Arabic Republic of Egypt
Ministry of Higher Education and Scientific Research

Egypt-Japan University of Science and Technology
Faculty of Engineering

Computer Science and Engineering Department

Graduation Project
Bachelor in Computer Science Engineering

GRADUATION PROJECT

Submitted to

Faculty of Engineering
School of Electronics, Communications and Computer Engineering
Computer Science and Engineering Department

In partial fulfillment of the requirements for the degree of

BACHELOR IN COMPUTER SCIENCE ENGINEERING

By

Abd Elrahman Khaled Abo Elsherbini

Lamyaa Samir Mohamed Ahmed Elmahi

Menna Tullah Abdelrahman Abdelbadei

Omar Bahaa Abdelmonein Sayed El Ahl

AUTO-GENERATION OF CLOUD COMPUTING SYSTEMS

Prof. Ahmed Elmahdy	Supervisor
Prof. Samir Elsagheer	Co-supervisor

February 2023

Acknowledgments

First and foremost, we would like to express our gratitude to Allah for his countless blessings, including the knowledge and ability to complete this thesis. We also extend our thanks to Professor Ahmed Elmahdy and Professor Samir Elsagheer for their valuable guidance and support throughout our thesis. Their expertise and feedback greatly contributed to the quality of our work, and we are grateful for the opportunity to learn from them. Their mentorship has been a source of inspiration, and we appreciate their dedication to our academic growth.

Abstract

The ever-increasing demand for cloud computing has led to a rise in the number of cloud deployments. However, the manual process of configuring these deployments has become a bottleneck, leading to numerous configuration errors. These errors not only waste valuable time and resources but can also impact the performance and security of the deployed systems. This thesis presents a cloud auto-generation system that overcomes these problems by automating the deployment process through the use of ready-to-use templates that are free of configuration and human errors. The cloud auto-generation system connects to the target bare-metal servers via a connection module, configures the SAS switches, automates the operating system installation process including partitioning and RAID creation, and configures the machines to be nodes by installing OpenNebula. The system developed in this study allows users to interact with the system through a user-friendly GUI to choose and apply entire system templates or customize their own templates, including fine-grained templates for the operating system installation or the SAS configuration steps. The customized templates are captured and can be used over and over. The use of automatically generated configuration templates makes the deployment process faster, more efficient, and more scalable, in addition to maintaining consistent configurations and reducing the time spent on manual configuration tasks, freeing up valuable resources. It is designed to help organizations of all sizes and industries streamline their cloud deployment process and ensure the reliability and security of their deployed systems. This thesis also presents an evaluation of the system, demonstrating its ability to significantly reduce the number of configuration errors, improve deployment times, and increase efficiency in the cloud deployment process. The results of the evaluation show that the system has the potential to revolutionize the way organizations deploy and manage their cloud environments.

List of Abbreviations and Terms

SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
HPC	High Performance Computing
FAI	Fully Automated Installation
PXE	Preboot Excution Environment
BIOS	Basic Input/Output System
NBP	Network Boot Programs
OOP	Object-Oriented Programming
OS	Operating System
PC	Personal Computer
IP	Internet Protocol
HTTP	Hyper Text Transfer Protocol
NFS	Network File System
FTP	File Transfer Protocol
TFTP	Trivial File Transfer Protocol
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
USB	Universal Serial Bus
DVD	Digital Versatile Disk
VM	Virtual Machine
SSH	Secure Shell
RAID	Redundant Array of Independent Drives or Redundant Array of Inexpensive Disks
RAIN	Redundant Array of Independent Drives or Redundant Array of Inexpensive Nodes
ISCSI	Internet Small Computer System Interface
WAN	Wide Area Network

Table of Contents

List of Figures

List of Tables

1. Introduction, Motivation, and Project Objectives

1.1 Introduction

In today's fast-paced world, technology plays a crucial role in businesses, organizations, and daily life. Cloud computing has revolutionized the way we do computing and has made it easier for organizations to scale their computing resources on demand. It has become an integral part of modern businesses, providing a flexible and scalable infrastructure for organizations of all sizes. The global cloud computing market is expected to reach $623 billion by 2023, growing at a compound annual growth rate of 18.3% from 2018 to 2023.. Furthermore, according to a survey by IBM, 68% of organizations are using or planning to use cloud computing to improve their competitiveness and reduce costs. [1]

However, deploying cloud systems has its own set of challenges, one of which is configuration errors that can occur due to human errors and other factors. We define the configuration to be the operations that determine the values of system settings. The goal of configuration is to transform the system from an unsatisfying state to the intended state by changing the settings. The state of a system can be measured by a few aspects of system behavior, such as functionality, performance, reliability, and security. Software configuration errors fall into one of the following three categories:

1. Parameter: erroneous settings of configuration parameters (either an entry in a configuration file or a console command)
2. Compatibility: configuration errors related to software incompatibility
3. Component: the other remaining configuration errors (for example, missing a specific software module)

Most of the existing research efforts focus on parameter misconfigurations because they account for the majority of real-world configuration errors (70.0%–85.5% of the studied misconfiguration cases in Yin et al. [2011]). [2] Moreover, the configurations of compatibility and components can also be modeled and represented by parameters.

Unlike software bugs, configuration errors are not defects inside the the software itself (source code). Even perfectly coded software can be misconfigured by the users. Traditional bug detection approaches based on program analysis, either statically or dynamically, cannot deal with configuration errors. Furthermore, software bugs are supposed to be fixed by developers, but configuration errors would be fixed directly by the users themselves. Debugging may not be an option for users or administrators by which to troubleshoot configuration errors, mainly because of users' lack of expertise or not having debugging information. These factors amplify the problem of configuration errors. These configuration errors can lead to system crashes, slow performance, and data loss, making the entire deployment process a costly and time-consuming exercise. The manual process of configuring cloud deployments can be time-consuming and error-prone, leading to costly downtime and security risks. In order to overcome these challenges and improve the efficiency of cloud deployments, the development of automated deployment systems is crucial. Deployment automation improves a facility's software deployment infrastructure. It enables rapid, efficient delivery of software services as well as eliminates the need for human involvement in application deployments.

To overcome these problems, this thesis proposes a cloud auto-generation system that automates the cloud deployment process, eliminating the risk of human error and improving the efficiency and reliability of cloud deployments. The system utilizes ready-to-use templates that are free of configuration errors and can be quickly deployed, reducing the time and resources required for manual configurations. The proposed system is designed to address the current challenges in cloud deployment and to help organizations of all sizes and industries to streamline their deployment processes. This thesis provides an evaluation of the cloud auto-generation system, demonstrating its ability to reduce configuration errors, improve deployment times, and increase efficiency in the cloud deployment process.

The user can choose from entire system templates or customize their own template, making use of finer-grained templates for the operating system installation or SAS configuration step. This customization process allows for greater flexibility and allows the user to configure their cloud system according to their specific needs. Additionally, by automating the deployment process, our system significantly reduces the time required to deploy cloud systems, making it a scalable and cost-effective solution. Our system connects to the target bare-metal servers via a connection module and uses kickstart to automate the operating system installation process, including partitioning and RAID creation. It also handles the SAS configuration process, ensuring that the final deployment is free of configuration errors.

Deployment automation is especially important for any business that wants to reduce or eliminate the problems of manual deployments or improve the speed and quality of software releases. In reality, there are multiple benefits to deployment automation. For starters, anyone on the company may deploy the software, which is a huge advantage for any firm. Understanding how to deploy or release software does not require a lot of brainpower, but it might be a finished procedure in the system. Typically, this responsibility is assigned to a single individual on a project team. If that person is absent, unwell, or otherwise unavailable on short notice, the job is delayed, putting pressure on other deadlines.[3]

Moreover, deployment automation boosts productivity. Automated software installations are completed in seconds reducing deployment from a laborious half-day operation to a simple task that is scarcely considered. Furthermore, automated deployments are substantially less error-prone than human deployments. Manual deployments take several steps and are prone to human error. Important procedures may be overlooked by mistake, errors may go undetected during the process, or wrong software versions may be used, etc.

Furthermore, deployment automation for a single software deployment has little overhead and may be performed as many times as needed. In general, this rapid performance level results in a greater frequency which, in turn, encourages improved adoption of agile software development. This is desirable because when a team can deliver more rapidly, this provides value to users more frequently and in smaller increments. Moreover, this together with the fact that deployment automation is less prone to errors, obtaining feedback easier and quicker. This feedback can be used for increased performance and efficiency.

Cloud automation is concerned with the deployment and administration of automated tasks, and employing cloud management technologies to do operations without the need for user intervention. Some of the jobs that cloud automation might do without real-time human input include auto-provisioning servers, data backup, and identifying and reducing unneeded processes. Cloud automation automates actions or processes to increase productivity and minimize human labor.

1.2 Motivation

While developing advances in computer science and engineering indicate tremendous achievements for developed nations, developing nations are still a long way from adopting such strategies in order to upgrade and fulfill the future standards for sustainable growth. One major source of this issue is the shortage of finances to support such advanced applications. Adopting the cloud paradigm is part of this. The lack of funds to encourage the adoption of such technologies extends to governmental entities such as universities and research centers, which require high-performance computation given the nature of these environments' requirement of research and business functions.

The cost of migrating to a cloud platform has constantly decreased, making it an appealing alternative for businesses of all kinds. Incorporating cloud automation as part of a cloud migration might result in significant cost savings owing to increased productivity and mistake reduction. A study by Gartner found that cloud computing can reduce IT operational costs by 30-40%, compared to traditional on-premise solutions [4].

As virtualization and cloud computing becomes more widespread, the activities associated with administering these systems, as well as the accompanying burden, expand. Manually installing and operating services like as scaling, provisioning, configuring resources, configuring virtual machines, and monitoring performance are time-consuming, inefficient, and frequently prone to mistakes that can impact availability.

Modern technological stacks are huge, dynamic, and ever-changing, with several moving elements. Data center administrators are unable to bear the strain of configuring and deploying each component in real-time while simultaneously recognizing and implementing possible enhancements. [5]

Furthermore, there are numerous patterns of failure in manual deployments. Using a broad definition of manual deployments, it is easy to understand where they fall short. A deployment is considered manual if it is characterized by operators logging into numerous machines and running programs. This manual process is slow and unreliable, prone to mistakes and failure, and inadequate in visibility and traceability. Visibility and traceability are important to be achieved in a deployment process because a deployment process is not a separate and independent event in time; once the deployment is completed, team members still need to know what was deployed, where it was sent, why it was deployed, and who executed the deployment—especially if a failure happened.[6]

For example, if a push-button deployment exists but the system does not give insight into the process, team members must dig through logs on several workstations to trace out the error which is a manual and time-consuming procedure. Keeping this expanded concept of deployment in mind may aid in identifying a pattern of activities that demonstrate why manual deployments are sluggish, error-prone, and "black boxes." As a result, these "poor" behaviors will give insight and motivation into what is required from a system that will run rapidly, successfully, and with high traceability (i.e., an automated system).

Cloud automation is critical to a successful transition to cloud infrastructure and can involve operations such as automatic storage and backups, security and compliance management, updating configurations and settings, and code deployment.

Automation solutions assist assure maximum performance from the system and its resources by automating cloud computing-related processes. They can also enhance efficiency by decreasing the need for IT professionals to handle repetitive tasks or make real-time choices regarding capacity or performance. Using automation to operate workloads in the cloud rather than on-premises can help save money and resources.

Manual deployments are flawed, and more disciplined deployment engineers cannot save them. The tedious job of conducting a deployment should be assigned to an automated system that can consistently execute a procedure. But not all automated solutions are the same. There are several "must-have" goals to keep in mind whether you are seeking to buy or construct a deployment automation solution. A mature deployment system should include the following characteristics:

1. Reliable, high-performing installations, particularly in production
2. Simple deployments encourage teams to adopt new releases more quickly.
3. Rapid deployments allow early test environments to be on the most recent build feasible.
4. Complete audit trail that spans all environments
5. Strong security and division of duties—controlling who can do what, when, and where—are essential.[6]

The motivation for the cloud auto-generation project stems from the increasing demand for cloud computing services in various organizations and the need to scale the cloud deployment process. One of the primary challenges in deploying cloud services is the configuration of the target servers, which is prone to errors due to human intervention. The manual configuration process is time-consuming and requires a high level of technical expertise, making it a major bottleneck in the scaling of cloud services.

In addition to human errors, other factors such as lack of standardization, inconsistent configuration, and mismanagement of configurations can result in configuration errors and cause the system to break. These errors not only result in downtime but also pose a significant risk to the security of the cloud system. A study by the Cloud Security Alliance found that the most significant security risk in cloud computing is misconfigured systems and human error, with 83% of respondents reporting that misconfigurations have caused a security incident. [7]

To overcome these problems, the cloud auto-generation project aims to automate the cloud deployment process by creating ready-to-use templates that are free of configuration errors. The cloud auto-generation system connects to the target bare-metal servers via a connection module and uses kickstart to automate the operating system installation process, including partitioning and RAID creation, and handles the SAS configuration processes, enabling the user to choose entire system templates or customize their own template and save it for future use.

The advantages of automating the cloud deployment process are numerous. By automating the configuration process, we can ensure that the same configuration is applied to all machines, reducing the chances of configuration errors. Additionally, the use of templates allows us to standardize the configuration process, making it easier to manage and maintain the cloud system. Furthermore, the user can make use of finer-grained templates for the operating system installation or the SAS configuration step, enabling greater flexibility and customization.

It is critical to optimize software development lifecycles. It is primarily about producing higher-quality products in a more efficient and productive manner. A cloud approach that combines automation can benefit IT infrastructure by allowing the system to send out numerous modifications while automatically monitoring and identifying errors and rolling back changes if performance issues arise. The ability to release new code without jeopardizing system stability is crucial to continuous delivery since it allows for quicker resource deployment and scalability.

Managing cloud services through automation results in a defined set of predictable procedures and policies that can be quickly updated to nowadays business's changing demands, resulting in improved control and enhanced agility of the system.

As cloud platforms gains popularity, finding ways to automate key activities becomes critical to the capacity to manage existing systems while providing time for innovation and upgrades. Cloud management automation enables improves IT governance while using fewer resources and/or employee effort.

1.3 Project Objectives

Over 75% of respondents in a continuous poll conducted by UrbanCode prior to its acquisition by IBM classified their deployment procedure as "Entirely Manual," "Mostly Manual," or "Mostly Scripted." Few companies claimed to have push-button deployment capabilities. [8] This outcome is not surprising: Until recently, deployment-automation options were limited, with vendors offering "solutions" that do not scale to the enterprise, need significant process re-engineering, and provide no visibility into the "who, what, when, where, why, and how" of deployments.[6]

Companies have resorted to internal solutions and increased scripting to assist their team members in achieving more regular deployments and keeping up with the rising demands of competitive marketplaces. Even after early success, the solution frequently fails to stand the test of time—changes in techniques, such as the implementation of Agile, need a fresh rewriting of the endeavor. These modest improvements, however, do not adequately address the core issues of manual deployments. They still rely on humans to connect to a system, run programs in a specific order, and alert interested parties.[6]

The primary objectives of the cloud auto-generation system developed in this study are:

1. To overcome the problems of configuration errors due to human errors and other factors by automating the cloud deployment process.
2. To create ready-to-use templates that are free of configuration errors, and that can be easily scaled as per the needs of the deployment.
3. To automate the operating system installation process, including partitioning, RAID creation, and SAS configuration, through the use of kickstart and other tools.
4. To provide users with the ability to choose from entire system templates or to customize their own templates, and to capture their configurations in configuration-error-free templates and save these templates for reuse.
5. To make use of finer-grained templates for the operating system installation or the SAS configuration step, to provide users with greater control over the configuration of their cloud deployments.
6. To create a connection module via which the system connects to target bare-metal servers.
7. To include a post-installation utility that configures the machines as nodes automatically.
8. To provide users with the ability to add classes to let certain machines be configured differently, while still maintaining consistency across the deployment.

Overall, the goal of this project is to provide a robust and scalable cloud auto-generation system that streamlines the deployment process, reduces the risk of configuration errors, and empowers users to fully control their cloud deployments. The cloud auto-generation system developed in this study is a game-changer in the world of cloud computing. It eliminates the risk of configuration errors, saves time, and ensures that the final deployment is scalable and cost-effective. We are confident that our solution will revolutionize the way cloud systems are deployed and help organizations achieve their computing goals more efficiently.

