details, such as the required packages and configuration of services. This means that the manuscript does not focus on what attribute each role entails, but instead connects the role with the appropriate class.

The dialogue is in the form of benchmark commands, which are sensitive to the specific role and in addition have specified performance constraints. The dialogue is collected in scenes. For example, there might be a scene where a webserver has to serve a page within a certain rate and where the webserver depends on a database for content. Here, the database acts in a support role that is provided by the Audition tool as part of the preparation for the scene, i.e. before the candidates play that scene. The support role is presented in the manuscript, but is not a candidate for a role itself.

When executed, Audition will parse the manuscript and run the casting call. From the casting call a series of candidates are established. The next step will be to organize the actual audition in which each combination of Amazon Machine Image (AMI) and type can audition for each role. This list is manifested in a MLN project description. Audition will populate the MLN template with all the image/type/role combinations with specific links to superclasses representing the role-specific configurations.

All roles are defined specifically as classes in the Puppet configuration management framework. MLN is aware of this and will register each candidate in puppet with the appropriate class by creating a node-block for each candidate. This is achieved using a Puppet plugin in MLN. See Figure 2 for an overview of the architecture.

When the organization of the casting is finished, the rest of process will have the following form:

```
foreach role {
  boot support roles
  foreach candidate {
    boot the candidate
    apply role configuration or finish
    foreach scene {
      run dialogue or finish
      run improvisation
      store results
    }
    shut down candidate
  }
  shut down support roles
}
```

Audition will utilize MLN to start/stop each candidate and insert commands to make it install the puppet agent and connect to the puppet master for configuration. The Audition tool will monitor whether the role-specific policy is implemented successfully, as there is no use continuing otherwise. If the policy was successful, all scenes are played out in order. However, if one of the candidates does not meet the specified outcome in a specific scene, the entire audition is finished for that candidate to save time. Finally, if all scenes are successful, the improv part will be run and its output stored. Improv does not have specific end-requirements.

Each scene includes a dialogue, which may be one or more lines which the candidate or support roles will execute in order. The dialogue may vary depending on the role and the type of action, so it is implemented in a plugin fashion. Every line of the dialogue is a separate plugin, which can be written and modified by third parties. The only requirement is that the dialogue plugin returns whether or not the line was executed successfully. An examples of a dialogue is a web benchmark, where the desired url and response rate will be the parameters. The dialogue plugin will then return true if the candidate was able to meet the required rate. Another example could be a URL checker that looks for a specific regular expression match in a webpage. This plugins may be run before the benchmark to first ensure correctness of the page. The scene may include support actors, as well, and have dialogue lines that involves them too. The dialogue

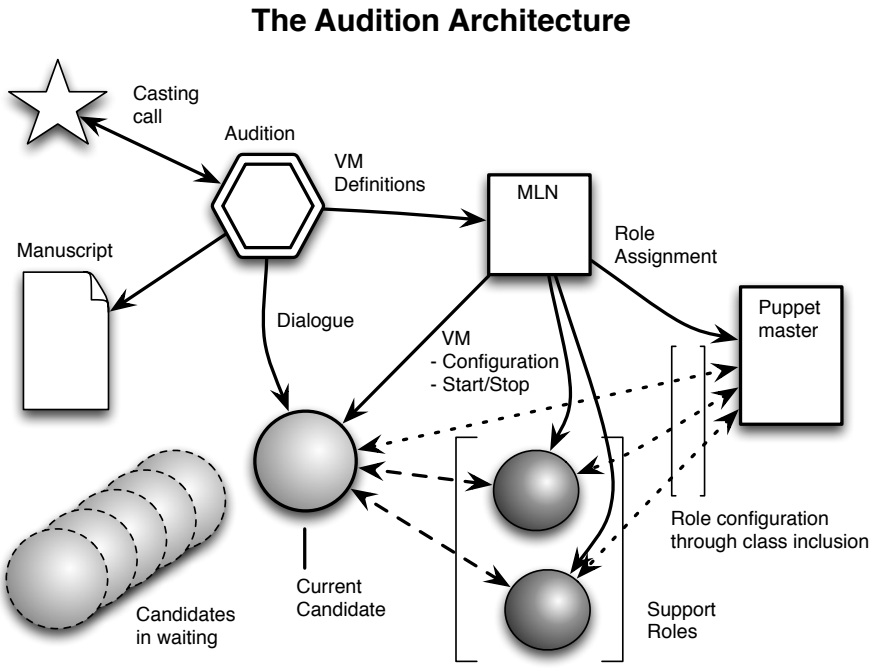

Figure 2: The Audition tool will read the manuscript before issuing the casting call. Once candidates are established, they are configured and booted using MLN and Puppet. The dialogue is in the form of tailored commands, like benchmarks.

may be as complex as desired and allows for coordinated orchestration if necessary.

