National College of Ireland

Enabling Automated Multi-cloud Migration For SMEs With A Standard Three-Tier Infrastructure

MSc Research Project
Cloud Computing

Achyut Gawade
Student ID: 22103228

School of Computing
National College of Ireland

Supervisor: Prof. Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Achyut Gawade
Student ID:	22103228
Programme:	Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Prof. Sean Heeney
Submission Due Date:	14/12/2023
Project Title:	Enabling Automated Multi-cloud Migration For SMEs With A Standard Three-Tier Infrastructure
Word Count:	XXX
Page Count:	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	14th December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Enabling Automated Multi-cloud Migration For SMEs With A Standard Three-Tier Infrastructure

Achyut Gawade
22103228

Abstract

The proposed solution is a practical middleware-based approach for seamless multi-cloud migration and portability of standard three-tier infrastructure, including virtual machines, databases, and object storage. The proposed system utilizes Python SDKs and APIs provided by the major public cloud providers (AWS, Azure, and GCP) to achieve resource discovery, VM migration, database migration, and object storage migration. The feasibility of heterogeneous VM migration across different hypervisors (XEN, Hyper-V, and KVM) and various disk formats (AMI, VHD, and RAW) has been evaluated and demonstrated throughout the study. A hybrid database migration approach combining snapshot and streaming data has been implemented to optimize data transfer and minimize downtime. Object storage migration has been streamlined using multi-part uploads for faster upload speeds and ARIA2 for parallel downloads, particularly for large files. The proposed solution has been successfully tested for migrating a sample Ubuntu 22.04.6 LTS virtual machine with 30 GB of disk space, a MySQL database with various object types and millions of records, and storage objects across different cloud providers, demonstrating its ability to achieve seamless multi-cloud interoperability. The design and evaluation of the tool have been performed considering the requirements of SMEs. The proposed solution can be used by SMEs to effectively utilize the benefits of multi-cloud computing while mitigating the risks associated with vendor lock-in and limited portability. In summary, the proposed solution addresses the key challenges of multi-cloud migration and portability, enabling SMEs to adopt a more flexible and cost-effective cloud strategy.

1 Introduction

A multi-cloud system refers to an approach where an organization uses services from multiple cloud service providers (CSPs) to meet its computing needs. Instead of relying on a single cloud provider, a multi-cloud strategy involves distributing workloads and applications across different cloud platforms. This approach aims to leverage the strengths of each provider, enhance redundancy, mitigate risks, and avoid vendor lock-in. Multi-cloud migration systems are essential for organizations looking to obtain benefits such as scalability, pay-per-use, and increased availability. A well-planned and executed multi-cloud migration can lead to improved efficiency, cost savings, and flexibility for businesses. Some of the common challenges in multi-cloud migration systems include data management, security, and integration with existing systems Success factors include a well-defined migration strategy, proper planning, and the selection of suitable migration

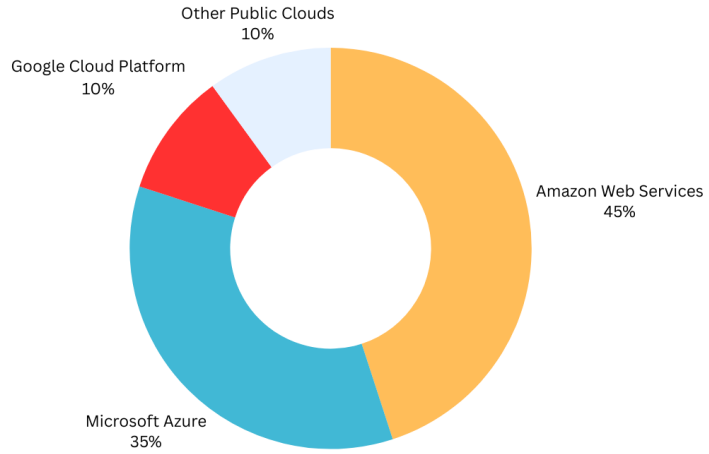


Figure 1: EU Public Cloud Market Share Based On Revenues In IaaS And PaaS Only

service providers. A recent economic analysis of cloud services in Europe Joshua Gans and Masri (2023) claims that Amazon Web Services (AWS) and Microsoft Azure hold up 80% of the market share combined, with Google Cloud Platform (GCP) holding the third-largest market share of around 10% is the primary concern for cloud market regulation. Due to this high concentration of market share, high switching costs, "data transfer-out" or "DTO" fees, and a lack of interoperability between services are emerging, resulting in vendor lock-in. Hence, it is extremely important to improve interoperability and migration between these three major players to mitigate vendor lock-in risks. The next section briefly highlights the prime complications of asserting seamless multi-cloud strategies.

- Complex compatibility issues: Migrating from a complex existing cloud infrastructure to a new cloud-based infrastructure can be difficult, as interdependencies and relationships between systems need to be identified and documented. The system and services provided by each cloud provider are based on proprietary formats, hardware, middleware, and software modules, which do not directly support services from different cloud providers. Hence, it is the primary challenge to ensure compatibility between source and destination clouds, leading to potential issues during the migration process.

- Long migration process and downtime: Manual cloud migrations often involve multiple stages with in-depth testing and validation between each stage, making the process time-consuming and prone to delays. Migrating applications and services from one cloud to another can result in downtime. Minimizing downtime during migration is crucial, especially for mission-critical systems.

- High cloud migration costs: The cost of migrating to the cloud, including data migration, application migration, and ongoing operations, can be a significant concern for organizations

- Data security and compliance: Ensuring data security and compliance with the migration process is essential, especially in cases where third-party service providers

are involved in the process. Inadequate security measures can expose sensitive information during the migration process.

- Skill gap: Cloud migration requires a diverse set of skills and knowledge in areas like cloud deployments, data migration, and application migration, which can be challenging to find and retain within the organization.

The above challenges associated with the cloud migration process often result in vendor lock-in, which is a situation where a customer becomes heavily dependent on a particular vendor's products or services to the extent that switching to an alternative vendor becomes difficult, expensive, or impractical. This dependency can arise due to proprietary technologies, formats, or standards used by the vendor, making it challenging for the customer to migrate to a different solution without significant effort, cost, or disruption. Small and medium enterprises (SMEs) face challenges with vendor lock-in in cloud computing due to limited resources, dependence on provider ecosystems, cost implications, a lack of negotiating power, limited expertise, long-term commitments, data migration challenges, and vendor-specific skills. These challenges can hinder SMEs' ability to switch between cloud providers efficiently, impacting their flexibility and adaptability in the rapidly evolving cloud landscape.

Research Question : How to feasibly migrate standard three tier infrastructure comprising of compute, storage and database between current three major public cloud service providers namely AWS, Azure and GCP to promote interoperability and mitigate vendor lock-in problem for SMEs ?

