Received 25 April 2021
Revised 22 July 2021
Accepted 26 August 2021

Design of a small-scale and failure-resistant IaaS cloud using OpenStack

Samuel Heuchert
Dakota State University, Madison, South Dakota, USA
Bhaskar Prasad Rimal
The Beacom College of Computer and Cyber Sciences, Dakota State University, Madison, South Dakota, USA
Martin Reisslein
School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, Arizona, USA, and
Yong Wang
The Beacom College of Computer and Cyber Sciences, Dakota State University, Madison, South Dakota, USA

Abstract

Purpose – Major public cloud providers, such as AWS, Azure or Google, offer seamless experiences for infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS). With the emergence of the public cloud's vast usage, administrators must be able to have a reliable method to provide the seamless experience that a public cloud offers on a smaller scale, such as a private cloud. When a smaller deployment or a private cloud is needed, OpenStack can meet the goals without increasing cost or sacrificing data control.

Design/methodology/approach – To demonstrate these enablement goals of resiliency and elasticity in IaaS and PaaS, the authors design a private distributed system cloud platform using OpenStack and its core services of Nova, Swift, Cinder, Neutron, Keystone, Horizon and Glance on a five-node deployment.

Findings – Through the demonstration of dynamically adding an IaaS node, pushing the deployment to its physical and logical limits, and eventually crashing the deployment, this paper shows how the PackStack utility facilitates the provisioning of an elastic and resilient OpenStack-based IaaS platform that can be used in production if the deployment is kept within designated boundaries.

Originality/value – The authors adopt the multinode-capable PackStack utility in favor of an all-in-one OpenStack build for a true demonstration of resiliency, elasticity and scalability in a small-scale IaaS. An all-in-one deployment is generally used for proof-of-concept deployments and is not easily scaled in production across multiple nodes. The authors demonstrate that combining PackStack with the multi-node design is suitable for smaller-scale production IaaS and PaaS deployments.

Keywords Business enablement, Cloud computing, Distributed systems, High availability, OpenStack, Scalability

Paper type Technical paper

Applied Computing and Informatics
Vol. 21 No. 1/2, 2025
pp. 164-183
Emerald Publishing Limited
e-ISSN: 2210-8327
p-ISSN: 2634-1964
DOI 10.1108/ACI-04-2021-0094

The authors would like to thank the OpenStack Foundation.

Funding: This research was supported by the State of South Dakota Board of Regents' Competitive Research Grant.

1. Introduction

In the present day, information technology (IT) requirements change rapidly. This includes infrastructure and platforms that support development and operations in an organization. In order to fulfill the dynamic nature of these requirements, including, but not limited to, on-demand virtual machines, development platforms and production platforms; resilient and elastic infrastructure and platforms must be available on-demand for developers. Cloud computing is a solution to such problems because it allows for geographically-unique servers that can be configured redundantly on a large scale in order to provide low-cost, elastic platforms for development and operations tasks [1–4].

While there are public clouds available to provide such infrastructure and platform services [5], there are situations that require cloud services to be administered on-premise and on a smaller scale. Whether it is used for the purpose of compliance, data retention, ease of administration or cost reduction; having an on-premise cloud infrastructure can enable business processes in many ways. One such way that is pervasive in many modern practices is the need for seamless resilience and elasticity for end users, in most cases for in-house developers. The public clouds, such as Azure and AWS, do this very well with little to no knowledge from the end user due to their vast size and allow for a rich feature set to be produced and computing instances to be resilient and elastic [5]. However, public clouds are often black boxes and are proprietary to the provider. In order to provide the same seamless features in a private cloud deployment, OpenStack can be used [6, 7]. OpenStack is a platform that controls multiple types of resources, often called nodes, in order to provide different cloud services. For example, one of the services is the compute service of Nova that allows users to create virtual machine instances as needed [8]. These virtual machines can be started to provide IaaS or PaaS for developers and allow much of the administration workload to be removed from the development and operations department.

This paper focuses on building a resilient, elastic private cloud using OpenStack while concentrating on infrastructure as a service (IaaS) and platform as a service (PaaS). In this private cloud deployment, OpenStack's elasticity features will be used in order to demonstrate how to provide a seamless user experience when accessing IaaS and PaaS in OpenStack. To do this, an all-in-one OpenStack deployment will *not* be used so as to avoid the lack of scalability [9]. While private clouds usually offer a smaller feature set than public clouds [5], such a limitation is an advantage for IaaS and PaaS offerings to enable the business. We selected the PackStack utility as a deployment method to demonstrate how a proof-of-concept tool can be used to provide a small-scale deployment with high rates of resiliency and elasticity. The Packstack utility allows for IaaS and PaaS to be scaled by simply changing one configuration file. While not recommended in a large-scale deployment due to a lack of further customization, the easy-to-use nature of Packstack's IaaS and PaaS scalability features allow it to be utilized on a smaller-scale by not requiring in-depth knowledge of OpenStack deployment commands. This paper focuses on a small-scale deployment that can be imitated by small companies to provide a public cloud-like experience for end users who need the IaaS and PaaS realms of the public cloud.

In this paper, the small-scale aspect is directly applied by focusing on providing IaaS and PaaS options within an organization. The control, storage, networking, compute, identity and images services will be combined into one or more nodes that are spread across the deployment in order to provide a resilient and elastic private cloud deployment using OpenStack. The motivation of the paper is to demonstrate how OpenStack can be useful on a small scale while planning for future growth. The term "small-scale deployment" is used in this paper to describe a IaaS and PaaS cloud environment where advanced features, such as billing or analytics are not needed.

The contributions of this paper are summarized are follows:

(1) Providing a comprehensive tutorial outlining how to build an OpenStack-based private cloud that can be used on a small scale while leaving room for future growth.

(2) Demonstrating how a proof-of-concept tool, such as PackStack (a utility that uses Puppet modules to deploy various parts of OpenStack on multiple pre-installed servers), can be used to create a failure-resistant deployment in small-scale production scenarios to deploy OpenStack.

(3) Stress testing an OpenStack deployment to accurately measure the expected performance against the actual performance. Note that stress-testing is very unique in this study and has not been previously investigated in-depth in such a specific type of deployment.

2. Background

The public cloud provides a rich feature set when providing both IaaS and PaaS. However, it is not always feasible for a business to relinquish full control over the data and infrastructure due to compliance, retention or cost [10]. Numerous business cases and applications may require strict full local (private) control over the data and infrastructure [11–14], yet may pose substantial computing demands, e.g. for document management [15, 16], image processing [17, 18], surveillance applications [19], education system management [20, 21] and process simulation [22]. This is where the private cloud can come into play for both IaaS and PaaS. Hosting traditional infrastructure and platforms on-premises usually requires manual intervention for infrastructure resources, such as virtual machines to be scaled and for software development platforms, e.g. Eclipse, NetBeans, Visual Studio or web server shells, to be installed for software developers. In order to provide the enabling nature of the public cloud, IaaS and PaaS can be provided using OpenStack. When software developers are enabled with resilient and elastic development and production environments, they will have greater efficiency [23]. In addition to increasing efficiency, resilient and elastic development and production environments are also cost-effective in the aspects of power usage, central processing unit (CPU) usage and memory usage because instances of infrastructure and platforms are able to be powered on and off at appropriate times.

2.1 OpenStack

OpenStack is an open-source cloud platform. As the OpenStack community grows, the offered components grow in number as well. This paper focuses on seven core components in the OpenStack in order to provide Iaas and PaaS. Figure 1 shows a common deployment model referenced in the OpenStack community.