2. Literature Review

2.1 Private Cloud

A private cloud is a cloud computing environment that is dedicated to the use of a single organization. It is typically operated on a cloud infrastructure owned, managed, and maintained by the organization itself or by a third-party service provider. In contrast to public clouds, which are shared among multiple organizations and customers, private clouds offer more control, customization, and security to their users.

Private clouds can be implemented on-premises, in a dedicated data center, or in a virtual environment. They can support a wide range of cloud services, such as infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS), and can be deployed using a variety of technologies, such as virtualization, containers, or microservices. The use of private clouds can offer organizations the benefits of cloud computing while also addressing concerns around data privacy, security, and regulatory compliance. The infrastructure and deployment process involved in setting up such a system. The infrastructure includes hardware, network, and software components required to host the DSS in a private cloud. The deployment process involves installing and configuring the necessary components, as well as testing and maintaining the system. [9]

Building a private cloud involves several steps defining the requirements and identifying the business objectives, performance, security, and compliance requirements of the private cloud. Also, specify the hardware and network infrastructure the type of hardware and network infrastructure that will support your private cloud. This can include servers, storage, switches, routers, firewalls, and other components. Selecting the cloud software that meets your requirement. Installing and configuring the hardware and configuring it according to requirements. This may involve setting up network connections, configuring storage, and defining resource allocation policies. After that create virtual machines to host the applications and data and manage these VMs using the cloud platform. Implementing security measures, such as firewalls, network segmentation, encryption, and access control, to ensure that the private cloud is secure. The next step is testing and validating the environment to ensure that it meets the requirements and validates its performance, security, and compliance. Finally, Monitoring the private cloud environment to ensure that it continues to operate smoothly and perform optimally, and perform regular maintenance as needed.

2.2 High Performance Computing

High-Performance Computing (HPC) refers to the use of supercomputers and other high-end computer systems to perform intensive and complex calculations and simulations. HPC is necessary because many important computational tasks, such as weather forecasting, scientific simulations, financial modeling, and big data analytics, require a large amount of computational power and memory.

Traditional computer systems are not capable of handling these complex tasks efficiently, so HPC is necessary to tackle these challenges. HPC systems are designed to provide parallel processing, high-speed communication between processors, and large amounts of memory, making them ideal for demanding computational tasks. HPC is also necessary for many fields that rely on advanced simulations and modeling, such as engineering, physics, and medicine. By using HPC, scientists and engineers can gain insights into complex systems, perform accurate predictions, and make informed decisions. In short, HPC is needed to address the increasing computational demands in various fields, provide more accurate results and predictions, and drive scientific and technological advancements. [10]

2.2.1 GPU Cluster for High-Performance Computing

A GPU cluster is a group of multiple GPUs connected together in a computer system to work in parallel and perform intensive computations faster. The benefits of using GPU clusters for high-performance computing include improved performance, scalability, and cost-effectiveness. There are also challenges and limitations of GPU cluster systems and the steps needed to implement them, such as hardware selection, software configuration, and networking setup. [11]

2.2.2 Next in High-Performance Computing

In this paper, the authors discuss the challenges and opportunities associated with the development of exascale computing and provide an overview of the technologies and approaches that are being developed to achieve this goal. They also provide insights into the future of HPC and the role that exascale computing will play in advancing science and engineering disciplines. This paper provides a comprehensive overview of the current state of HPC and the future trends and directions of the field.

The field of high-performance computing (HPC) is constantly evolving, with new developments and innovations constantly emerging. Some of the trends and future directions in HPC include The integration of artificial intelligence (AI) and machine learning (ML) into HPC systems is becoming increasingly important. AI and ML algorithms can be used to analyze large amounts of data and perform complex simulations, making them critical for a wide range of applications, from drug discovery to weather forecasting.

Edge computing involves bringing HPC capabilities closer to the data source, rather than relying on central data centers. This will enable real-time processing of large amounts of data, making it possible for organizations to make informed decisions quickly and effectively. Quantum computing has the potential to revolutionize HPC by offering exponential increases in processing power compared to traditional computer systems. Quantum computing is still in its early stages, but it has the potential to become a key component of HPC in the future. The growing demand for HPC has resulted in a significant increase in energy consumption, leading to a need for more environmentally sustainable computing solutions. The development of green computing technologies, such as energy-efficient processors and renewable energy sources, will be critical for the future of HPC.

2.3 General Installation Script

The "General Installation Script" is a term that could refer to a variety of scripts used in different contexts. It likely refers to a script that automates the process of installing software or other components on a computer system. A general installation script typically includes a series of commands and instructions that are executed in a specific order to install the desired software or components. These scripts can be written in various programming languages, such as bash, Python, or Ruby, and are used to simplify and standardize the installation process.

The use of general installation scripts can greatly reduce the amount of time and effort required to install software and components, as well as minimize the possibility of errors or misconfigurations. These scripts can also make it easier to deploy the same software or components on multiple systems, saving time and ensuring consistency. In general, the term "General Installation Script" is used to describe a script that is designed to be flexible and reusable and can be adapted for use in a variety of installation scenarios.

2.4 Fully Automatic Installation

It is a free, open-source software tool used for unattended operating system installation and configuration management. FAI is used to automate the installation process of one or many computers, allowing system administrators to quickly and easily deploy and configure new systems. FAI works by creating a master image of a configured system, which can then be used to automate the installation of new systems. The master image includes all necessary software, configurations, and user accounts so that the installation process can be completed without any user intervention.

FAI provides a flexible and scalable solution for system administrators, allowing them to manage large numbers of systems and ensure consistency and reliability. It supports a wide range of Linux distributions and provides a variety of configuration options, making it suitable for use in a variety of environments and use cases. Overall, FAI is a valuable tool for system administrators who need to manage large numbers of systems, automate the installation process, and ensure consistency and reliability across their infrastructure.

A fully automatic installation system for heterogeneous distributed systems, which includes automatic discovery and configuration of system components, automatic deployment of software components, and automatic configuration of system and software components. The system is designed to simplify the installation process and reduce the manual effort required to install and configure distributed systems.

The system uses a combination of technologies, including network scanning, agent-based deployment, and XML-based configuration files, to achieve its goals. The authors evaluate the performance and reliability of the system in a variety of scenarios and compare it with other existing installation systems. Overall, this paper provides a comprehensive overview of a fully automatic installation system and may be a useful resource for researchers and practitioners in the field of distributed systems and software engineering. [12]

2.5 Kickstart

Kickstart is a Linux tool used to automate the installation of the operating system. It allows you to create a customized installation script that can be used to install the operating system on multiple systems in a standardized and consistent manner, without requiring manual intervention. The script includes information such as the installation source, the language and keyboard settings, the partitioning scheme, and the packages to be installed. When the installation process is started, the Kickstart script is read and used to guide the installation, eliminating the need for manual input. This makes the installation process more efficient and reduces the likelihood of errors and inconsistencies. The Kickstart script can be stored on a network server or on a local disk, making it easy to deploy the Linux operating system on multiple systems in a large organization.

The paper "Installing Linux OS Over Network in Area of ITCGH (Using 'Kickstart')" describes a method for installing the Linux operating system over a network in the Information Technology department of a hypothetical organization called the ITCGH. The authors use a tool called "Kickstart" to automate the installation process, which allows the installation to be performed in a standardized and consistent manner. The use of Kickstart eliminates the need for manual intervention during the installation process and reduces the likelihood of errors and inconsistencies. This makes the installation process more efficient and cost-effective, as multiple systems can be installed simultaneously. The authors also describe the steps involved in the Kickstart installation process and provide details on how the tool can be configured to meet the specific needs of the ITCGH. Overall, the paper provides a practical guide for using Kickstart to install Linux over a network in an IT environment.[13]

2.6 Difference between Software Misconfigurations and Software Bugs

Unlike software bugs, configuration errors are not defects inside the software itself (source code). Even perfectly coded software can be misconfigured by the users. Traditional bug detection approaches based on program analysis (either statically or dynamically) cannot deal with configuration errors. Software bugs are supposed to be fixed by developers, but configuration errors can be fixed directly by users themselves. Debugging may not be an option for users (administrators) by which to troubleshoot configuration errors, mainly because of users' lack of expertise or not having debugging information.

2.7 Causes of Misconfigurations

Can be caused by data corruption as a result of system errors. Examples of system errors include failed uninstallation of applications, applying patches that are incompatible with existing applications, and software bugs that corrupt the configuration files on the disk. Misconfiguration errors can also be caused by users' inadvertent action slips when setting configuration parameters. Examples of human errors include typos (missing a dot in a domain name of DNS configurations for example). [2]

2.8 State-of-the-art Systems Approaches

2.8.1 Building Configuration-Free Systems

1. Eliminating Configurations
   Since the flexibility comes with a high cost (complexity and error proneness), the following two types of configurations should be eliminated:
   (a) Configurations that are not needed by most of the users
   (b) Configurations that are hard for users to set correctly
      i. Eliminating unused configurations.
         Xu et al. [2015] report that 31.1%–54.1% of the configuration parameters were seldom set by any users in four mature, widely used system-software projects Hiding or removing these parameters can significantly simplify configurations with little impact on existing users

Some parameters could be automatically set according to the information of the current system instance
Example:
PostgreSQL has several preset parameters whose values are determined during the installation of the software. For example, the blockSize (the data-block size) parameter value is automatically set to be aligned with the size of the file system's disk blocks in order to optimize the disk usage To understand which configurations the users need, system vendors should maintain user-feedback loops for configuration design Such data provide great opportunities to understand users' configuration settings in the field, including how the users set each parameter and which parameters are rarely set. [2]

ii. Eliminating hard-to-set configurations
Hippodrome analyzes a running workload to Determine its requirements Calculate a new storage system configuration Migrate the existing system configuration to the new configuration It makes better configuration decisions by systematically exploring the large space of possible configurations based on the predefined storage system models. MUSE applies similar principles to generate the service and server configurations toward provisioning energy-conscious resources for Internet hosting data centers DAC automates IP-address configurations for data center networks, considering both the locality and topology information for performance and routing purposes In this way, it eliminates the error proneness of traditional address configuration protocols such as Dynamic Host Configuration Protocol (DHCP) that require a huge amount of manual inputs. [2]

iii. Automatic Performance Tuning
One specific class of hard-to-set configuration parameters is those related to system performance. Large systems usually include a large number of performance-related parameters (memory allocation, I/O optimization, parallelism levels, logging). Mature systems provide default values for these configuration parameters. The default values are usually carefully selected by developers based on their experience and/or in-house performance testing results. However, given a particular workload and system runtime, it is hard for the static default values to deliver the optimized system performance. The basic idea is to model

the performance as a function of configuration settings.

With such a model, the performance tuning problem is to find the value set v that achieves the best performance (argmax) in terms of the metric. The performance function is often unknown or does not have a closed form A number of black-box optimization algorithms have been proposed including the recursive random algorithm, the hill-climbing algorithm, and the neighborhood-selection algorithm. Other studies try to capture the performance characteristics using specific models like Using a Gaussian process to represent the response surface of performance functions in relational database systems and applying dependency graphs to describe the performance dependencies among different parameters of Web applications.

It is fundamentally difficult to precisely model the system performance in the field due to many unexpected factors. Consequently, some of the models are limited to certain workloads and environments, making them less appealing in practice. [2]

2. Reusing Configuration Solutions

   As the configuration is difficult, the experience and efforts toward the correct configuration solutions should be shared and reused Previous working solutions are likely to be helpful (and even directly applicable)

   (a) Configuration File Templates:

   Typical config file template is a working solution for one particular use case. Users can start with the templates based on the upper-level applications, and edit system-specific parameters accordingly Templates can also be made for different hardware or environmental conditions. Can be distributed by third-party tool providers or among user communities. For example, the user communities of configuration management tools (Puppet, Chef, and CFEngine) have the tradition of sharing configurations for a variety of common software systems. Static configuration file templates have two major limitations. They cannot capture configuration actions. For example, setting file permissions, installing software packages, creating new user groups.

   The solution provided by the templates may not work for the new system Applying a wrong solution may have side effects (changes of the system states) that need to be undone, Manually undoing system changes is not only tedious but also difficult as users may not be aware of some implicit changes of system states

(b) Record-and-Replay Systems

Record-and-replay systems are proposed to address the preceding two limitations and to make configuration reuse fully automated and its methodology is as follows:

i. Record the traces of a working configuration solution on one system
ii. Replay the traces on other systems under configuration

    If the replay fails (not passing predefined test cases), the system will be automatically rolled back to its original state.

Traces for the same configuration problem may differ (due to system and environment settings). Multiple raw traces could be merged to construct the canonical solution. A centralized database is usually deployed to maintain all the solutions for easy retrieval.

- Autobash Record-and Replay System

  AutoBash is designed for recording and replaying configuration tasks performed in Bash (or other UNIX-style shells). It leverages kernel speculation to automatically try out solutions from the solution database one by one until it finds a working one for a configuration problem. The kernel speculator ensures that the speculative state is never externalized (isolates AutoBash's replay activities from other non-configuration tasks). AutoBash requires users to provide test cases and oracles (called predicates) in order to decide whether or not the configured system is correct. Later, Su and Flinn further propose the automatic generation of predicates by observing the actions of users' troubleshooting processes

- KarDo Record-and Replay System

  KarDo is designed for automating Graphical User Interface (GUI) -based configurations on personal computers It records the window events when users perform configuration tasks based on OS-level accessibility support KarDo analyzes the raw traces and identifies the window events specific to the task and the events for state transition events It then constructs the canonical solution for a configuration task The canonical solution works for systems with different internal states

2.8.2 Making Systems Easy To Configure

The key to this goal is to treat configuration as user interfaces and adopt the user-centric philosophy for configuration design and implementation. In the declarative configuration, the idea is to help users state what is to be configured, but not necessarily how it is to be configured. Users can work with high-level primitives instead of the minutiae of configuration settings.

1. Declarative Configuration
   This is done with a hierarchical system model to express rules that define how various components (sub-models) compose a system. The models are in the form of functions defining the configuration constraints between different components. A system instance is produced by assigning the system parameter settings as the inputs of these model functions.
   With this model, users can establish system policies to define whether a system instance is acceptable or not. For example, they can establish the system policy that a system model is valid only when all its sub-models are valid.
   Puppet Declarative Configuration Tool
   Puppet configuration snippet declares two system services: apache and apache-ssl. apache requires a package installation. apache-ssl is based on the apache service and requires an additional file. Puppet would ensure the dependencies according to the declarations.

```
class apache {
    service{   apache   : require -> package[   httpd   ] }
}
class apache-ssl inherits apache {
    // Host certificate is required for SSL to function
    service[   apache   ] { require -> file[  apache  . p e m ]
     }
}
```

   Declarative Methods Applied to Resource Configuration
   Hewson design an object-oriented configuration language that allows administrators to provision physical machines by describing the constraints (hardware limit), and describing requirements of resources. It translates users' declarations into a Constraints Satisfaction Problem (CSP) and uses a CSP solver to find a valid solution. Other tools model the declared problems into Boolean

Satisfiability Problems (SAT) and Constraint Logic Programming (CSP). Most existing declarative configuration approaches only allow users to declare the procedures instead of expressing their intentions of configuration.
Tools should be helping users express high-level intentions. For example, users can express an intention such as “I want my data to be more reliable”; accordingly, the system can suggest, or automatically adjust, relevant configuration parameters such as RAID level or the number of data replicas. [2]

2. Design and Implementation of User-Friendly Configuration Constraints
   Design and Implementation of User-Friendly Configuration Constraints Design principles. The constraints are one part of the configuration interface presented to the users, and their design should follow the interface design principle. The following principles should be carefully applied and evaluated in the design of configuration constraints.
   - Intuitiveness: Being intuitive means speaking the users’ language and to minimize the users’ memory load.
   - Consistency: computer system users tend to derive operations by analogy with other similar aspects of the system. Inconsistent constraints lead to confusion.
   - feedback: When users’ configuration settings violate constraints, the system should pinpoint the error and provide potential solutions via the display or log messages.
   - Minimalism: constraints should be designed as simple as possible.
   - Help and documentation: The constraints of each parameter should be rigorously documented not only in user manuals but also in the systems themselves.[2]

2.8.3 Hardening Systems Against Configuration Errors

Systems themselves should anticipate potential configuration errors and be hardened against these errors. Ideally, when the users misconfigure the system, the system should not fail or crash; instead, the system should deny the erroneous settings and print log messages to pinpoint the errors.