2 Related Work

The multi-cloud standardisation efforts started with the survey Kaur et al. (2017), which critically assesses the adoption of standards such as OCCI, SAML, and IPv6 in addressing interoperability challenges within interconnected clouds. Despite the potential benefits, the limited embrace of these standards by global providers is underscored as a significant drawback. It emphasizes the imperative of extending support across various service models (IaaS, PaaS, and SaaS) and encourages the exploration of additional technologies such as OpenFlow and NOX for inter-cloud networking. The analysis urges a broader adoption of diverse standards and technologies to effectively tackle the complexities of interconnected cloud environments. followed by a study by Tomarchio et al. (2020) that evaluated existing cloud resource orchestration frameworks (CROFs) like Terraform, CloudFormation, Brooklyn, Cloudify, and Heat, finding that they rely on diverse open standards and abstraction libraries such as jclouds. The research concludes that the prevalence of open standards primarily in academic projects for cloud interoperability, like the Open Cloud Computing Interface (OCCI), the Cloud Data Management Interface (CDMI), and the Topology and Orchestration Specification for Cloud Application (TOSCA) by OASIS, is not widely adopted in the architectures of commercialized cloud providers.

In their review of modern interoperability in multi-cloud systems, Caceres and Globa (2022) observed that many existing tools predominantly utilize low-level APIs and mechanisms for migrating infrastructure-as-a-service (IaaS). However, they noted a lack of interoperability for most high-level services, primarily due to the absence of standards. The research concludes by suggesting that standardization of higher-level services can be

achieved through the HTTP protocol, leveraging APIs provided by public cloud vendors. The proposed approach involves a top-to-bottom strategy to ensure effective standardization and interoperability in multi-cloud environments.

Comprehensive research by Alonso et al. (2023) highlighted the challenges of multi-cloud native applications, emphasising that the heterogeneity of cloud service providers' (CSPs) architecture is a major contributor to vendor lock-ins. The absence of standard protocols and specifications compounds this issue, with a notable lack of efforts to establish standardisation across CSPs. The study mentions a few standards set by the industry, such as OASIS CAMP, OASIS TOSCA, NIST, OAuth, CSA STAR Program, and SAML/2, for the integration and security of heterogeneous clouds. However, it lacks a specific focus on multi-cloud migration and portability in major public clouds such as AWS, Azure, and GCP and provides limited discussion on the technologies, protocols, and algorithms.

This highlights a gap in the literature, as there is inadequate coverage of multi-cloud migration and portability in AWS, Azure, and GCP, and insufficient exploration of the specific technologies, protocols, and algorithms relevant to this context. To address these gaps, further research questions need to be formulated to study the impact of different technologies, protocols, and algorithms on the efficiency and success of multi-cloud migration and portability and the specific requirements for achieving seamless multi-cloud migration and portability across these major public clouds.

2.1 Cloud Resource Discovery

There are several tools available for cloud resource discovery, each with its own features and capabilities. The popularity of these tools varies based on specific use cases, preferences, and the cloud platform being used. AWS Config Piper and Clinton (2023), Azure Resource Graph [1], Google Cloud Asset Inventory [2], Terraform Brikman (2019), and Hashicorp Consul are a few of the popular options extensively used in the industry. Of which AWS Config, Azure Resource Graph, and Google Cloud Asset Inventory only work for their own resources.

In most research, Terraform is used as a resource provisioning IaaC tool. For example, in research conducted by Bahaweres and Muhammad Najib (2023) and Liu et al. (2022), Terraform is used to mitigate disaster recovery scenarios as it provides a consistent workflow for provisioning and managing resources, regardless of the underlying cloud provider. Few more studies by Bhalla et al. (2023) and Gupta et al. (2021) utilized Terraform for cloud Hadoop cluster provisioning and automation.

In the research paper de Carvalho and Patricia Favacho de Araujo (2020), compare the performance of Terraform and Cloudify as prominent multi-cloud orchestrators. Through practical experiments, it was observed that Terraform surpassed all other orchestrator frameworks, with notable superiority over Cloudify.

From all this research, it is evident that Terraform is only looked upon as a multi-cloud orchestrator. While it's primarily used for infrastructure provisioning, its state files can be used for discovering existing resources. This shows a gap in existing studies, which will be explored in this research.

[1] `https://azure.microsoft.com/en-us/get-started/azure-portal/resource-graph`

[2] `https://cloud.google.com/asset-inventory`

2.2 Virtual Machine Migration

An initial effort in heterogeneous cloud VM migration, as outlined by Kargatzis et al. (2017), established the foundation for recognizing diverse virtualization formats employed by various CSPs such as OpenStack and VMWare. This includes hypervisors, disk formats, container packaging formats, and various supported migration types within the context of multi-cloud VM migration. A similar study on heterogeneous VM migration by Raj et al. (2020) analyzed the migration between OpenStack and VMWare using a volume-based technique and an image-based technique. Both of these studies employed a middleware-based approach for implementing a script that will download the image from the source cloud, convert it if needed, and then upload and deploy it on the destination cloud. These studies lack a wider migration experimentation and feasibility study on AWS, GCP, and Azure, which are the major players in the cloud industry.

There has been plenty of research on task scheduling for multi-cloud computing and live VM migration techniques, considering security and reliability constraints. The research conducted by Zhu et al. (2021) focused on a multi-round allocation algorithm and discussed the challenges of resource scheduling to optimise task execution time and total cost in a heterogenous multi-cloud system with the proposed Matching and Multi-Round Allocation (MMA) algorithm that uses the variance of the estimated completion time of tasks on resources as a metric to schedule tasks. Whereas Choudhary et al. (2017) briefly discusses various live VM migration techniques, for example, pre-copy and post-copy approaches. While the research paper provides valuable insights into task scheduling in a multi-cloud environment and live VM migration with optimisation techniques, it does not specifically focus on the migration and portability aspects of public clouds like AWS, Azure, and GCP.

Anglano et al. (2020) introduced a toolkit designed to facilitate the creation and utilization of multi-cloud systems (MSs) in both cloud and edge environments. The study emphasizes the significance of interoperability among diverse heterogeneous clouds to mitigate the risk of vendor lock-in. The toolkit, named "EasyCloud," is Python-based and encompasses three primary subsystems: VM management, VM monitoring, and a user interface for overseeing multi-cloud operations. In successive research Anglano et al. (2021), "EasyCloud" underwent enhancements to include additional monitoring capabilities with diverse sink options. While the focus of this research centers on virtual machine migration and monitoring, it falls short in executing stateful migration of VMs across different CSPs.

Addya et al. (2023), presented a study on VM coalition for multi-cloud systems, introducing a live migration strategy for virtual machines called CoMCLOUD, utilizing a broker-based technique. The approach aims to minimize downtime through pre-copy-based parallel migration. The research primarily focuses on explaining the VM placement strategy and associated trade-offs but does not delve into the practical implementation using specific frameworks. While the study provides results through simulations of a multi-cloud environment, practical validation is recommended through live migration with CSPs. Further exploration and implementation with real-world CSPs would contribute to the practical evolution of the proposed VM migration strategy.