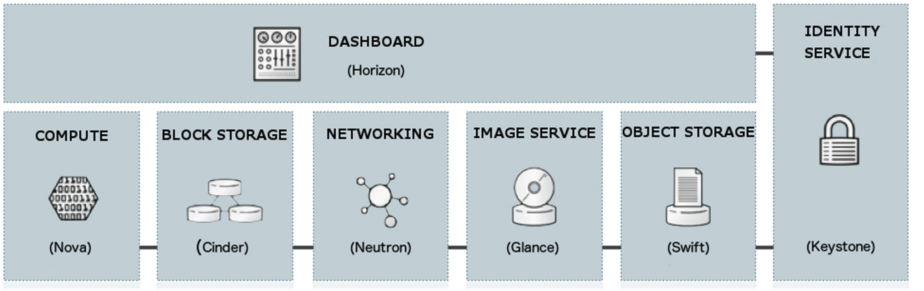


Figure 1.
Illustration of OpenStack deployment model for IaaS with seven core OpenStack components

(1) Compute (Nova): Provides a mechanism to provision instances for computing, also known as virtual servers.

(2) Block storage (Cinder): Provides a mechanism to present storage resources to be used by Nova. On the backend, Cinder virtualizes the management of devices that are used for block storage so that the end-users can consume the resource without knowledge of where the storage is physically located.

(3) Networking (Neutron): Provides network as a service between network interface devices and the OpenStack services, such as Nova.

(4) Image service (Glance): Provides the service of being able to upload and discover assets that can be used with other services. For example, Glance provides pre-configured images for compute instances.

(5) Object storage (Swift): Provides a highly available and distributed storage for objects or blobs.

(6) Identity service (Keystone): Provides client authentication, service discovery and multi-tenant authorization through the OpenStack-specific API. Keystone can use a wide array of protocols, such as the Lightweight Directory Access Protocol (LDAP), OAuth, Security Assertion Markup Language (SAML) and OpenID Connect (identity running on top of OAuth 2.0) [24].

(7) Dashboard (Horizon): Provides a web-based interface to manage services, such as Nova, Swift, Keystone and others.

2.2 Resiliency and elasticity

Cloud service is synonymous with redundancy and high availability, both characterizing resiliency. In the event of a hardware failure or a disaster in a data center, when a service is resilient, all services continue to run with little to no intervention from administrators and users have no insight into the situation. This provides the seamless user experience that public clouds provide. Public clouds provide this with their rich feature set and a private OpenStack deployment can provide a high level of resilience as shown by the enterprise usage in such platforms, e.g. Walmart's data center [25]. While Walmart most likely uses a more robust platform to orchestrate OpenStack than a utility, such as PackStack, the foundation of OpenStack remains the same. Benkhelifa *et al.* have discussed the dependence of society on cloud computing and how distributed and fault-tolerant systems are the backbone to this dependence [26]. They discuss the ResiliNet Group's [27] model of survivable and resilient networks. The work by the ResiliNet Group includes a resilience model known as the $D^2R^2 + DR$ model in which resiliency is defined as a life cycle. The model outlines the need to defend, detect, remediate, recover, diagnose and refine. This life cycle can be seen as the foundation on which the dependence of cloud computing is built.

Traditional resource scaling and elasticity usually require hardware procurement, operating system installation and platform configuration. However, with a cloud platform, scaling is accomplished seamlessly through the use of pre-configured images, flavors and hardware allocations that react in real-time to changing needs or notify administrators of needs [28]. In OpenStack, elasticity-management permissions can be granted to the end-user. These permissions are crucial in a situation where a developer locates a need to increase or decrease the size of a running platform instance. As discussed below in Section 4 regarding the proposed design, scaling and elasticity in OpenStack in this project are made possible by horizontal scaling. Horizontal scaling has been described by Millnert and Eker [29] who describe HoloScale and discuss how horizontal and vertical scaling differ. In brief, horizontal

scaling expands a whole resource unit; whereas, vertical scaling increases the capacity of already existing resources.

3. Related work

There are many solutions for resilient elastic private cloud OpenStack deployments. These solutions have been published in research papers and documentation, specifically on the OpenStack Foundation Documents site [30] and on many Linux Community sites. Many of these solutions incorporate a full-service stack for large-scale deployment or are all-in-one systems for proof-of-concept deployments.

Two examples of the existing multi-node enterprise solutions to deploy, manage and upgrade OpenStack in a data center or enterprise are TripleO [31] and Kolla–Ansible [32]. TripleO is a project in the OpenStack community which aims at deploying, managing and operating OpenStack-based clouds using OpenStack's own cloud facilities built on a foundation of Nova, Ironic, Neutron and Heat. Heat is the automation tool commonly used in OpenStack to inject logic to different deployment scenarios. A TripleO deployment is called a cloud-on-cloud deployment [33]. Kolla–Ansible focuses on using the scalable Ansible automation to deploy OpenStack.

Table 1 summarizes the main distinctions of our study with respect to the related studies on small-scale private clouds, which we review in the following. We begin by reviewing the study by Lebre *et al.* [34] on converting mega data centers to micro data centers by using OpenStack. The idea of using multi-site data centers to control cloud computing infrastructures is one that is worthwhile to explore and implement due to the need of distributed systems contributing to the aspects of latency and high-availability. However, many small businesses would not benefit from such a highly distributed design [34]. In contrast, we focus in this paper on the scalable nature of cloud computing which could benefit small businesses in using OpenStack. The design of a small scale, failure-resistant IaaS cloud in OpenStack that this paper introduces allows novice administrators to start small and grow. This approach allows room for substantial growth in respect to the size of the industry. The example of a small software development company starting with two to three employees and quickly growing to one hundred can be used to illustrate a good example of how OpenStack's scalability can be leveraged.

In [35], discussing OpenStack and Docker, Calinciuc *et al.* consider building a fully functional IaaS platform for social media applications. The discussion of Kernel-based Virtual Machine (KVM) virtualization vs Docker containerization specifically for social media applications in [35] is important because it highlights the fundamental idea of containerization being more efficient than traditional virtualization.

Table 1. Comparison of our study against the main related OpenStack-based private cloud studies

Study	Multinode	IaaS	Small scale	Dynamic Configurability	Generic virtualization	PackStack
Lebre *et al.* [34]	Yes	Yes	No	Yes	Yes	No
Calinciuc *et al.* [35]	Yes	Yes	No	Yes	No - Docker only	No
Awasthi *et al.* [36]	Yes	Yes	No	Yes	Yes	No
Sheela *et al.* [37]	Yes	Yes – Paas as well	Yes	No	Yes	No
Kengond *et al.* [38]	Yes	Yes – PaaS as well	No	Yes	Yes	No
Bathia *et al.* [39]	Yes	Yes – PaaS expandable	Yes	No	Yes	Yes
Bathia *et al.* [40]	No	Yes	Yes	No	Unknown	Yes
Suriansyah *et al.* [41]	Yes	Yes	Yes	No	Yes	Yes
Our study	Yes	Yes – PaaS expandable	Yes	Yes	Yes	Yes

In 2016, a couple years after OpenStack had started to gain traction in the industry, Awasthi *et al.* discussed OpenStack in detail and how its scalable nature was key to its success [36]. Awasthi *et al.* explain in detail how OpenStack operates and how this operation allows for OpenStack to simultaneously act as a scalable, hybrid cloud without the burdens of a public cloud. Having full control of the ecosystem in an open source environment is key [36].