Misconfiguration vulnerabilities are unexpected system behavior under certain configuration errors, including crashes, hangs, missing functionalities, producing incorrect results, and mistranslating users' settings.

Fixing these vulnerabilities includes enhancing system resilience to the errors (adding proactive checking code) and improving system reactions to the errors (denying the errors and printing error messages).

The key is to have a small set of representative configuration errors as test cases. The errors used for testing should be efficient so that they can expose misconfiguration vulnerabilities in a short amount of time, realistic so that they are likely to be made by real-world users, and systematic to expose as many vulnerabilities as possible.

- ConfErr
  ConfErr is a black-box configuration testing tool. It uses human error models rooted in psychology and linguistics to generate realistic configuration errors. The configuration errors generated by ConfErr include:
- Spelling errors (omissions, insertions, substitutions, case alterations)
- Structural errors (reverting two sections in a configuration file)
- Semantic errors (violating RFC, inconsistency between functionality)

ConfErr understands neither the semantics nor the constraints of each configuration parameter. Configuration errors are generated by mutating the correct configuration settings in the default configuration files. [2]

2.8.4 Checking Correctness of Configurations

Most of today's detection approaches are rule-based: the detector checks the configuration settings against a set of predefined correctness rules (also known as constraints or specifications). If a configuration setting does not satisfy these rules, it will be flagged as a misconfiguration. The key challenge of rule-based detection is to define and manage effective and useful rules.

- Defining rules: Built-in configuration checking logic to detect syntax and format errors against basic rules.
  There is a domain-specific language called Configuration Predicate Language

(CPL) for cloud system operators to write configuration-checking logic in a compact, declarative fashion. Compared with traditional imperative languages, CPL can significantly reduce the efforts of writing checking code (up to 10% reduction of lines of code) in cloud systems. Manually specified rules face the following two problems. First, it is hard to make predefined rules complete. Second, it is costly to keep the rules updated with the software evolution since it requires repeating the manual efforts.[2]

- Learning rules
  The basic idea is to learn the "common patterns" from large volumes of configuration settings collected from a large number of healthy system instances. The patterns shared by most of the healthy instances are assumed to be correct and will be used as the correctness rules for misconfiguration detection. A configuration setting that does not follow these patterns is likely to be misconfigurations. Machine learning techniques are applied to learn configuration rules from different types of configuration data.[2]

3. Methodology

To set up a cloud from bare metal, some necessary series of steps should be followed to implement any cloud system application (e.g. OpenNebula, OpenStack, etc.).

1. Make sure of the physical connection among the servers and storage devices in a desirable network connection, and, of course, being connected to the internet. The topology of the servers' connection can be hard-fixed by the servers' company, as in our case with FUJITSU Server PRIMERGY BX900 S2 Blade System, and PRIMERGY CX400 M4 and these Blade Systems are connected together as well.
2. Associate the available storage pool to the servers as needed. This can be done using SAN, SAS, NAS, etc.
3. Install OS on the servers along with the needed firmwares, applications, configurations, and any necessary commands that shall be executed before or after installing the OS and/or the cloud technology.
4. Run our cloud technology on the servers and mointor our resources using any option we would like to use (e.g. external PC).

In our case, we target E-JUST's new cloud-to-be set of servers. The cloud technology that we use is OpenNebula, as it is one of the most famous open-source solutions out there. The OS we use is CentOS 7 since it free reliable option that provides a way to install the OS on many devices simultaneously (using Kickstart which would be explained later in this section). Also, we configure the present SAS switches to connect among the servers and hard disks. In this section, we cover the exact implementation details of what we did, along with the other possibilities that we could have used, and the reason for implementing what we did.

3.1 Physical Structure

The cloud's hardware consists of 5 racks (4 identical "racks" and a "primary rack" with 2 sets of GPU servers and a main server).

3.1.1 Hardware

Table 1. *E-JUST's CoE's cloud infrastructure specifications*

Specs	**BX900S2 (14 × BX2560M2)**	**Primergy RX2540 M4**	**Primergy CX2570 M4**	**Primergy CX2560 M4**
Processor	112 x Intel Xeon E5-2640v4	2 x Intel® Xeon® Processor 4110	6 x Xeon Gold 6140	6 x Xeon Gold 6132
Hard Disk	96 TB	19.2 TB	8125 TB	2.8125 TB
DRAM	3.5 TB	64 GB	768 GB	768 GB
Raid Controller	56			
Ethernet	112/10Gbits			
Ethernet Switch	72 ports1/10 Gbits/s (u) - 24ports 1/10 Gbits/s (d)			48 × 10GBASE-T
SAS Switch	1446 GB SAS port (I) - 486 GB SAS port (E)			

The above table shows the hardware of the cloud infrastructure that we tried our system on. The first column shows the 4 identical racks' specifications, while the other 3 are found in a separate rack. Our system was implemented on the fourth rack, on of the identical ones.

3.1.2 Topology

FUJITSU provided the connections amongst servers in the same chassis (as in the 4 identical racks, and the GPU servers). The connection between the racks and within the main rack can be viewed as follows

As shown in figure 1, the 4 identical racks are connected to the main rack's main switch through a quad fiber cable. Those 4 racks have their servers connected internally inside the chassis using the present switches. The main rack's servers are also connected to the main switch through ethernet switches.

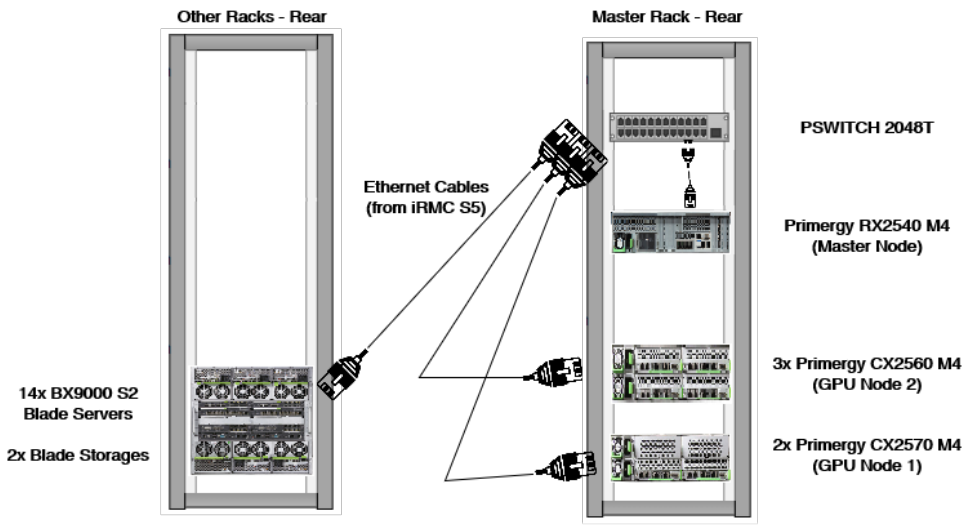


Figure 1. *Topology of the servers*

3.2 System Design

3.2.1 Object-oriented programming (OOP)

Object-oriented programming (OOP) offers numerous advantages for software development, including abstraction, encapsulation, inheritance, and polymorphism. In the context of our cloud auto-generation project, OOP greatly improves the development process by providing a clean and modular approach to designing the cloud deployment system. By using classes and objects, we encapsulate the details of the installation process, making it easier to write and maintain the code. OOP also enables us to create a hierarchy of classes, allowing for the inheritance of common properties and methods from a parent class, making the code more reusable, and reducing the risk of errors due to copy-pasting. Additionally, OOP provides a more intuitive way of thinking about the components of the system, making it easier for future developers to understand the code and continue its development. Overall, the use of OOP in our cloud auto-generation project leads to more efficient and effective code, while reducing the risk of configuration errors due to human error and other factors. We used Object-Oriented Programming (OOP) to make our code as modular as possible which would make a room for enhancements, and expand our scope later. The code structure is as shown in the system class diagram in figure 2 The Class

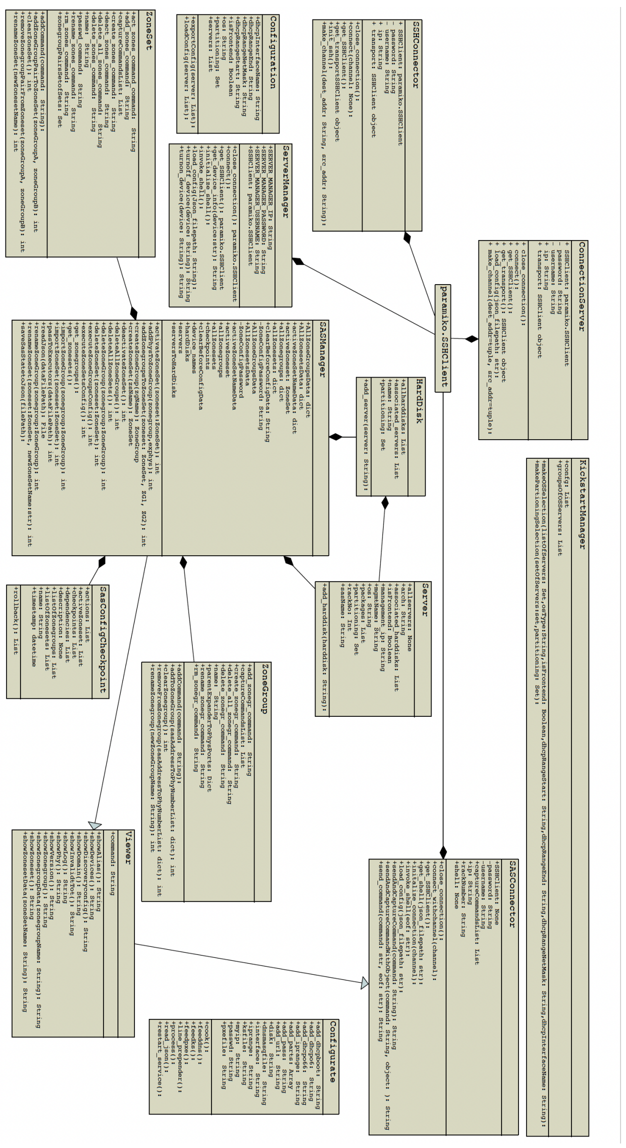


Figure 2. *System Class Diagram*

Diagram in figure 2 outlines the relationships and interactions between different classes in the Cloud Auto-Generation system. The diagram effectively demonstrates the object-oriented design principles utilized in the project to overcome the problems of configuration errors and to streamline the cloud deployment process. This diagram provides a comprehensive overview of how the different components work together to ensure a seamless and error-free deployment process.

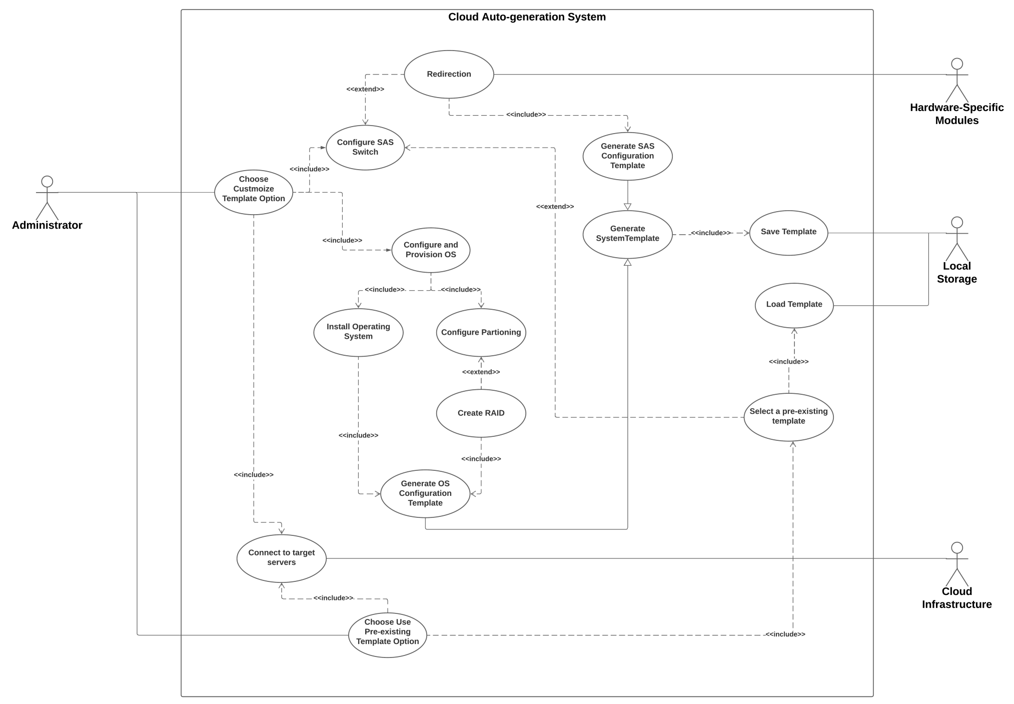


Figure 3. *UML use case diagram of the system*

Figure 3 presents the use case diagram for our cloud auto-generation system aimed at overcoming the problems of configuration errors and facilitating the scaling of the cloud deployment process through the creation of ready-to-use templates. It explains the actors, how they interact with the system and the different cases that the administrator might experience.

Figures 4 and 5 present the UML sequence diagrams built for our cloud auto-generation project providing a comprehensive view of the implementation and accurately capturing the various steps involved in the process. The diagrams highlight the flexibility provided to the user in choosing entire system templates 4 or customizing their own templates 5, which is a great feature of the system developed in this study.

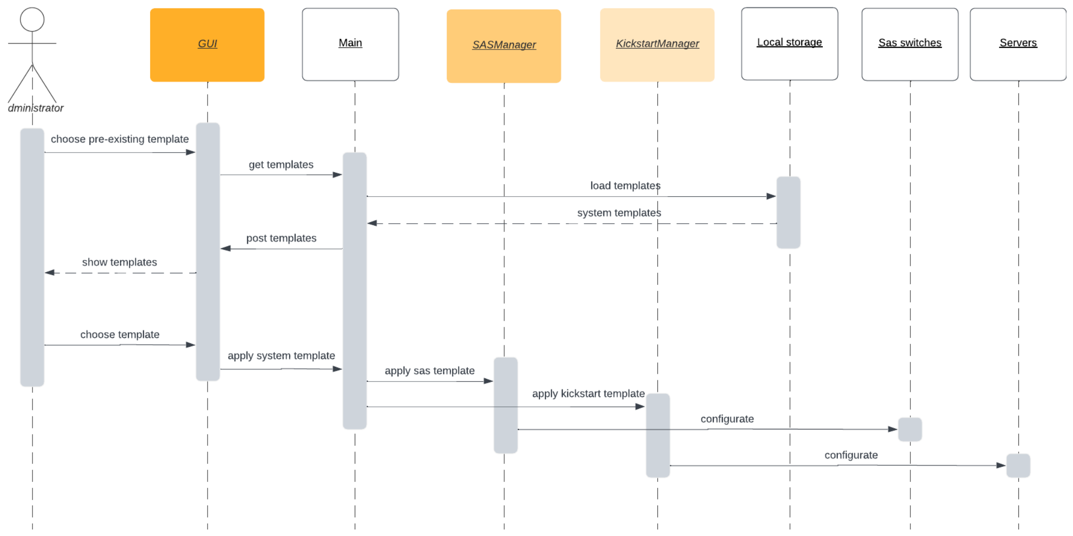


Figure 4. *UML sequence case diagram of the system: Choose Pre-existing Template Case*

The UML sequence diagrams in figures 4 and 5 clearly show the sequence of events and the order in which they occur, including the conditional and iterative steps. They also indicate the types of data and information exchanged between the components, and the constraints and limitations that apply. They depict the interactions and message flow between various system components and the administrator as the actors involved in the process.

These diagrams are useful as visualization tools that can provide a clear understanding of the underlying mechanisms and processes that enable our cloud auto-generation system to overcome the problems of configuration errors and scalability. Moreover, they can also help stakeholders better understand how the system works and identify any potential issues or areas for improvement. Furthermore, they highlight the key features of the system, such as the use of ready-to-use templates, the automation of the SAS configuration and operating system installation processes, and the ability to customize templates according to the user's needs.

