Survey of Automated Software Deployment for Computational and Engineering Research

James O. Benson, John J. Prevost, and Paul Rad
Electrical and Computer Engineering Department
The University of Texas at San Antonio
{james.benson, jeff.prevost, paul.rad}@utsa.edu

Abstract— Automated, efficient software deployment is essential for today's modern cloud hosting providers. With advances in cloud technology, on demand cloud services offered by public providers are becoming increasingly powerful, anchoring the ecosystem of cloud services. Cloud infrastructure services are appealing in part because they enable customers to acquire and release infrastructure resources on demand for applications in response to load surges. This paper addresses the challenge of building an effective multi-cloud application deployment controller as a customer add-on outside of the cloud utility service itself. Such external controllers must function within the constraints of the cloud providers' APIs. In this paper, we describe the different steps necessary to deploy applications using such external controller. Then with a set of candidates for such external controllers, we use the proposed taxonomy to survey several management tools such as Chef, SaltStack, and Ansible for application automation on cloud computing services based on the defined model. We use the taxonomy and survey results not only to identify similarities and differences of the architectural approaches of cloud computing, but also to identify areas requiring further research.

***Keywords*—Distributed computing, Software, Open source software, Public domain software, Software as a service, Software maintenance, Software packages.**

I. INTRODUCTION

Cloud and cloud services have become more popular in recent years. In the cloud hosting industry, companies have found that costs can be reduced by improving up-time in servers and creating a scalable server based on load. According to a white paper by Vision Solutions, 59% of Fortune 500 companies experienced a minimum of 1.6 hours of downtime per week [1]. This means that for a company who has 10,000 employees who on average make a salary of $30 per hour [2], or $60,000 per year, this downtime can potentially create a loss of $480,000 weekly or nearly 25 million dollars annually, not including the cost of benefits, loss of sales, or negative impact to the reputation of the provider from services being unavailable. Consequently, it is of the utmost importance for a company's servers to have their services installed, configured, and running as quickly as possible and as consistent as possible to help reduce costs. This translates into automated deployment and configuration. However, in academia cloud and cloud services have not been adopted as quickly as they have in industry. This slower movement towards the cloud in academia can be attributed to faculty and student researchers lacking the full scope of knowledge, support as well as the time and resources that companies are able to devote to integrating and efficiently using the cloud. [3]

It is realistic to assume that some cloud-based companies, like Amazon, host data centers that have over 50,000 servers per availability zone with more than one center per zone [4]. It is in centers like this or other large companies which host a large number of servers that it becomes critical for fast deployment of software and ease of deployment. Deployment methods such as Ansible become critical for these types of situations where simple bash scripting no longer becomes feasible.

The approach taken in this research is to create a one click system where a user is capable of deploying both infrastructure and software to heterogeneous cloud environments using standard deployment methods. The final objective is to develop a single framework which be used to reach out and configure the various cloud environments as is shown in Figure 1.

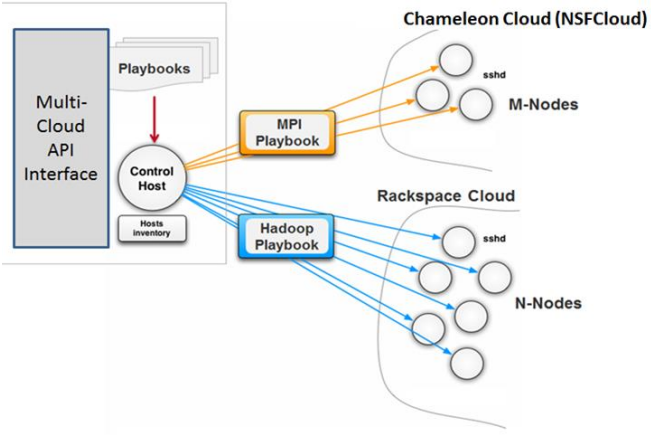

Figure 1: Multi Cloud Application Deployment Controller (Multi-cloud ADC)

This new framework contains a host system which orchestrates the scripting and deployment of N-nodes to various types of clouds operated by any of the major providers or even providers implementing private cloud in-house. The

The authors are with the University of Texas at San Antonio (UTSA) in the Electrical and Computer Engineering Department. This paper was supported by the Open Cloud Institute at UTSA.