There are many published papers and documents around the topic of OpenStack and the various methods to deploy it. In addition to the papers discussed above, a couple more examples include the work completed in [37] regarding deploying OpenStack on a university campus and orchestrating a distributed OpenStack in [42]. In [37], Sheela and Choudhary explore the idea of providing a test bed for students to deploy applications. This concept of using OpenStack for a specialized area is similar to the small-scale focus in our paper. However, the deployment strategy outlined in our paper allows for the ability to scale the IaaS workload to a much greater degree that can be deployed on a production level. Nevertheless, the common challenge among current OpenStack designs is the lack of documentation for the option of small-scale, dedicated node deployments that may grow in the future. During a growth period, more infrastructure is needed for development and production. In the current landscape of OpenStack, there is a gap in a simple deployment method to accomplish this need for more infrastructure. Many papers exist and discuss OpenStack and specialized uses, e.g. for social media applications [35] and software-defined network perimeters [43]. In contrast, this paper is unique in that we demonstrate how a resilient and elastic cloud service platform can initially be put in place and subsequently be scaled up. Kengond *et al.* [38] focus on Hadoop as a service; whereas we can allow arbitrary services since we provide a generic IaaS. While Hadoop could in principle be used to run a developed service on top of it, this would be much more complex than our direct approach of providing a small-scale private IaaS cloud. A related mechanism to automatically scale Hadoop clusters based on CPU utilization measurements has been presented in [44].

Relatively few studies have explored PackStack for configuring private OpenStack clouds. In the context of cloud computing for education [45], Bathia *et al.* [39, 40] have employed PackStack for the initial configuration of a private OpenStack cloud but have not considered dynamic reconfigurations. In contrast, in this study, we employ PackStack for both the initial configuration as well as the dynamic reconfigurations of a private OpenStack cloud. Moreover, Bhatia focus strictly on higher education implementations, whereas we provide a general-purpose private cloud configuration for arbitrary use cases. Similar to the Bhatia *et al.* approach, the recent Suriansyah *et al.* [41] approach is limited in terms of dynamic reconfigurability. The instance resizing approach in Suriansyah *et al.* imposes a downtime on the order of 10 s. In contrast, we provide a dynamic instance reconfiguration (resizing) approach that completely avoids downtime and only incurs a ping latency increase from sub-one millisecond to the order of four milliseconds during the reconfiguration.

For completeness, we note that ancillary aspects of operating OpenStack private clouds that are orthogonal to the design aspects covered in this article have been examined in a few recent studies. The energy-efficiency management of OpenStack private clouds has been examined in [46], while the studies [47, 48] have investigated the merging and consolidation of private clouds. Mechanisms to assure the integrity of the data in OpenStack private clouds with a blockchain have been explored in [49].

The target audience of this paper are small companies that have a relatively modest ceiling for growth, but do not want to be inhibited by a lack of infrastructure and platforms for their developers. A common resolution for the lack of infrastructure and platforms is usually the move to the public cloud. However, if a cloud offering is used from the beginning, companies do not have to factor in moving to a cloud platform in the future, which will allow

for maintaining control over the company data. Starting with a private cloud will allow for cost control in the future by utilizing the model that builds around resiliency and elasticity. In this paper, the focus is on starting at the simplest level in order to reduce the level of knowledge needed to enter the OpenStack arena and to use features that may be helpful to the aforementioned smaller companies. In this way, sub-par hardware can be used at first and can later be improved as needed. This allows for small companies with limited resources to get started with a plan for growth in the future while keeping cost initially low.

4. Proposed design

The proposed design in this paper is a small-scale deployment. This type of deployment focuses on a single feature of a public cloud model, such as IaaS or PaaS. Public cloud models also provide the features of a small-scale deployment but incur the complexity that a small-scale deployment avoids. Our design directly addresses the problem of starting an OpenStack deployment on a small scale while still allowing for substantial future growth in the IaaS and PaaS models. Our focus is to demonstrate how infrastructure and platforms can be provided as a service to developers while allowing developers to be enabled to start elastic instances that can scale at the click of a button. Our OpenStack design allows for the demonstration of physical compute node failure and virtual machine live migration between compute nodes while still providing a seamless end-user experience. In this case, the end-users are the developers who need transparent access to resources in the event of multiple failures.

Three compute nodes are included in the initial IaaS offering in this design. We then expand the infrastructure to a fourth compute node as part of an experiment to demonstrate the ability to dynamically add nodes without compromising the availability of the current nodes that provide IaaS services through OpenStack. The design allows for the use of the PackStack utility since it focuses on a small-scale deployment and a single feature while lacking the advanced features of public clouds. Since the focus of this paper is on resiliency and elasticity, our design includes redundancy wherever possible, such as hardware RAID for storage at the most basic level and multiple compute nodes on the logical level of OpenStack providing IaaS. While there is no documented best practice about the number of nodes that are deployed, we have created a five-node deployment to fully demonstrate how IaaS can be both resilient and elastic on a small scale. The deployment initially includes a controller, three compute nodes and one networking node. The controller and networking node are not logically redundant to simplify the deployment. The controller node hosts Glance, Swift, Horizon and Keystone while the compute and networking nodes host Nova and Neutron, respectively. As an experiment, a fourth compute node is added, bringing the total OpenStack node count to six which demonstrates the ability to scale the IaaS-focused OpenStack deployment without compromising availability. Availability is not compromised because the deployment does not need to be brought offline for a compute node to be added. During the scaling process, the deployment will gain a standby compute node that can eventually act as an active node after it is fully added. As discussed in further detail below, each of the nodes will be used to load balance the new virtual machines that are added. If one node fails, the other nodes are used to transfer the virtual machines off of the failed node to the active nodes.

The following assumptions have been made in the design of the deployment.

(1) OpenStack handles all software-defined networking concepts and the configurations of advanced features, such as Generic Routing Encapsulation (GRE) tunneling, virtual local area networks (VLANS) and Virtual Extensible LAN (VXLANS), are not complete. Only the basic level of networking configuration is complete to provide connectivity from the virtual machine LAN.

(2) The controller node acts as the command and control center for the deployment and all other nodes communicate directly with the controller node.

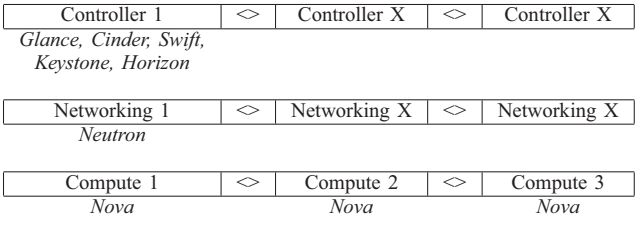
The design of the deployment is shown in Figure 2. Note that the basic design focuses on horizontal scaling to allow for resiliency, elasticity and scalability, as discussed in Section 2. In Figure 2, the "X" denotes the possibility for a redundant node to be added for scalability and redundancy. The numbered nodes are the nodes that are implemented in the demonstration.