Hence, this research does not discuss the impact of VM multi-cloud migration and portability on performance, security, and reliability. The above gaps in the research necessitate exploring the strengths and weaknesses of multi-cloud VM migration and portability in AWS, GCP, and Azure public clouds. The research paper does not look

into the specific technologies, protocols, and algorithms used for multi-cloud migration and portability.

2.3 Database Migration

Recent studies has covered Various aspects of database migration, such as The research article Namdeo and Suman (2021) introduces a database migration model, the Snapshot-Live Stream Db Migration Model (SLSDMM), designed for transitioning data from Relational Database Management Systems (RDBMS) to NoSQL databases. The proposed SLSDMM model incorporates both snapshot and live stream data migration techniques, addressing the dynamic nature of data in modern applications. The paper positions SLSDMM as a novel solution that combines the strengths of snapshot and live stream migration, presenting a hybrid approach for improved efficiency. The paper briefly mentions the performance of SLSDMM for large databases, but a more in-depth exploration of scalability issues and potential optimizations for cloud-based databases would enhance the practical applicability of the model. A middleware-based approach by Kiranbir Kaur (2020) and a successive study for energy efficiency improvement Kaur et al. (2022) for multi-cloud database migration facilitating migration across Azure, GCP, and AWS have made significant contributions to live database migration. The approach employed a middleware program written in C#, which replicates the code object by object and record by record. However, the impact of replicating records and database objects on database performance has not been studied. Further research is needed to address the optimization of the proposed approach, whereas A study on transactional database scaling by Georgiou et al. (2022) provides a commendable exploration of its scalability and performance, employing a range of benchmarks and workload mixes for transactional database scaling for MySQL, Oracle, and PostgreSQL DB. Strengths include the introduction of a parallel replication algorithm, the analysis of affected classes, and the implementation of statement-level load balancing. However, gaps in the research surface in the form of a limited focus on real-world use cases, a lack of exploration into scenarios with various sizes of databases, and the need for a more in-depth discussion on the impact of working with a cloud-managed database engine

2.4 Storage Migration

Shi et al. (2020), introduced a secure multi-cloud storage system based on an application programming interface (API) and software development kit (SDK). The system utilizes erasure code for blocking original data, followed by encryption using the Advanced Encryption Standard (AES) algorithm. Additionally, MD5 is employed for content verification of encrypted blocks. It's important to note that this research does not encompass a storage system for Azure Cloud and Google Cloud in terms of object storage, and it is limited to the Linux environment due to the implementation using the Jerasure library. followed by Mseddi et al. (2021) proposing an efficient replica migration scheme for distributed cloud storage. But the later one lacks the comparative study of techniques and performance on AWS, GCP, and Azure clouds. Finally, an article by Kumar et al. (2022) discussed the challenges of block-level deduplication in cloud storage due to maintenance difficulties and high processing power requirements. It introduces file-level deduplication as an alternative, highlighting its benefits in handling a large number of blocks and reducing processing power needs. The proposed approach focuses on file-level deduplication

and compression to enhance cloud storage efficiency, allowing for the storage of a single copy of redundant data and improving availability through replication.

The examination of existing literature exposes several deficiencies in effectively implementing simulated strategies in a multi-cloud setting. A substantial number of tools and proposed frameworks do not possess a consolidated interface for migrating the infrastructure, applications, data, and storage of SMEs. Additionally, many systems exhibit platform dependencies, requiring programming skills to expand functionalities, leading to substantial costs in both expertise and time.

3 Methodology

To work upon the gaps perceived in Section 2, this research utilizes a few prominent conclusions drawn from the critical analysis. The proposed study is based upon a suggestion by Caceres and Globa (2022), which emphasizes the use of higher-level APIs, services, and protocols such as HTTP for interoperability. Also, this study extends the use of SDKs and APIs for complete infrastructure migration, as implemented by Kiranbir Kaur (2020) and Shi et al. (2020) for migration purposes.

Grozev and Buyya (2014) laid the foundational work for load distribution of a three-tier application in a multi-cloud environment based on a definition of standard three-tier applications given in Fowler et al. (2003), which layers the whole web application in three distinct yet interconnected layers:

- Presentation Layer: This is usually the graphical user interface layer that provides the user with the ability to interact with the system.
- Business Layer: executes the primary business logic by accepting requests from presentation layer while accessing and modifying the data layer
- Data Layer: Manages the persistent data by accepting data queries from the business layer and returning the result of an operation.

From a cloud architectural perspective, presentation and the business layer are coupled together as a modern application that could be written in any programming language, e.g., Java or Python, and run on virtual machines provisioned by the cloud providers. In contrast, the data layer functions independently as a standalone service, encompassing relational database engines, NoSQL storage, and object or file storage for persistent storage solutions. The majority of the applications developed in the current era by SMEs follow a three-tiered approach. Hence, a solution needs to be developed for achieving seamless multi-cloud migration and portability of virtual machines, database engines, and object storage across the public clouds. To achieve this goal, comprehensive tools and techniques need to be assessed to enhance interoperability from the IaaS level to the SaaS level.

For a middleware-based cloud VM migration, this study explored potential object-oriented languages that have rich SDK support from CSPs. A few of the prominent choices were Java, Python, JavaScript, .NET, and GO. Out of this list, Python was finalised for implementation purposes due to it's features such as rapid development, compatibility with cloud platforms, developer community support, interoperability, and most of the recent work done on migration used python as a programming language.

Table 1: CSPs Virtualization format comparison

Cloud Provider	**Hypervisor**	**Container Type**	**Disk Format**
Amazon Web Services	XEN	AMI	AMI
Microsoft Azure	Hyper-V	Bare	VHD
Google Cloud Platform	KVM	Bare	RAW, AMI, VDI

3.1 Automated Cloud Discovery

Cloud service discovery refers to the process of identifying and locating available services in a cloud computing environment. It is a crucial aspect of managing and interacting with services in a dynamic and distributed system. As discussed in 2.1, while Terraform's core focus is on defining and deploying resources, it can be utilized for certain aspects of multi-cloud infrastructure discovery. Terraform employs a declarative language that provides a unified way to describe infrastructure across multiple cloud providers. Terraform uses state files to track the current state of infrastructure. This can be valuable for understanding the existing state and changes in the infrastructure. This research proposes a novel approach to utilizing a Terraform state file for resource discovery in a multi-cloud environment.

3.2 Virtual Machines Compatibility

As discussed in Section 2.2, migrating a virtual machine from one CSP to a different CSP has a lot of challenges associated with it, especially for GCP, Azure, and AWS, as all of them use different sets of virtualization formats in terms of hypervisor, container file type, and disk format. A summary of these various formats is outlined in Table 1. As the current research is dealing with interoperability between three different types of hypervisors (XEN, Hyper-V, and KVM), it can be considered a case of heterogeneous VM migration. These three CSPs not only use different formats for the underlying disk images but also container-type descriptors that specify the metadata regarding the instance launch. Whereas AWS wraps the metadata descriptor in Amazon Machine Image (AMI) format, Azure and GCP use the bare format, signifying the absence of an initiation descriptor.