3.3 Example: A Wordpress site

In the following example, we see the deployment of a Wordpress site which consists of two roles: a webserver (web) and a database (db). The manuscript describes two scenes, each with a dialogue specific to the role being tested. The first scene will be played by all candidates who audition for the webserver role. This scene has three lines for the webserver. First, it needs to successfully connect to the database, next it has to provide the content "Lorem Ipsum" on the url */index.php* and finally it needs to deliver the page */index.php* at a rate of 50 pages per second. The plugin that corresponds to each line is responsible for executing the task. For example, the `benchmark.web` plugin executes the httperf benchmark tool and compares the results to the required performance. A database is mentioned as a support role. Support roles are not candidates and are already known to support the required performance. The support role database will be a special virtual machine which is booted in the beginning of the audition and re-used each time this scene is played by a candidate.

```
mln_file mln-template-wordpress.mln

scene frontpage {
  role web
  support_roles database
  dialogue {
    [web]: connect.db [database]
    [web]: content.web /index.php Lorem Ipsum
    [web]: benchmark.web 50 /index.php
  }
}

scene backend_performance {
  role database
  dialogue {
    [database]: restore.db wordpress_base.sql
    [database]: transactions.db 300
  }
}
```

The database is a different role and here the dialogue is about first restoring a database from a provided backup file. This will provide content for the next line, which is a database benchmark test to check if the performance level of 300 transactions per seconds can be reached.

Dialogue Illustration

```
dialogue {
      [web]: connect.db $[database]
      [web]: content.web /index.php Lorem Ipsum
      [web]: benchmark.web 50 /index.php
}
```

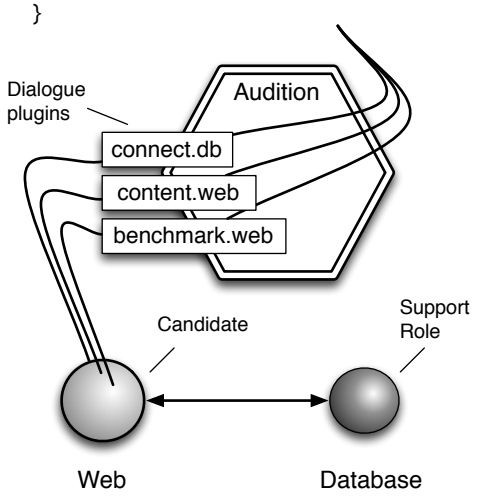

Figure 3: Dialogue illustration. Each line in the dialogue is handled by a plugin. This enables the audition to easily be tailored to local needs.

4 The Audition workflow

An audition is always for a specific play. In our case, it means it is specific to the project and the desired technical solution. The project architects along with the technical release and test leads, need to write the manuscript in the beginning of the project. The manuscript might change along the way, as more features need to be tested for each role.

The MLN project template, which contains the superclasses and definitions for each role, has to be established early but will probably not change. We anticipate that larger organizations have standardized services to such an extent that the role ”webserver” will mostly be the same platform throughout all solutions. The Puppet configurations may evolve along with the project and will be maintained by the DevOps personnel. The plugins, which make out the dialogue, are relatively simple and may be shared across the organization and community as a whole. The idea is that a plugin written for a web benchmark can be re-used every time this functionality is required in a role. The parameters of the lines will be specific to the manuscript, in order to localize the way it works.

Running the audition will be the task of the test and release group, but assisted by developers and operations. One may either go directly to a deployment based on the

MLN template

```
global {
   project wordpress
}

superclass basic {
     puppet {
        nodename $hostname.$project
        include {
           ssh
        }
      }
      ec2 {
        key ec2_key
        user_file {
              [redacted]
        }
     }
}

superclass web {
    superclass basic
    puppet {
         include {
            web
         }
    }
}

superclass database {
    superclass basic
    puppet {
         include {
            database
         }
    }
}

superclass sup_database {
   superclass basic
   puppet {
         include {
            sup_db
         }
   }
   ec2 {
      type m1.large
      ami ami-894221e
   }
}
```

Configuration management installation and agent initialization to the master. Executed when instance boots

Specialized superclasses for each role in the manuscript. Each candidate wil inherit the properties from one of them.