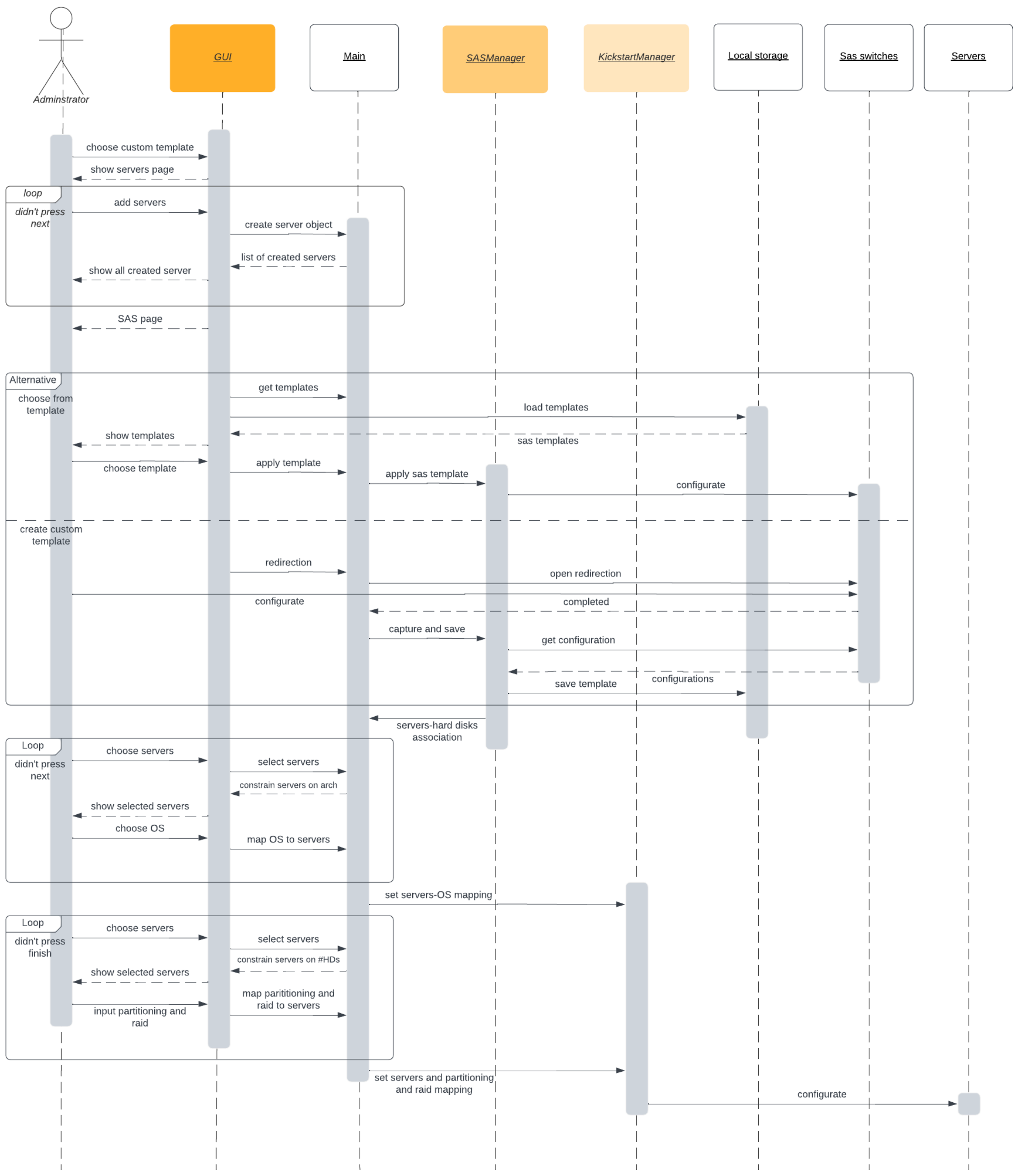


Figure 5. *UML sequence case diagram of the system: Customize Template Case*

3.2.2 Graphical User Interface (GUI)

This Graphical User Interface helps the user describe the private cloud system he aspires. The GUI helps the User specify the Hardware specs, Storage (SAS) specs, Servers, Partitioning, Raid systems, and more. It also could provide some templates to suggest from your previous work.

GUI User Interactions Diagram

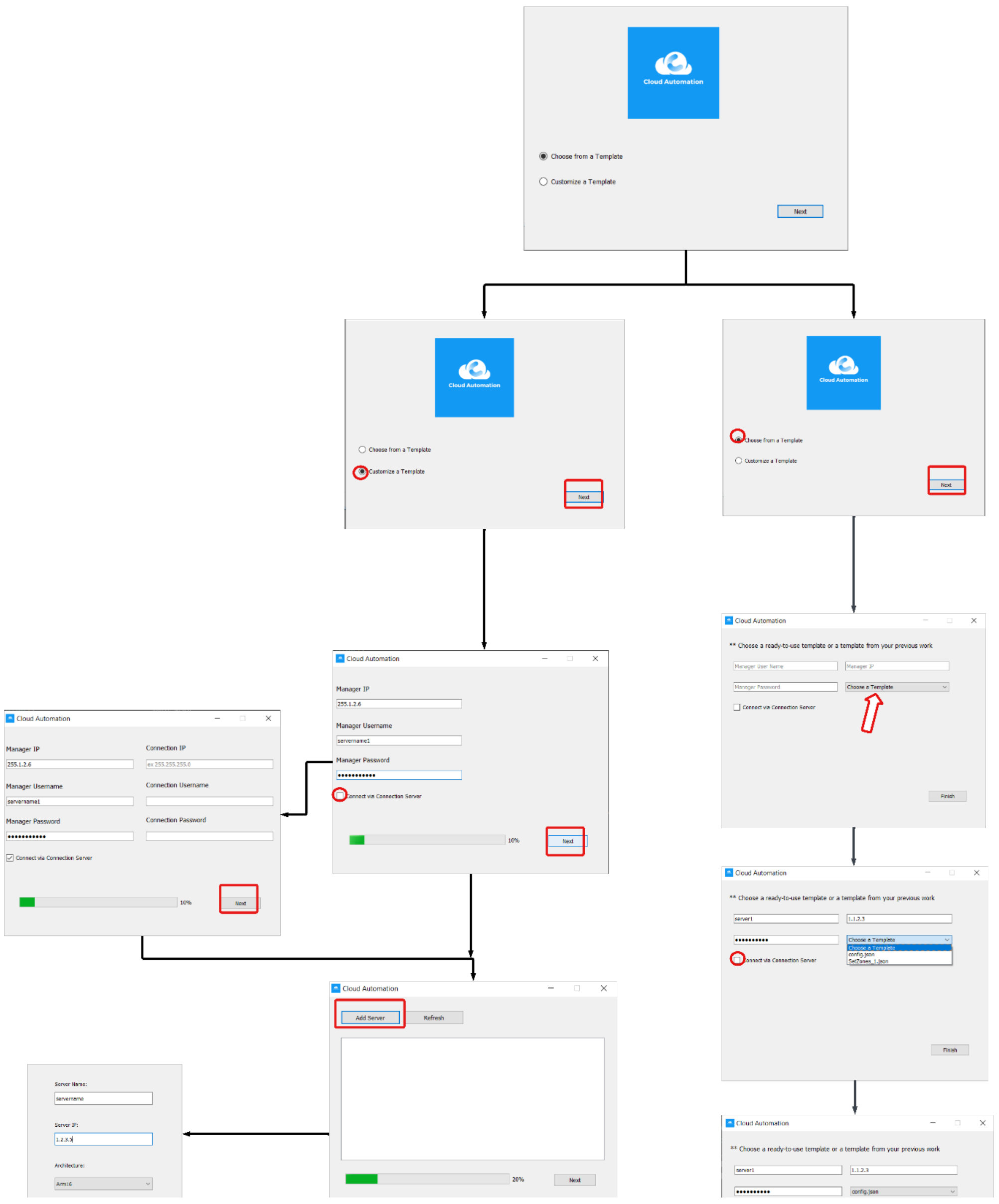


Figure 6. *GUI User Interactions Diagram part 1*

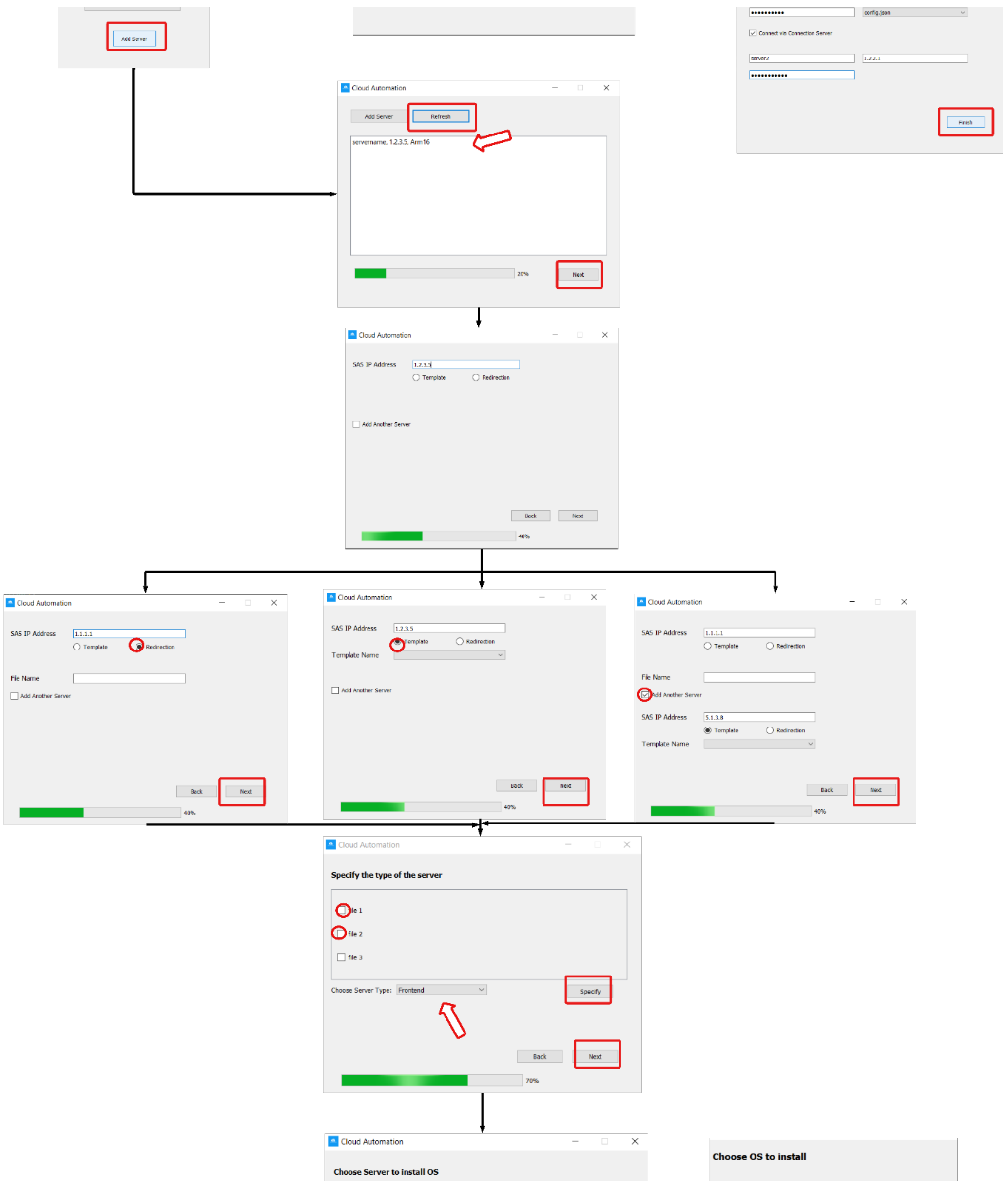


Figure 7. *GUI User Interactions Diagram part 2*

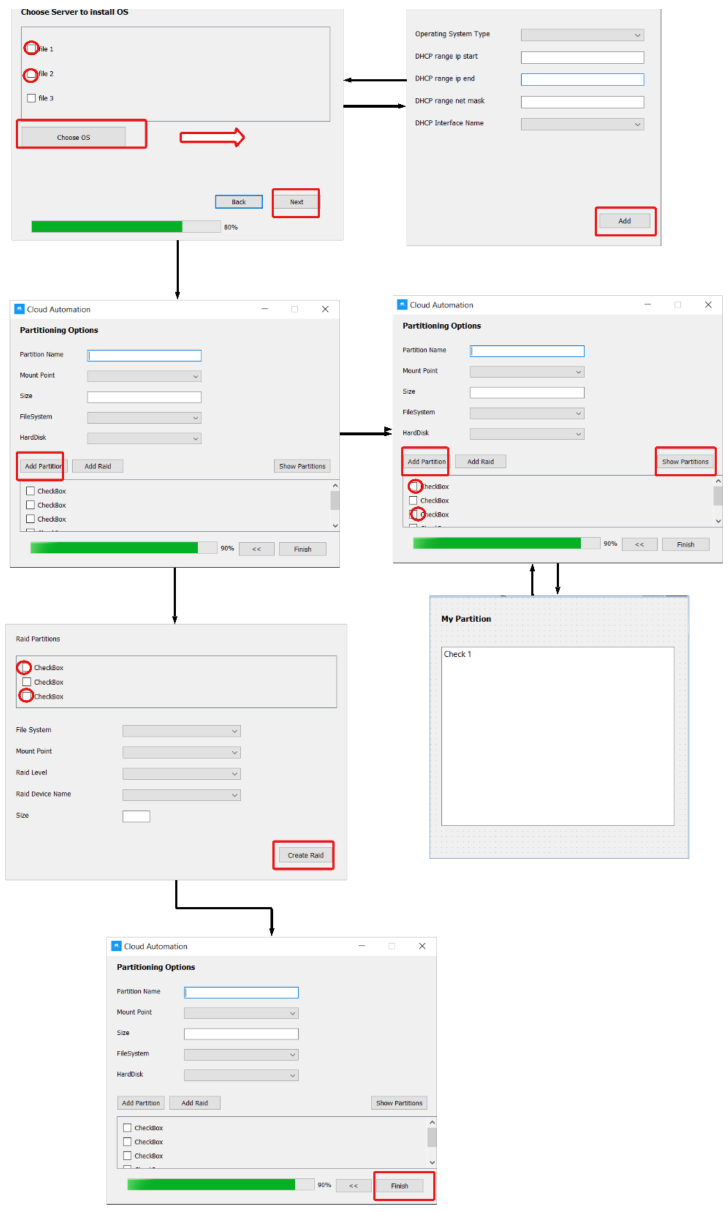


Figure 8. *GUI User Interactions Diagram part 3*

3.2.3 Connection Module

The connection module is used to establish connections from the system to the servers, the server manager, and the SAS switch, in addition to sending the needed commands to them for deployment. The connection can be done via various ways, such as Ethernet connection, ssh connection, VPN connection, or HTTP connection. In our system, we use an ssh connection.

The ssh module connects to the connection server, then from the connection server, it makes nested ssh to connect to the server manager, SAS switches, and the servers. We used such a technique manually in order to explore our connection options.

Choosing to continue with nested ssh as our way of communication with the hardware, we automated this step using Paramiko, a pure-Python implementation of the ssh protocols.

3.2.4 SAS Configuration

The SAS System is responsible for managing the sas switch present in the chassis of our system. As the chassis consists of switches, server blades, storage blades (hard disks), etc. We need to have some sort of system that can control the connections between the server blades and the storage blades. Such a system is provided by the SAS switch's functionality.