Hence, this research extends the work of Kargatzis et al. (2017) and Raj et al. (2020) to assess middleware-based heterogeneous migration in the case of XEN, Hyper-V, and KVM hypervisors. This research also examines the feasibility and interoperability of varied disk formats such as AMI, VHD, and RAW with the use of Python SDKs and APIs provided by the CSPs under consideration. This feasibility study is extremely important, as even if the CSPs are utilizing standard open formats for their infrastructure, they always introduce a few proprietary modifications and advancements, which eventually result in a lack of interoperability.

3.3 Database Migration

A database is a structured collection of data that is organized and stored in a way that allows for efficient retrieval, management, and manipulation of information. Databases are a fundamental component of information systems and are used to store, organize, and manage data for various applications. The majority of small and medium enterprises use relation database management system (RDBMS) due their several advantages, such as Atomicity, Consistency, Isolation, Durability (ACID), etc. On average, SMEs running a three-tier application store data sizes of around 5 GB to 10 GB. Out of which MySQL, PostgreSQL, Oracle, and Microsoft SQL Server are the most widely used databases in the industry, [3].

As the data under migration could be from the enterprise's production database, it is extremely crucial to ensure the reliability of the proposed solution along with the accuracy of the data transfer. Hence, the proposed system will be based on migrating the relational databases, namely MySQL, PostgreSQL, Oracle, and Microsoft SQL Server, but the design will be able to extend to other databases in the future with minimal effort. For the performance evaluation, reliability and accuracy shall be assessed. This can be done with schema-level validation and data-level validation after the migration is performed. This will also verify compatibility between the source and target database schemas.

3.4 Object Storage Migration

Object storage is a type of data storage architecture that manages and organizes data as distinct objects rather than in traditional file hierarchies or block structures. Each object typically consists of data, metadata, and a unique identifier called a key. Object storage is commonly used for large-scale, unstructured data such as multimedia files, backups, and archives. In context of SMEs, it could be any kind of object, such as text files, audio files, images, etc. AWS, Azure, and GCP all support object storage; however, the terminology for all of them varies, such as in AWS it is called Simple Storage Service (S3), in Azure it is called blob storage, and in GCP it is cloud storage. SMEs can have varied storage requirements depending on the functional requirements, and object size can vary from a few KBs to GBs per object. Hence, a comprehensive storage migration is required that can reliably handle large file migrations with minimal time.

4 Design Specification

As per the discussion in Section 3, the entire proposed solution is based on a middleware design for facilitating the migration. Middleware refers to software that acts as an intermediary layer between different applications or components within a computing environment. It serves as a bridge that facilitates communication, interaction, and data exchange between disparate systems, applications, or services. Middleware plays a crucial role in simplifying the complexity of integrating diverse technologies and enabling them to work together seamlessly. Middleware's key characteristics include communication facilitation, integration support, data management, security implementation, transaction

[3] https://www.statista.com/statistics/1131568/worldwide-popularity-ranking-relational-database-management-systems/

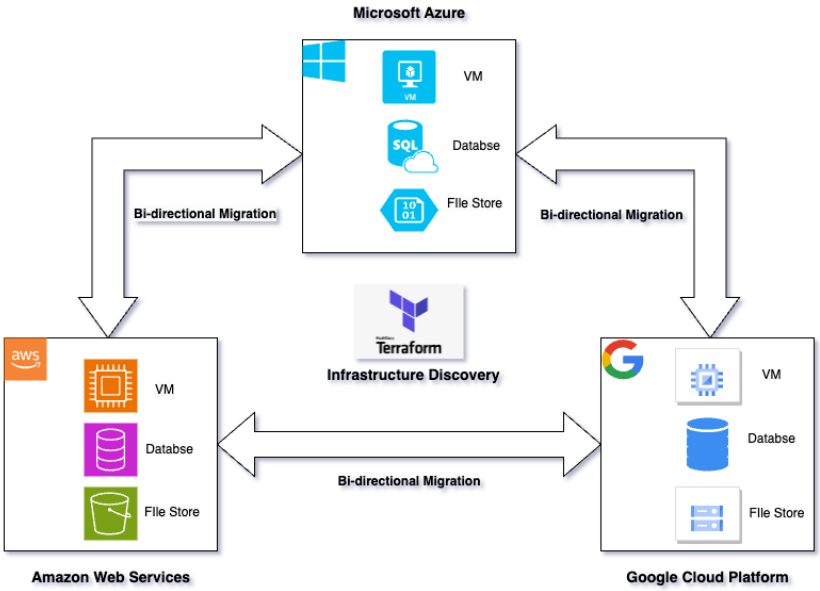


Figure 2: Proposed System Overview

management, and load balancing, providing advantages such as enhanced interoperability, scalability, and streamlined development and maintenance processes. Middleware often utilizes SDKs and libraries to build up new functionalities over existing ones. Sometimes middleware is also established to build a wrapper or an abstraction around APIs or heterogeneous systems, similar to the current study. In this research, a Python-based middleware is designed and developed to perform interactions with all three CSPs. For prototyping purposes, each script is developed individually and takes pre-defined variables for cloud account credentials.

To achieve resource discovery in a cloud environment, a Python script is designed along with a Terraform script, where the Python script provides the inputs to the Terraform script and invokes it by making a CLI command, which can in turn provide the details of each cloud resource along with all the dependent resource information. Terraform is an open-source infrastructure as code (IaC) tool developed by HashiCorp that enables users to define and provision infrastructure resources in a declarative configuration language. To achieve this, AWS, Azure, and GCP are added as providers in the Terraform script along with their respective cloud account credentials. The key resource used by the Terraform tool is the import command here. The `Terraform import` command is used in Terraform to import existing infrastructure into the Terraform state. This is useful when resources that were created outside of Terraform are needed to be managed using Terraform. The terraform import command associates existing resources with a Terraform configuration.