The core methodology in this design focuses on a small-scale deployment at first while still planning for future growth. To do this, multiple considerations are made. First, the hardware is considered. In a small-scale deployment, the most robust hardware is not needed on all nodes. However, it must support the most basic functions of the particular service node. For example, the hardware used for the controller node is a modern HP server with 16 GB of RAM, dual quad-core processors and 256 GB of RAID1 SSD storage. Since the controller node will not have redundancy in this OpenStack deployment, reliable hardware must be used. The same is true for the networking node because there will be no redundancy on the hardware level outside of the mirrored storage and a backup network interface card. On the other hand, the hardware for the hypervisors are older SuperMicro servers with 8 GB of RAM, dual hex-core hyper-threaded Xeon processors and 128 GB of storage. This method allows for a low-cost, scalable hardware solution that can be added to in the future by adding compute nodes at a low cost. In total, four compute nodes, one controller node and one networking node are used. The first three compute nodes are the aforementioned SuperMicro servers with the final added node being an older HP server, as outlined in Section VII.E. The controller and networking nodes are modern HP servers as noted above.

Overall, the methodology focuses on building a solid foundation that can be added to as growth happens. This includes using lower-cost hypervisor hardware that can be purchased second-hand in many cases. If a hypervisor fails due to a hardware failure, the redundancy compensates for it and the deployment is not compromised. This paper demonstrates how to add a compute node to the node list after the initial configuration has been completed. This aspect is key for the feature of horizontal scalability. Note that OpenStack is an open source-based cloud operating system and this paper's focus is starting small while planning for future growth. Therefore, our initial deployment provides a solid foundation and has low cost. These two foci allow for a low-cost startup while maintaining the ability to grow the IaaS and PaaS models at scale. This methodology allows for a small company with limited funds to build a robust infrastructure backbone for the future at a very low startup cost. The methodology also allows for the demonstration of tolerating failure and proving elasticity for instances.

5. Implementation strategy

Implementing a small OpenStack deployment can be done in a multitude of ways [50], such as manually or through a utility, such as DevStack [51] or PackStack [52]. DevStack, which is an



Note(s): The 'X' nodes, which allow for future growth, are not used in this study

Figure 2.
The OpenStack private cloud deployment design focusing on horizontal growth, which allows for high-availability, high-scalability and elasticity

open-source shell script, is well-suited for large-scale deployments due to the modularity of the DevStack extensible scripts. As we target a small-scale deployment, we use PackStack in conjunction with CentOS 7. The simplicity that PackStack affords makes it ideal for dedicated, small-scale deployments that focus on IaaS and PaaS.

While PackStack is generally considered a deployment method for proof-of-concept cloud services [53], there are many considerations to take into account when deploying PackStack for small-scale production. These are shown by the process outlined in detail below for each respective node. At the end of the implementation, the OpenStack deployment is prepared to launch instances on the same private network subnet, migrate instances between compute nodes, scale instances, dynamically add Nova compute nodes and manage permissions for different users in different projects once the project preparations in Section 6 are complete. This paper focuses heavily on the IaaS part of OpenStack, which can be easily expanded to PaaS by creating relevant images to be launched in virtual machine instances. In addition, security and tracking are demonstrated by available permission management and usage metrics; although they will not be as robust as in a public cloud. The implementation of OpenStack through PackStack relies on the assumption that IaaS must come before PaaS. Therefore, it should not be expected to see robust PaaS, but rather resilient and elastic instances that can be scaled at the click of a button in the Horizon dashboard.

As a first step, each of the five nodes is prepared as detailed in the supplementary material to this article [54] to configure static Internet Protocol (IP) addresses and to prepare the controller node. The installation uses PackStack-specific Puppet modules and manifests to install all the required dependencies on each node. While it is possible to install each service manually on each node, that would be impractical given the amount of time it takes in comparison to using a tool, such as PackStack. The utility also reduces the room for error that a manual installation presents. In a larger-scale production deployment, customizing each node more fully or using a larger-scale tool would be preferred. This could be done with TripleO or Kolla–Ansible.

6. Project preparation

6.1 Project and project member creation

To help with security, isolation and best practice adherence; a separate project and dedicated project member user is created, as described in more detail in [54]. From the Identity Tab, a new project is created by clicking Create Project. Quotas are set here since resources are limited for the project. For this demonstration, a project called project1 is created with a single member user called project1. Quotas are also set to limit the vCPUs and RAM available to the project. When isolating projects and users in Horizon, Keystone is used on the backend to verify the identity of users and their different project permissions.

6.2 Image and flavor creations

Before virtual machine instances can be launched, flavors and images are defined on the administrator side of OpenStack. After the image is created, a flavor is defined and assigned to the new project. Although there are defined default flavors, a custom flavor that defines virtual central processing unit (vCPUs), RAM, Root Disk Size, Ephemeral Disk Size and permissions is created and assigned to the project as to adhere more closely to the previously assigned quotas. A flavor called a1.small with 1 vCPU, 512 MB of RAM, and a 1 GB Root Disk provides a lightweight instance option to demonstrate scalability. Another flavor called a2.medium with 2 vCPUs, 1024 MB of RAM, and a 1 GB Root Disk provides a higher performing option that can be used to scale up an instance.

7. Results and experiments

Following the completion of the setup and project preparation, virtual instances can now be created. OpenStack will automatically load balance new instances across compute nodes by default, but instances can be live migrated to different compute nodes with no downtime. In addition, instances can have flavors changed, allowing for elasticity. As shown in the last stress test experiment in Section 7.6, the auto scheduling by the "nova-scheduler" service for instances is corrupted, forcing all instances to be scheduled to the same node, which caused the deployment to crash, as elaborated in Section 7.6. All reported quantitative evaluation results are based on the averages of three independent measurement replications.

7.1 Instance creation

To create a virtual machine instance, the dedicated project1 user logs into the Horizon dashboard and expands the Compute tab under the Project section. The "Launch Instance" option is chosen. Figure 3 shows the newly created instances with the a1.small flavor. The prior created internal network with IP address range 172.25.1.0/24 has randomly assigned an IP address in the address range for each instance.

7.2 Instance elasticity

As an experiment of elasticity and latency comparisons, instance1 at 172.25.1.178 is changed from the a1.small flavor to the a2.medium flavor. Prior to the change, the blue line in Figure 4 shows samples of the latency between the two instances when both instances are at the same a1.small flavor as measured in milliseconds on ping commands. The blue line includes the pings with sequence numbers 0-9, whereby the pings are spaced one second apart, and is considered the control to which other tests will be compared. Note that all ping latency values are below 0.8 ms and the average is 0.665 ms. There are no outlying values outside of the vertical axis.

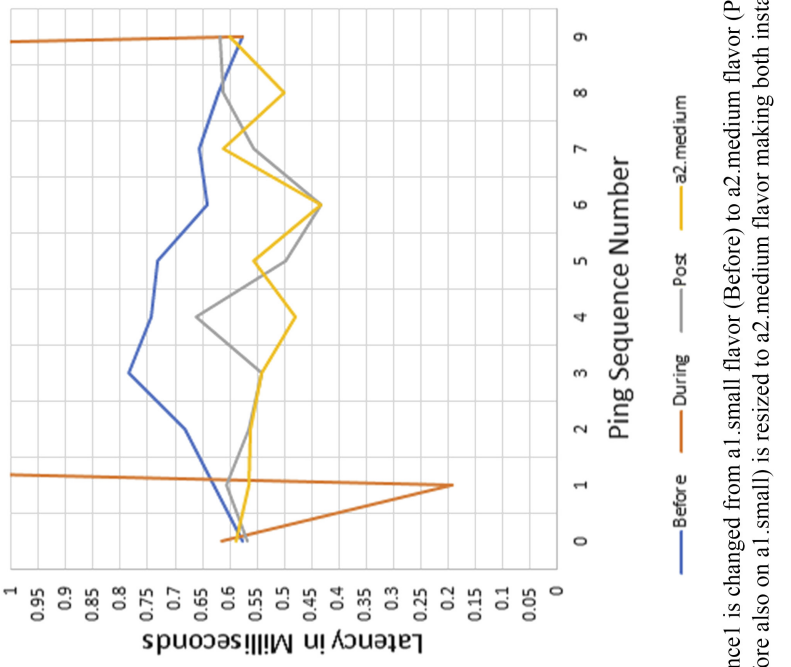
Following the control group being created, an experiment of changing instance1 at 172.25.1.178 to the a2.medium flavor is started. The orange line in Figure 4 shows the latencies of the pings during which the resize from a1.small to a2.medium was accomplished. Many of the data points run above the vertical axis of the diagram after the ping with sequence number 1 while the average latency comes to 4.723 ms. At the ping with sequence number 9, the value returns to the graph area. This indicates that the connectivity remains intact while the resize takes place and resumes to controlled latency levels at the ping with sequence number 9.

