Virtual infrastructure orchestration for cloud service deployment

Qadeer, Arsalan; Malik, Asad Waqar; Rahman, Anis ur; Hamayun, Mian Muhammad; Ahmad, Arsalan

DOI:
10.1093/comjnl/bxz125

Document Version
Peer reviewed version

Citation for published version (Harvard):
Qadeer, A, Malik, AW, Rahman, AU, Hamayun, MM & Ahmad, A 2019, 'Virtual infrastructure orchestration for cloud service deployment', *The Computer Journal*. https://doi.org/10.1093/comjnl/bxz125

Virtual Infrastructure Orchestration for Cloud Service Deployment

Arslan Qadeer[1], Asad Waqar Malik[1,2], Anis Ur Rahman[1,2], Mian Muhammad Hamayun[1,3] and Arsalan Ahmad[1]

[1]*School of Electrical Engineering & Computer Sciences, National University of Sciences & Technology (NUST), Islamabad, Pakistan*
[2]*Department of Information System, Faculty of Computer Science & Information Technology, University of Malaya, Malaysia*
[3]*School of Computer Science, University of Birmingham, Dubai, United Arab Emirates*
Email: {14msitaqadeer, asad.malik, anis.rahman, mian.hamayun, arsalan.ahmad}@seecs.edu.pk

Cloud adoption has significantly increased using the infrastructure-as-a-service (IaaS) paradigm, in order to meet the growing demands of computing, storage, and networking, in small as well as large enterprises. Different vendors provide their customized solutions for OpenStack deployment on bare metal or virtual infrastructure. Among these many available IaaS solutions, OpenStack stands out as being an agile and open source platform. However, its deployment procedure is a time taking and complex process with a learning curve. This paper addresses the lack of basic infrastructure automation in almost all of the OpenStack deployment projects. We propose a flexible framework to automate the process of infrastructure bring-up for deployment of several OpenStack distributions, as well as resolving dependencies for a successful deployment. Our experimental results demonstrate the effectiveness of the proposed framework in terms of automation status and deployment time; that is, reducing the time spent in preparing a basic virtual infrastructure by four times, on average.

Keywords: Cloud Computing; Cloud Deployment; Virtual Infrastructure; Orchestration; OpenStack

Received The Computer Journal; revised Preprint version, accepted: Sept 2019

1. INTRODUCTION

With the ever-increasing need of IT infrastructure for computing and storage in almost all domains, the traditional approach of installing and maintaining local computing resources has become risky, costly, inflexible and non-fault-tolerant. These issues are effectively addressed by cloud computing, as it provides an alternate solution for computing, storage and application services. Additional benefits of cloud services include scalability and flexibility for growing businesses such as scale-up and scale-down for a capital expenditure savings (pay-as-you-go) including upfront capital cost reduction [1]. Further, all major industry players are offering cloud solutions. For instance, Amazon EC2, Microsoft Azure, Google Cloud and IBM blue cloud are available to their clients globally.

Cloud computing offers different service models such as infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS) and software-as-a-service (SaaS). The IaaS service model is used to host operating systems, servers, storage, networking and other infrastructure components to build a cloud based hardware platform. In this model, the end users manage their operating system and application requirements. In the PaaS service model, a cloud provider delivers the aforementioned infrastructure resources, including operating system, as well as middleware, runtime environments and a set of tools required for application development. Lastly, in the SaaS model, a provider hosts all resources with pre-installed software applications and makes them available for its customers on the cloud. The end-users are not required to maintain anything, including hardware and software.

A cloud deployment model defines how a cloud environment and its services are deployed and made accessible to its clients. These models govern the purpose and nature of the cloud deployed. The deployment models are often influenced by the proprietorship, size and access patterns under consideration. There are three basic types of cloud deployment models *i.e.* a private cloud owned and used by a single organization, a public cloud provisioned for public use and a hybrid cloud, which is a mix of the

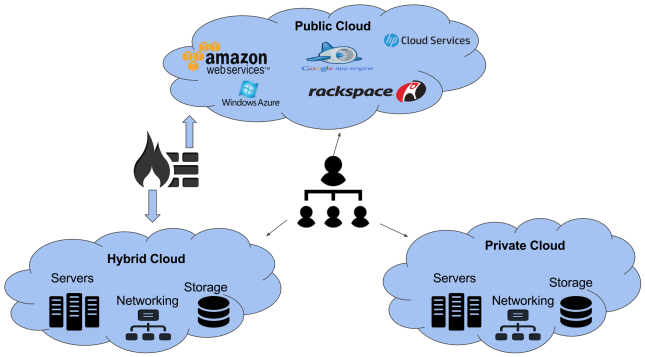

FIGURE 1. Cloud Deployment Models

previous two models and provides greater flexibility as illustrated in Fig 1.

The main focus of this paper is the deployment of a cloud setup using the IaaS service model, as it is the fundamental cloud service model and poses many technical challenges. All of these challenges are also *inherited* in the PaaS and SaaS models, as both of these models use IaaS as their foundation. Currently, several proprietary and a few open-source IaaS solutions exist, including Amazon AWS, Windows Azure, Google Compute Engine, IBM SmartCloud, Rackspace Open Cloud and HP Enterprise Converged Infrastructure.

Over the years, OpenStack [2] has received significant acceptance from the cloud community due to its open source nature and pluggable architecture. The OpenStack solution provides IaaS with computing, network, and storage services, which are all managed via a dashboard. Many industry players offer their own flavors of such solutions and most of them are based on OpenStack; that is, around thirty distributions are available that deploy and setup OpenStack in private as well as public clouds [2]. Moreover, according to a survey report [3], OpenStack is being used in IT/telecommunications, academia/research, media and financial institutions. Notably, IT covers 68% of the total usage of OpenStack with 65% of the total deployments running at the production stage. The main reasons behind the popularity of OpenStack include cost savings, increased operational efficiency and the ability of organizations to innovate and compete by deploying applications faster.

Many studies on the OpenStack deployment process, as discussed in detail in the next section, have highlighted the potential limitations of the current deployment strategies. Some of the significant limitations motivating this work includes the time taken per deployment, the need for extensive manual efforts and the high probability of human errors. We propose an automated OpenStack deployment framework for the IaaS service model that shows a significant reduction in the overall deployment times, reduces manual efforts drastically and decreases the probability of human errors to almost zero. With the help of the proposed virtual infrastructure toolkit, researchers and developers will be able to efficiently deploy any number of OpenStack based cloud instances in no time and will be able to test and validate different features quite easily. This will have a significant positive impact on the cloud adoption in all domains, and even naive users are expected to deploy OpenStack for their organizational needs.