978-1-4673-9519-9/16/$31.00 ©2016 IEEE

host system would first ask you to choose which application you need to build, for example, a Chef server, WordPress, Rails, database server such as Cassandra or MongoDB, or even another cloud OS such as OpenStack. Next, the system would determine if one or multiple servers are necessary. The framework would then provide a list of common operating system and an option to select the of quantity of nodes. Once a few other basic items are selected, the cloud and infrastructure would be built and the software deployed along with any SSH keys, usernames/passwords, etc. Likewise, a user could create a cluster of servers to be installed with a specific operating system and pre-selected software chosen from a list of available software.

For educational facilities and researchers, previous research has been focused on simplifying the cloud. One such method uses a program called *Blender* [3]. Blender helps simplify the building of a cloud infrastructure and installation of programs on the cloud. While this project is notable, it has not been widely adopted and vigorously tested unlike other commercial products like Juju or OpenStack Heat, for infrastructure deployment, or Ansible, Chef, and SaltStack for software deployment.

In Scientific and Engineering computational research there is no "one type fits all solution" due to the fact that researcher needs vary greatly based on the problem domain and the researcher's knowledge and preference. It is therefore important to survey the needs of researchers in academia to identify specific needs in terms of computational power, storage requirements, required programs and system configurations. Once this information is determined, it would be possible to have predefined scripts run, build out the required infrastructure and automatically install and configure the necessary software. This would provide a complete solution to the researcher significantly reducing the effort required to get their custom cloud implemented. The work proposed here is an alternative method to Blender [3] as it uses commercial products to help automatically deploy software packages as necessary based upon the needs of a specific user.

In this paper, we chose two specific use-cases based on two research domains. Each research domain requires specific software packages to be deployed and configured. The first package is focused on chemistry consisting of four applications which are all in the apt-get repository. The second package is more focused on data analytics consisting of three applications one of which, octave, that has two additional packages to allow for parallelization across several different nodes. These research domains will allow us to test different deployment models to characterize effectiveness, ease of use and speed of deployment. The key metrics for evaluating the ease of deployment is the code required to execute it, time to execute both a clean install and existing installation on a single node verse two nodes. Likewise, summaries of the software deployment and what they are capable of will be described as well, including if they have a GUI, encrypt traffic, and what language it is based off of and others.

II. History: Images vs Scripting

There are two main approaches to setting up the infrastructure of standup servers: imaging and scripting. Both approaches have a number of advantages and limitations, which are detailed below.

Once an image has been set up, imaging a server is a very fast and dependable way of deploying a preconfigured system, including authentication and potentially difficult service infrastructures. Presently, companies such as rightscale.com can stand up one of countless images on any cloud provider a customer would like in just a matter of minutes. However, these images have their limitations. Typically, images may not have been recently updated which means that several pieces of software may be outdated or critical security patches may be missing. Further, if a customer desires a slightly different configuration, a completely new image needs to be set up. The inflexibility of the final image is one of the major drawbacks to imaging. Every desired change of an image requires a new image to be saved. These duplicate images which may vary only minimally can occupy a large volume of space. Further, incorporating software updates, patches and any end user customizations are slow to save.

Scripting while it may not be as instantaneous as imaging, especially for complex or lengthy tasks, does overcome many of the limitations of imagine. Specifically, it allows the flexibility of installing all of the necessary and desired software, including updates, and architecting between systems. Likewise, scripting can be dynamic enough to distribute custom files to each of the N-nodes to ensure that necessary files work appropriately, something that is impossible when using imaging for server set up. An example of this might be distributing lists of IP addresses, custom hostnames, or specific license files for software, or the system, to operate correctly.

Another advantage of scripting is that it requires very little space. The recipes or scripts typically are only kilobytes large and the instead of having duplicates of the same operating system for each image, in scripting only one version of the operating system and one version of the software is needed. Figure 2 - Images vs Scripting provides an overview of the different space requirements of imaging vs. scripting. The schematic highlights how imaging can become very resource intensive while scripting allows for more flexibility with minimal resource allocation.