Next, the light grey line in Figure 4 shows the latency after the resize has taken place and the instances a have been stable for 30 s. As shown, the latency resumes to a stable, sub-1 ms delay and the data points remain on the graph. Due to the increase of resources, specifically vCPUs, the latency improved slightly to an average of 0.566 ms in comparison to the 0.665 ms when both instances were at the a1.small flavor. This demonstrates that the assigned resources have a direct relationship to the latency in regard to the OpenStack internal networking technologies.

As a final test to further examine the relationship between assigned resources and latency, instance2 at 172.25.1.101 is resized to the a2.medium flavor which increases the RAM and

Instance Name	Image Name	IP Address	Flavor
instance1	cirros	172.25.1.178	a1.small
instance2	cirros	172.25.1.101	a1.small

Figure 3.
The created instances with their IP addresses in the range of the internal network



Note(s): Instance1 is changed from a1.small flavor (Before) to a2.medium flavor (Post); moreover, instance2 (before also on a1.small) is resized to a2.medium flavor making both instances a2.medium

Figure 4.
Instance elasticity and latency experiment

vCPU count by two-fold. The gold line in Figure 4 shows that the average latency very slightly improved over a sequence of ten pings further from the previous configuration of instance1 being a2.medium and instance2 being a1.small.

Overall, elasticity in OpenStack is seamless as shown by the experiment. Even with increased latency during a resize for a ten-ping sequence, the difference is unnoticeable in most situations since there are no ping drops. A common example of this would be resizing an instance with a web server platform. If the web server has some additional latency on the order of a few seconds or less for a duration on the order of 10 s, the clients it is serving will barely notice. In addition to demonstrating the ability to have elastic instances, we validated that the amount of hardware allocated to an instance directly affects its performance.

7.3 Instance migration

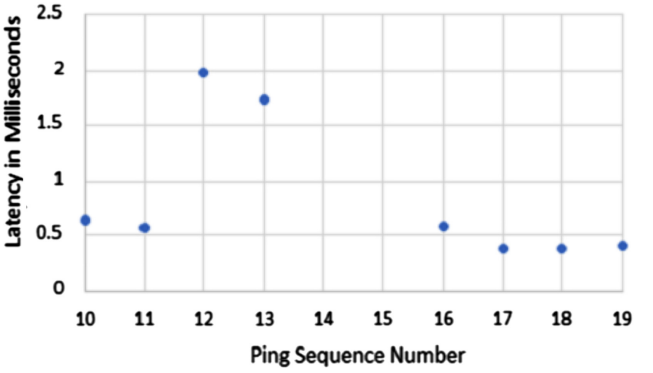
Instance are migrated in OpenStack for a variety of reasons. Two reasons are hardware maintenance or hardware failure. The hardware failure case is further explored in Section 7.4 as a demonstration of OpenStack providing a platform for instances to be failure-resistant, also known as the feature of instance resilience.

When hardware maintenance is needed in an OpenStack deployment, it can generally be assumed that it is planned. Unplanned migrations leading to instance resilience are discussed in the next section. For planned migrations, two different scenarios are tested. The first scenario is migrating an instance with no downtime. In OpenStack, this is known as a live migration. The second scenario is known as a cold migration, where the instance is shut down for a small amount of time. Cold migration can be used if multiple instances providing the same service are behind a load balancer that can determine when an instance is down and direct requests to an instance that is functioning.

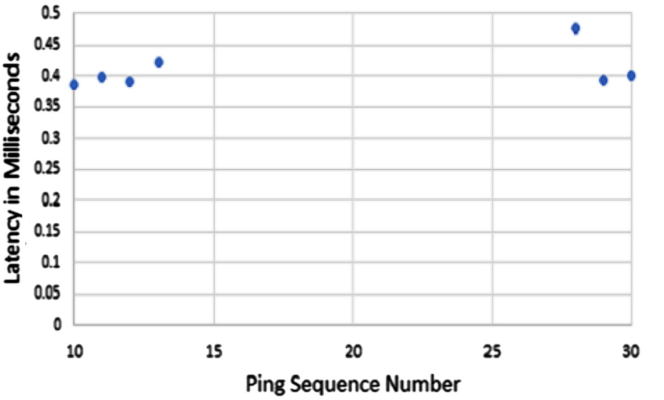
7.3.1 Live migration. Live migration is completed through the Admin tab of the Project under the Compute Instances subsection. In this live migration, the instance is not shut down and the migration is seamless for the end user. To demonstrate this, instance2 on compute1 is live migrated to compute2 which also hosts instance1. Figure 5a shows the latency during the live migration. The live migration takes place starting at the ping with sequence number 12 and ends at the ping with sequence number 18. Note that there is the loss of the pings with sequence numbers 14 and 15 along with the increase in latency of the pings with sequence numbers 12 and 13. Following the live migration, the latency returns to similar levels as prior to the migration. The ping losses for sequence numbers 14 and 15 could be due to many variables. Further study is required to investigate this behavior. However, two packet drops is a relatively small loss.

7.3.2 Cold migration. In the cold migration scenario, the instance is shut down, causing more downtime. However, the persistent storage volume and the networking settings are kept so that no settings are changed. To fully demonstrate this cold migration scenario, a larger ping sequence sample consisting of 30 pings (each corresponding to 1 s) is taken and displayed in Figure 5b. Following the cold migration of instance2 from compute2 to compute1, after one outlier at 0.46 ms, the latency returns to the expected value of around 0.4 ms that was present before (when the two instances were hosted on two separate compute host nodes).

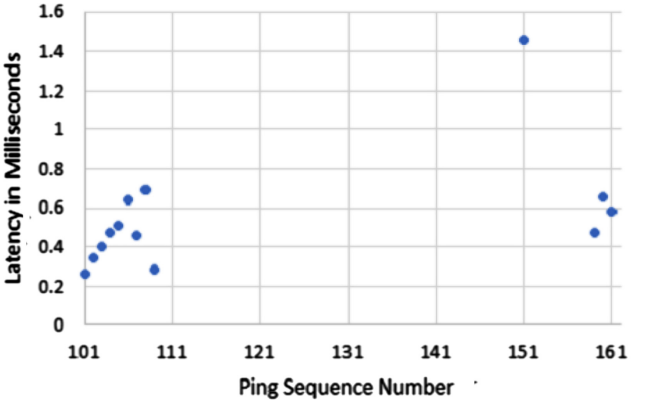
Overall, in both the live and cold migration scenarios, the instance migration—when needed in the context of hardware maintenance—is seamless with close to zero downtime. The cold migration is the safer option due to OpenStack completely shutting down the instance and starting it back up on the new compute node. This cold migration will reduce the risk of data loss; although a live migration is fairly low risk. As shown by the extensive documentation on OpenStack [55], live migrations have more room for error but are highly desirable due to next to zero downtime.