The rest of the paper is organized as follows: Section 2 presents a comparative study of existing IaaS deployment solutions and discusses OpenStack and its partially automated deployment strategies. Section 3 formulates the problem by outlining the system model and its requirements. Section 4 describes the proposed automation framework to bring-up a complete virtual infrastructure, which is ready for deployment of multiple OpenStack distributions. Sections 5 and 6 include the implementation details and experimental results for the proposed framework. Finally, Section 8 concludes the paper with a general summary and possible future directions to enhance the proposed solution.

2. RELATED WORK

Automatic infrastructure deployment plays a significant role in increasing the adoption of a particular framework.

In general, the main objectives of automation include: i) to increase productivity due to simple and repeatable deployments, ii) reduce the overall deployment costs both in terms of time and money, and iii) ensure quality and functionality of the deployment, with no deployment errors. Particularly, in the context of cloud framework automation, for instance, the cloud software developers can focus on particular research areas such as provisioning of resources in real-time, infrastructure components, resource monitoring and adaption of cloud features to the existing platforms. However, the existing open-source cloud deployment solutions lack the necessary features, which in turn makes them difficult to use for the development and testing of cloud software.

Generally, two types of approaches are used for deployment *i.e.* top-down and bottom-up [4]. The former is mainly model-driven and enforces middleware abstraction and promotes flexibility. However, this approach is semi-automatic and requires serious efforts to replicate. These solutions often involve the use of a description standard to specify application deployment and management parameters, for instance, TOSCA [5] is an XML-based language using a typed topology graph for application deployment. The format used is ideally portable and modular making the deployment easier across multiple platforms. On the other hand, the bottom-up approach uses some kind of DevOps Some example of such DevOps are:

Chef[1], Puppet[2], Crowbar[3], Control-tier, Capistrano[4], CFEngine 3[5], MCollective, Juju[6], AWS cloudformation and AWS opsWorks[7], which are client-server based, whereas SmartFrog and SLIM [6] use a P2P setting. In all cases, the approaches are application-specific and offers limited flexibility.

Many of the cloud stacks use the aforementioned automation approaches primarily to create and manage cloud services including resource management, user management, access APIs and GUIs. Some example of cloud stacks are: OpenNebula [7], Nimbus [8], OpenStack [2], Eucalyptus [9], CloudStack [10] and vCloud[8]. Many of these are open-source solutions allowing the users to write scripts for cloud deployment and management; but, they lack in automation of basic networking infrastructure setup, as well as creation and deployment of compute nodes.

To support the cloud stacks, many open-source libraries and frameworks are available that allow the users to get started with cloud application deployment. Generally, the libraries including jCloud [9] and MADCAT [11] provide an abstraction of different cloud features for cloud application development, deployment and control. Other libraries like PaaSage [12] supports model-based development, configuration, optimization and deployment; MODAClouds enables the cross-cloud migration to exploit available resources on multiple clouds; ARTIST [13] supports re-engineering and migration of legacy software to cloud-based environments; CELAR [14], 4CaaSt [15] and CloudScale [16] help manage allocation of resources for cloud applications, as well as, provisioning of cloud services and applications.

Apart from the aforementioned stacks and libraries, OpenStack is getting popular among the different open-source cloud solution providers. In contrast to the other solutions, it supports multiple cloud API sets, providing a comfort level preventing lock-in of cloud workloads [17, 18]. Moreover, with a faster release cycle, it is evidently emerging as an industry leader in cloud solutions. From a functional and architectural perspective, a number of studies have compared OpenStack to other IaaS solutions and concluded it as being massively scalable and agile as discussed in [19, 20]. For instance, Bashir et al. [21] present a case study using OpenStack to demonstrate the feasibility of migration from a traditional web service to a cloud-based solution. A similar study to analyze migration of scientific applications to a cloud-based environment concludes that cloud-based solution results in negligible performance degradation [22]. It is worthwhile to note that even such short-term experiments consume significant amount of time and effort, just to bring-up the basic infrastructure for OpenStack or similar cloud environments. Thus, the cloud infrastructure automation toolkit can significantly reduce time and effort.

Similar experimental cloud bring-up challenge is demonstrated in [23, 24] where the authors deploy different open source cloud solutions for comparison and present a list of steps followed during the deployment process. It is evident that the task is hectic, even a minor human error can lead to a cloud setup in an unstable state. Additionally, the steps followed are far more difficult to revert back, and in most cases the installation effort has to be restarted from scratch. This becomes more challenging when deploying a private cloud [25] or a cloud on a multi-node environment [26] using OpenStack. Notably, most of the aforementioned studies have either ignored the pre-requisites of OpenStack deployment or lack focus on basic infrastructure automation.

Particular to this study, even the latest distributions of OpenStack require manual infrastructure setup. The OpenStack Ubuntu distribution [27], an open-source IaaS solution for both private and public clouds, uses Juju and Charms to deploy OpenStack. Moreover, the distribution requires VM called metal-as-a-service (MaaS) node where all the required packages must be installed and configured.

As the initial steps to install the distribution are manual, they take considerable effort, and are even troublesome as they can lead to an error-prone configuration. DevStack [2] does provide automation to deploy a minimal OpenStack setup on a single or multiple nodes but still assumes that the basic infrastructure is already available.

RackSpace[10], an OpenStack distribution for public and private clouds, uses Ansible [28] for configuration management. It does offer an all-in-one solution for testing and demonstration purposes but still lacks the basic infrastructure bring-up; that is, installation of operating system, network configurations, nodes interface configurations, setting-up of bridges, password-less SSH access and many other settings have to be performed manually. Similarly, RDO OpenStack [29] is an RPM distribution for stable, scalable and secure cloud deployment. However, the deployment work-flow is quite different and has a steep learning curve involving many manual steps for setting up the environment for cloud including the installation of operating system, the network configurations of director and overcloud nodes, packages installation etc.

In another comprehensive study [30] of a substantial approach to fully automate the key phases of NFV service lifecycle, software-based management,

[1]https://www.chef.io/
[2]https://puppet.com/solutions/cloud-management
[3]http://crowbar.github.io/
[4]http://capistranorb.com/
[5]electric-cloud.com
[6]https://jujucharms.com/
[7]https://aws.amazon.com/cloudformation/
[8]https://www.vmware.com/
[9]https://jclouds.apache.org/
[10]https://www.rackspace.com

and orchestration (MANO) stack is presented which operates with OpenStack and OpenDaylight controllers. The key phases specifically include resource discovery and matching, service mapping, service deployment, and monitoring. The authors present a modular approach in a service-oriented architecture, where each of the components can be discretely consumed in a plug-and-play manner. As stated, this framework only addresses the native service lifecycle challenges of NFV and does not consider the underlying infrastructure orchestration for OpenStack which is under discussion. In another study, [31] learning scenarios in the area of IaaS, SaaS, and academia are presented by deploying OpenStack cloud for University campus. The cloud deployment began after manual configuration of hardware nodes and network interfaces. The manual setup is time-consuming, prone to errors and required expertise in Linux environment. It is evident that all the distributions discussed above require manual infrastructure setup and configurations, which is a tedious and error-prone task. Table 1 illustrates a summary of the currently available solutions offering automated tasks to bring-up an OpenStack cloud. It is evident that to prepare the initial infrastructure for OpenStack deployment, all of the dependencies must be resolved manually, requiring hectic efforts. Thus, an automated infrastructure deployment solution is strongly desired to avoid misconfiguration errors.