For VM migration, middleware is designed to create an abstraction over heterogenous VM migration, which involves dealing with various hypervisors and multiple disk formats such as RAW, VHD, VMDK, and AMI. The scripts are written in such a format that they can be exposed as web services with the help of representational state transfer (REST) APIs. Figure 3 is the flowchart of the entire VM migration process. The migration cloud takes place in six possible ways by permutation of three cloud vendors. Each direction has a set of generic processes, such as exporting VM from the source cloud account, converting

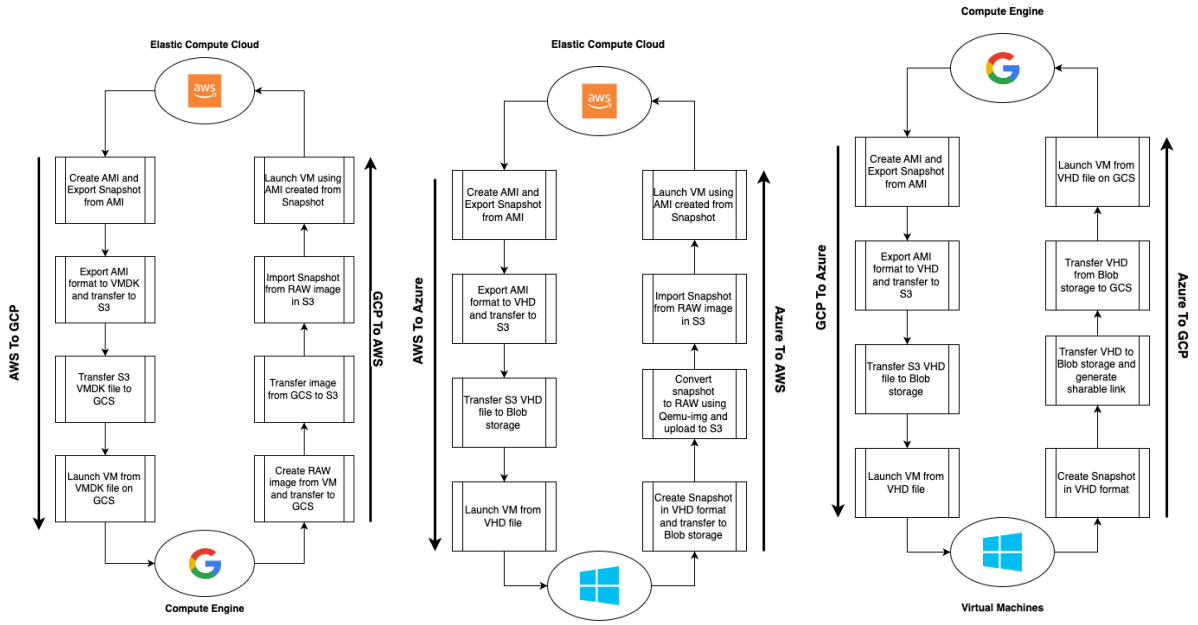


Figure 3: VM Migration Middleware Designed Steps

it to the required format, transferring the image file to the cloud storage service of the destination cloud platform, and then importing the image file to launch VM.

A common and comparatively simple method to perform DB migration is the snapshot method, also termed a dump transfer. A snapshot migration involves taking a point-in-time snapshot or copy of the entire database and transferring it to a new environment. It's a relatively straightforward approach but may involve downtime during snapshot creation due to the lock set on the source database to maintain the integrity of the data. To mitigate this, a hybrid approach is exercised to find a trade-off between speed and performance. This research is utilizing the hybrid strategy proposed by Namdeo and Suman (2021) to migrate databases using the Snapshot-Live Stream Db Migration Model (SLSDMM), where DB is migrated using both snapshot and streaming data. For designing such a strategy, the research is utilising dump file export and restoration from the source database to the destination database, followed by the data synchronisation step, which will migrate the data record by record from the source database to the destination database until both databases are synchronised.

The storage migration will follow a middleware approach where the Python middleware will migrate the object files from source storage to destination, iterating object by object. For upload, the SDKs support multi-part upload for faster file migration; however, download functionality by SDKs does not support such optimization; hence, this study is making use of a utility called aria2[4]. Aria2 is an open-source, cross-platform software that supports downloading files from various protocols, making it a powerful tool for managing and automating downloads. Aria2 is capable of downloading files concurrently; it can split a file into multiple parts and download them simultaneously. This can lead to faster download speeds, especially for large files.

[4]https://aria2.github.io/manual/en/html/README.html

5 Implementation

5.1 Cloud Resource Discovery

5.1.1 Azure Resource Recognition

The implemented code interacts with Microsoft Azure services using the Azure SDK for Python. This script employs the ServicePrincipalCredentials class from `azure.common.credentials` for authentication and utilizes the ResourceManagementClient class from the `azure.mgmt.resource` package. The code lists resources of specific types, including virtual machines (Microsoft.Compute/virtualMachines), disks (Microsoft.Compute/disks), snapshots (Microsoft.Compute/snapshots), network interfaces (Microsoft.Network/networkInterfaces), storage accounts (Microsoft.Storage/ storageAccounts), and various types related to databases such as PostgreSQL, flexible PostgreSQL, Azure Data, MariaDB, MySQL, flexible MySQL, SQL servers, and SQL Virtual Machines. The script then iterates through the listed resources and fetches their IDs, names, and locations, providing a detailed view of the Azure resources present in the specified resource group. This implementation showcases how to use the Azure SDK for Python to programmatically interact with Azure resources, list them based on resource types, and extract relevant information about each resource within a designated resource group.

5.1.2 AWS Resource Recognition

The developed Python script utilizes the Boto3 library to interact with AWS services, specifically focusing on EC2 instances, RDS databases, and S3 buckets. The AWS credentials, including access key, secret key, and session token, are configured to authenticate the Boto3 clients. Three distinct clients are initialized for EC2 (ec2_client), RDS (rds_client), and S3 (s3_client) services, each associated with the specified AWS region.

The script then fetches details about EC2 instances using the describe_instances method of the EC2 client. It iterates through the retrieved reservations and instances, printing relevant information such as instance ID and state. Similarly, the script obtains information about RDS database instances using the describe_db_instances method of the RDS client. It iterates through the retrieved DB instances, printing details like DB instance ID and engine.Lastly, the script fetches details about S3 buckets using the list_buckets method of the S3 client. It iterates through the retrieved buckets, printing the name of each bucket. This script provides a comprehensive overview of the AWS resources within the specified region, facilitating easy monitoring and management of EC2 instances, RDS databases, and S3 buckets through the Boto3 library.

5.1.3 GCP Resource Recognition

One more Python script is being developed that utilizes the Google Cloud Client Libraries to interact with Google Cloud Platform (GCP) services, specifically focusing on compute engine instances, cloud storage buckets, and database instances. The script begins by importing the necessary modules from the `google.cloud` package. The script initializes a compute engine client, a cloud storage client, and a cloud SQL client for interacting with resources from the GCP cloud and fetching metadata. This script provides a comprehensive overview of GCP resources, facilitating easy monitoring and management of compute engine instances, cloud storage buckets, and cloud SQL instances through

the Google Cloud Client Libraries. Ensure that the specified service account has the necessary permissions to access and list resources in the given project.