(a) Live migration; pings 14 and 15 lost



(b) Cold migration; pings 14–27 lost



(c) Evacuation migration; pings 110–158 lost

Figure 5.
Migration scenarios of instance2 from compute1 to compute2; the pings with the indicated sequence numbers are lost

7.4 Instance resilience: evacuation migration

Hardware outages can be both planned or abrupt (unplanned). For abrupt hardware outages, OpenStack offers a feature known as evacuation. Similar to all other demonstrated OpenStack features, evacuation can be done via the command line or via the graphical user interface (GUI). Having a physical compute node fail but the instance surviving is known as resilience. It is expected that an instance has resilience to survive failure and continues to function after a reasonably short period of time.

To demonstrate instance evacuation, instance2, hosted on compute1, is evacuated after compute1 is purposefully forced to fail by pulling the physical power cord to the SuperMicro server. This demonstrates a real-life disaster scenario of an abrupt node failure.

Prior to the disaster affecting compute1, the latency between pings is close to the average latency as previously shown, around 0.5 ms. Starting at ping sequence 110, compute1 is lost after the power cord is pulled, see Figure 5c. Following this loss, the admin user from the GUI Admin Compute Hypervisor section is leveraged to evacuate the failed compute1 host, which is reported as being failed after 30 s in Horizon. Note that the 30 s is the delay for OpenStack to detect the outage. After Evacuate Host is chosen, the target host compute2 is selected. Then, with about 19 s of additional delay [pings 110-158 lost in Figure 5c corresponding to 49 s of lost pings = 30 s (for OpenStack) + 19 s (additional delay)], instance2 is back up and instance1 returned to being able to ping instance2 with delays in the 0.5 ms range. Figure 5c shows a summary of the downtime that the compute1 disaster caused. The dropped packets result in a generally acceptable amount of downtime given the unique circumstances.

7.5 Compute node addition

One of the primary OpenStack features is scalability. Scalability is commonly known as the ability to add resources horizontally [29]. In this paper, we explore why PackStack (which is generally reserved for proof-of-concept clouds) is desirable for small-scale deployments due to the simplicity of the initial deployment and the addition of compute nodes.

In this experiment, PackStack facilitates a compute node addition named compute4. This expands the entire compute node farm to 4 nodes instead of 3 and distinctly demonstrates the scalability that OpenStack and PackStack offer. By adding an older HP server with 24 vCPUs and 16 GB of RAM as compute4 node, the compute node farm is increased. This demonstrates how PackStack facilitates business by allowing for uninterrupted scalability to increase capacity while also showing that heterogenous hardware can easily be used. Note that this node addition is conducted live and no downtime is needed.

In order to add the additional compute node, the original PackStack answer file is duplicated. This duplicated file is then changed in two ways. The fourth compute node is prepared with the same steps as in Section 5. The "EXCLUDE_SERVERS" parameter is changed to include all the current working nodes. This includes the controller, networking, compute1, compute2 and compute3. Finally, the "CONFIG_COMPUTE_HOSTS" is appended with the static IP address set on the new compute-node-to-be. Once these parameters are changed, the answer file is run again with the "packstack" command. This time, the configuration of the current nodes will be skipped and the new node will be added without any downtime to the cloud deployment. After the fourth compute node has been added, instances can be migrated or built on top of it, just as if it was part of the original deployment. Two use cases for this scenario may be a planned decommissioning of a compute node or the need to further scale up the compute node farm.

Overall, the addition of compute4 to the OpenStack deployment demonstrates the immense flexibility for compute node scalability offered by PackStack.

7.6 Stress test experiment

To further test the resiliency, one final experiment is conducted to test the robustness of the deployment. A new project is created that does not include any quotas, allowing for an unlimited number of instances to be created across the four nodes. The motivation for this experiment is to observe how long the compute nodes can function while instances are over-allocated in regard to the physically available hardware. This experiment can be viewed as a stress test.

The first step of this experiment is creating a project and dedicated user as done prior. The project is then verified that quotas (set to unlimited) are set above the actual resources available (80 vCPUs and 46 GB RAM) in the OpenStack deployment. To further simulate this, a flavor is created called b1.small. The b1.small flavor has an allotment of 2 vCPUs and 1 GB of RAM. This should allow for 40 simultaneous instances to be run at one time fairly easily while consuming all of the vCPUs in the deployment and 40/46 = 87% of the RAM.

The stress test is conducted as follows. In total, 40 instances are created in groups of 10. At each group of 10, the latency average of 10 pings is recorded. The latency average is recorded on each addition of 10 instances after a 30 stability period is seen. Finally, groups of 10 instances are added until the OpenStack deployment crashes. The experiments demonstrated that the deployment crashes between 50 and 60 instances, specifically allowing 58 instances consuming more resources than the Horizon dashboard reported as available. However, the experiment also demonstrated that OpenStack's logical allocation of RAM is not the limiting factor since 46 instances would consume 100% of the available RAM, even though the hardware's physical usage was not 100% of the available RAM at the 46 instance mark based on the "top" command in CentOS 7. However, we were able to go past that mark, up to 58 instances, even though we should not have been able to according to the amount of logically available RAM that OpenStack was reporting as available in Horizon. This is most likely due to instances not always consuming the entire allocation of RAM on the physical hardware. However, it is best practice to adhere to the logical amount of RAM that OpenStack presents to avoid cases where increased load on an instance pushes physical RAM usage. Figure 6 shows the average 10 ping latency between 2 instances at each node count mark starting at 10. As shown in Figure 6, as more nodes are added, the latency increases.

During the experiment, the resource usage on the compute nodes is observed, and it is noted that the logical static usage statistics reported from the controller node are not accurate since the controller reports on a block static level based on the number of instances. For example, when 50 instances are running, the controller node reports that the compute1 and compute2 nodes each were using 16 GB of RAM, even though each compute node had only 8 GB of RAM available. However, the accurate statistic pulled from the "top" command on an

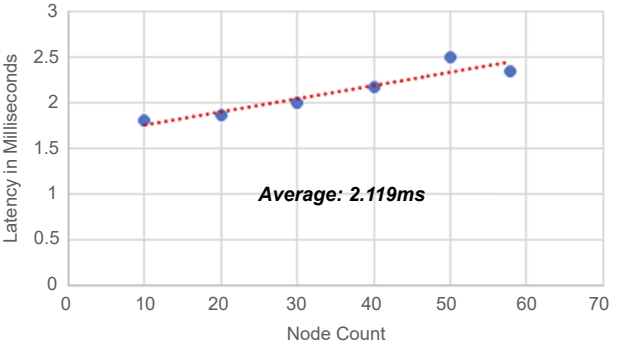


Figure 6.
Stress test latency as a function of node count

SSH session indicated that only 4 GB of RAM were used on each compute node (of the 8 GB or RAM available on each compute node).

Figure 7 shows the evidence that as the controller becomes more resource-starved, instances begin to fail to spawn before finally crashing at the 59 instance count. With each failure we manually spawned an additional group of the number of failed instances. This ensures that the node count is accurate in each block of 10 spawned instances. OpenStack was able to continue well past the 80 vCPU allocation that 40 instances used. We observe from Figure 7 that the spawning issues started above the 20 instance count; and then increased exponentially.