3. PROBLEM FORMULATION

In this section, we present the system model assumed in our work and formulate the problem of deployment orchestration for the cloud.

3.1. Terminology

We define a virtual infrastructure as a set of functions that enable the sharing of physical resources of machines across the entire infrastructure in a prodigious way. It distributes the software layer from underlying hardware in order to aggregate physical servers, storage cabinets and underlay networks into a shared pool of resources. Thereafter, allowing the provision of these accumulated inexpensive building blocks to applications dynamically, securely and reliably.

We depict an orchestrator as a tool automating the entire application life cycle on a virtual infrastructure, we follow a simple definition of an application. This includes its creation and deletion following a set of steps, commands, and instructions for provisioning the aforementioned building blocks.

The primary feature among all is virtual machine (VM) configuration wherein computing resources are provisioned to a VM for an actual program to execute. Prior to this, a virtual network is set up to render isolated paths among communicating VMs, usually homogeneous belonging to the same application or tenant. This is followed by the installation of packages acting like tools and daemons for use with the applications. The result of such virtual infrastructure orchestration is a set of VMs organized in a specific topology aiming to provide resources and failovers to maintain high availability of services.

3.2. System Model

For virtual cloud orchestration model, consider a cloud computing platform providing IaaS. The platform supports execution of N physical systems acting as servers. These servers are used to run various services on the platform. Moreover, the servers are used to assign VMs to users U. Every physical node can support up to k virtual machines. The system configuration of each machine is represented as C_i. Based on the physical system and VM configuration, only q VMs can be hosted on a single server. In the proposed framework, we assume that servers are directly connected or placed at the same location, and hence, the network delay for VM migration and the time taken to perform other tasks at servers is negligible.

Let η represent the number of nodes available for cloud orchestration. The node responsible for network management is denoted as ϑ, ψ represents the controller nodes used to provision cloud resources ξ over computing nodes. Furthermore, a separate node is required to configure storage represented as ϱ. The proposed infrastructure deployment framework starts its execution from a configuration file. The user needs to specify core deployment parameters such as the type of cloud computing infrastructure, the number of nodes, and related configurations for the control, storage and compute nodes. User-initiated computing requests are first received at the ψ node, which provisions resources at ξ nodes and configures the network with the help of ϑ.

Once a specified cloud configuration has been deployed, a ξ module tests all interfaces through control messages. The ψ node checks the entire system set up as per the initial configuration specified by the user. Furthermore, the ψ node checks its connection with ξ and ϱ nodes. For a successful deployment, the user is required to mention ψ, ξ and at least one ϱ in the configuration file. A typical distributed system framework relies on multiple levels of services; that is, services executing on l_m layer may coordinate with other services executing at l_{m-1} layer for correct execution. Notably, manual deployment of such multi-layer services can become a time-consuming effort, which is prone to errors. Similarly, other essential services including monitoring, synchronization, and resource allocation need such automation for efficient cloud orchestration and management.

TABLE 1. Distribution-wise task automation support.

Distribution	Infrastructure setup	Network configuration	OS installation	Packages installation
DevStack	×	×	×	×
Rackspace	×	×	×	×
Canonical	×	×	✓	✓
RDO	×	×	✓	✓

3.3. Requirement Analysis

A virtual infrastructure orchestration is a complex deployment system and relevant research efforts have proposed a wide range of goals for an orchestrator. We identify the following functional properties:

Coordination: Monolithic server-side applications are being migrated into microservices based architecture comprising front-end, business logic, database access and application integration with other services via REST API. This layered approach aims at the decomposition of an application into a set of manageable services for faster development, making them easier to understand and maintain. Notably, coordination in a large complex and distributed system is critical, which can be ensured by implementing effective deployment procedures. The challenges in such deployment are keeping track of services and their synchronization. Furthermore, the definition of proper networking work-flows and installation steps is important to guarantee coordination among services through inter-service communication.

Automation: Monolithic applications can be replicated on different servers. In contrast, a microservice poses a distributed system containing a large number of services. Each of these services further relies on other services that need to be spawned, configured, scaled and monitored. Such multi-level complex deployment requires a high level of automation.

Resource provision and monitor: The most significant part of orchestration is the effective scheduling of resources, which is necessary for cost saving. Traditional approaches lead to inefficient utilization of underlying hardware resources, as it is difficult to manually monitor setup of modules for rescheduling resources and to manage faulty nodes and services. Therefore, the goal is to tackle the aforementioned issues by abstracting rudimentary operations, while ensuring the integrity of coordination, automation, resource provisioning and monitoring.

3.4. Problem Statement

OpenStack is emerging as an industry leader in open source cloud solutions. Figure 2 shows a bare minimum OpenStack deployment. In order to setup OpenStack, all components including the controller, the compute nodes, storage and network must be installed and configured correctly with all of the dependencies resolved beforehand. This process requires prodigious manual effort and sound technical knowledge, as there is no proper automation mechanism available that can perform all of the initial set-up for a quick OpenStack deployment. Thus, a manual cloud bring-up can lead to an error-prone set-up and subsequently a misconfigured OpenStack deployment.

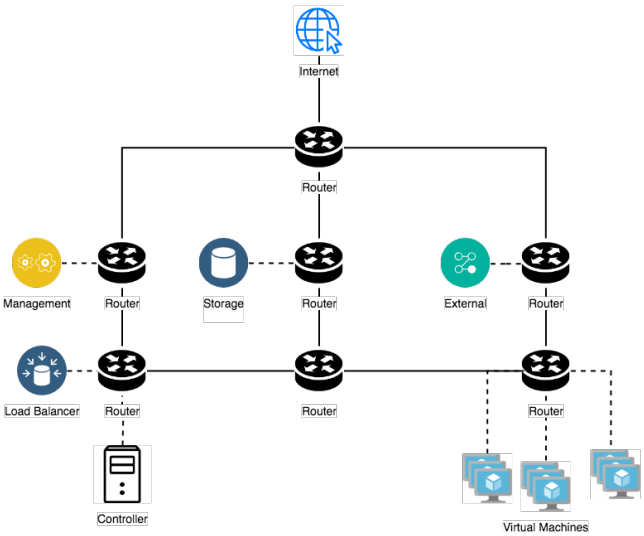

FIGURE 2. OpenStack deployment setup and configuration

Due to the complex set-up procedure, a systematic and automated mechanism is required that can rapidly deploy complex open source cloud infrastructures to facilitate researchers, developers and even those who are low on technical skills. The proposed framework, as explained in the following section, is built around the afore-mentioned OpenStack distribution with the design flexibility to make it work with any available open source distribution. In order to increase the proliferation of OpenStack and facilitate technology adoption, automation of cloud deployment has become a necessity.

