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 theater 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

Automation, Configuration Management, Virtualization, Cloud, Tools, Best Practices

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 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

This paper was presented at the NIK-2013 conference; see http://www.nik.no/.

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.

DevOps, short for Developer Operations, is about streamlining the way developers and system administrations work together to deploy code faster and with a higher quality. It represents the *de facto* best practices for agile development groups with rapid release cycles. Imagine a company whose primary business is running a website. Using agile development and streamlined releases in the DevOps spirit, they anticipate releases often, sometimes only days 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, load balancer 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.

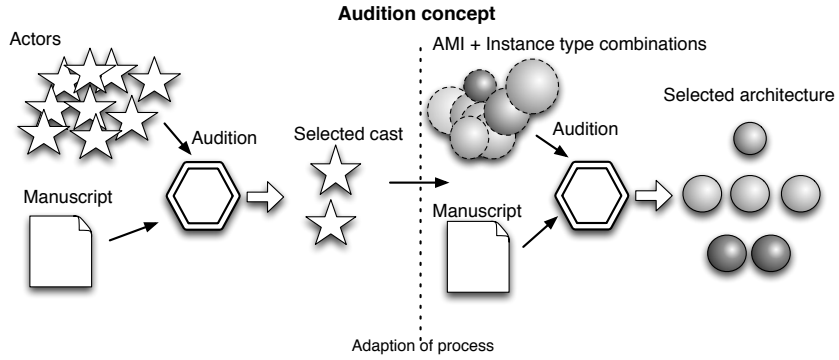

Figure 1: Concept illustration. Just like actors auditioning for a role, so do virtual machines in a cloud environment. The resulting *cast* will be the chosen virtual machine types and images for the architecture in question.

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.)

Furthermore 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 with them. 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 theater. For every part described, we link it to the domain of system administration and describe how it is incorporated into the model.

The manuscript

At the heart of every major theater 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.

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.

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 [16, 17]. 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 first two steps, 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. Performance tests will 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: Whenever seasoned technicians, be that car mechanics or system administrators, work 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.

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. If 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 resulting set of candidates that 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.

- **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].

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 web server 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.

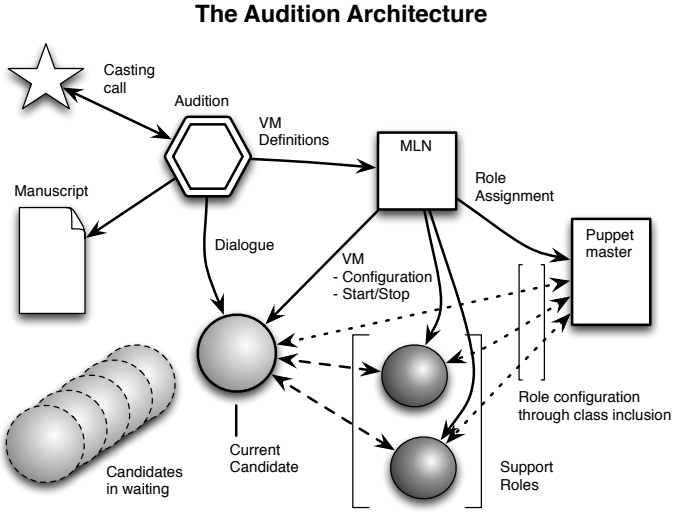

Figure 2: The Audition tool will read the manuscript before issuing the casting call. Once candidates are established, they are configured and booted using MLN and Puppet. The dialogue is in the form of tailored commands, like benchmarks.

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 web server has to serve a page within a certain rate and where the web server 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 may be as complex as desired and allows for coordinated orchestration if necessary.