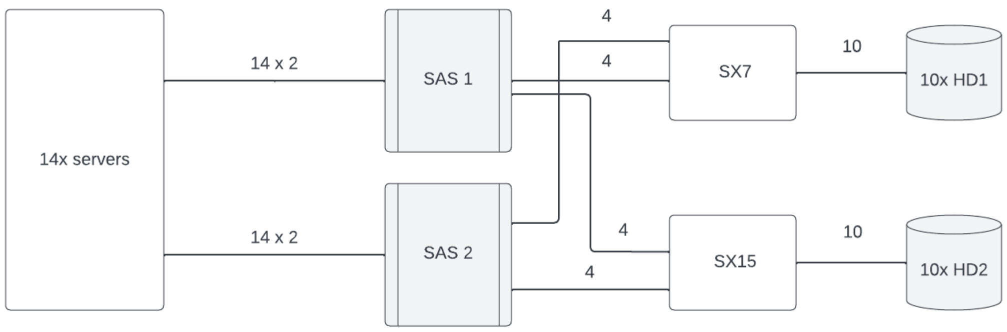


Figure 9. *Topology of the servers with SAS*

Figure 9 shows the connections among the 14 blade servers, 2 SAS switches, 2 storage expanders, and the 10*2 hard disks. The physical connections present don't allow arbitrary visibility among the servers and hard disks. However, configurations must be made, to the SAS switches (one of which is the main, the other is redundant) to allow the creation of **zone groups** and **zone sets**.

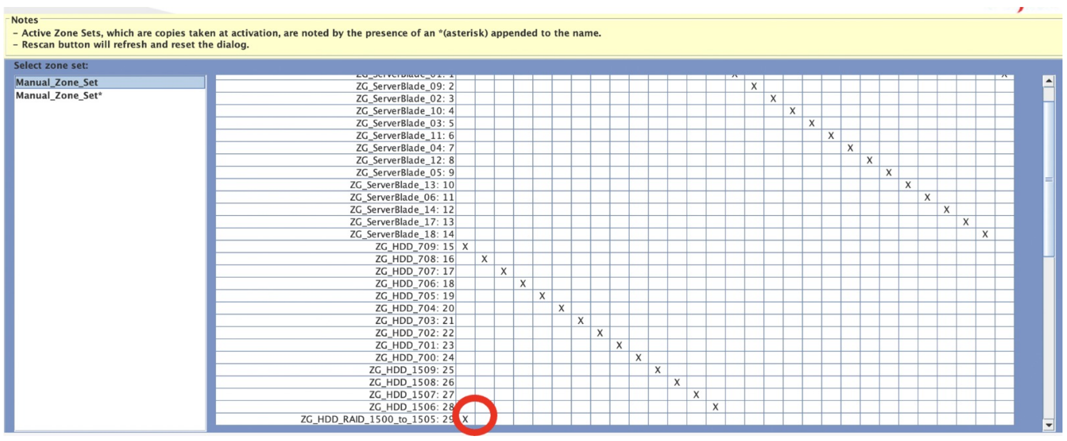
A **zone group** is a collection of phys on one or more SAS zoning expanders (like LSI SAS switches). Each phy on an expander corresponds to a device connected to that phy. Zone groups can then be given permission to communicate with phys in other zone groups (including permission to phys within the zone group itself as such permission is not granted by default) by adding the zone groups to a zone set.

A **zone set** is a collection of zone group pairs that have been given permission to communicate with each other. Zone groups and zone sets have some properties that shall be put into consideration as:

- Members of a zone group cannot by default see each other.
- Making changes to a zone group will not affect active zoning.
- Making changes to a zone set will not affect active zoning.
- Only when a zone set is activated are changes made to the domain.
- A zone group can be a member of more than one zone set.
- Devices can belong to more than one zone
- At most one zone group may be active.
- When you activate a zone (from any switch), all switches in the fabric receive the active zone set
- After completing this procedure, the connection blade (switch) can provide the basic, minimal Fibre Channel services necessary to enable hosts to access their storage.

Manual Configurations

In light of finding how to automate the processes of creating, deletion, renaming, and editing zone groups and zone sets, we tried applying different manual SAS configurations to become more familiar with the commands and understand the mentioned processes.



Notes
- Active Zone Sets, which are copies taken at activation, are noted by the presence of an *(asterisk) appended to the name.
- Rescan button will refresh and reset the dialog.

Select zone set:

Manual_Zone_Set
Manual_Zone_Set*

```
           ZG_ServerBlade_09: 2
           ZG_ServerBlade_02: 3
           ZG_ServerBlade_10: 4
           ZG_ServerBlade_03: 5
           ZG_ServerBlade_11: 6
           ZG_ServerBlade_04: 7
           ZG_ServerBlade_12: 8
           ZG_ServerBlade_05: 9
          ZG_ServerBlade_13: 10
          ZG_ServerBlade_06: 11
          ZG_ServerBlade_14: 12
          ZG_ServerBlade_17: 13
          ZG_ServerBlade_18: 14
              ZG_HDD_709: 15
              ZG_HDD_708: 16
              ZG_HDD_707: 17
              ZG_HDD_706: 18
              ZG_HDD_705: 19
              ZG_HDD_704: 20
              ZG_HDD_703: 21
              ZG_HDD_702: 22
              ZG_HDD_701: 23
              ZG_HDD_700: 24
             ZG_HDD_1509: 25
             ZG_HDD_1508: 26
             ZG_HDD_1507: 27
             ZG_HDD_1506: 28
ZG_HDD_RAID_1500_to_1505: 29
```

Figure 10. *Topology of the servers with SAS*

We managed to create 28 zone groups each corresponding to either a server blade or a hard disk. After that, we created a zone set with the mappings as in figure 10, then we activated the zone set in the current domain to apply the changes on all servers, switches, and hard disks. The red circle in fig 10 shows a mapping between 2 zone groups.

Automatic Configurations

To automate such actions to the user's desires, we have 2 ways that the user can use to apply configurations on the SAS switches. The first is to allow the user to freely create zone groups and zone sets using the SAS's default GUI and our system would capture what the user did and save the configurations for later usage. And the second is to allow the user to choose from the available configuration templates (including the ones they make from the redirection).

In the sasConfigurator.py, there are the zone group and zone set classes, where each zone group is to have a name, a list of ExpandersToPhys, and a list of commands applied to the zone group, and each zone set to have a name, a set of zone group pairs (representing the mappings), and a list of commands applied to the zone set. In the sasConnector.py, we have the grandparent class, SASConnector, that's responsible for the connection to the SAS switch, sending needed commands, and capturing these commands for future analysis. In the sasViewer, we have the Viewer class that inherits from the SASConnector. The Viewer is responsible for sending 'show' commands and receiving the outputs of such commands. These outputs are used to capture the SAS configurations of the SAS switch. In the SASManager, we have the SASManager class that inherits from the Viewer. The SASManager is responsible for managing the entire SAS switch, keeping track of the zone groups and zone sets, applying the needed configurations to the SAS switch from JSON files, and saving the configurations for future usage in JSON files. All SAS switch interactions are done through this class.

The process of configuring the SAS starts in the GUI when the user is asked to provide details about the SAS switch to configure and how they would like to configure it. We provide the user with 2 options that they might use:

1. Redirecting to the SAS switch's IP and using its built-in GUI to configure the SAS as needed. After the user completes the configuration in the redirection window, the made configuration are saved automatically by our system and exported to a JSON file for future use, if needed.
2. Using an already present template, either created by the user from the redirection or present with the system.

After completing one of the above for each SAS switch, the system takes the user to the next step, which is about kickstarting the servers. In the meantime, the SAS module continues working to provide crucial information to the kickstart partitioning GUI window, which is the associated hard disks to each server, according to the currently active zone set. As a server can have multiple associated hard disks, and a hard disk can have multiple associated servers, finding these associations is very important when it comes to partitioning the servers. When sending a kickstart configuration file, the receiving servers must have the same associated hard disks to avoid any errors in the partitioning stage. Also, if a hard disk has more than one associated server, then we need to keep this in mind when partitioning this hard disk on those different servers to avoid any conflicts among the servers' partitions.

3.2.5 PXE and Kickstart Configuration

The manual work is connecting the nodes and the server of opennebula with each other as automating their connection was seen as a huge security risk The automatic work is the tools to achieve a working open nebula on a bare metal machine

Configurate

The code defines a class called Configurate, which is used for configuring various files related to the installation of an operating system via PXE boot.

The class has the following attributes:

- passwd: the root password for the operating system being installed
- iprange: the IP address range for the DHCP server
- myip: the IP address of the network interface specified in the interface attribute
- ksfile: the file path for the Kickstart file
- dnsmasqfile: the file path for the dnsmasq configuration file
- disk: a list of disk partitions for the operating system being installed
- pxefile: the file path for the PXE boot configuration file

The class has the following methods:

- read_json: reads a JSON file specified by json_filepath and stores information from the file in the class attributes
- line_prepender: takes a file path ksfile and a string line as inputs and adds the string as a new line at the beginning of the file
- cook: generates various strings based on the information in the class attributes and stores them in the class attributes add_pass, add_url, add_parts, add_raid_-parts, add_raid_partis, add_iprange, add_dhcpboot, and add_dhcpo6. These strings will later be used to update the various configuration files.
- feedks: adds the information stored in the class attributes add_parts, add_url, add_pass, add_raid_parts, and add_raid_partis to the Kickstart file.
- feeddns: adds the information stored in the class attributes add_iprange, add_-dhcpo6, add_dhcpboot, and add_dhcpo66 to the dnsmasq configuration file.
- feedpxe: reads the PXE boot configuration file, performs some modifications to the file, and writes the new content back to the file.

The read_json method reads the JSON file and stores the information from the file in the relevant class attributes. The information includes the root password for the operating system, the IP address range for the DHCP server, and the disk partitions for the operating system. The line_prepender method is used to add a new line at the beginning of a specified file. The cook method generates various strings based on the information in the class attributes, such as the root password for the operating system and the disk partitions for the operating system. These strings are used to update the Kickstart, dnsmasq configuration, and PXE boot configuration files. The feedks method adds the information stored in the class attributes add_-parts, add_url, add_pass, add_raid_parts, and add_raid_partis to the Kickstart file, which is used to automate the installation process. The feeddns method is used to feed the dnsmasq configuration file with the necessary DHCP information. The add_iprange, add_dhcpo6, add_dhcpboot, and add_dhcpo66 class attributes are used to store the DHCP settings, which are then written to the dnsmasq file using the line_prepender method. The feedpxe method is used to feed the PXE file with the required information. The method opens the PXE file, reads its content, and modifies it to add the required information. The new content is written back to the file. The Configurate class provides a convenient way to manage the configuration files required to set up a PXE server and automate the installation process. The class can be used to read the configuration information from a JSON file, process it, and write it to the relevant files. This approach makes it easier to maintain the configuration files and reduces the risk of manual errors.

Flask Server

The purpose of this script is to create a RESTful API endpoint that updates a specific file, "/var/lib/tftpboot/pxelinux.cfg/default", with a new file name in a series of file names, stored in the "kicklist" list, once a certain number of updates, stored in the "batchlist" list, have been performed. The number of updates performed is stored in the "start" variable and the current index of the "kicklist" and "batchlist" is stored in the "indexx" and "vari" variables respectively.

When the endpoint is accessed with the "param" GET parameter equal to "secret" meaning that the post installation script started that server machine being installed on almost finished and doesn't need the pxe server anymore, the "start" variable is incremented by 1 and then checked if it is equal to the sum of "vari" and the current "batchlist" value, represented by "vari + batchlist[indexx]". If this is true and "start" is not equal to the sum of "vari" and the last value of "batchlist", represented by "vari + batchlist[-1]", the "indexx" is incremented by 1, "vari" is updated to the value of "start", and the "/var/lib/tftpboot/pxelinux.cfg/default" file is updated with the new file name in "kicklist" using the "re" module's "re.sub()" function. The "content" of the file is first read using the "with open()" statement, and then written back to the file using another "with open()" statement with "w" as the second argument. The "os.system()" function is then used to restart two services, "vsftpd.service" and "dnsmasq". Finally, a message of "Word changed successfully." is returned to the client.

If "start" is not equal to "vari + batchlist[indexx]", a message containing the current value of "start" and the target value, "vari + batchlist[indexx]", is returned to the client. If the "param" GET parameter is not equal to "secret", the message "Param not matched." is returned, and we can read these error messages to postscript log stored at the machine. The script uses the Flask module to create the RESTful API endpoint, and the app is run on "0.0.0.0" using the "app.run()" statement.

Post Installation Script

The following code as mentioned in the Appendix 5.2 performs several tasks:

It first adds autologin to the machine as a root to perform the next tasks with the first five lines reason for that is through trial and error the more consitent and smooth way to run as root from that start was that way, then it adds an initd.d script to be run at the start and we can add our installation script to it the reason for that is because the post script that is installed with kickstart is started before many basic services is started which can be very limited so that was one way to have flexibility on our installation script and then delete the init.d service and finally our installation script in is ‘msg.sh‘ which is run by the init.d service

1. The first line removes the symlink to the tty1 service in the getty.target.wants directory
2. The second line copies the getty@.service file from the /lib/systemd/system directory to the /etc/systemd/system directory and renames it to getty@tty1.service
3. The third line modifies the ExecStart line in the getty@tty1.service file to include the –autologin option with the root user and the –noclear option
4. The fourth line adds an alias to the getty@tty1.service file
5. The fifth line creates a symlink from /etc/systemd/system/getty@tty1.service to /etc/systemd/system/getty.target.wants/getty@tty1.service

6. The sixth line creates a new file /etc/init.d/srwr.sh
7. The following lines use the cat command with process substitution («EOT) to write a shell script to the /etc/init.d/srwr.sh file.
8. The following line makes the /etc/init.d/srwr.sh file executable: chmod +x /etc/init.d/srwr.sh
9. The following line adds the /etc/init.d/srwr.sh file to the system's init configuration: chkconfig –add /etc/init.d/srwr.sh
10. The next set of commands is used to install and configure OpenNebula. The opennebula.repo file is created and populated in /etc/yum.repos.d/ directory. This file is used to configure the yum repository for OpenNebula, from which the OpenNebula packages can be installed. The following line creates the opennebula.repo file with the specified content
11. The msg.sh file is created and populated in the /root/ directory. This file contains the script to install OpenNebula. The following line creates the msg.sh file with the specified content:

The msg.sh script is used to set up and configure OpenNebula with taking into consideration any error that may appear and fixing it to have a fully functional OpenNebula Server on Centos 7 system.

3.2.6 iSCSI

iSCSI (Internet Small Computer System Interface) is a protocol that enables the transfer of data over a network between a server and a client, using SCSI commands. It is an economical and effective way to share data and storage resources between multiple systems. We managed to apply iSCSI on virtual machines where a node, namely node0, shared storage partitions with another virtual machine, node1. This was done using the following steps:

First step in setting up iSCSI is to configure the network. We need to have two Linux virtual machines connected over a network, one acting as the iSCSI target and the other as the initiator. We also need to make sure that the two systems can communicate with each other using the iSCSI protocol. After that we install the iSCSI target software on node0, which is the component that allows the server to share its storage resources over the network. There are many different options for iSCSI target software on Linux, including open-source solutions like Open-iSCSI and commercial offerings like Microsoft iSCSI Software Target.

Next, we configure the iSCSI target. We do so by making physical volumes from our existing hard disks, creating a logical volume group from these physical volumes, and finally creating the (logical) partitions from these volume groups. We specify the partitions we wish to share and set up access controls to restrict access to the shared storage. After that, we install iSCSI initiator software on node1, which is the component that allows the client system to access the shared storage on the target system. There are many different options for iSCSI initiator software on Linux, including open-source solutions like Open-iSCSI and commercial offerings like Microsoft iSCSI Initiator.

Then, we connect from the iSCSI initiator VM to the iSCSI target VM. This involves specifying the IP address or hostname of the target system and logging in using the credentials we configured when setting up the iSCSI target. The iSCSI initiator shall have the same iqn name that has permission in the iSCSI target VM. Finally, we mount the shared storage to make it accessible on the client system. This is done using the operating system's file manager or by specifying a mount point in the iSCSI initiator software. We have to try reading and writing data to the shared storage to make sure that it is accessible and functioning as expected.

4. Results and Discussion

4.1 Results

Trying different schemes to build our system, we found that building our system using OOP resulted in the best practices in terms of the system being modular, where one module wouldn't affect the other, for example, failure of the SAS module wouldn't affect the performance nor functionality of the kickstart module. In addition, OOP allowed ease of work distribution among the team members, clarity of contribution, and allowance for easy integration.