4. DESIGN AND ARCHITECTURE

In this section, we discuss core modules of our proposed toolkit. The modules are flexibly designed to support various software configurations. Figure 3 shows the main modules: toolkit deployment manager, network infrastructure manager, platform setup manager, software manager and service manager. The details and working of each of these modules are discussed in the

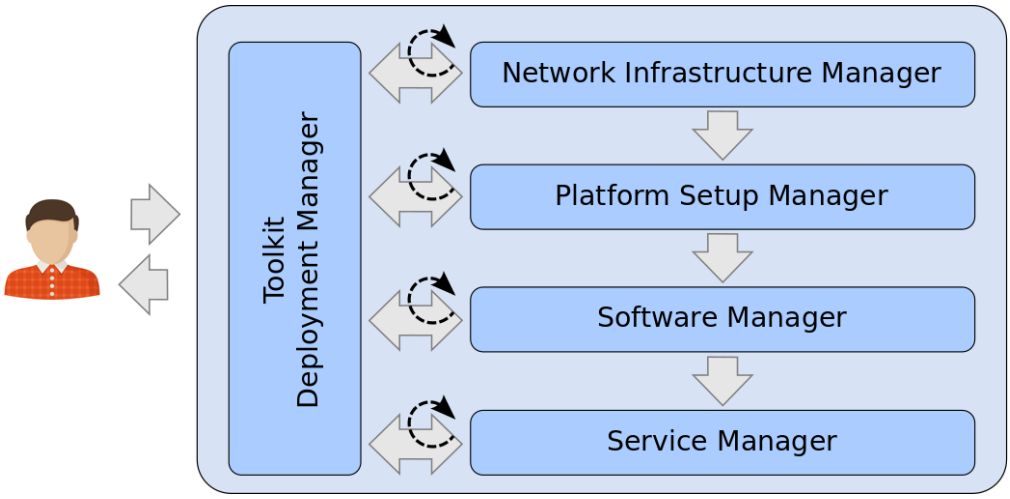

FIGURE 3. Architecture Design of the Proposed Toolkit

following sections.

4.1. Toolkit Deployment Manager

Toolkit deployment manager (TDM) module acts as the overall in-charge of the proposed toolkit. It is responsible for monitoring other modules and upon failure, it re-initiates the failing process. It also maintains all logs and the running history of all modules. The user interacts with the deployment module for setup configurations. Furthermore, the module also manages user network specification that may vary depending on the OpenStack distribution. User deployment-specific configuration includes features like DHCP-enabled or NAT-based networking. Previously, these configurations were managed via a command line interface; in contrast, the proposed deployment toolkit handles all of these dependencies and other network related issues itself and the end user is not required to intervene.

As discussed above, the users interact with the toolkit deployment module, state initial user setup parameters in the form of an XML file. These parameters are used to provision the virtual machines and configure them. The file also includes information such as the number of VMs, the type of VMs, the OS image, the number of network interfaces attached to each VM and many other parameters. Currently, the XML file is written in Ruby syntax. Furthermore, a single configuration XML file defines one project that is transparent to other provisioned resources. Since one project may contain more than one VM, at present, there is no restriction on the number of resources that can be provisioned at any time, but the resulting performance is strongly linked to the underlying hardware.

4.2. Network Infrastructure Manager

Network infrastructure manager (NIM) is responsible for managing network interfaces within every project. Note that every OpenStack distribution requires different network settings; that is, comprising of a single or multiple network interfaces and bridges connected to other virtual machines within the same cloud setup. For example, RackSpace requires a single management network supporting NAT. It also needs multiple bridges like *br-vxlan*, *br-vlan*, *br-mgmt* and *br-storage* to support Linux Containers [28]. Similarly, other cloud distributions have their own requirements for networking that are catered using the NIM module. Moreover, it automatically configures the maximum transmission unit (MTU) of the network interfaces, if required.

4.3. Platform Setup Manager

Platform setup manager (PSM) is responsible for provisioning of virtual machines to deploy identical working environments. Generally, it uses a customized package containing an operating system with pre-installed libraries and applications. The customization is in terms of the operating system distribution, the kernel version and other system and/or application-specific settings. The PSM module provides the aforementioned functionality using a pre-configured vagrant box added in the vagrant domain. We have used the *vagrant-libvirt* to bring-up the configured number of virtual machines with specified network settings. It should be noted that *libvirt* has been used instead of the default provider *i.e.* VirtualBox, in order to support the kernel virtual machine (KVM) module.

4.4. Software Manager

Software manager (SWM) module is responsible for the resolution of missing dependencies on the newly created hosts. This is accomplished via a dependency checking procedure that verifies all pre-requisites for a successful installation. The SWM module can support different distributions, setup/create networks in the

libvirt domain, create the required configurations for VMs and trigger other events.

4.5. Service Manager

Service manager (SM) module aims to configure virtual machines automatically. This module uses Ansible [28], which is an open source automation engine for cloud provisioning, configuration management, and application deployment. Ansible provides an agent-less architecture, which does not require any servers, daemons or databases. The Ansible engine is installed on a single node, referred to as the *deploy node* and it requires password-less SSH access to an inventory file containing information about the remote machines. The SM module is also responsible for configuring and deploying all required applications for successful OpenStack deployment. It works by connecting to the remote machines over SSH and pushing customized programs called *Ansible Modules* to the remote VMs, where these modules perform configuration management tasks.

4.6. Toolkit Work-flow

Using the proposed work-flow, a user can download and complete the cloud deployment procedures using a wrapper script. The script requires necessary details and performs the needed required steps without any user intervention. This can be done in two ways:

(a) Set the wrapper to ask for the detailed parameters as input, as per requirement.
(b) Define a configuration file with some variables and provide this as a parameter file to the wrapper.

The later option is more user friendly as it avoids most of the human errors, and hence has been adopted in this work. The configuration file can be re-used multiple times and would result in exactly the same cloud configurations.

To make it simpler for end users, the proposed framework requires fewer parameters related to deployments such as the number of nodes to bring-up, number of controllers/compute nodes and some other information depending upon the selected distribution. During the later stages, it allows the users to enter subnet details for the mentioned network type. Since different and multiple networks are required for different distributions; for example, for DevStack a management network is required, whereas RDO OpenStack needs two networks: a provisioning network and an external network (under-cloud) in a converged networking model. Once the network is created in libvirt and the corresponding bridges are setup on the base machine, a Vagrantfile is populated automatically. All dependencies are then automatically resolved, packages are downloaded and installed. Once the dependencies are resolved, vagrant brings up the virtual machines using Vagrantfile and libvirt provider. Vagrant then invokes Ansible on the up and running virtual machines, which uses *playbooks* to provision machines with specific package installations and configurations. The initial infrastructure setup is hence complete and ready to deploy an OpenStack cluster, which has been chosen as a case study for this work. Figure 4 illustrates the details of the proposed work-flow for all currently supported OpenStack distributions.