Example: A Wordpress site

In the following example, we see the deployment of a Wordpress site which consists of two roles: a web server (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 web server role. This scene has three lines for the web server. 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. 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
```

```
  }
}
```

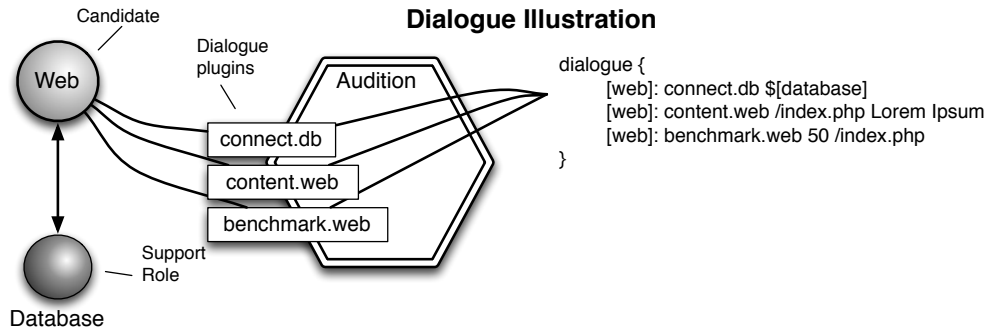

Figure 3: Dialogue illustration. Each line in the dialogue is handled by a plugin. This enables the audition to easily be tailored to local needs.

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.

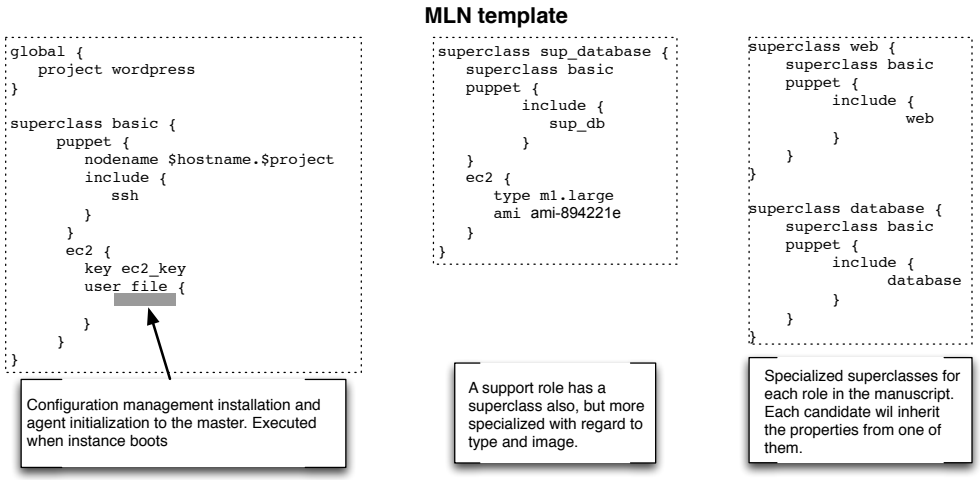

Figure 4: MLN template illustration. The template contains the technical details of all the roles. Every role has a superclass definition.

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.

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 [15]. 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: (i) As a tool to track how performance demands increase across releases as a form of A/B testing system, or (ii) as a tool to test configuration management policies or lastly (iii) as a simple smoke test / integration 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 web servers 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.

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.

In the work by Singer et al.[14] a cost model is proposed to estimate the running costs of an in-house system, should it be moved into a cloud. The model utilizes monitoring data to make predictions on resource demand such as CPUs, memory, network and storage. Based on this data, the optimal instance type is identified. The contrast to Audition is that our model does not assume an existing setup that has supplied data for several months. Also, the type thresholds found in the manuscript's dialog represent values not on hardware demand but rather performance values one would find in an SLA.

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].

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] LIVENSON, I., SINGER, G., SRIRAMA, S. N., NORBISRATH, U., AND DUMAS, M. Towards a model for cloud computing cost estimation with reserved resources. In *Proceeding of 2nd International ICST Conference on Cloud Computing, CloudComp 2010* (2010).

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

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

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