4.1.1 Connection and SAS Modules

Regarding the SAS module, it was tested on the SAS switch in the fourth rack. We started by testing the connection of the module with the switch and if it does send the needed commands or is not as in the manual configuration using the SAS switch's CLI. It was found that the commands' output captured by our module wasn't a normal string, as it was not readable when parsing it using python's regex. To overcome this problem, a new directory, `/tmp`, was made and when new output was captured a file in the `/tmp` is created/modified accordingly with the content of the output, which is read after that to parse using regex. Such a method worked well and didn't waste considerable time, since the size of the output wasn't relatively big.

In the SAS switch configurations, our SAS module was able to function the following correctly:

1. Initalize connection with the switch, with the help of the connection module, invoke a shell for command sending and output retrieval, and close the connection after work completion.
2. Send the right commands according to the needed functionality such as creating a zone group/set or modifying them.
3. Perform complex functionalities such as:
   - Capture SAS switch's state (zone groups with their expander to phys,

zone sets with their zone groups mapping, and active zone set).

- Export SAS switch's state to a JSON file.
- Read and apply SAS switch's state from a JSON file.
- Get servers' associations with hard disks according to a given zone set mapping

The SAS module offers a wide variety of functionalities that allows the expansion of the SAS switch configuration to the user's needs.

4.1.2 Kickstart Module

While building the Opennebula frontend, we ran into many issues. Opennebula runs mainly on ruby and installs its dependencies along with it, so it needed a version compatible with its functionality that uses the same version over all of the system and all users, so a ruby version manager was used. Also, there were known issues for the last versions of opennebula that we encountered. Such problems were `oned` program cant initialize because of failed dependencies so the latest version is used, and connecting the KVM nodes to the frontend we needed to SSH passwordless between all nodes and frontend so they can communicate.

While building the PXE server we wanted to boot an OS on fresh machines and install the Opennebula and KVM nodes on these machines using the kickstart script, so we built a cloud from fresh virtual machines with automation. We tried first to build `http` server for the machines to grab the OS drive and the kickstart, but the server couldn't handle it and was disconnecting halfway. Consequently, we had to switch to `ftp` server, which was consistent. After that, we configured the network for the machines to get an IP from our `DHCP` server and then grab the necessary files to install the OS and run the kickstart file and installation scripts.

The post-installation scripts failed to run on the machine as they run before the system modules' initialization. For example yum packages are not yet started which is considered to control half of the installation script, so it needed to be saved in a file and then run as a startup script once so the system and all necessary modules would have started. This could be done using multiple methods. We used `init.d` with the `/etc/init.d` folder containing the lifecycle executables of the services managed by the system. Furthermore, we can add our own by creating an LSB-compliant wrapper that starts our service which is the installation script.

All the configurations were based on legacy BIOS machines. When testing the kickstart, DHCP server, and PXE server on virtual machines, everything worked smoothly. However, on the actual infrastructure of E-JUST, the servers only read the files but didn't execute them since the servers used UEFI. So we needed to configure the server to run on hybrid BIOS setup legacy and UEFI.

4.2 Discussion

4.2.1 Deployment Speed

Network Connection Time

There are several ways for an administrator to connect to the target servers in a cloud system to install an operating system and configure a SAS switch. Some examples are:

1. Remote desktop protocol (RDP): This protocol allows to remote control of the target server and can be used to install an operating system and configure a SAS switch.
2. Secure Shell (SSH): This protocol allows you to securely connect to the target server and perform administrative tasks, including installing an operating system and configuring a SAS switch.
3. Virtual Private Network (VPN): A VPN can be used to securely connect to the target server and perform administrative tasks, including installing an operating system and configuring a SAS switch.
4. Out-of-band management: This method allows to connect to the target server using a separate network connection, such as a serial or modem connection. This is typically used when the target server is not accessible via the main network.

For the sake of the analysis of the speed of the cloud auto-generation system, we will assume that the manual configuration is done after connecting to the target servers via SSH protocol. We will be calculating the SSH connection time

including connection initialization, sending commands, and receiving responses. It is assumed that the connection protocol followed in the cloud auto-generation system is SSH protocol as well. SSH connection time can vary depending on several factors such as network latency, bandwidth, and encryption overhead. In this study, it is assumed that for both the manual process and the auto-generated configuration case these factors have the following typical values:
Network speed = 6.7 Mbps
Character Size = 2 bytes
Data Size = Number of Characters $\times$Character Size

We assume an average network speed of 6.7 Mbps and a typical latency of 50 to 150 milliseconds and an overhead in the range of 1 to 10 milliseconds for a Wide Area Network (WAN). This type of network is used to connect computer networks in different geographic locations. WANs are typically used by large organizations such as multinational corporations, government agencies, and educational institutions to connect their branch offices and data centers. WANs can also be used to connect remote employees to the main network, as well as to provide internet connectivity to users in remote locations. Examples of technologies used to implement WANs include leased lines, Frame Relay, MPLS, and VPNs. [14]

Therefore, a rough estimate of the time for an SSH connection can be calculated using the following formula:

$$\text{SSH Connection Time} = \frac{\text{Data Size}}{\text{Network Speed}} + \text{Latency} + \text{Overhead}$$

$$\text{Data Size} = \text{Number of Transmitted Characters} \times 2 \text{ bytes/character}$$

$$\text{Latency} = 150 \text{ ms}$$

$$\text{Overhead} = 10 \text{ ms}$$

Let Number of Transmitted Characters be n

$$\text{SSH Connection Time} = \frac{16\text{n bit}}{7.025 \times 10^6 \text{ bps}} + 1.50 \times 10^{-1} \text{ s} + 1.0 \times 10^{-2} \text{ s}$$

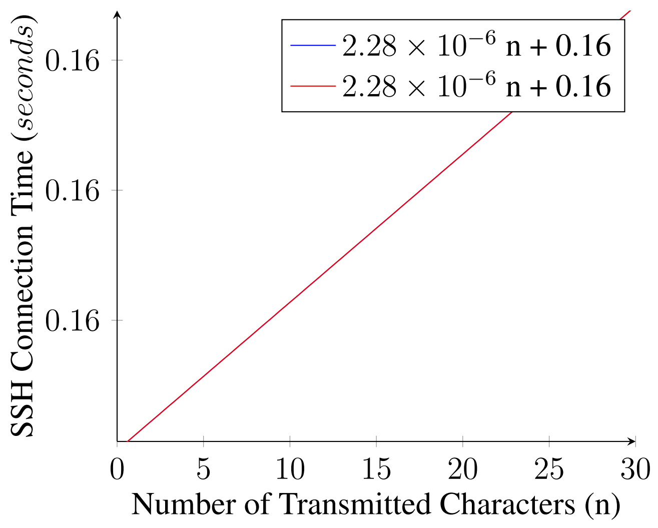
Where:

- Data Size is the size of the data being transferred over the network
- Network Speed is the bandwidth of the network in bits per second
- Latency is the time it takes for a data packet to travel from one point to another in the network
- Overhead is the time required for the SSH protocol to handle encryption, compression, and other functions

This formula provides an estimate of the time required for a single round-trip of data transfer, and the actual time for an SSH connection will depend on the specifics of the implementation and the network environment.

Using the above function as a proof-of-concept estimation for SSH connection time in the graphs below represents this relation. Since it is assumed that both the manual and the auto-generated configuration scenarios will be applied in the same environment with the factors mentioned above being identical, therefore, the relationship between SSH connection time and data size in both cases is identical and it is shown in the two graphs below. As the size of the data being transferred over an SSH connection increases, the connection time will also increase. This is due to the fact that larger amounts of data require more time to be transmitted and processed.

It is possible to see non-linear relationships in some cases, where the connection time increases at a faster rate than the data size. This can be due to limitations in the network or the devices, and could indicate a bottleneck in the transmission.



SAS Switch Configuration Time

The amount of time it takes to manually configure a SAS switch can vary greatly depending on various factors such as the size of the switch, the complexity of the configuration, the familiarity of the person performing the configuration, and the tools and resources available. Typically, a basic configuration of a small SAS switch may take anywhere from 30 minutes to an hour, while a more complex configuration for a larger switch could take several hours to a day or more. In our calculations, we assume a moderate level of complexity for the SAS switch and the configuration complexity. Thus, the average time assumed to configure a SAS switch based on our experience manually configuring the Fujitsu SAS switch on the Cloud Computing Lab at E-JUST is 1 hour per SAS switch.

$$\text{Average Total Time to Configure SAS Switches Manually} = \text{Time to Configure One SAS Switch} \times \text{Number of SAS Switches}$$

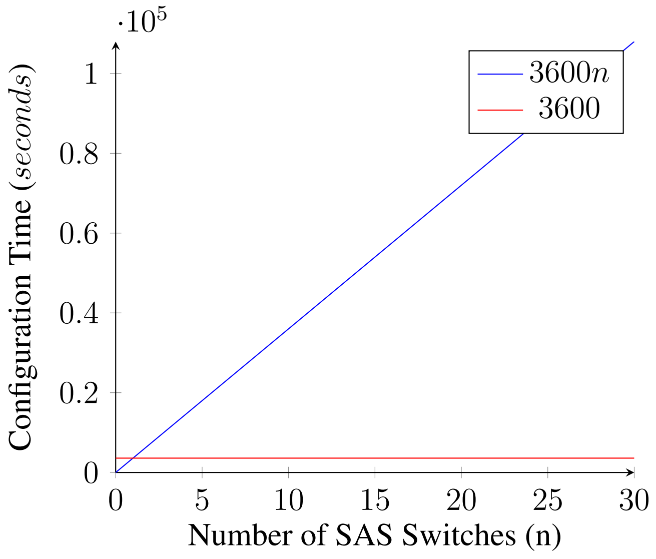
$$\text{Time to Configure One SAS Switch} = 1 \text{ hour}$$

$$\text{Average Total Time to Configure SAS Switches Manually} = 1 \text{ hour} \times 60 \text{ minutes/hour} \times 60 \text{ seconds/minute} \times \text{number of SAS switches}$$

The graph below depicts the time it takes to configure SAS switches as the number of switches increases.

In the manual case, the time required to configure each additional switch grows linearly, as depicted by the blue line reflecting $\mathcal{O}(n)$ time complexity. This means that the more switches that are added, the more time it takes to configure each one.

On the other hand, the automated case shows that the time required to configure each switch remains constant, as indicated by the red line reflecting $\mathcal{O}(1)$ time complexity. This provides strong evidence for the advantages of automation over manual processes in terms of time efficiency, especially when scaling to a large number of switches. Automated configuration of SAS switches leads to quicker and more consistent deployments, with reduced risks of human error.



Operating System Installation Time

The time it takes to manually install an operating system depends on various factors, such as the type of operating system, the hardware specifications of the machine, and the person's level of experience and familiarity with the installation process. On average, it can take anywhere from 30 minutes to a few hours to manually install an operating system. Thus, the average time assumed to install an operating system on a server based on our experience manually installing an operating system on a Fujitsu server in the Cloud Computing Lab at E-JUST is 30 minutes per server.

Average Total Time to Install Operating Systems Manually $=$

Time to Install an Operating System on One Server $\times$ Number of Servers

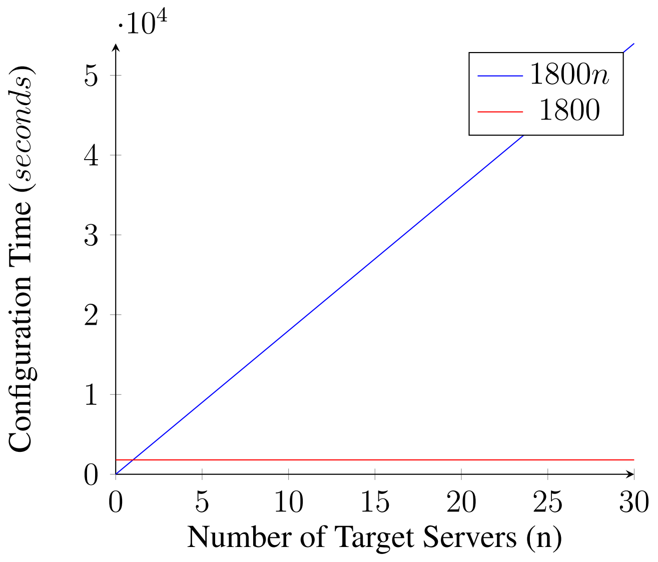
Time to Install an Operating System on One Server $= 30$minutes

Average Total Time to Install Operating Systems Manually $=$
30minutes $\times$ 30seconds/minute $\times$ Number of Servers
The graph below shows the time of installing an operating system on multiple servers for both cases; the manual configuration case and using the auto-generated configuration.

For the blue plot referring to the manual case, the time to install the operating system increases linearly as the number of servers increases, meaning that for each additional server, the time to install the operating system increases proportionally. This is a linear time complexity in terms of "Big O" notation O(n).

For the red plot referring to the automated case, however, the time to install the operating system remains constant, regardless of the number of servers. This is a constant time complexity, in terms of "Big O" notation O(1).

This graph highlights the efficiency of automation in reducing the time complexity of tasks, as compared to manual processes. Automation can significantly reduce the time and effort required to perform repetitive or complex tasks, which is especially important when scaling to large numbers of servers. The graph clearly illustrates the advantages of configuration auto-generation over manual processes in terms of time efficiency, and is a testament to the value of automation in reducing time complexity for repetitive and complex tasks.



4.2.2 Infrastructure Cost

Cost Objectives

1. **Reducing Cloud Deployment Costs**
   The cloud auto-generation system aims to reduce the costs associated with deploying and scaling cloud infrastructure. This is achieved through automation. The use of ready-to-use configuration templates in this project eliminates the dependency on manual configuration processes. Manual configuration processes can increase the cost of cloud deployment in several ways.
   First of all, manual configuration processes increase time and labor costs because configuring cloud resources manually can be time-consuming and require significant human effort, increasing the overall cost of deployment. Moreover, manual configuration processes can result in inefficient resource utilization. It can lead to sub-optimal configurations that result in inefficient use of resources, such as over-provisioning or under-utilization. For example, the administrator responsible for performing the manual configuration can mistakenly forget to associate a certain array of hard disks to any of the available servers by SAS.
   Furthermore, manual configurations can result in inconsistent configurations which can cause problems with compatibility, performance, and security. Manual configuration processes also increase the risk of errors as it increases the risk of human error, which can result in misconfigurations, security vulnerabilities, or other issues that can be costly to fix. By automating the cloud deployment process through the use of ready-to-use templates in the tool developed in this study, the risk of configuration errors due to human error is reduced. This leads to fewer mistakes and more consistent results, which can save time and resources in the long run.
   Moreover, manual configuration processes have slow deployment times as it takes longer to set up and configure resources manually than it does to automate the process.
   By automating cloud deployment processes and using the tool developed in this study, organizations can reduce the cost of cloud deployment and ensure that resources are deployed consistently and efficiently. The tool developed in this study reduces the risk of human error and speeds up the deployment process, enabling organizations to get up and running in the cloud faster and more cost-effectively. Using ready-to-use templates can streamline the cloud deployment

process, making it faster and more efficient. This can result in significant cost savings, especially for organizations that deploy large numbers of applications or services to the cloud. Moreover, the use of ready-to-use templates makes it easier to scale cloud deployments, as templates can be used to quickly deploy additional instances of applications or services. This can help organizations to respond more quickly to changing business needs, without incurring additional costs for manual configuration. By using ready-to-use templates, organizations can standardize their cloud deployments, which can help reduce the risk of compatibility issues and improve the overall quality of the deployment. This can result in cost savings by reducing the need for manual intervention, and by improving the stability and performance of the deployment. In conclusion, using ready-to-use templates in cloud deployment can help reduce costs by reducing human error, increasing efficiency, improving scalability, and promoting standardization. By automating the deployment process and ensuring consistency, organizations can achieve more reliable and cost-effective cloud deployments.