5. IMPLEMENTATION DETAILS

The primary purpose of this work is to automate the complex procedure of configuring and installing a cloud environment. Hence, the cloud users are required to launch the proposed framework that manages the complete process and provides the user with ample options to configure the desired cloud setup. The main objective of this work is to automatically handle all dependencies and install the pre-requisite packages. Interestingly, the proposed framework can attract more researchers to adopt the use of a cloud platform for research and development. It is worthwhile to mention that users with limited knowledge of the Linux usually feel reluctant to use it due to the lack of technical expertise to resolve different system related issues. In this work, we aim to bridge this gap as well.

5.1. Toolkit Deployment Manager

The TDM is responsible for collecting all the relevant information from the system as well as the user, validating it and storing it for later use. This information is gathered using a terminal comprising all configurations necessary to bring up the desired infrastructure or distribution. Afterwards, the user entry is validated using regular expressions and other algorithms to identify any invalid user inputs. This step also includes provision of an IP address range in a specific format. In order to build a virtual network, XML file format is used with a separate section for IP, DHCP and other required network configurations.

Furthermore, the TDM is also responsible for resolving dependencies after checking of available packages on the required systems, if necessary, the manager can download and install any missing packages. Basically, it is a helper module that contains the list of packages, libraries, and dependencies required by different components of the toolkit. For instance, all packages required are installed before the installation of Vagrant, Ansible, libvirt etc. In order to install the packages, a simple package manager provided with various operating systems is used, such as APT/DPKG in Ubuntu while Yum installer in CentOS. Moreover, to install the required plugins, the vagrant command line is used.

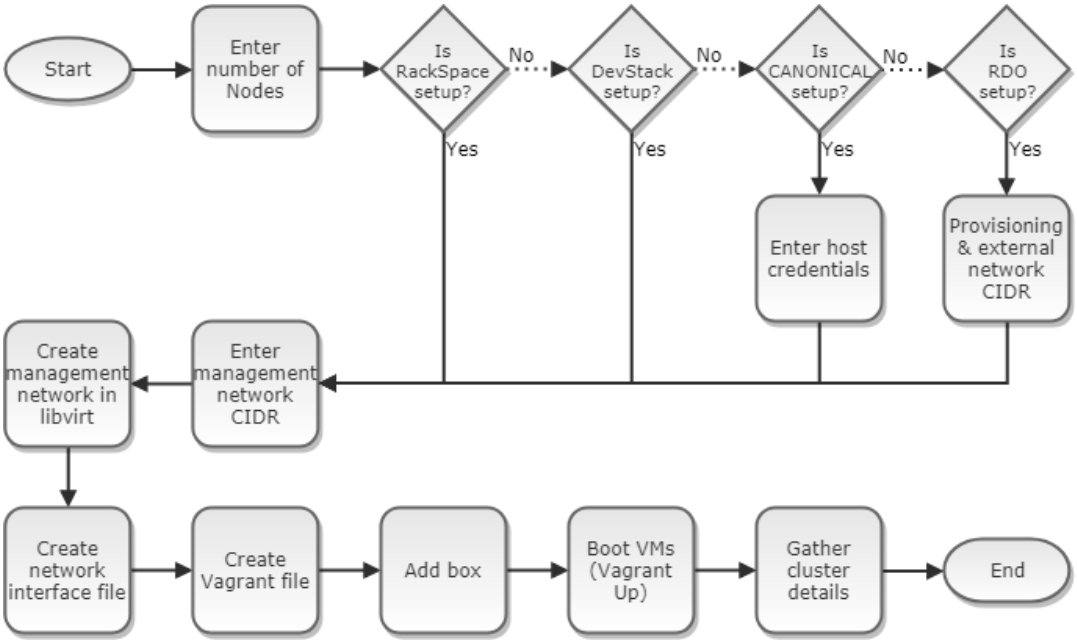

FIGURE 4. Detailed Workflow of the Proposed Deployment Toolkit

5.2. Network Infrastructure Manager

The NIM is responsible for creating a virtual network that is required by the specified distribution. It starts the required distribution followed by virtual network deployment in the libvirt domain. Moreover, the network interfaces are identified and created. The manager uses libvirt CLI to create a virtual network using the XML configuration file recorded by the toolkit deployment manager. All the required network-related information is extracted from the XML file such as IP ranges, DHCP enabled (yes/no) and other settings. Thus, as a result of these steps, a complete Vagrant box with pre-packaged development environment is added, and a virtual network is made available for the selected cloud distribution.

5.3. Platform Setup Manager

The PSM module is responsible for the platform setup. It invokes the virtual machines using the specified drivers such as libvirt. Primarily, Vagrant is used to create virtual machines whereas libvirt is the main provisioning tool that provides a runtime for the virtual machines to run and operate. All nodes related information is maintained by Vagrant including the number of nodes and their specifications. These specifications include information regarding the CPUs, RAM, NICs, hard disk, provisioning tools etc. The vagrant box is created using all the required information and added to the vagrant domain. During the last step, the PSM module spawns all of the VMs using the libvirt as provisioner and Vagrant as configuration provider.

5.4. Software Manager

Once the VM is setup, the next step is to install the packages and resolve dependencies within the VMs required for OpenStack setup. Ansible is mainly used at this stage, which is initialized by vagrant. The tool is used to configure and setup nodes remotely; that is, to prepare a complete node for deployment. It starts with the package installation that is common in all the distributions. However, Ansible uses *playbooks* for each individual role. These playbooks define each individual task and configuration settings that are to be performed inside a VM. Note that the software manager module also includes a proper validation mechanism to identify any missing packages and installs them before proceeding further. The SWM module results in up and running nodes required for the setup requested at the initial stage. The checklist of these dependencies (and sub-dependencies) is listed below:

- libvirt (libvirt-bin, qemu-kvm, ubuntu-vm-builder, bridge-utils, qemu, qemu-system, libvirtd, virt-manager)
- ansible
- vagrant
- vagrant-libvirt (ruby-libvirt, libxslt-dev, libxml2-dev, libvirt-dev, zlib1g-dev, ruby-dev)
- fog-libvirt

5.5. Service Manager

The Service Manager module is responsible for configuring all the required settings and gathers clustered information to show it to the user. Ansible is used again in this module. For gathering IP address related information from VMs, the traditional

Linux command *ip* is used. Moreover, libvirt CLI is used to find the MAC addresses whereas the director node credentials are already known to this module from the initial configuration file generated using the user input. Later on, the networks inside the VMs are configured. This step is also common across all distributions. Moreover, the SM module is also responsible for distribution-specific configuration such as SSH settings. In addition to the afore-mentioned procedures, Canonical and RDO distributions require some additional information such as credentials of the MaaS node (in case of canonical) and the director node (for RDO), as well as the MAC addresses of all target nodes to be used for OpenStack installation. All these node-specific settings are shown to the user in an easy and readable format. The outcome of this module is a complete installation of required packages and dependencies, and all of the required configurations for installation of a specific OpenStack distribution. Furthermore, it displays the clustered information and exits the toolkit workflow with a final status message that the infrastructure is ready for OpenStack installation.