A support role has a superclass also, but more specialized with regard to type and image.

Figure 4: Dialogue illustration. Each line in the dialogue is handled by a plugin. This enables the audition to easily be tailored to local needs.

recommended cast from the audition or one might review the cast in more detail. This can provide interesting information about the overall performance of the solution. For instance, if the database code has been modified heavily from one release to another, one might check if the cast has changed, e.g., if the database role is played by a more powerful server than in the last release. Remember, Audition will pick the cheapest server that still is able to perform relative to the requirements in the manuscript. Therefore, the resulting deployment may scale as the solution matures, driving costs up.

4.1 Execution

Audition is executed from the command line:

```
$ audition -m wordpress.manuscript
```

The output focuses on listing only the candidates which completed successfully and highlights the most affordable of them:

```
<omitted output>
Role: wordpress_web
  2nd place: m1.large with ami-def89fb7
       $0.240 per instance-hour
  1st place: c1.medium with ami-def89fb7
       $0.145 per instance-hour

Creating MLN code for role:
wordpress_web: c1.medium/ami-def89fb7
```

From the output, we see that two candidates were successful. The both use the same machine image but vary in price. The best pick is the hardware type which favors CPU performance (c1.medium) as opposed to memory (m1.large).

5 Discussion

DevOps is all about adopting methodologies that bring developers and operations closer together [14]. A strong focus on automation is needed to facilitate the release cycles of agile and continuous integration processes. The approach used by Audition is in line with DevOps as it supplies a tool that can automate an otherwise cumbersome analysis process. With this automation in place, the project is allowed to use tailored deployments for each release, which continue to respect the SLA and keep avoiding over-provisioning. Furthermore, it is a tool where both operations and developers partake. For instance, setting up the configuration management and MLN project will be the focus of operations, while the dialogue in the manuscript will be something the developers have best knowledge of.

The dialogue may not be all about performance, though. We see the following applications for Audition:

- As a tool to track how performance demands increase across releases
- As a tool to test configuration management policies
- As a simple smoke-test that checks for correct functioning of software before deployment.

Since there is no real difference between puppet classes for the manuscript and for the production environments, Audition allows for heavy re-use of configuration code.

Obviously, components such as configuration management, need to be in place before one can start with Audition. Audition is meant to enhance automation without replacing the established tools used in-house. Puppet was used in our implementation, but others could be implemented as Audition is really oblivious to it. The connection to the configuration management is in the MLN project description, which can be expanded through plugins to support other tools, like Cfengine [7]. In addition, other platforms can be used for deployment as well, such as local OpenStack, VMware, Xen or KVM, as long as the corresponding MLN plugin is utilized. However, we chose AWS as it showcases the price dimension.

One of the benefits of cloud deployments is the ability to scale resources based on demand. One may therefore question the need to assess the performance of a deployment, when scaling already is in place. We do not see Audition and scaling as mutually exclusive. In fact, just because one can scale, does not mean one should not pay attention to the performance of webservers and to minimize the cost for the current constellation of virtual machines. Automatic scaling does not provide a cost-projection in the same way Audition does. Furthermore, knowledge from the Audition will supply the administrator with useful insight that can be used to optimize scaling decisions.

5.1 The cost of an Audition

The execution time of one audition is mostly influenced by the number of candidates, roles and comprehensiveness of the dialogue. The cost model of Amazon AWS is based on a per-hour price. Meaning that even if a candidate is running for only 5 minutes, a whole hour is charged. The support roles are kept running for as long as it takes to let all candidates for a role play the scenes. If Audition runs from an Amazon instance as well, then the network traffic would be internal and constitute no extra cost.

As an example, consider an audition with three images and the instance types m1.large ($0.240/h), m1.medium ($0.120/h), c1.medium ($0.145/h) and m1.small ($0.060/h). This makes for a total of 12 candidates. Considering only a single role which with booting, scenes and dialogue amount to 10 minutes each. Additionally a support role using the instance type m1.large is used. The number of candidates would require the support role to be up for 120 minutes, rounding to 3 hours charged. Considering only charges for time used, the cost for the audition would only be $2.415. For practical, additional improvements should be considered to cut the time of the audition through parallel testing of candidates. However, this would not reduce the cost of each candidate as one is charged for a full hour.