Overall, this final experiment shows important attributes in regard to the controller and compute nodes. The controller's CPU is the main bottleneck. Once the instance count hits 50, the controller routinely uses anywhere from 80%–90% of the 8 core CPU. At the 50 instance count, an abnormality of the controller scheduling all instances to one compute node is observed and this is the first sign of failure even though all 10 instances from the 40 to 50 instance mark count did spawn successfully after having to manually re-spawn 3 instances. Normally, the controller would be able to equally distribute instances across the 4 compute nodes, similar to the distribution of instances 1 through 40.

Building on the insights from the present stress text experiment, a future experiment could monitor the controller's load closely as instances spawn and provide a ratio for the specifications needed on the controller to how many instances it can support. However, the controller being fully loaded in the conducted stress test experiment was expected since the controller had Glance, Cinder, Swift, Keystone and Horizon all running on it. If these services are split up among different controller or storage nodes, it is highly likely that the deployment could support more running instances.

8. Conclusions and future direction

We have addressed a research gap for OpenStack in the applied computing area, namely the use of an OpenStack deployment utility, such as PackStack (which is generally considered a proof-of-concept tool) to deploy a small-scale private IaaS OpenStack cloud. We demonstrated that the PackStack utility provides an effective way for small companies to take advantage of the convoluted OpenStack software platform to orchestrate distributed computing. The paper has shed light on the needed components to deploy a fully functioning IaaS cloud so as to ease the development and operations burdens compared to the full suite of OpenStack features. In addition, this paper demonstrated how using PackStack as a deployment tool is a viable option for a private production OpenStack IaaS cloud due to the ease of implementing elasticity and resiliency through node addition and migrations.

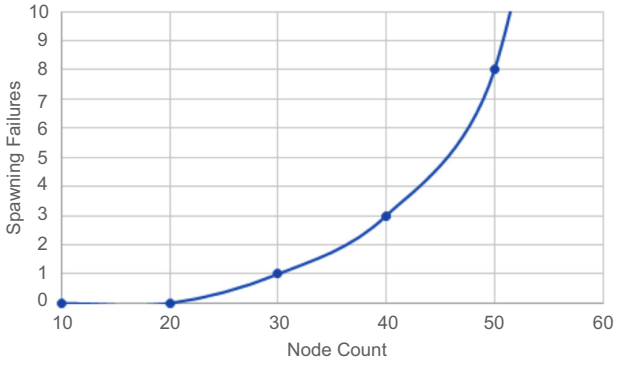


Figure 7.
Spawning failures per 10 instance spawns

Although OpenStack in the capacity demonstrated in this paper may not be ideal for all deployments, the concept of using a highly-supported and developed tool on a small scale presents many advantages. While the design in this paper is able to be horizontally scaled rather easily, the efficiency of the hardware and the administration time were not taken into account. An interesting future research direction is to examine the latency impact of hardware acceleration modules [56, 57] as well as fast packet processing paradigms, e.g. [58–61], for low-latency network packet processing. Moreover, future research should explore high-availability strategies for private clouds, e.g. utilizing OpenStack with clustering.

References

1. Hwang K, Dongarra J, Fox GC. Distributed and cloud computing: from parallel processing to the internet of things. Burlington, MA: Morgan Kaufmann; 2012.
2. Potdar AM, Narayan D, Kengond S, Mulla MM. Performance evaluation of Docker container and virtual machine. Proc Comp Sci. 2020; 171: 1419-28.
3. Rimal BP, Choi E, Lumb I. A taxonomy and survey of cloud computing systems. In: Proc. IEEE Fifth International Joint Conference on INC, IMS and IDC; 2009: 44-51.
4. Rimal BP, Lumb I. The rise of cloud computing in the era of emerging networked society. In: Cloud computing: principles, systems and applications. Springer; 2017: 3-25.
5. Foster I, Gannon DB. Cloud computing for science and engineering. Cambridge, MA: MIT Press; 2017.
6. Kai Z, Youyu L, Qi L, Hao SC, Liping Z. Building a private cloud platform based on open source software OpenStack. In: Proc. IEEE International Conference on Big Data and Social Sciences (ICBDSS); 2020: 84-87.
7. Silverman B, Solberg M. OpenStack for architects: design production-ready private cloud infrastructure. Birmingham: Packt Publishing; 2018.
8. OpenStack. OpenStack docs: OpenStack compute (nova); 2020. Available from: https://docs.openstack.org/nova/latest/ [accessed 27 September 2020].
9. OpenStack. OpenStack docs: all-in-one single machine (nova); 2020. https://docs.openstack.org/devstack/ [accessed 27 September 2020].
10. Microsoft. What are public, private, and hybrid clouds?; 2020. Available from: https://azure.microsoft.com/en-us/overview/what-are-private-public-hybrid-clouds/#public-cloud/ [accessed 27 September 2020].
11. Hoeschele T, Dietzel C, Kopp D, Fitzek FH, Reisslein M. Importance of internet exchange point (IXP) infrastructure for 5G: estimating the impact of 5G use cases. Telecommun. Pol.. 2021; 45(3): 102 091.1-18.
12. Navarro-Ortiz J, Romero-Diaz P, Sendra S, Ameigeiras P, Ramos-Munoz JJ, Lopez-Soler JM. A survey on 5G usage scenarios and traffic models. IEEE Commun Surv Tutorials. 2020; 22(2): 905-29.
13. Rostami A. Private 5G networks for vertical industries: deployment and operation models. In: Proc. IEEE 2nd 5G World Forum (5GWF); 2019: 433-9.
14. Varga P, Peto J, Franko A, Balla D, Haja D, Janky F, Soos G, Ficzere D, Maliosz M, Toka L. 5G support for industrial IoT applications–Challenges, solutions, and research gaps. Sensors. 2020; 20(3): 828.1-43.
15. Abidin SSZ, Husin MH. Improving accessibility and security on document management system: a Malaysian case study. App Comp Info. 2020; 16(1/2): 137-54.
16. Thorat C, Inamdar V. Implementation of new hybrid lightweight cryptosystem. App Com Info. 2020; 16(1/2): 195-206.