6. EVALUATION RESULTS

In this section, we present a short description of the measurement metrics used to evaluate the proposed toolkit, followed by details of an experiment to gather these metrics. The obtained results are then presented and analyzed in terms of effort required to setup a basic virtual infrastructure environment for different cloud platform distributions.

6.1. Measurement of Deployment Effort

For this study, deployment effort corresponds to the effort of deploying a cloud computing platform. This quality attribute is used to evaluate deployability using the proposed orchestration toolkit. In order to measure this, we use a measurement framework based on the ISO/IEC SQuaRE quality model [32] including metrics average deployment time (ADT), deployment reliability (DR), number of deployment steps (NDS), and deployment steps parameters (DSP). The last three metrics are summed up to get two more metrics: effort of deployment steps (EDS) and deployment effort (DE). A description of each of these metrics is given below:

- **Average Deployment Time (ADT)** is the average duration between invocation of deployment script and its completion, leading to the cloud up and running.
- **Deployment Reliability (DR)** corresponds to the deployment reliability. It is the fraction of successful deployments among a number of attempts; that is, $DR = 1$ represents 100% success rate.
- **Number of Deployment Steps (NDS)** is the total number of steps to complete a task. It corresponds to the effort required for a successful deployment; that is, more the steps greater the effort.
- **Deployment Steps Parameters (DSP)** is the number of times a deployment script prompts for user intervention or asking for user input for custom parameter configuration.
- **Effort of Deployment Steps (EDS)** is the effort done to initiate a deployment script and its completion. A simple sum of NDS and DSP.
- **Deployment Effort (DE)** totals up the effort for a successful deployment.

$$DE = ((1 - DR) + 1) * EDS$$

6.2. Execution of Measurements

To compute the aforementioned metrics, we conduct a deployment experiment. Each individual deployment script invocation is timed using a typical timer function. Basically, the test comprises an initial IaaS infrastructure deployment followed by different cloud platform deployments. This test is run ten times to get an evaluation for each platform. The resulting metrics are shown in Table 2.

6.3. Effort Analysis

- **Average Deployment Time (ADT).** The mean of the deployment time is 60 min for manual deployment with a deviation of 22 min, while the automated solution takes only 14 min on average. We observe that the latter produces small differences among different distributions used for deployment of a cloud platform with a deviation of 4 min. Overall, the deployments are on average almost ∼4 times faster than the manual deployment. Note that timing results exclude download and troubleshooting times.
- **Deployment Reliability (DR).** For some distributions, we experience deployment failures while setting up the basic infrastructure ready for cloud deployment. Some recurrent problems result in lower DR values of 0.6, especially when preparing the physical nodes (OS and package installations with network settings), which is followed by the creation of virtual machines and bringing up of the underlying virtual network infrastructure for cloud deployment. In the case of automated deployment, on average, we experienced fewer failures resulting in higher DR values of 0.8.
- **Number of Deployment Steps (NDS).** In the case of automated deployment, we use scripts, and hence, the effort represents the total lines of code executed excluding logging and comments.

TABLE 2. Deployment efforts.

	devStack	RackSpace	Canonical	RDO
Effort of deployment steps (EDS)				
Manual	> 444	> 344	> 550	> 389
Automated	381	235	438	314
Number of deployment steps (NDS)				
Manual	> 376	> 230	> 431	> 309
Automated	376	230	431	309
Deployment steps parameters (DSP)				
Manual	> 68	> 114	> 119	> 80
Automated	5	5	7	5
Deployment reliability (DR)				
Manual	0.6	0.7	0.6	0.5
Automated	0.8	0.8	0.8	0.8
Average deployment time (ADT) in minutes				
Manual	> 60	> 40	> 90	> 50
Automated	9	15	18	15
Deployment effort (DE)				
Manual	> 622	> 447	> 770	> 583
Automated	457	282	526	377

On average this number is 336 operations with a deviation of 87. Note that in both deployment cases: manual and automated, the total operations are necessarily same; however, in the former case it is done by manual entering commands into the terminal while in the latter being a script runs automatically.

- **Deployment Steps Parameters (DSP).** Often deployment process prompts a user to input parameters to set some variables or configurations. This is an effort when deploying manually or using automation. But some of these parameters are either asked redundantly or are dependent on previous configurations. A script used for automation can store these parameters in variables to reuse for upcoming deployment steps, and hence, reducing effort. On average, manual deployment asks for 95 parameters with a deviation of 25. On the other hand, automated deployment requires 5 parameters with a deviation of 1 parameter across different distributions.
- **Effort of Deployment Steps (EDS).** The combined metric uses NDS and DSP to compute the effort of deployment steps to successfully deploy a cloud platform. On average, the metric for manual deployment is 432 with a deviation of 89, whereas the metric for automated deployment is significantly low with an effort of 342 ranging within a deviation of 87 for different distributions. Some distributions require more steps, whereas others require fewer steps but more parameters. DevStack is driven by a concise deployment workflow. In contrast, Rackspace, Canonical, and RDO require a cumbersome configuration of the initial infrastructure. Moreover, in the case of Rackspace, the low EDS value of 235 is contrasted by a large configuration file. The automation reduces effort with respect to the EDS through parameter reuse, even though the metric does not take into account the automation of execution of deployment steps.
- **Deployment Effort (DE).** The metric takes into account the failures and computes the total deployment effort after automation, with an average of 410 and a standard deviation of 105. This shows a significantly lower effort compared to manual deployment averaging at 606.

6.4. Summary of deployment times

Figure 5 illustrates the overall reduction in time to setup and configure infrastructure for the mentioned distributions. Note that download and troubleshooting times are excluded; however, during experimentation, the time is longer. For devStack virtual setup, it takes more than two hours to prepare the host for the first time. The preparation includes setting virtual machines in multi-node setup and OS installation, network settings, and installing dependencies. With our proposed work-flow, the complete procedure takes around 8 to 9 minutes to bring-up and provision virtual infrastructure, which is ready to run DevStack. Similarly, Rackspace needs one deployment node, three controller nodes and two compute nodes for a standard deployment, which takes vast time (almost 6 hours) from bringing-up virtual machines and setting the networks. It may also include VLANs configuration at switches. Using the proposed work-flow, it

only takes 12 to 15 minutes to bring-up a ready-to-use Rackspace standard virtual infrastructure for OpenStack deployment.