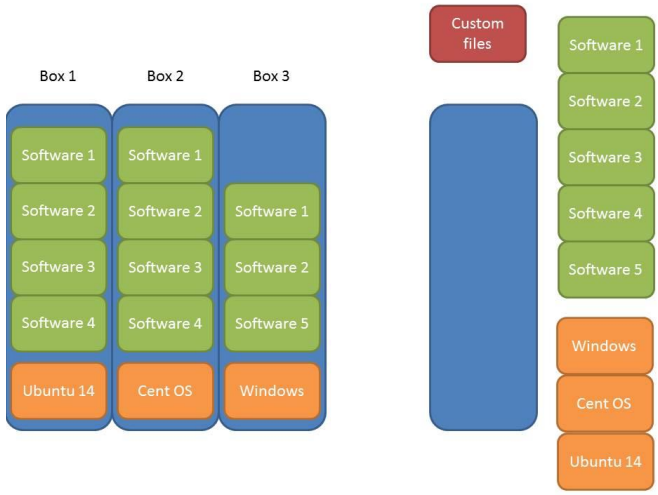

Figure 2 - Images vs Scripting

In summary, scripting seen at the high level is superior to imaging due to its increased flexibility, adaptability, and lower space requirements. In addition, as scripting languages such as Ansible, Chef, and Salt and others are becoming increasingly more human readable the field of automating tasks becomes more accessible. Likewise, scripting is becoming smarter by incorporating error testing and simplifying reporting to pass/fail. Modern scripting can accomplish a variety of images in parallel each with complex tasks. Further, frequently each of these scripting tools have modules or libraries for more specific actions that might otherwise require bash. For example, creating users can be a simple, "user: name=JohnDoe" in Ansible. [5]

III. Decoupling Application and Infrastructure Delivery

Many of the previous works on resource provisioning assume a central controller that combines application control and arbitration policy [6, 7, 8, 9]. Parekh et al. [6] demonstrated the use of classical control theory to fine tune the parameters of a controller of a cloud server, such as email. Soundararajan et al. [7] present control policies for dynamic provisioning of a database server. Urgaonkar et al. [8] use queueing theory to model a multi-tier application and to determine the amount of physical resources needed by the application. Paul et al. [9] used network resource partitioning using Infiniband network.

Cloud Infrastructure platforms present a well-defined REST-full application programing interface (API) service to their end users. Today, the cloud provider APIs are stateless service without direct knowledge of the application performance metrics, or the impact of allocation and placement choices on the service quality of the applications. The API service provides a useful separation of concerns: the end user is insulated from the details of the underlying physical resources, and the provider is insulated from the details of the end user application

Our position is that the application delivery model should also reflect the decoupling of application topology and configuration as an external service from infrastructure deployment defined by cloud providers. In this model application playbooks should be decoupled from the cloud provider and customizable by the end user. A clean decoupling of application control policy and playbooks from the cloud provider infrastructure is a necessary architectural step to prevent cloud platforms from becoming complex and inflexible as application demands change overtime.

The Multi Cloud Application Deployment Controller (Multi Cloud - ADC) facilitates the separation of end user applications to define and customize their optional playbook and policies, outside of the cloud provider using the provider infrastructure APIs. A principled layers approach offers the best potential for application developers to innovate in their control policies and customize their playbooks to the needs of their applications, while deploying on multi-clouds

This separation implies that the application deployment controller consists of user defined playbooks function independently of one another and can run on multi-clouds using the Cloud interface translation plugin.

The (multi-cloud ADC) service is based on resource reservation contracts whose terms are negotiated and expressed through exchanges of property lists. The reservation terms define the timeframe for active resources such as cloud servers

The main goals of all three of these configuration management tools are consistency, reliability, speed, minimal system impact and easy to learn. Overall, the end-user should be able to know that the software is deployed to all of the specified nodes and configured as described. They should also know if something fails where it fails and what if anything else is going to be installed. Likewise, the deployments should be fast. In large companies, a package may need to be deployed to thousands of nodes and if the management tool goes one-by-one it may take days or weeks to complete. As the software is deployed, the management tools should not be demanding most of the resources either so the server can still operate as normal.

IV. A COMPARISON OF SOFTWARE DELIVERY TOOLS

A. Ansible

Ansible was first released in February 2012 and over the past three years it has gathered over 1,041 contributors, 65 releases, and nearly 14,000 commits.[1] It has an agentless architecture which is different than Chef and SaltStack which rely on an agent based architecture. An agentless architecture means that no agent is installed on the nodes which will be controlled. This type of management can be seen in Figure 3. This is helpful in that if a process is not running, then there is no checking-in by the nodes resulting in no additional network traffic and no background daemons running. Additionally, Ansible differentiates itself from Chef and Salt by not requiring SSH keys, no additional ports need to be opened besides SSH, no root access is necessary although it can configure things using ***sudo*** if necessary. These items make it an ideal case for situations when high security is necessary or greater stability/performance by not installing additional monitoring software [5].