6 Related work

Automated software testing is a well established field with many approaches to assert the correct execution of applications when certain inputs are given. Several projects within this domain are moving towards cloud-based solutions, such as Cloud9 [10] and the $TaaS_D$ framework designed by Candea et.al [9]. Likewise, the Test support as a service (TSaaS) by King et.al suggests that more existing tools are utilizing the scalability and orchestration of virtualized infrastructures through migration [13]. However, their advances are within execution simulation for developers in order to establish software quality. Our approach is from the perspective of operations where we build on existing concepts of configuration management and automated virtual machine deployment. Audition as such comes in at a later stage where deployment configurations are optimized relative to software. The quality of the software itself is not tested explicitly.

There are industry solutions, such as RedHat CloudForms[4], for automated capacity planning and resource allocation. However, we argue that our approach is freely available and adaptable to various workloads and tools from the ground up. CloudForms, on the contrary, uses a specific set of tools for management and does not utilize third-party images in the same way as offered by Amazon AWS. It is possible to use Audition as a supplement to other solutions, e.g. as a preliminary testing tool before architectures are orchestrated through CloudForms.

The use of analogies and mimicry is a known approach to simplify otherwise complex processes. Finstadsveen used concepts from biology and animal survival strategies to model how a group of servers could collectively ensure a high level of service. The result was a terminology and concept which allowed non-technical people to engage in discussions about advanced scaling approaches and intrusion prevention for cloud-based services[11].

7 Future work

While the test implementation presented in this paper has been of limited scope and time span, a future work item is to utilize Audition in a development project and observe how much the cast changes as the developed solution matures.

Furthermore, allowing for a closer integration with other arenas, such as the AWS Marketplace, would be very helpful as it would allow third-parties to register their DevPay images for auditions in order to compete for business. This form of API is not present at the time of writing. A general role-description, like apache2 webserver and loadbalancer, could be made available so that the third-party images could be tailored and optimized for that purpose.

References

[1] Amazon aws. https://aws.amazon.com/, April 2013. [Online; accessed April 2013].

[2] Chef. http://www.opscode.com/chef/, April 2013. [Online; accessed April 2013].

[3] Openstack open source cloud computing software. http://openstack.org, April 2013. [Online; accessed April 2013].

[4] Red hat cloudforms. http://www.redhat.com/products/cloud-computing/cloudforms/, April 2013. [Online; accessed April 2013].

[5] The sleuth kit. http://sleuthkit.org/, April 2013. [Online; accessed April 2013].

[6] BEGNUM, K. Simplified cloud-oriented virtual machine management with mln. *The Journal of Supercomputing 61*, 2 (2012), 251–266.

[7] BEGNUM, K., BURGESS, M., AND SECHREST, J. Adaptive provisioning using virtual machines and autonomous role-based management. In *Autonomic and Autonomous Systems, 2006. ICAS'06. 2006 International Conference on* (2006), IEEE, pp. 7–7.

[8] BLANK-EDELMAN, D. Selected talks by david blank-edelman. http://www.otterbook.com/the-talks/, April 2013. [Online; accessed April 2013].

[9] CANDEA, G., BUCUR, S., AND ZAMFIR, C. Automated software testing as a service. In *Proceedings of the 1st ACM symposium on Cloud computing* (2010), ACM, pp. 155–160.

[10] CIORTEA, L., ZAMFIR, C., BUCUR, S., CHIPOUNOV, V., AND CANDEA, G. Cloud9: A software testing service. *ACM SIGOPS Operating Systems Review 43*, 4 (2010), 5–10.

[11] FINSTADSVEEN, J., AND BEGNUM, K. What a webserver can learn from a zebra and what we learned in the process. In *Proceedings of the 5th ACM Symposium on Computer Human Interaction for Management of Information Technology* (2011), ACM, p. 5.

[12] KANIES, L. Puppet: Next-generation configuration management. *The USENIX Magazine. v31 i1* (2006), 19–25.

[13] KING, T. M., AND GANTI, A. S. Migrating autonomic self-testing to the cloud. In *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on* (2010), IEEE, pp. 438–443.

[14] SACKS, M. Devops principles for successful web sites. In *Pro Website Development and Operations*. Springer, 2012, pp. 1–14.

[15] SPOLIN, V., SILLS, C. B., AND REINER, R. *Theater games for rehearsal: A director's handbook*. Northwestern University Press, 2011.

[16] SPOLIN, V., SILLS, P., AND SILLS, C. *Theater games for the lone actor*. Northwestern University Press, 2001.