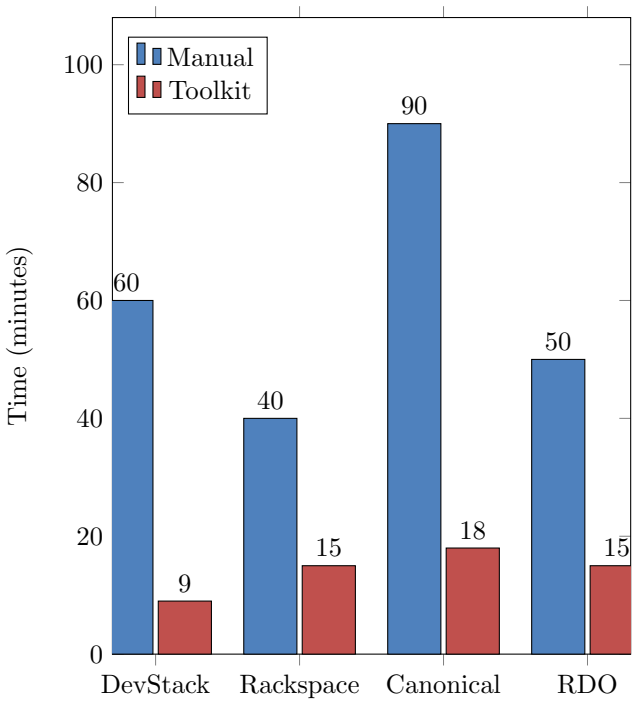

FIGURE 5. Time Spent on Preparing Basic Virtual Infrastructure with and without Toolkit for OpenStack Distributions/Installers

Canonical and RDO have some additional responsibilities such as apart from installing some additional packages which take more than usual time, canonical also downloads a bootstrap image, which is used to commission nodes (Controllers, Compute Nodes, bootstrap node) [27]. RDO brings-up the configured number of controllers, compute nodes and shuts them off, which is required and returns their MAC addresses that are used later for introspection [33]. Manually, in regular lab exercises it takes more than 8 hours to reach until this point but the proposed work-flow has not only reduced the manual efforts but also significantly saved the required time.

6.5. Summary of task automation

The steps followed to deploy a cloud platform using different available distributions are somewhat similar; notably, there exists no standardized set of these steps. This comes with a learning curve every time; that is, understanding workflow of the distribution for its automation.

Table 3 illustrates the status of task automation when using the proposed toolkit. It shows that everything is automated by the toolkit as compared to Table 1, which helps to remove all of the manual steps and improves user experience by reducing tremendous amount of time consumed in manual efforts. Moreover, the proposed framework has significant impact on removing the human errors due to the automation process. We expect that the proposed work-flow gives significant boost to the OpenStack development community and its users to easily try it in-house for multiple distributions and adopt the suitable for production use.

The proposed framework is to facilitate OpenStack deployment over local bare-metal systems. Primarily, targeted at applications in a controlled cloud environment for testing and research purposes. Nevertheless, the framework can be adapted to work over Amazon EC2 and other cloud infrastructures. In the case of EC2, network-specific constraints need to be considered to allow packet flow between the deployed VMs such as their MAC and IP address bindings. For a stand-alone installation over EC2, no such modifications are required to support OpenStack deployment using the proposed framework. Therefore, making the framework architecture flexible and capable of replication across any cloud orchestration engine or edge/cloud fabric.

The currently implemented work-flow does not provide support for vagrant to create virtual machines on multiple hardware machines. Apart from that a specific network setting such as bonds (provides HA and fail-over) cannot be tested with the current release of the proposed toolkit.

7. AVAILABILITY

The proposed framework is an open source project and is anonymously available on a source repository hosted by GitHub at https://github.com/arslan-qadeer/droid. We expect other researchers to utilize the proposed framework for their in-house cloud deployment.

8. CONCLUSIONS

With the growing need of IT infrastructure in Cloud computing, multiple open source solutions like OpenStack to provide IaaS, are gaining significant pace in academia and industry. Developers have to face a number of issues while validating a small change in the code to verify a bug fix in the absence of any infrastructure bring-up tool. Without proper knowledge of OpenStack, it is also difficult for a researcher to establish cloud provisioning infrastructure for experimental purposes. The proposed toolkit based work-flow helps researchers as well as developers, to deploy OpenStack based cloud environments, in a quick and efficient manner. It also helps professionals for urgent and in-house proof of concept deployments without worrying about the underline details. It brings manual efforts to minimal, while sustaining the stability of complete cycle, ultimately providing a solution that is fully automated and human error free. Last but not

TABLE 3. Distribution-wise task automation support with the proposed toolkit.

Distribution	Infrastructure setup	Network configuration	OS installation	Packages installation
DevStack	✓	✓	✓	✓
Rackspace	✓	✓	✓	✓
Canonical	✓	✓	✓	✓
RDO	✓	✓	✓	✓

the least, it also helps to setup demos in academic sector for experimental purposes and to share the state-of-the-art work in cloud computing domain. The proposed framework is designed in a way that it can be enhanced for any type of cloud deployment to provide not only IaaS model but also PaaS and SaaS models without much effort. However, in PaaS or SaaS, the deployment lifecycle has to follow the same conventional way of manual operations. Though the issues are not particularly related to PaaS or Saas but still similar to IaaS orchestration. To further elaborate, consider an example application that is intended to be run in cloud environment. To validate such application one has to replicate the exact cloud environment. Even though the application could be deployed, configured and scaled in a fully automated way, but the preparation of the underlying infrastructure is similar for the entire cloud ecosystem including PaaS or SaaS.

9. FUTURE ENHANCEMENTS

The current design of toolkit allows to test OpenStack distributions on a single hardware node requiring slightly more resources to hold several virtual machines running cloud inside, which lacks scalability testing and can cause bottleneck for performance testing. This can be enhanced to multiple hardware nodes, which can be a scalable as well as an efficient solution for performance testing. As mentioned earlier, the proposed toolkit supports OpenStack deployments to provide IaaS service model but support for other service models e.g PaaS, SaaS and for latest agile solutions like “Docker”, “Mesos” and “Kubernetes” can also be added.

10. FUNDING

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

REFERENCES

[1] Sefraoui, O., Aissaoui, M., and Eleuldj, M. (2014) Cloud computing migration and it resources rationalization. *Multimedia Computing and Systems (ICMCS), 2014 International Conference on*, pp. 1164–1168. IEEE.

[2] Ismail, M. A., Ismail, M. F., and Ahmed, H. (2015) Openstack cloud performance optimization using linux services. *Cloud Computing (ICCC), 2015 International Conference on*, pp. 1–4. IEEE.

[3] Org., O. (2016) Openstack user survey. Technical report. OpenStack Org.

[4] Evans, N., Bozonnet, S., Wang, D., Fredouille, C., and Troncy, R. (2012) A comparative study of bottom-up and top-down approaches to speaker diarization. *IEEE Transactions on Audio, Speech, and Language Processing*, **20**, 382–392.