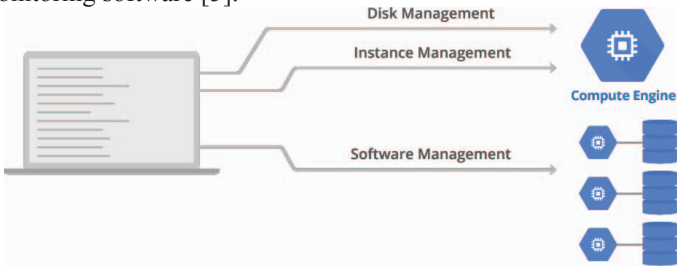

Figure 3 - Standalone management [10]

To control the nodes, Ansible retrieves the necessary hosts from the inventory file and then executes the code over SSH. While all of the software operates at reasonable speeds, SSH can slow the process down compared to using ZeroMQ for communication like Salt does [11]. The execution of the code comes from a YAML file which is a descriptive language and relatively easy to learn. Companies like Twitter [12],

[1] These statistics are as of April 26, 2015 from https://github.com/ansible/ansible

Evernote, EdX [13] and care.com all use Ansible for their software deployment.

B. Chef

While Chef is one of the longest lived software configuration management tools still around being founded in 2008, the base of contributable users is considerably smaller at 375 users, 239 releases, with 12,286 commits[2]. A simple graphical overview of how a master/slave management looks

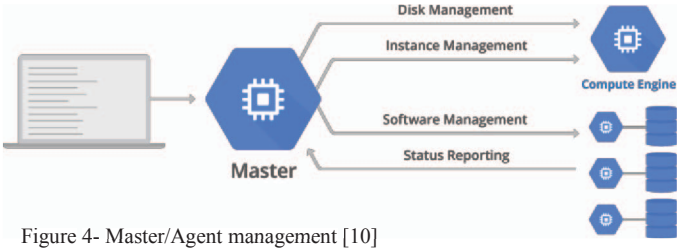

Figure 4- Master/Agent management [10]

can be seen in Figure 4. Chef typically depends of a master/slave configuration where there is a main master node or server that recipes are uploaded to and then is deployed to the clients. Where Chef really excels is the contribution of cookbooks to help configure and deploy systems already built and field tested by other users. As of May 2, 2015, there are nearly 2200 cookbooks and over 62,500 chefs supporting the supermarket on supermarket.chef.io. In addition, Chef's website interface also allows the administrator to view and search node activity, assign cookbooks, roles and nodes to tasks.

To control the clients in Chef, you can import a cookbook or create one of your own. The cookbooks are written in Ruby, so some knowledge of Ruby programming is very helpful, especially since official documentation can be vague at times for more advanced functions. Once imported into the server, the administrator can use the WEBUI, which is easy to use, assign the recipe to the nodes and execute it or the user can proceed through a command line. Chef uses SSH to communicate and authenticates via the use of certificates. Companies like Ancestry.com, Bonobos, Etsy and Bloomberg use Chef to manage their systems [14].

Table 1 - Features of Configuration Management Software

	Ansible	Chef	SaltStack	Shell
Language	Python	Ruby	Python	Shell
License	GPL	Apache	Apache	GNU
Mutual Auth	Yes - SSH	Yes- Keys	Yes	No
Encrypts	Yes – SSH	Yes - SSL	Yes- SSH***	No
Verify mode	Yes	Yes- why run mode	Yes	No
Agent-less	Yes	No**	Both	Yes
GUI	Yes	Yes	Yes	No

[2] These statistics are as of April 26, 2015 from https://github.com/chef/chef

C. SaltStack

SaltStack is slightly older than Ansible with its initial release in March of 2011, has a base of 1,078 contributors, over 52,000 commits, and 86 releases as of May 2, 2015. SaltStack depends on a master/minion system as well, however, in this case, the clients make a request to join the master node and when the master accepts, the client can then be controlled. One of SaltStack's main advantages is in optimizing the speed, scalability and resiliency; this is achieved through the ability to use of asynchronous file servers, multiple tiers of masters and the ability to distribute loads and redundancy. One of the benefits of this peering system is that unlike Chef, minions can ask questions to the master in order to get a more complete picture, for example, they can use a database to look up, in real-time, information in order to help complete the configuration of a minion [15].

A SaltStack operation has several steps. The first is to define a state tree and assign parts of it to servers. The minions at this point will download the state definition from the tree and compare its current state to the state tree. At this point, the minion will make the appropriate changes to make the two match, and report back to the master. For example, the state tree could ask that Apache is installed and running, the minion may see that it is neither installed nor running and proceed with installing it and setting it up as defined in your tree. Salt also has various other grains to refine tasks on nodes from coarse grains that are applied to all of the nodes to fine grains that are node(s) specific. Companies like Lyft [16], Overstock.com [17], Rackspace [18] and others all use SaltStack to manage their servers, clouds and workstations.