5.1.4 Terraform State Discovery

The Terraform scripts are developed, which work in conjunction with the previously developed Python scripts. The developed scripts serve as an infrastructure-as-code blueprint for configuring Azure, AWS, and GCP resources using the HashiCorp Terraform platform. The script begins by specifying the required AWS, Azure and google provider, detailing the source and version which establishes the necessary authentication credentials, including the client ID, client secret, tenant ID, and subscription ID. Following this, distinct resource blocks that define various Azure, AWS and GCP resources to be managed by Terraform. These resources encompass an Azure virtual machine, a storage account, a PostgreSQL server, and a resource group. Notably, fields within these resource blocks, such as names, locations, and configurations, are intentionally left blank and are expected to be populated with the actual details retrieved from the existing cloud account resources using a prior Python script. Upon execution, this Terraform script facilitates the import of the identified cloud resources into Terraform's state, enabling streamlined infrastructure management and future updates through Terraform workflows. After executing this script with the `terraform import` command, the terraform.tfstate file will be populated with the state of the Azure resources in the given cloud account.

5.2 Virtual Machine Migration

The implementation of VM migration primarily uses SDKs for AWS, Azure and GCP, with various libraries involved, such as *boto3, google.cloud, azure.mgmt*, etc. There are various steps associated with VM migration, as discussed in Section 4. The migration process deals with various disk formats; most of them are being converted using in-built APIs provided by the SDK; however, there is no built-in support for some format conversions, such as VHD to RAW in the case of the Azure to AWS migration. QEMU-img [5] is utilized to perform such disk conversions to make the migration process compatible. QEMU-img is a command-line tool used in the QEMU (Quick Emulator) virtualization environment. It is primarily used for disk image manipulation. The Python scripts are developed for each migration direction. The scripts perform authentication with each cloud provider and take a snapshot of a running VM using SDK client methods such as `ComputeManagementClient.snapshots()`, `boto3.client.import_snapshot()`, and `google.cloud.InsertImageRequest()`. This process is followed by transferring the image file to the destination cloud storage service with the help of the respective SDK functions. Finally, the image file is imported into the destination compute subsystem and VM is initialised.

5.3 Database Migration

The implemented Python script focuses on the synchronization of tables and data between a source relational database and multiple destination relational databases. The implementation utilizes key libraries, including SQLAlchemy for database operations, pandas

[5] https://www.qemu.org/docs/master/tools/qemu-img.html

for data manipulation, and the database Connectors for connectivity between SQL engine and the python middleware. The script defines functions for obtaining database engines, fetching source and destination databases from user input, and synchronizing database structure and data. The main execution section orchestrates the entire process by obtaining source and multiple destination database engines, fetching table names from the source, and iteratively synchronizing the databases and their corresponding data. Notably, the script employs a modular approach, encapsulating functionalities within functions for enhanced readability and maintainability. As discussed in Section 4 DB migration utilize a mixed approach where bulk data is transferred initially using dump files followed by synchronization of data records for live migration. For this, the scrips uses os module from python for performing command-line SQL execution for example `mysqldump & mysql` for dumping bulk data from source to destination.

5.4 Storage Migration

The implementation of the Storage Migration system includes a Python script that facilitates inter-cloud object migration among AWS S3, GCS, and ABS. It is written in Python, a versatile and widely-used programming language, and employs specific libraries for each cloud provider interaction: Boto3 for AWS, google-cloud-storage for Google Cloud, and the Azure SDK for Python for Azure Blob Storage. The script defines functions for each cloud provider, encompassing operations like getting files, getting a specific file, putting a file, and deleting a bucket. The main function consists of the migration process based on the specified configuration options. These options include all bidirectional transfer links in AWS, GCP, and Azure.

Throughout the execution, the script outputs messages to the console, providing information on the progress and status of the migration process. It details the start of the process, the type of transfer being executed, and concludes with a completion message upon successful execution. The script offers flexibility by allowing the option to delete objects from the source bucket after a successful transfer, as controlled by the `DELETE_AFTER_TRANSFER` configuration. Additionally, error handling mechanisms are implemented using exception handling, ensuring that relevant error messages are printed to the console if issues arise during file download, upload, or bucket deletion. Aria2 utility is used to speedup the object download which perform faster download by parallel disk allocation and download.

The overall result is a tool that facilitates seamless inter-cloud object migration, leveraging the capabilities of Python and cloud-specific libraries to interact with AWS, Google Cloud, and Azure services.

6 Evaluation

The next section lists a set of experiments to validate the proposed resource discovery and migration strategy for VMs, databases, and storage objects. The aim is to perform application profiling in terms of feasibility, migration time, accuracy, and reliability.

6.1 Cloud Testbed Specifications

A testbed is a dedicated platform or system where developers and engineers can conduct experiments, evaluate, and validate the functionality, performance, and compatibility

Figure 4: Terraform.tftstate File Populated With Resources From Cloud Environment

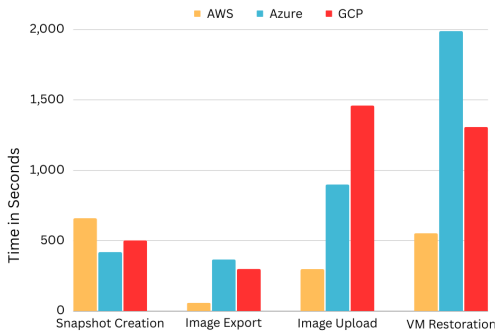
of software or hardware products in a controlled setting before deployment to ensure reliability and effectiveness. The below-outlined experiences are performed on AWS Linux 2 AMI with EC2 instance type c5.9xlarge, which is a compute-optimized instance class with 36 vCPU, 72 GiB of memory, and a network bandwidth of 12 Gbps.

6.2 Evaluation of Cloud Resource Discovery

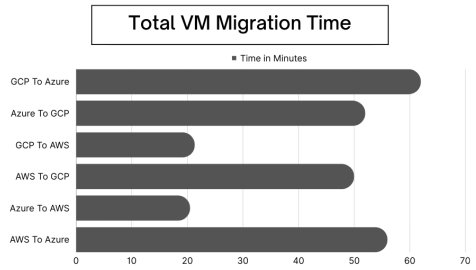
This experiment validates the compatibility and generalizability of the developed Terraform script for AWS, GCP, and Azure environments. The single script should be able to import resource configurations from all three CSPs. This shall result in the generation of a terraform.tfstate file, which will populate the discovered resources in JSON format. For this experiment, three resources from various resource types, such as virtual machines, managed disks, storage buckets, and database instances, were initiated in each cloud account. The results from Figure 4 validate the application of Terraform as a resource discovery tool. Terraform was able to discover the detailed configuration of each cloud resource available in the account. This result gives a future direction for developing a system that can perform cross-platform infrastructure management with the use of Terraform.

6.3 Evaluation of Cloud VM Migration

The below set of experiments is based on a migration analysis of a standard Ubuntu 22.04.6 LTS instance with 2 vCPU, 4 GiB of RAM, and 30 GB of disk space. This same image was initialized on each of the cloud platforms to perform the evaluation under identical conditions.