[5] Binz, T., Breitenbücher, U., Kopp, O., and Leymann, F. (2014) Tosca: portable automated deployment and management of cloud applications. *Advanced Web Services*, pp. 527–549. Springer.

[6] Papaioannou, A., Metallidis, D., and Magoutis, K. (2015) Cross-layer management of distributed applications on multi-clouds. *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pp. 552–558. IEEE.

[7] Milojičić, D., Llorente, I. M., and Montero, R. S. (2011) Opennebula: A cloud management tool. *IEEE Internet Computing*, **15**, 11–14.

[8] Muralidharan, K. and Gupta, P. (2006) Nimbus: A task aware/context aware mobile computing platform. *Software Technologies for Future Embedded and Ubiquitous Systems, 2006 and the 2006 Second International Workshop on Collaborative Computing, Integration, and Assurance. SEUS 2006/WCCIA 2006. The Fourth IEEE Workshop on*, pp. 6–pp. IEEE.

[9] Baun, C. and Kunze, M. (2009) Building a private cloud with eucalyptus. *E-Science Workshops, 2009 5th IEEE International Conference on*, pp. 33–38. IEEE.

[10] Mullerikkal, J. P. and Sastri, Y. (2015) A comparative study of openstack and cloudstack. *Advances in Computing and Communications (ICACC), 2015 Fifth International Conference on*, pp. 81–84. IEEE.

[11] Inzinger, C., Nastic, S., Sehic, S., Vogler, M., Li, F., and Dustdar, S. (2014) Madcat: A methodology for architecture and deployment of cloud application topologies. *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on*, pp. 13–22. IEEE.

[12] Achilleos, A. P., Kapitsaki, G. M., Constantinou, E., Horn, G., and Papadopoulos, G. A. (2015) Business-oriented evaluation of the paasage platform. *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on*, pp. 322–326. IEEE.

[13] Menychtas, A., Santzaridou, C., Kousiouris, G., Varvarigou, T., Orue-Echevarria, L., Alonso, J., Gorronogoitia, J., Bruneliere, H., Strauss, O., Senkova, T., et al. (2013) Artist methodology and framework: A novel approach for the migration of legacy software on the cloud. *2nd Workshop on Management of resources*

and services In Cloud And Sky computing (MICAS 2013).

[14] Giannakopoulos, I., Papailiou, N., Mantas, C., Konstantinou, I., Tsoumakos, D., and Koziris, N. (2014) Celar: automated application elasticity platform. *Big Data (Big Data), 2014 IEEE International Conference on*, pp. 23–25. Citeseer.

[15] García-Gómez, S., Escriche-Vicente, M., Arozarena-Llopis, P., Jimenez-Ganan, M., Lelli, F., Taher, Y., Biro, J., Momm, C., Spriestersbach, A., Vogel, J., et al. (2012) 4caast: Comprehensive management of cloud services through a paas. *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pp. 494–499. IEEE.

[16] Shen, Z., Subbiah, S., Gu, X., and Wilkes, J. (2011) Cloudscale: elastic resource scaling for multi-tenant cloud systems. *Proceedings of the 2nd ACM Symposium on Cloud Computing* 5. ACM.

[17] Sefraoui, O., Aissaoui, M., and Eleuldj, M. (2012) Comparison of multiple iaas cloud platform solutions. *Proceedings of the 7th WSEAS International Conference on Computer Engineering and Applications,(Milan-CEA 13). ISBN*, pp. 978–1.

[18] Voras, I., Mihaljević, B., Orlić, M., Pletikosa, M., Žagar, M., Pavić, T., Zimmer, K., Čavrak, I., Paunović, V., Bosnić, I., et al. (2011) Evaluating open-source cloud computing solutions. *MIPRO, 2011 Proceedings of the 34th International Convention*, pp. 209–214. IEEE.

[19] Sotomayor, B., Montero, R. S., Llorente, I. M., and Foster, I. (2009) Virtual infrastructure management in private and hybrid clouds. *IEEE Internet computing*, **13**.

[20] Malik, S. U., Khan, S. U., and Srinivasan, S. K. (2013) Modeling and analysis of state-of-the-art vm-based cloud management platforms. *IEEE Transactions on Cloud Computing*, **1**, 1–1.

[21] Mohammed, B. and Kiran, M. (2014) Experimental report on setting up a cloud computing environment at the university of bradford. Technical report.

[22] Niranjani, S., Jehadeesan, R., Vanishree, S., and Karthick, P. (2015) A prototype for private cloud implementation using open-source platform. *International Journal of Emerging Technology in Computer science and Electronics (IJETCSE)*, **13**.

[23] Kumar, R., Gupta, N., Charu, S., Jain, K., and Jangir, S. K. (2014) Open source solution for cloud computing platform using openstack. *International Journal of Computer Science and Mobile Computing*, **3**, 89–98.

[24] Li, Z., Zhang, Y., and Liu, Y. (2017) Towards a full-stack devops environment (platform-as-a-service) for cloud-hosted applications. *Tsinghua Science and Technology*, **22**, 1–9.

[25] Girish, L. and Guruprasad, H. (2014) Building private cloud using openstack. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, **3**, 142–145.

[26] Bonner, S., Pulley, C., Kureshi, I., Holmes, V., Brennan, J., and James, Y. (2013) Using openstack to improve student experience in an he environment. *Science and Information Conference (SAI), 2013*, pp. 888–893. IEEE.

[27] Pepple, K. (2011) *Deploying openstack.* " O'Reilly Media, Inc.".

[28] Hall, D. (2013) *Ansible configuration management.* Packt Publishing Ltd.

[29] Jeswani, J. T., Kurian, J. J., and Santiago, J. R. (2016) Installation of openstack (liberty release) using packstack. *International Journal of Computer Applications*, **138**.

[30] Kourtis, M., McGrath, M. J., Gardikis, G., Xilouris, G., Riccobene, V., Papadimitriou, P., Trouva, E., Liberati, F., Trubian, M., Batall, J., Koumaras, H., Dietrich, D., Ramos, A., Ferrer Riera, J., Bonnet, J., Pietrabissa, A., Ceselli, A., and Petrini, A. (2017) T-nova: An open-source mano stack for nfv infrastructures. *IEEE Transactions on Network and Service Management*, **14**, 586–602.

[31] Sheela, P. S. and Choudhary, M. (2017) Deploying an openstack cloud computing framework for university campus. *2017 International Conference on Computing, Communication and Automation (ICCCA)*, May, pp. 819–824.

[32] Suryn, W., Abran, A., and April, A. (2003). Iso/iec square. the second generation of standards for software product quality.

[33] Markelov, A. (2016) Getting to know openstack. *Certified OpenStack Administrator Study Guide*, pp. 1–6. Springer.