V. Methods

For the system testing performed in this paper, all were run under Ubuntu 14.04 LTS server edition with 40GB of storage and 3.86 GB of RAM. All of the software deployment packages were configured to be run in a stand-alone fashion, if possible, and no system or package optimization was attempted. After each installation, all packages and unused libraries were removed to provide a clean slate and to allow for adequate comparability of the metrics collected during the automated software deployment process. As mentioned above, the key metrics included amount of code (lines of code, word count, character count, Table 2) and software deployment times both at full run as well as at no change for the two packages, separately and combined (

Table 3). Details about the features of the three software deployment tools can be found in Table 1. All of the execution of the code was timed using the shell command "*time*" in which the real-time was reported. The total file size of the three packages (gnuplot, pspp, and octave) being installed in the analytics package consumed 210.158MB of disk space as reported by apt-get and the four packages (lammps, mricron, mricron-data, and tm-align) being installing in the chemistry package consumed 74.854MB of space.

All of these tools are capable of being used for software deployment. However, tying these tools into a platform that can easily do infrastructure deployment also needs consideration. It is important that after deployment of the software and infrastructure the system is left intact with as little perturbation from the deployment method as possible. As such, Ansible is used in deployment method as our primary tool.

Table 2 - Software Configuration Code Characteristics

	Lines of Code[3]	Word Count[3]	Character Count[3]
Ansible	19	74	469
Salt Stack	11	31	166
Chef	21	69	328
Shell	8	52	316

Table 3 - Software Deployment Times in Seconds

	Time(s)	Chemistry Package	Analytics Package	Chemistry & Analytics
Ansible	Full Run	329.7	289.1	513.7
	No Change	434.7	153.6	431.8
Salt Stack	Full Run	172.7	205.9	201.8
	No Change	35.7	35.7	35.7
Chef	Full Run	44.2	87.9	101.5
	No Change	27.2	27.2	27.3
Shell	Full Run	117.6	128.6	383.4
	No Change	102.1	76.6	204.2

[3] This number is based off of the combined package code.

VI. Results and Conclusion

All three of these configuration management tools are more than capable of achieving the simple tasks that were assigned to them along with much more complex jobs if necessary. Likewise, all of them have comparable features with the exception of the agentless feature of Ansible which is unique in this environment. This gave Ansible an advantage when it came to configuring the host machines. All that is required of Ansible is the system be available with an IP Address and that it can host an SSH connection [19]. Both SaltStack and Chef required agents be installed on the host machines [20].

When it comes to execution however, both SaltStack and Chef significantly outperformed Ansible. Full run Ansible took over 513 seconds, compared to 202 seconds for SaltStack and a little over 101 seconds with Chef. Ansible was 1.54X slower than SaltStack and 4X slower than Chef.

That being said, there are some limitations of this work that need to be acknowledged. First, the code executed was not optimized and kept as simple as possible to prevent any bias based on coding. Second, it should be expected that the code would scale out linearly as more additional nodes are added since each of these tools are capable of parallelized connections to nodes limited only by the bandwidth connecting them. However, this parallelization was not tested beyond 2 nodes. Third, the deployment packages (chemistry and analytics) tested here were limited to installing a few pieces of software. A more comprehensive test involving deploying specific packages to hundreds of targeted nodes based on their roles along with configuring the software and starting their services, etc. would certainly impact the deployment time and more rigorously evaluate how efficiently the management tools work.

In conclusion, the end use of the product depends on the use case of the company. If simplicity of code is critical, Salt Stack is the clear choice with close to half the amount of code needed compared to Chef and nearly four times less code than Ansible. Likewise, if speed is important, then Chef is the winner. It took half the time compared to Salt Stack and a fifth of the time compared to Ansible. Lastly, if neither speed of deployment nor size of code is the primary concern then Ansible is the most interesting project of them all considering no additional agents need to be installed. Likewise, the time shown in Table 3 does not reflect any additional time required to install and configure the agents on each host which may not play a significant factor in the overall situation, it should be considered. One additional benefit for Ansible is that while the process may take slightly longer, there is an option that allows for dry-runs to be executed to verify that your code will work prior to execution which could save time overall, especially for complex tasks.