17. Akbar JM, *et al.* Joint method using Akamatsu and discrete wavelet transform for image restoration. App Com Info. 2021. doi: 10.1016/j.aci.2019.10.002.
18. Nannia L, Ghidoni S, Brahnam S. Ensemble of convolutional neural networks for bioimage classification. App Com Info. 2020; 17(1): 19-35.
19. Mahalingam T, Subramoniam M. A robust single and multiple moving object detection, tracking and classification. App Com Info. 2020; 17(1): 2-18.
20. Monsalve-Pulido J, Aguilar J, Montoya E, Salazar C. Autonomous recommender system architecture for virtual learning environments. App Com Info. 2021. doi: 10.1016/j.aci.2020.03.001.
21. Mustafa A. The personalization of e-learning systems with the contrast of strategic knowledge and learner's learning preferences: an investigatory analysis. App Com Info. 2020; 17(1): 153-67.
22. Pinho T, Coelho J, Oliveira P, Oliveira B, Marques A, Rasinmäki J, Moreira A, Veiga G, Boaventura-Cunha J. Routing and schedule simulation of a biomass energy supply chain through SimPy simulation package. App Com Info. 2020; 17(1): 36-52.
23. TechGenix. Software development in the cloud: benefits and challenges; 2020. http://techgenix.com/software-development-in-the-cloud/ [accessed 18 November 2020].
24. OpenStack. OpenStack documentation; 2020. https://docs.openstack.org/keystone/latest/getting-started/architecture.html [accessed 21 February 2021].
25. SuperUser OpenStack. Software development in the cloud: benefits and challenges; 2020. https://superuser.openstack.org/articles/inside-walmartlabs-and-its-openstack-core/ [accessed 27 September 2020].
26. Welsh T, Benkhelifa E. On resilience in cloud computing: a survey of techniques across the cloud domain. ACM Comput Surv. 2020; 53(3). [Online]. doi: 10.1145/3388922.
27. Resilinets Group. ResiliNetsWiki; 2016. Available from: https://resilinets.org/ [accessed 21 February 2021].
28. OpenStack. Capacity planning and scaling; 2020. https://docs.openstack.org/operations-guide/ops-capacity-planning-scaling.html [accessed 18 November 2020].
29. Millnert V, Eker J. Holoscale: horizontal and vertical scaling of cloud resources. In: Proc. IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC); 2020: 196-205.
30. OpenStack. Documentation for Ussuri; 2020. https://docs.openstack.org/ussuri/ [accessed 27 September 2020].
31. OpenStack. TripleO documentation; 2020. https://docs.openstack.org/tripleo-docs/latest [accessed 12 November 2020].
32. OpenStack. TripleO documentation; 2020. https://docs.openstack.org/kolla-ansible/latest/ [accessed 14 November 2020].
33. RedHat. TripleO documentation; 2020. https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/16.1/ [accessed 18 Nov 2020].
34. Lebre A, Pastor J, Simonet A, Desprez F. Revising OpenStack to operate fog/edge computing infrastructures. In Proc. IEEE International Conference on Cloud Engineering (IC2E), 2017: 138-48.
35. Calinciuc A, Spoiala C, Turcu C, Filote C. OpenStack and Docker: building a high-performance IaaS platform for interactive social media applications. In: Proc. 13th International Conference on Development and Application Systems, 05 2016: 287-90.
36. Awasthi S, Pathak A, Kapoor L. OpenStack-Paradigm shift to open source cloud computing & its integration. In: Proc. 2nd International Conference on Contemporary Computing and Informatics (IC3I); 2016: 112-19.
37. Sheela PS, Choudhary M. Deploying an OpenStack cloud computing framework for university campus. In: Proc. International Conference on Computing, Communication and Automation (ICCCA); 2017: 819-24.

38. Kengond S, Narayan D, Mulla M. Hadoop as a service in OpenStack. Emerg Res Elec. Computer Science and Technology, Lecture Notes in Electrical Engineering. 545. Singapore: Springer. 2019; 223-33. doi: 10.1007/978-981-13-5802-9_21.

39. Bhatia G, Al Noutaki I, Al Ruzeiqi S, Al Maskari J. Design and implementation of private cloud for higher education using OpenStack. In Proc. IEEE Majan International Conference (MIC), 2018, 1–6.

40. Bhatia G and Al Sulti IH, CASCloud. An open source private cloud for higher education," In Proc. IEEE International Arab Conference on Information Technology (ACIT), 2019: 14–20.

41. Suriansyah MI, Mulyana I, Sanger JB, Winata S. Compute function analysis utilizing IAAS private cloud computing service model in Packstack development. ILKOM J Ilmiah. 2021; 13(1): 10-16.

42. Haja D, Szabo M, Szalay M, Nagy A, Kern A, Toka L, Sonkoly B. How to orchestrate a distributed OpenStack. In: Proc. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS); 2018: 293-98.

43. Tkachova O, Salim MJ, Yahya AR. An analysis of SDN-OpenStack integration. In: Proc. Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S T); 2015: 60-2.

44. Hosamani N, Albur N, Yaji P, Mulla MM, Narayan D. Elastic provisioning of Hadoop clusters on OpenStack private cloud. In: Proc. IEEE 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT); 2020: 1-7.

45. Borse Y, Gokhale S. Cloud computing platform for education system: a review. Inter J Comp Appl. 2019; 177(9): 41-5.

46. Prameela P, Gadagi P, Gudi R, Patil S, Narayan D. Energy-efficient VM management in OpenStack-based private cloud. Adv Comp Net Comm. 2020; 1: 541.

47. Andreetto P, Chiarello F, Costa F, Crescente A, Fantinel S, Fanzago F, Konomi E, Mazzon PE, Menguzzato M, Segatta M, *et al.* Merging OpenStack-based private clouds: the case of cloudveneto.it. in EPJ Web of Conferences. 2019; 214: 07010.

48. Pyati M, Narayan D, Kengond S. Energy-efficient and dynamic consolidation of virtual machines in OpenStack-based private cloud. Procedia Computer Science. 2020; 171: 2343-52.

49. Patil A, Jha A, Mulla MM, Narayan D, Kengond S. Data provenance assurance for cloud storage using blockchain. In: Proc. IEEE International Conference on Advances in Computing, Communication and Materials (ICACCM); 2020: 443-8.

50. Awasthi A, Gupta PR. Comparison of OpenStack installers. Inter J Innov Sci, Eng Techno. 2015; 2(9): 744-8.

51. OpenStack. OpenStack documentation; 2020. Available from: https://docs.openstack.org/devstack/latest/ [accessed 25 October 2020].

52. Various Contributors. redhat-openstack/ packstack; 2020. Available from: https://github.com/redhat-openstack/packstack [accessed 25 September 2020].

53. RDO Project. PackStack: create a proof of concept cloud; 2020. Available from: https://www.rdoproject.org/install/packstack/ [accessed 25 September 2020].

54. Heuchert SA., Rimal BP, Reisslein M Wang Y. Design of a small-scale and failure-resistant IaaS cloud using OpenStack (supplementary material); 2021. Available from: https://github.com/socketsetter/openstack/blob/main/SmallScaleOSCloud_Suppl.pdf.

55. OpenStack. Live-migration; 2020. Available from: https://docs.openstack.org/neutron/pike/contributor/internals/live.html [accessed 8 November 2020].

56. Linguaglossa L, Lange S, Pontarelli S, Rétvári G, Rossi D, Zinner T, Bifulco R, Jarschel M, Bianchi G. Survey of performance acceleration techniques for network function virtualization. Proc IEEE. 2019; 107(4): 746-64.

57. Shantharama P, Thyagaturu AS, Reisslein M. Hardware-accelerated platforms and infrastructures for network functions: a survey of enabling technologies and research studies. IEEE Access. 2020; 8: 132 021-085.

58. Cerović D, Del Piccolo V, Amamou A, Haddadou K, Pujolle G. Fast packet processing: a survey. IEEE Commun Surv Tutorials. 2018; 20(4): 3645-76.

59. Fei X, Liu F, Zhang Q, Jin H, Hu H. Paving the way for NFV acceleration: a taxonomy, survey and future directions. ACM Comput Surv (Csur). 2020; 53(4): 1-42.

60. Fujimoto K, Matsui K, Akutsu M. KBP: Kernel enhancements for low-latency networking without application customization in virtual server. In: Proc. IEEE 18th Annual Consumer Communications Networking Conference (CCNC); 2021: 1-7.

61. Xiang Z, Gabriel F, Urbano E, Nguyen GT, Reisslein M, Fitzek FH. Reducing latency in virtual machines: enabling tactile Internet for human-machine co-working. IEEE J Selected Areas Commun. 2019; 37(5): 1098-116.

Corresponding author
Martin Reisslein can be contacted at: reisslein@asu.edu