2. **Minimizing Cloud Cost Overruns**
   According to a study by Flexera, 86% of organizations have reported at least one cloud cost overrun. The cloud auto-generation system in this study aims to minimize these cost overruns by ensuring that cloud deployments are efficient, reliable, and free of errors. Cost overrun refers to a situation where the actual cost of a project exceeds the estimated cost. This can occur due to a variety of factors, such as unexpected changes in scope, delays in project completion, unforeseen challenges or problems, and inadequate cost-estimating methods. Cost overruns can have significant impacts on a project, such as delaying completion, reducing the budget available for other activities, and potentially leading to project cancellations.
   In this project, the use of ready-to-use templates can provide more accurate cost estimates for cloud deployments, as the templates can be pre-configured with the necessary resources and settings to deploy specific applications or services. This reduces the risk of cost overruns due to unexpected changes in scope or unanticipated resource requirements.
   Furthermore, the use of ready-to-use templates can provide a more flexible and scalable approach to cloud deployment, allowing organizations to optimize their costs by deploying resources only when needed and scaling up or down as required.
   In conclusion, the cloud auto-generation project can help minimize cloud cost

overruns by providing more accurate cost estimates, ensuring a consistent deployment process, improving resource utilization, and providing greater flexibility and scalability in the deployment process. By reducing the risk of cost overruns, organizations can achieve more predictable and cost-effective cloud deployments. Cloud infrastructure refers to the physical and virtual resources and services that support the delivery of cloud computing services. It includes the hardware, software, and networking components that are necessary to provide computing services over the internet. This can include data centers, servers, storage systems, networking equipment, and other components necessary to deliver cloud services. The cloud infrastructure is managed by a cloud service provider, who is responsible for maintaining and updating the infrastructure to ensure the availability and performance of cloud services. Cloud infrastructure can be used to deliver a wide range of services, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The use of cloud infrastructure enables organizations to reduce the cost and complexity of managing their own IT infrastructure while providing greater flexibility and scalability for their computing needs.

3. **Optimizing Cloud Infrastructure Costs**
   The cloud auto-generation system aims to optimize cloud infrastructure costs by creating ready-to-use templates that are tailored to the specific needs of an organization. This will enable organizations to deploy cloud infrastructure more quickly and efficiently, reducing the costs associated with cloud deployments. The use of ready-to-use templates can help organizations manage their cloud resources more effectively, as templates can be designed to match the specific requirements of each deployment. This can lead to improved resource utilization and cost optimization. Furthermore, the use of ready-to-use templates can also provide organizations with the ability to perform predictive cost optimization. This involves using data and analytics to predict future cloud infrastructure costs and adjusting and optimize those costs.
   In conclusion, the cloud auto-generation project can help optimize cloud infrastructure costs by improving resource optimization, improving resource management, automating monitoring and management, and providing the ability to perform predictive cost optimization. By optimizing their cloud infrastructure costs, organizations can achieve more cost-effective and efficient cloud deployments.
4. **Increase Cloud Cost Transparency**

The cloud auto-generation system should aim to increase transparency in cloud costs by providing detailed reports and insights into the cost of deploying and maintaining cloud infrastructure. This will enable organizations to make more informed decisions about their cloud deployments, reducing the risk of cloud cost overruns.

5. **Improve Cloud Cost Management**
   The cloud auto-generation system should aim to improve cloud cost management by providing organizations with tools and resources to monitor and manage their cloud infrastructure costs. This will enable organizations to optimize their cloud deployments and reduce the costs associated with cloud infrastructure.

4.2.3 Security

Following the standard of ISO 27001:

Security Risks and Assessment

1.1 Security Risks in the System The following assumptions should be taken into account when considering security risks in our system:

- Weak Passwords: This risk can arise from poor password management practices, such as using easily guessable passwords or failing to regularly update passwords.
- Insufficient Access Control in the GUI: This risk can occur when the graphical user interface (GUI) of the system does not properly restrict access to sensitive data or functions.
- Insufficient Access Control and Policies: This risk can arise when the system lacks proper access control mechanisms, such as role-based access control or two-factor authentication, to ensure that only authorized users can access sensitive data or perform critical actions.
- Configuration Errors: This risk can occur when the system is misconfigured, leading to errors or failures that can disrupt the operation of the system or expose sensitive data.
- Insecure APIs and Lack of Visibility: This risk can occur when APIs used by the system are not properly secured, allowing unauthorized access to sensitive data or functionality. Additionally, a lack of visibility into API usage can make

it difficult to detect and prevent malicious activity.

1.2 Risk Acceptance Criteria

- Weak Passwords: The acceptability of this risk will depend on several factors, such as the type of server being used and the services being provided (e.g. admin login for a MySQL server or oneadmin for an OpenNebula MySQL server).
- Insufficient Access Control and Policies/GUI: The acceptability of this risk will depend on the potential impact of a malicious actor outside of the system's trusted network.
- Configuration Errors: The acceptability of this risk can be measured by the number of times the system halts during the automation process and the severity of the errors that can stem from misconfiguration.
- Insecure APIs and Lack of Visibility: The acceptability of this risk can be evaluated by considering the potential impact of an unauthorized individual intercepting an API call and accessing sensitive parameters.

It is important to conduct regular and consistent security risk assessments to ensure that the results produced are valid and comparable. This will help to identify any potential security risks and to develop effective mitigation strategies.

1.3 Analyses of the Information Security Risks

- Weak Passwords: This risk is accepted as the systems and services in question are protected by an external barrier that cannot be breached from the outside.
- Insufficient Access Control and Policies/GUI: This risk is temporarily accepted as accessing the GUI without administrative or cloud architect authorization is not a concern, as the passwords for the servers would be required.
- Insufficient Access Control and Policies: This risk is not accepted as there is no two-factor authentication in place to prevent changes to policies or parameters. The solution is to implement user restrictions and create user groups.
- Insecure APIs and Lack of Visibility: This risk is not accepted as there is no encryption of data sent to the API, and the use of hardcoded secrets makes the system vulnerable. The solution is to add a layer of protection, such as TLS, to our API.
- Configuration Error: This is not an acceptable risk as the system has built-in handling for most expected errors and relies on templates to minimize human error.

Perform Vulnerability Scans

Vulnerability scans are automated assessments of an infrastructure that identify known vulnerabilities and weaknesses within the various components of the system, including the server manager, SAS, GUI, and PXE server. To ensure consistent, valid, and comparable results, regular security risk assessments should be performed.

One commonly used vulnerability scanner is Nessus, which can be used to test the security of a cloud deployment. The process of performing a Nessus scan involves the following steps:

1. Preparation: Ensure that we have the necessary credentials and permissions to access the cloud environment and perform a scan.
2. Scan Configuration: Configure the Nessus scan by specifying the target IP addresses or hostnames of the cloud servers, as well as the type of scan to be performed.
3. Scan Launch: Launch the scan by clicking the "Start Scan" button in the Nessus interface. The scan will then run and collect data on the security

posture of the cloud environment.

4. Scan Results: After the scan is completed, a report will be generated that summarizes the results and lists any vulnerabilities detected, along with recommendations for mitigation.
5. Mitigation: Based on the results of the Nessus scan, steps were taken to mitigate any detected vulnerabilities, such as patching software or implementing access controls, or performing extra manual testing on found weak points.

It is important to note that Nessus should be used as part of a broader security program that includes regular scans and ongoing monitoring of the cloud environment.

Another tool for testing the security of a cloud deployment is Nuclei, which is a fast and highly customizable scanner (Mostly checks known vulnerabilities or CVEs with certain behavior so a check after it is advised). The process of performing a Nuclei scan involves the following steps:

1. Preparation: Ensure we have the credentials and permissions to access the cloud environment and perform a scan.
2. Scan Configuration: Configure the Nuclei scan by specifying the cloud servers' target IP addresses or hostnames, as well as the type of scan to perform. In Nuclei, It is also applicable to specify custom rules to test specific vulnerabilities specific to the cloud environment.
3. Scan Launch: Launch the scan by using the CLI or web interface provided by Nuclei.
4. Scan Results: After the scan is completed, a report will be generated that summarizes the results and lists any vulnerabilities detected, along with recommendations for mitigation.
5. Mitigation: Based on the results of the Nuclei scan, take steps to mitigate any detected vulnerabilities, such as patching software or implementing access controls.

It is important to note that Nuclei should be used as part of a broader security program that includes regular scans and ongoing monitoring of the cloud environment, and in conjunction with other security tools and techniques to ensure a comprehensive view of the security posture of the cloud environment is obtained.

Conduct Penetration Testing

Penetration testing is a critical aspect of any system's security posture. It involves simulating an attack from an outsider or insider perspective to identify vulnerabilities, security weaknesses, and potential points of exploitation. This type of testing helps organizations understand the effectiveness of their current security measures and determine if additional measures are needed.

Performing a penetration test on the system can be done either by hiring a professional penetration tester or by conducting the test internally. In both cases, the testing process involves several key steps:

1. Preparation: Before conducting a penetration test, we should gather information about our system and its components, including the server manager, SAS, GUI, PXE server, and any other critical systems or applications. This information will be used to guide the testing process and help focus the effort on areas that are most likely to be vulnerable.
2. External Testing: In this phase, the tester will simulate an attack from an outsider perspective. This may involve attempting to penetrate the system from the internet or other public networks, exploiting known vulnerabilities in software or hardware, and performing social engineering attacks (Out of scope). The goal of this phase is to identify areas where an attacker may be able to gain unauthorized access to the system.
3. Internal Testing: In this phase, the tester will simulate an attack from an insider perspective, using techniques such as exploiting open ports, accessing panels or systems with weak authentication or access control, or attempting to gain access to sensitive information or secrets. The goal of this phase is to identify vulnerabilities that could be exploited by an attacker with physical or remote access to the system.
4. Results Analysis: After the penetration testing has been completed, the tester will analyze the results and provide a comprehensive report of findings. This report will include recommendations for mitigating vulnerabilities, improving security posture, and reducing the risk of future attacks.

It's important to note that penetration testing is just one aspect of a comprehensive security program. We should also implement regular vulnerability scans, maintain up-to-date software and hardware, and conduct ongoing security monitoring and incident response activities to ensure that our systems remain secure over time. Additionally, regular penetration testing should be performed to validate the effectiveness of security measures and identify

Evaluate the Security of the Deployment Pipeline

In addition to performing vulnerability scans on the cloud environment, it is equally important to review and secure the deployment pipeline. The deployment pipeline refers to the series of steps taken to deploy new changes to the infrastructure, and it plays a critical role in maintaining the security of the environment.

During the review process, it is important to examine the security measures in place for the pipeline, including access controls, code signing, and other security measures. Access controls ensure that only authorized individuals have access to the pipeline, while code signing verifies the authenticity and integrity of the code being deployed on the servers. This helps to prevent unauthorized access and the deployment of malicious code.

It's also a good idea to implement security practices such as using encryption in each module of our system to protect sensitive data during the deployment process, and regularly auditing the pipeline to identify any potential vulnerabilities. The deployment pipeline should be designed with security in mind and tested regularly to ensure that it is secure and reliable.

Finally, it's important to ensure that the pipeline is automated and streamlined to minimize the risk of human error, which can introduce vulnerabilities into the environment. By incorporating security into the deployment pipeline, It is applicable to ensure that the infrastructure remains secure and protected from potential threats.

Test the Disaster Recovery Plan

To make sure that it is effective and can be executed quickly and efficiently. A disaster recovery plan is a critical component of any cloud deployment, as it helps to minimize the impact of a security breach or other emergency. The plan should include procedures for backup and restore of data, redundancy of critical systems, and disaster recovery scenarios.

When testing the disaster recovery plan, it is important to consider the impact of various scenarios, such as the failure of a single server or the loss of an entire data center. This may involve simulating the failure of critical systems and assessing the effectiveness of the disaster recovery plan. The goal is to identify any potential weaknesses in the plan and to ensure that it can be executed quickly and efficiently in the event of a real emergency.

In the case of a cloud deployment with SAS GUI and PXE, it is important to test the disaster recovery plan to ensure that the SAS GUI and PXE servers can be quickly restored in the event of a failure restoring the environment, boot images, and network. This may include testing the backup and restore procedures, as well as the redundancy of the critical systems. Additionally, it may be necessary to test the disaster recovery plan in a live environment to ensure that it can be executed efficiently and effectively.

Ultimately, the disaster recovery plan should be reviewed and updated regularly to ensure that it remains effective and relevant. This can help to minimize the impact of a security breach or other emergency and help to protect the integrity of the cloud deployment with SAS GUI and PXE.

4.2.4 Scalability

Scalability Requirements: When designing a cloud system, it's crucial to consider several key requirements to ensure success, including:

- Elastic capacity
- Disaster recovery
- Security
- Cost optimization

Use Case 1: Comparing a Private Cloud with 10 Servers

A private cloud with 10 servers can be evaluated using traditional approaches and our system to determine which one meets all requirements. Our system offers greater flexibility, as the administrator can make updates to the design at any time during deployment. Additionally, our system has a strong focus on disaster recovery and provides the same level of security as traditional approaches. Furthermore, our system offers improved cost optimization during scalability, as outlined in the speed section.

Use Case 2: Evaluating OpenNebula Performance

To evaluate the scalability of OpenNebula as part of our system, a load testing scenario is set up. The scenario will simulate multiple user requests and measure OpenNebula's performance. A testing environment that closely resembles the production environment is created, including software versions, network settings, and storage options. Load testing tools, such as Apache JMeter, will be used to simulate user requests, and built-in monitoring tools, such as the OpenNebula Sunstone web interface, will track performance metrics like response time, CPU utilization, and memory usage.

After the load testing, the performance results are evaluated, and any signs of degradation, bottlenecks, or increased error rates are identified. If any issues are discovered, the data gathered during the testing process can be used to troubleshoot and resolve the problem. If necessary, the OpenNebula system can be scaled up by adding more resources or adjusting configurations, such as adding or removing instances or allocating resources differently. The testing process should be repeated regularly to ensure optimal performance as the load increases over time.

Use Case 3: Security Breach Emergency

In the event of a security breach or other emergency, it's important to have a disaster recovery plan in place. This step involves testing the disaster recovery plan to ensure that it works as intended in the event of an actual disaster. To evaluate the effectiveness of the disaster recovery plan for OpenNebula, a simulated disaster scenario is created, such as a data center outage or a network failure. The disaster recovery plan is then put into action and monitored to see how well it performs. Any issues or potential improvements can be identified and addressed during the testing process. This helps to ensure that the disaster recovery plan is robust and effective and that the OpenNebula system can be recovered quickly and effectively in the event of a disaster.

By conducting regular load testing and disaster recovery plan testing, organizations can ensure that their OpenNebula system meets all scalability requirements and continues to perform optimally, even as the load increases over time. This helps organizations to be prepared for any eventualities, and to have a system that is both scalable and resilient.

4.2.5 Deployment Success Rate

The success rate of cloud auto-deployment is a crucial aspect of cloud computing, as it directly affects the efficiency, scalability, and cost-effectiveness of the deployment.

By using an automation-based template, the deployment process becomes consistent and predictable, reducing the chances of failure. Measuring the deployment success rate in cloud auto-deployment is crucial to ensure that the system is functioning as expected and that the goals of the deployment have been achieved. There are several ways to measure the deployment success rate, including:

1. Monitoring System Availability and Performance: In the case of an OpenNebula cloud, the OpenNebula Sunstone web interface can be used to monitor the availability and performance of the system. This interface provides real-time information about the status of the cloud and the various resources, such as the virtual machines, networks, and storage, that are part of the system. By monitoring the system in this way, Identifying any issues could be quickly and taking the appropriate action to resolve them. Nagios can be used to monitor the CPU utilization, memory usage, and network traffic of the OpenNebula cloud and send alerting if any thresholds are exceeded.
2. User Satisfaction Surveys: at the end of the GUI there is a link of a form to ask users to provide feedback on the performance of the system and the deployment process.
3. Service Level Agreement (SLA) Compliance: As the user entered the specifications exactly in the system GUI, to make sure the system achieved all the required points.

5. Conclusion and Future Plan

5.1 Conclusion

The cloud computing industry is rapidly growing and presents a great opportunity for organizations to improve their competitiveness and reduce costs. However, organizations are facing challenges in efficiently deploying and scaling their cloud infrastructure, particularly due to configuration errors and human error. The proposed cloud auto-generation project addresses these challenges by automating the cloud deployment process through the creation of ready-to-use templates that are free of configuration errors.

The results of this study demonstrate the potential impact that this system could have on the cloud computing industry, reducing costs, improving the speed and efficiency of cloud deployments, and increasing the reliability and security of cloud infrastructure. As the cloud computing market continues to grow, it is expected that the demand for efficient and reliable cloud deployment solutions will also increase. The cloud auto-generation system presented in this thesis has the potential to meet this demand and make a significant contribution to the cloud computing industry. The project achieves these needs by automating the operating system installation process and the SAS configuration process, thus reducing the potential for human error and increasing the efficiency of the deployment process. Our cloud auto-generation system connects to the target bare-metal servers via our connection module and uses kickstart to automate the operating system installation process, including partitioning and RAID creation. The system also tackles the SAS configuration processes, giving users the option to choose entire system templates or customize their own templates. This customization process can also make use of finer-grained templates for the operating system installation or the SAS configuration step.