(a) Individual Process Time For Each Cloud



(b) End To End VM Migration Time

Figure 5: Ubuntu 22.04.6 LTS Migration Evaluation with 2 vCPU, 30 GB disk space

6.3.1 Experiment 1 : Individual Process Time

In this experiment, timed operations were performed with the standard image on each cloud platform and metrics such as snapshot creation, image export and download, image conversion and image upload on the destination cloud, and finally the image import operation. The time taken by each process is shown in the Figure 5(a)

The comparison of the experimental outcome in Figure 5(a) depicts that snapshot creation and format conversion are instantaneous processes as compared to image transfers. Azure took the least time for exporting the snapshot from the running virtual machine, which is 420 seconds; in contrast, AWS took the highest time for the snapshot creation process with 660 seconds. In the restoration part, Azure took most of the time as compared to its creation, export and upload duration contributing to the vast end to end timings associated with Azure cloud as discussed in the following experiment.

6.3.2 Experiment 2 : Total Transfer Time

This experiment is about migrating the virtual machine end-to-end by making a snapshot of the running image from the source cloud and initialising the virtual machine on the destination cloud. All the migration directions were tested for AWS-GCP-Azure and time duration were noted as given by Graph 5

This experiment confirms the feasibility of the VM migration between these three major public CSPs. The analysis from the graph signifies that the VM migration time for Azure to AWS is the least with end-to-end migration time of 20 minutes 50 seconds; on the other hand, GCP to Azure migration takes the maximum time out of the tested scenarios. Notably, the experiment was also carried out with different disk formats, which were not part of the design proposal as per the claims from the CSPs documentation, but most of such approaches failed the restoration stage on the destination cloud. Hence, from the results, the compatible disk formats are confirmed as per the design proposal.

6.4 Evaluation of Cloud Database Migration

The below experiments were designed, per the direction of Section 3.3 to evaluate the reliability and accuracy of the proposed cross-cloud database migration system.

Table 2: End-to-End Object Migration Timings

Cloud Object Migration Direction	Time In Seconds
AWS to GCP	123.642
AWS to Azure	491.420
GCP to AWS	52.419
GCP to Azure	41.336
Azure to AWS	549.460
Azure to GCP	531.095

6.4.1 Experiment 1 : Reliability Analysis

This experiment validates the reliability of the system by performing schema-level validations. In this experiment, various database objects such as tables, indexes, views, stored procedures, triggers, constraints, indexes, and sequences were created in the source database. For this MySQL 5.7 DB server, it was setup in the Azure and AWS clouds, with the source DB in the Azure cloud containing a total of 100 DB objects.

The reliability of the system was analysed on the basis of the successful migration of each object from source to destination. The proposed solution is found to be 100% reliable, with no DB objects getting dropped during migration. All the object types were found to be compatible across the clouds and functioned normally after the migration.

6.4.2 Experiment 2 : Synchronization Accuracy Analysis

This experiment evaluates the accuracy with which the DB migration and synchronization occur. To study the accuracy of the system, 5,000,000 (5 million) rows were migrated from the source DB to the destination DB. Out of which 4000,000 (4 million) records were migrated initially with dump functionality, and then new 1 million records were inserted for checking the synchronisation functionality performed by SQLAlchemy with the help of Pandas. After running the sync function, the overall records from the destination database were evaluated.

On cursory analysis, it was observed that the destination database had a total of 5 million records that were migrated from the source database. For measuring the accuracy of the system, the SHA256 hash was calculated from the source DB and matched with the data level hash from the destination DB. Both the hashes match, concluding that all the data records from the source were replicated to the destination accurately. Proving the successful execution of the snapshot-streaming model using the dump and SQLAlchemy synchronisation techniques.

6.5 Evaluation of Cloud Storage Migration

6.5.1 Experiment 1 : Individual Transfer Time

This experiment deals with migrating a large file object across different cloud directions. A sample file of 5 GB was used to evaluate and compare transfer performance along the cloud directions. In the Python script, the flag was set according to the particular scenario, and the output time was noted from the console as mentioned in Table 2.

From the observation, it is evident that the migration time from Azure to GCP is the maximum, taking 531 seconds. where the GCP to Azure migration took the least time for transferring a 5 GB object which is 41.336 seconds.

6.5.2 Experiment 2 : Reliability And Accuracy Analysis

To assess the reliability of the migration system, 10,000 objects of small and medium size were migrated from the source to the destination object store. For the system to provide 100% reliability, it was expected to receive all 10,000 objects at the destination cloud storage. This experiment involved a negative testing scenario where script execution was terminated abruptly and restarted to examine the fault tolerance and resiliency features of the proposed design. To evaluate the accuracy, the MD5 hash of the source files was compared with the destination file.

Despite an unexpected interruption that forced the script to terminate abruptly, the remaining files were seamlessly resumed and migrated without any data loss, corruption, or duplication. This remarkable resilience was achieved through a robust mechanism that enabled the script to pick up where it left off without compromising the integrity of the data. To confirm the accuracy of the migration, the hash values of the files in the source and destination folders were matched, confirming that the data was transferred without any errors or inconsistencies.

6.6 Discussion

Experimental analysis of the proposed system is extremely promising, as the developed implementation showcases the successful migration of a three-tier infrastructure. Section 6.2 depicts a new use case of Terraform as a resource discovery tool. This could benefit the organizations by expanding upon the functionality of the existing Terraform setup to utilise it's discovery features to explore the live infrastructure and migrate it to a multi-cloud architecture. The results shown in 6.3 validate the interoperability of virtual machines between the three major cloud providers and a workflow to migrate VMs across CSPs. The approach is satisfactory and in alignment with the research question, as it performs VM migration without any downtime. The implementation and experiment can be improved by using compression techniques for the VM images to reduce the download and upload time, which will not only speed up the migration but also save bandwidth and the DTO cost associated with each cloud provider. Also, VMs with more types of guest OS or larger disk sizes can be experimented with for a much wider feasibility study.

Analysis from 6.4 results signifies that the middleware-based DB migration system is capable of migrating all the DB objects without any record loss to the destination database with 100% accuracy, as verified with the hash calculation. This tool is designed to utilize dump commands that are specific to the database engine, which limits its generalizability, and extensions support more database engines without any modification. A more sophisticated approach could be developed to build an engine-independent bulk migration system. The experiment performed in Section 6.5 successfully validated the migration system's ability to handle unexpected terminations and maintain data integrity for migrating large and extensive numbers of objects. However, it is important to note that the negative testing scenario employed a single abrupt termination. Further testing with multiple interruptions and longer downtime would be beneficial to fully assess the system's resilience under real-world conditions. Additionally, the performance of the

system could be further improved in terms of speed by utilising appropriate compression techniques.

7 Conclusion and Future Work

This research aims to devise a feasible method for migrating standardized three-tier infrastructure, comprising compute, storage, and databases, among the three major cloud providers, namely AWS, Azure, and GCP. The objective is to promote interoperability and mitigate the issue of vendor lock-in for SMEs. The primary challenge in designing a multi-cloud-compatible VM migration system is the lack of standardization among cloud providers. The proposed design overcomes the challenge of standardization by leveraging high-level protocols provided by CSPs and abstracting away the complexities of the migration process.