Regardless of which deployment tool or tools a company uses, these tools and this level of automation is critical in todays world of cloud computing which systems can be brought up and down in an instance. Which tool is chosen depends on the company policies and level of comfort of the user, but the benefits of reproducibility, speed, and mass deployment is significant.

VIII. REFERENCES

[1] Vision Solutions, "Assessing the Financial Impact of Downtime: Understand the factors that contribute to the cost of downtime and accurately calculate its total cost in your organization.," Vision Solutions Inc., Irvine, CA, 2014.

[2] U.S. Bureau of Labor Statistics, "Table B-3. Average hourly and weekly earnings of all employees on private nonfarm payrolls by industry sector, seasonally adjusted," 03 04 2015. [Online]. Available: http://data.bls.gov/cgi-bin/print.pl/news.release/empsit.t19.htm. [Accessed 30 04 2015].

[3] R. Di Cosmo, A. Eiche, J. Mauro, G. Zavattaro and S. Zacchiroli, "Automatic Deployment of Software Components in the Cloud with the Aeolus Blender," Inria Sophia Antipolis, Valbonne, 2015.

[4] T. P. Morgan, "A Rare Peek Into The Massive Scale of AWS," enterprisetech.com, 14 11 2014. [Online]. Available: http://www.enterprisetech.com/2014/11/14/rare-peek-massive-scale-aws/. [Accessed 02 06 2015].

[5] Ansible Inc., "WhitePaper: Ansible In Depth," ANSIBLE, INC, 2014.

[6] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. "Using control theory to achieve service level objectives in performance management". In Proc. of IM, 2002.

[7] G. Soundararajan, C. Amza, and A. Goel. "Database replication policies for dynamic content applications". In Proc. of EuroSys, 2006.

[8] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. "Dynamic provisioning of multi-tier internet applications". In Proc. of ICAC, 2005

[9] P. Rad , R. Boppana, P. lama, G. Berman, and Mo Jamshidi, "Low-Latency Software Defined Network for High Performance Clouds", in Proceedings of the 8th IEEE System of Systems, 2015, pp. 805-812

[10] Google Cloud Platform, "Compute Engine Management with Puppet, Chef, Salt, and Ansible," [Online]. Available: https://cloud.google.com/developers/articles/google-compute-engine-management-puppet-chef-salt-ansible/. [Accessed 02 05 2015].

[11] SaltStack - Github, "RAET (Reliable Asynchronous Event Transport) Protocol," GitHub, 28 02 2015. [Online]. Available: https://github.com/saltstack/raet/blob/master/README.md. [Accessed 02 05 2015].

[12] Ansible, "How Twitter Uses Ansible," Twi, 21 05 2014. [Online]. Available: https://www.youtube.com/watch?v=fwGrKXzocg4. [Accessed 02 05 2015].

[13] J. Jarvis, "How edX uses ansible," Speaker Deck, 20 05 2014. [Online]. Available: https://speakerdeck.com/jarv/how-edx-uses-ansible. [Accessed 02 05 2015].

[14] Chef Software, Inc., "Chef Customers," Chef Software, Inc., [Online]. Available: https://www.chef.io/customers/. [Accessed 02 05 2015].

[15] SaltStack, "6. Reactor System," SaltStack Inc., [Online]. Available: http://docs.saltstack.com/en/latest/topics/reactor/index.html. [Accessed 02 05 2015].

[16] J. Miranda, "Lyft Replaces Puppet with SaltStack," InfoQ, 22 08 2014. [Online]. Available: http://www.infoq.com/news/2014/08/lyft-moves-to-saltstack. [Accessed 02 05 2015].

[17] SaltStack, "SaltConf15 - Overstock.com - Automate Self-Service Provisioning of Developer Workstations," 26 03 2015. [Online]. Available: https://www.youtube.com/watch?v=qqPRNU1tia4. [Accessed 02 05 2015].

[18] SaltStack, "SaltConf15 - Rackspace - Automating Hybrid Cloud Infrastructures," 14 04 2015. [Online]. Available: https://www.youtube.com/watch?v=yDv_H9j1PI0. [Accessed 02 05 2015].

[19] "Inventory," Ansible, Inc., 22 04 2015. [Online]. Available: http://docs.ansible.com/intro_inventory.html. [Accessed 24 04 2015].

[20] W. E. E Wong, "OpenSource Automation in Cloud Computing," in *Proceedings of the 4th International Conference on Computer Engineering and Networks*, Springer International Publishing, 2015, pp. 805-812.