In conclusion, the cloud auto-generation system presented in this thesis provides a solution to the problems of configuration errors and manual deployment processes in cloud computing. The system offers significant benefits to organizations and has the potential to become an essential tool in the cloud computing industry.

5.2 Future Plan

The project is not without limitations and there is still room for improvement. In the future, the following work can be done to further enhance the performance and functionality of the cloud auto-generation system:

1. Making the system more general and integrability as being integrable with cloud management platforms such as OpenStack and Amazon Web Services (AWS), and cloud orchestration tools such as Ansible, Puppet, and Chef to expand the deployment capabilities and automate the entire cloud deployment process end-to-end.
2. Adding security features to the system to ensure the privacy and protection of user data during the deployment process.
3. Improving the user interface to make it easier for users to customize templates and manage their cloud deployments.
4. Developing a rollback mechanism to undo changes made during the deployment process in case of any errors.
5. Implementing automatic testing and validation of templates to ensure that they are error-free and ready to use and making room for CI/CD.
6. Introducing version control for templates to allow users to keep track of changes made over time and revert to older versions if needed.
7. Increase the stability of the process and make it less error-prone
8. Making the process available remotely
9. providing support for more hardware infrastructure.

These are just some of the potential directions for future work, and we believe that the cloud auto-generation system has the potential to become a leading tool for cloud deployment and management.

Appendix

Hardware

Rx2540 M4

The FUJITSU Server PRIMERGY RX2540 M4 is the new 2U dual socket rack server for high usability, scalability and cost-efficiency. Its Intel® Xeon® Processor Scalable Family CPUs in conjunction with DDR4 memory technology with up to 3TB increases performance to meet the requirements for data center processing, enterprise applications as well as collaboration. The modular design offers excellent expandability of up to 28 disk drives, up to 8 PCIe Gen3 expansion cards, and best-in-class energy efficiency thanks to two hot-plug power supplies with up to 96% efficiency to meet the future demands of data growth. To complete the picture, DynamicLoM makes network connections more flexible and ready for future modifications. Operation in higher ambient temperatures is ensured by the Cool-safe® Advanced Thermal Design resulting in lower OPEX.

Feature:

- Versatile Performance for any computing need
  - Intel® Xeon® Processor Scalable Family CPUs with up to 28 cores relying on Intel® UltraPath Interconnect for an increased data rate between the CPUs.
  - Up to 3,072 GB DDR4 memory with 2,666 MT/s (24 DIMM slots).
  - 8x PCIe Gen3 slots.
- Enhanced Features for enhanced Computing
  - Onboard LAN via OCP for basic LAN, DynamicLoM for extended requirements.
  - Mix&Match storage drive bays: Ideal scalability of either up to 12x 3.5-inch or up to 24x 2.5-inch HDD/SSD/PCIe SSD+ an additional rear option of 4x 2.5-inch drives.
  - 2x internal M.2 devices support for hypervisor installations or mirroring.

  - Power supply units with 96% energy efficiency.
  - Fujitsu’s Cool-safe® Advanced Thermal Design for higher ambient temperatures in the data center.
  - Optional liquid-cooled base unit (on special request).
  - Up to 2x GPGPU support within one system.
- Foundation for Trust and Security
  - BIOS, firmware, and selected software are updated free of charge.
  - TPM2.0 modules and latest operating system support.
- Simplified management
  - iRMC S5 comes with a new interactive web UI and conforms to Redfish providing unified API support for heterogeneous environments.
  - RAID Controller embedded onboard
- Benefits
  - Ready for the future and data growth scenarios with the performance of two processors – marking the standard of tomorrow with an increase in computing power.
  - DDR4 memories with higher bandwidth and lower consumption are the enabler; optimized for virtualization and clouds, data centers and high performance computing.
  - Flexible expandability and diverse options for storage devices permits for the integration of existing and new SSD and HDD as needed. Less today, more in future – or vice versa
  - The right Ethernet connection for all: Basic via onboard LAN, extended with DynamicLoM guarantees the highest flexibility to integrate the server into existing infrastructures – now and in future without overhauling the existing infrastructure.
  - Flexible expandability and diverse options for storage devices permits for the integration of existing and new SSD and HDD as needed. Less today, more in future – or vice versa.
  - Not only “greener”, also less expensive over time: Highly efficient hot-plug power supplies save energy costs and make it easy to maintain the running system and ensure industry-leading uptime.
  - Higher ambient temperatures lead to lower costs for cooling the data center.
  - Less noise, latest technology to cool processors and memory directly where the heat is being generated.

- Optimal for VDI, CAD or future technologies such as Artificial Intelligence of Virtual Reality applications.
- Lifecycle investment protection.
- The comprehensive tools of the Fujitsu ServerView Suite eases the administrators life.
- Hardware and Software driven security features are very important in a fast-paced world, especially considering cybercrime.
- Optimized for both: data centers and SMEs can now rely on latest generation iRMC S5 increasing security and server admin productivity.
- RAID support for the most common configurations is conveniently embedded on the system board and does not require a dedicated controller.

Figure 11. *RX2540 M4 Server*

CX2570 M4

FUJITSU Server PRIMERGY CX scale-out systems are built for cloud computing scenarios, high-performance computing, service providers, and large server farms. They focus on providing large data centers with massive scale-out x86 server power while at the same time delivering the best economics for server density, energy consumption, heat optimization, and lower overall operational costs.

The FUJITSU Server PRIMERGY CX400 M4 is a modular enclosure for the Fujitsu multi-node ecosystem that combines the density and efficiency of blade-like servers with the simplicity and cost benefits of rack-based systems. It helps to provide a scalable IT infrastructure enabling flexible and fast responses to rapidly changing IT demands. The system with only 2U holds hot-plug and redundant components and can be equipped with up to four server nodes featuring the latest generation Intel® Xeon® processors and Intel® Optane™ DC persistent memory technology to best match particular workloads.

CX2560 M4

The FUJITSU Server PRIMERGY CX2560 M4 is designed to be a workhorse for data centers looking for new levels of efficiency and density in an outstanding compact form factor. The CX2560 M4 is well-suited for mainstream enterprise workloads, web serving, dedicated hosting, infrastructure virtualization as well as analytics thanks to the high performance of the new Intel® Xeon® Processor Scalable Family with up to 28 cores and the latest DDR4 technology supporting up to 2,048 GB of main memory. The server node is prepared for individual future demands by offering various modular options. In addition to the basic onboard LAN, the node provides the option of using the DynamicLoM technology as well as two additional PCI Express® (PCIe) expansion slots. The CX2560 M4 nodes are housed in the PRIMERGY CX400 M4, a 2U modular chassis that delivers the density and efficiency of bladelike servers with the simplicity and cost benefits of rack-based systems. The CX400 M4 delivers efficiency through shared power, cooling, and management.[15]

Figure 12. *CX2560 M4 Server*

BX900 S2

The Fujitsu Server PRIMERGY BX900 S2 is the accelerated dynamic server infrastructure in a single cube. This blade server can be dynamically adapted to various IT requirements and provides significant economic advantages for a large and growing number of applications. The PRIMERGY BX900 S2 has space for up to 18 server and storage blades in a 10U chassis. Thus it's the leader in its class for density in a compact form factor. Fujitsu's Cool-safe™ cooling concept, combined with power supply units certified with 80Plus Platinum and holistic power management, reduce your costs and ensures more efficient use of energy and cooling capacity. Centralized management of physical and virtualized environments and comprehensive I/O virtualization capabilities, combined with a fully redundant system design, supports business agility. Furthermore, the PRIMERGY BX900 S2 provides future-proof connectivity enhancements with the support of high-speed InfiniBand (FDR InfiniBand) with 56 Gbit/s bandwidth. Server blades, equipped with CPUs from the next generation Intel® Xeon® processor E5-2400v2 / E5-2600v3 product family, offer scalable performance to meet the highest requirements of extensive virtualization/ consolidation scenarios for business-critical applications on the one hand and demanding high-performance computing applications on the other hand. Packed with advanced features, up to 24 high-speed DIMM slots, and a flexible 10Gbit/s Universal Converged Network Adapter (UCNA) on board, the new PRIMERGY server blades provide the highest memory modules density in a dual-socket blade and allow doing more with a two processor server than ever before.

BX2560 M2

The FUJITSU Server PRIMERGY BX2560 M2 is a dual-socket server blade that further extends the capabilities of the Fujitsu blade ecosystem and provides improved performance, especially on large data sets. It addresses a broad set of workloads, from IT and web infrastructures up to high-performance computing applications. The PRIMERGY BX2560 M2 harnesses the power of the latest Intel® Xeon® processor E5-2600 v4 product family, with up to 1024 GB of RAM (16 DIMM slots), two hot-plug disk drives (HDD/SSD/ PCIe SSD), and two slots for additional mezzanine cards. The processor features up to 22 cores per CPU and uses DDR4 memory technology that runs at speeds of up to 2400 MHz. The system also includes an Emulex-based dual port 10 Gigabit Ethernet, Data Center Bridging

Figure 13. *BX900 S2 Server*

(DCB)-capable LAN on motherboard (LOM). This LOM solution enables an agile server infrastructure that can be partitioned into up to eight PCIe functions, partially configurable to NIC, iSCSI, or FCoE personalities with offload features, and present up to 128 MAC/VLAN addresses to the host that can be dynamically configured.

Figure 14. *BX2560 M2 Server*

OpenNebula installtion code

```
curl 'http://192.168.1.104:5000/?param=secret'
rm -rf /etc/systemd/system/getty.target.wants/getty\@tty1.service
```

```
cp /lib/systemd/system/getty\@.service /etc/systemd/system/getty\
    @tty1.service
sed -i "s/ExecStart.*$/ExecStart=-\/sbin\/agetty --autologin root
    --noclear %I/g" /etc/systemd/system/getty\@tty1.service
echo ";Alias=getty@tty1.service" >> /etc/systemd/system/getty\
    @tty1.service
ln -s /etc/systemd/system/getty\@tty1.service /etc/systemd/system/
    getty.target.wants/getty\@tty1.service
touch /etc/init.d/srwr.sh
cat <<EOT>/etc/init.d/srwr.sh
#! /bin/sh
# chkconfig: 345 99 10
case "\$1" in
  start)
    sh /root/msg.sh
    ;;
  *)
    ;;
esac

exit 0
EOT
chmod +x /etc/init.d/srwr.sh
chkconfig --add /etc/init.d/srwr.sh
cat << EOT > /etc/yum.repos.d/opennebula.repo
[opennebula]
name=opennebula
baseurl=https://downloads.opennebula.io/repo/6.4/CentOS/7/x86_64
enabled=1
gpgkey=https://downloads.opennebula.io/repo/repo2.key
gpgcheck=1
repo_gpgcheck=1
EOT
touch /root/msg.sh
cat << EOT > /root/msg.sh
sed -i 's/^SELINUX=.*/SELINUX=disabled/' /etc/selinux/config
yum -y install epel-release
yum -y update
yum -y install mariadb-server mariadb
yum -y install centos-release-scl-rh
cat <<EOA >/root/bla
\#!/bin/bash
gpg --keyserver keyserver.ubuntu.com --recv-key
```

```
\curl -sSL https://get.rvm.io | bash
source /usr/local/rvm/scripts/rvm
rvm install 2.6.3
export PATH="/usr/local/rvm/rubies/ruby-2.6.3/bin:$PATH"
EOA
chmod 755 /root/bla
bash /root/bla
systemctl start firewalld
firewall-cmd --state
firewall-cmd --add-port=9869/tcp --permanent
firewall-cmd --add-port=2633/tcp --permanent
firewall-cmd --reload
yum -y install opennebula opennebula-sunstone opennebula-fireedge
    opennebula-gate opennebula-flow opennebula-provision
systemctl start mariadb
systemctl enable mariadb
mysql_secure_installation <<EOF

Y
qwe
qwe
Y
n
n
Y
EOF
kill -9 \$(ps aux|grep "/bin/mysqld" |head -n 1|cut -d " " -f 6)
mysqld_safe --skip-grant-tables --skip-networking &
a
mysql --user="root" --password="qwe" --execute="FLUSH PRIVILEGES;
    SET PASSWORD FOR 'root'@'localhost' = PASSWORD('qwe');FLUSH
    PRIVILEGES;"
systemctl restart mariadb
mysql --user="root" --password="qwe" --execute="CREATE DATABASE
    opennebula;CREATE USER 'oneadmin' IDENTIFIED BY 'qwe';GRANT ALL
     PRIVILEGES ON opennebula.* TO 'oneadmin';"
mysql --user="root" --password="qwe" --database="opennebula" --
    execute="CREATE USER 'oneadmin' IDENTIFIED BY 'qwe'; GRANT ALL
    PRIVILEGES ON opennebula.* TO 'oneadmin';"
sed -i 's/DB.*sqlite/#&/' /etc/one/oned.conf
sed -i ':a;N;/DB.*sqlite/!s/^\s\{3,\}TIME.*$/#&/;ta;P;D' /etc/one/
    oned.conf
cat << EOF >> /etc/one/oned.conf
```

```
DB = [ BACKEND = "mysql",
 SERVER = "localhost",
 PORT = 0,
 USER = "oneadmin",
 PASSWD = "qwe",
 DB_NAME = "opennebula",
 CONNECTIONS = 25,
 COMPARE_BINARY = "no"]
EOF
systemctl start opennebula opennebula-sunstone opennebula-fireedge
     opennebula-gate opennebula-flow
systemctl enable opennebula opennebula-sunstone opennebula-
    fireedge opennebula-gate opennebula-flow
firewall-cmd --add-port=9869/tcp --permanent
chkconfig --del /etc/init.d/srwr.sh
EOT
chkconfig --add /etc/init.d/srwr.sh
chkconfig --add /etc/init.d/srwr.sh
```

Bibliography

[1] "Cloud computing market by component (solutions and services), service model (infrastructure as a service, platform as a service, and software as a service), deployment model, organization size, vertical, and region - global forecast to 2023," https://ResearchAndMarkets.com, published: 2019.

[2] T. XU and Y. ZHOU, "Systems approaches to tackling configuration errors: A survey," University of California San Diego, 2022, accessed: 2022-08-01.

[3] I. (n.d.)., "Manual deployment," IBM, 2022.

[4] "Cost reduction in cloud computing," https://Gartner.com, published: 2018.

[5] "Additional installation methods," https://www.ibm.com/docs/en/linux-on-systems?topic=servers-additional-installation-methods, accessed: 2022-07-05.

[6] I. (n.d.)., "Deployment automation basics," https://www.ibm.com/downloads/cas/WVPED3LY, IBM, pp. 1–8, 2021.

[7] "The notorious nine cloud computing top threats in 2013," https://CloudSecurityAlliance.org, published: 2013.

[8] "Ibm global technology outlook: Cloud," https://IBM.com, published: 2017.

[9] B. I. et al. (2016), "Implementing a cloud-based decision support system in a private cloud: The infrastructure and the deployment process, international journal of decision support system technology," https://www.igi-global.com/article/implementing-a-cloud-based-decision-support-system-in-a-private-cloud/148625, accessed: 2022-07-17.

[10] K. M. Diab, M. M. Rafique, and M. Hefeeda, "Dynamic sharing of gpus in cloud systems," in *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, 2013, pp. 947–954.

[11] ——, "Dynamic sharing of gpus in cloud systems," in *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, 2013, pp. 947–954.

[12] "Centos and scientific linux cern installation with fai," https://lists.uni-koeln.de/pipermail/linux-fai/2011-September/009243.html, accessed: 2022-07-17.

[13] I. e. a. . Benatia, "Installing linux os over network in area of using kickstart," https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4089437, accessed: 2022-07-17.

[14] "What is wan: Wide area network definition: Computer networks," https://www.comptia.org/content/guides/what-is-a-wide-area-network, accessed: 2023-02-11.

[15] "Fujitsu primergy cx2560 m4 manuals," https://www.manualslib.com/products/Fujitsu-Primergy-Cx2560-M4-9011731.html, accessed: 2022-06-25.