The proposed system's compatibility and migration capabilities demonstrate its effectiveness in mitigating vendor lock-in for businesses. Using the proposed tool, an enterprise can move their infrastructure, such as compute instances, database servers, and storage buckets, across the clouds with minimal configuration, giving them flexibility and a wider choice of platform. This system could be integrated with a decision-making system to generate an automated migration trigger as per cost-saving or network optimization preferences. The primary focus of the initial design was to achieve interoperability, while the optimization of migration time through the implementation of suitable compression and encryption techniques was not explicitly considered. This enhancement, however, could significantly improve the proposed system's security and efficiency.

Future research directions could explore the feasibility of conducting live migrations of virtual machines across interconnected cloud environments while operating in heterogeneous hypervisor setups. This could be achieved by employing various migration techniques, such as pre-copy, post-copy, hybrid, and block-level migration, to address the challenges inherent in such cross-hypervisor migrations.

References

Addya, S. K., Satpathy, A., Ghosh, B. C., Chakraborty, S., Ghosh, S. K. and Das, S. K. (2023). Comcloud: Virtual machine coalition for multi-tier applications over multi-cloud environments, *IEEE Transactions on Cloud Computing* **11**(1): 956–970.

Alonso, J., Orue-Echevarria, L., Casola, V., Torre, A. I., Huarte, M., Osaba, E. and Lobo, J. L. (2023). Understanding the challenges and novel architectural models of multi-cloud native applications – a systematic literature review, *Journal of Cloud Computing* **12**(1).

Anglano, C., Canonico, M. and Guazzone, M. (2020). Easycloud: a rule based toolkit for multi-platform cloud/edge service management, *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 188–195.

Anglano, C., Canonico, M. and Guazzone, M. (2021). Easycloud: Multi-clouds made easy, *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 526–531.

Bahaweres, R. B. and Muhammad Najib, F. (2023). Provisioning of disaster recovery with terraform and kubernetes: A case study on software defect prediction, *2023 10th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pp. 183–189.

Bhalla, Y., Hemamalini, V. and Mishra, S. (2023). Automating hadoop cluster on aws cloud using terraform, *2023 International Conference on Networking and Communications (ICNWC)*, pp. 1–10.

Brikman, Y. (2019). *Terraform: Up amp; running*, O'Reilly Media, Inc.

Caceres, A. and Globa, L. (2022). State-of-the-art architectures for interoperability of heterogeneous clouds, *2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, pp. 704–709.

Choudhary, A., Govil, M. C., Singh, G., Awasthi, L. K., Pilli, E. S. and Kapil, D. (2017). A critical survey of live virtual machine migration techniques, *Journal of Cloud Computing* **6**(1).

de Carvalho, L. R. and Patricia Favacho de Araujo, A. (2020). Performance comparison of terraform and cloudify as multicloud orchestrators, *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pp. 380–389.

Fowler, M., Stafford, R., Mee, R., Hieatt, E., Foemmel, M. and Rice, D. (2003). *Patterns of enterprise application architecture*, Addison-Wesley.

Georgiou, M. A., Paphitis, A., Sirivianos, M. and Herodotou, H. (2022). Hihooi: A database replication middleware for scaling transactional databases consistently, *IEEE Transactions on Knowledge and Data Engineering* **34**(2): 691–707.

Grozev, N. and Buyya, R. (2014). Multi-cloud provisioning and load distribution for three-tier applications, *ACM Trans. Auton. Adapt. Syst.* **9**(3).
URL: *https://doi.org/10.1145/2662112*

Gupta, M., Chowdary, M. N., Bussa, S. and Chowdary, C. K. (2021). Deploying hadoop architecture using ansible and terraform, *2021 5th International Conference on Information Systems and Computer Networks (ISCON)*, pp. 1–6.

Joshua Gans, M. H. and Masri, M. (2023). Economic analysis of proposed regulations of cloud services in europe, *European Competition Journal* **19**(3): 522–568.
URL: *https://doi.org/10.1080/17441056.2023.2228668*

Kargatzis, D., Sotiriadis, S. and Petrakis, E. G. (2017). Virtual machine migration in heterogeneous clouds: from openstack to vmware, *2017 IEEE 38th Sarnoff Symposium*, pp. 1–6.

Kaur, K., Bharany, S., Badotra, S., Aggarwal, K., Nayyar, A. and Sharma, S. (2022). Energy-efficient polyglot persistence database live migration among heterogeneous clouds, *The Journal of Supercomputing* **79**(1): 265–294.

Kaur, K., Sharma, D. S. and Kahlon, D. K. S. (2017). Interoperability and portability approaches in inter-connected clouds, *ACM Computing Surveys (CSUR)* **50**: 1 – 40.
URL: *https://api.semanticscholar.org/CorpusID:30625897*

Kiranbir Kaur, Sandeep Sharma, K. S. K. (2020). A middleware for polyglot persistence and data portability of big data paas cloud applications, *Computers, Materials & Continua* **65**(2): 1625–1647.
URL: *http://www.techscience.com/cmc/v65n2/39897*

Kumar, P. M. A., Pugazhendhi, E. and Nayak, R. K. (2022). Cloud storage performance improvement using deduplication and compression techniques, *2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pp. 443–449.

Liu, B., Xin, Y. and Zhang, C. (2022). A solution for a disaster recovery service system in multi-cloud environment, *2022 International Applied Computational Electromagnetics Society Symposium (ACES-China)*, pp. 1–4.

Mseddi, A., Salahuddin, M. A., Zhani, M. F., Elbiaze, H. and Glitho, R. H. (2021). Efficient replica migration scheme for distributed cloud storage systems, *IEEE Transactions on Cloud Computing* **9**(1): 155–167.

Namdeo, B. and Suman, U. (2021). A model for relational to nosql database migration: Snapshot-live stream db migration model, *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, Vol. 1, pp. 199–204.

Piper, B. and Clinton, D. (2023). *CloudTrail, CloudWatch, and AWS Config*, pp. 193–221.

Raj, S., Mangal, N., Savitha, S. and Salvi S., S. (2020). Virtual machine migration in heterogeneous clouds - a practical approach, *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pp. 1–6.

Shi, W., Liu, T. and Huang, M. (2020). Design of file multi-cloud secure storage system based on web and erasure code, *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*, pp. 208–211.

Tomarchio, O., Calcaterra, D. and Modica, G. D. (2020). Cloud resource orchestration in the multi-cloud landscape: A systematic review of existing frameworks, *J. Cloud Comput.* **9**(1).
URL: *https://doi.org/10.1186/s13677-020-00194-7*

Zhu, Q.-H., Tang, H., Huang, J.-J. and Hou, Y. (2021). Task scheduling for multi-cloud computing subject to security and reliability constraints, *IEEE/CAA Journal of Automatica Sinica* **8**(4): 848–865.