National College of Ireland

Cloud Native Application Disaster Recovery in a Multi-Cloud Environment – A DevOps Approach using Terraform

MSc Research Project
Cybersecurity

Wei Tong
Student ID: X21202648

School of Computing
National College of Ireland

Supervisor: Ross Spelman

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Wei Tong
Student ID:	X21202648
Programme:	MSc. in Cybersecurity **Year:** 2022
Module:	Final Thesis
Supervisor:	Ross Spelman
Submission Due Date:	14th December 2023
Project Title:	Cloud Native Application Disaster Recovery in a Multi-Cloud Environment – A DevOps Approach using Terraform
Word Count:	6305 **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: Wei Tong

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):	

Cloud Native Application Disaster Recovery in a Multi-Cloud Environment – A DevOps Approach using Terraform

Wei Tong
X21202648

Abstract

In today's business landscape, the ability to recover from unforeseen disasters and continue critical business operations is crucial for enterprises, irrespective of their size. Any additional minute when the system is offline could cause substantial financial losses and irreversible reputation damage to the organisation. Hence, finding the quickest and the most reliable solution to recover the failed system can be extremely beneficial. In recent years, one noticeable trend in the software industry is that cloud-native applications are becoming increasingly popular. These applications provide developers great flexibility when architecting complex distributed systems. With the advantage of leveraging multiple platforms, developers can create highly available, fault-tolerant applications across several cloud ecosystems. However, this trend also introduces many unique challenges to the development and operation (DevOps) team, such as the complexity of infrastructure orchestration and management between multiple cloud providers. To address these difficulties, Infrastructure as Code (IaC) tools like Terraform have been created. To find the most suitable disaster recovery solution for cloud-native applications, we conducted a comprehensive literature review of the existing research papers on the topic. However, none of them provided an in-depth exploration of performance evaluation. Therefore, we propose to develop a more robust and efficient disaster recovery solution for multi-cloud environments utilizing the IaC tool Terraform. In this research, we try to bridge the gap between literature and contribution value into the implementation of more efficient disaster recovery solutions for cloud-native applications in the multi-cloud environment.

1 Introduction

Reducing project implementation costs, increasing software development agility, and ensuring robust system security and compliance are the primary driving forces for businesses to adopt a cloud strategy. Gartner projected that, by the year 2025, over 95% of the new digital workload will be hosted in the cloud environments. The cloud's pay-as-you-use model gives enterprises of any size seamless access to an extensive pool of computing resources, which allow them to perform research and development activities at a fraction of the cost. This model offers businesses the opportunity to implement software project without the burdens of procuring hardware or installing operating systems. Instead, the process only involves provisioning the

required resources in the cloud. The flexibility offered by this pay-as-you-use model encourages businesses to start new projects and pay only for the resources consumed. Additionally, the ability to delete cloud resources effortlessly without penalties is another advantage to leverage the cloud strategy.

A vast range of services is offered by the Cloud Service Providers (CSP), including but not limited to Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Notably, Amazon Web Services (AWS), Microsoft Azure, and Google Cloud stand out as the three major players in this domain. Business often chooses a specific CSP according to their needs and migrates on-premises applications to the cloud to leverage many advantages, such as cost efficiency and convenience. Furthermore, new software is also developed and deployed within the same cloud infrastructure, primarily driven by the familiarity of developers with the specific CSP.

However, it is important to acknowledge that many potential issues exist when utilizing a single cloud approach, including concerns such as vendor lock-in, cost considerations, and business continuity. Organizations may only become aware of these issues when an unexpected interruption occurs, such as when their applications go offline due to unforeseen situations with the CSP's load balancer.

1.1 Motivation and Background

Recovering from unforeseen disasters swiftly is critical for organizations, regardless of their size and industry. Small e-commerce platforms must maintain an online presence so that their customers can purchase products from them. Air traffic control systems require continuous online operation so that all air traffic activities can be monitored and managed accurately. Even a brief downtime of the system can cause significant issues for the business. Therefore, find the best and most reliable disaster recovery solution for the failing system can be extremely important.

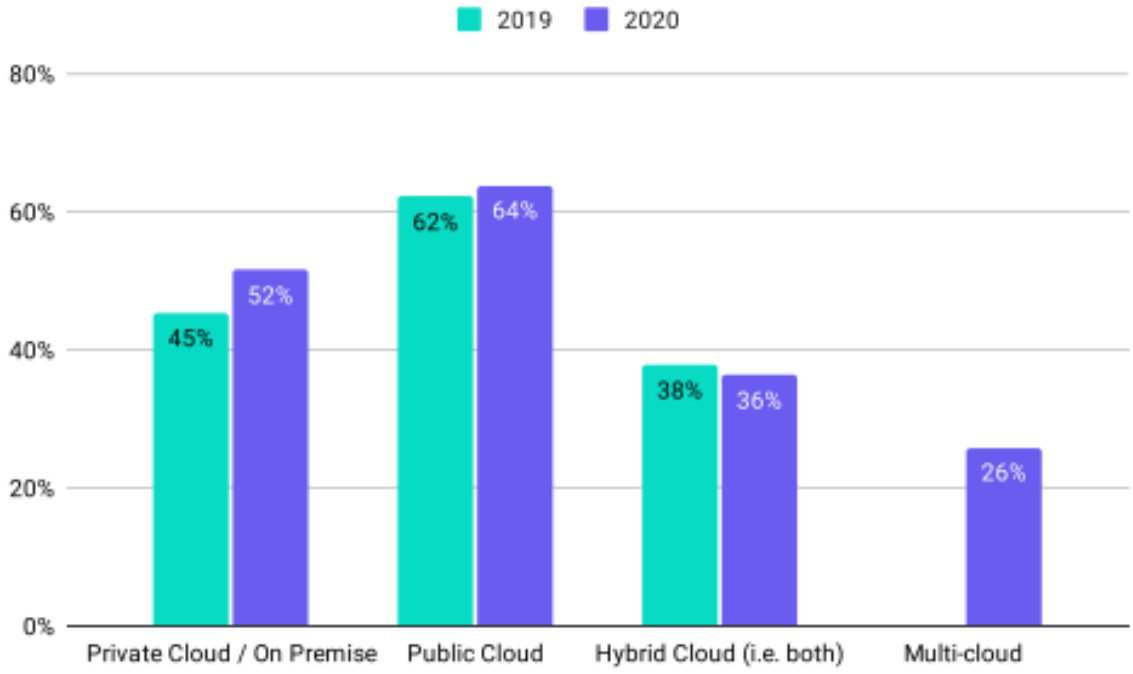

Figure 1. CNCF survey 2020. Source: CNCF

A 2020 survey conducted by The Cloud Native Computing Foundation (CNCF) reveals the growing popularity of the term multi-cloud, see Figure 1. It shows that cloud-native applications are gaining more interest due to the capability of running in any CSP that supports the Docker running environment. This gives developers more options and flexibility to architect applications that can leverage resources from multiple cloud providers.

However, the multi-cloud application architecture brings many advantages but also some challenges, especially for the Development and Operation (DevOps) teams. DevOps, as a cultural movement, makes the collaboration between the development and operation teams easier. It has one single goal, which is to release high-quality software efficiently. It also has many other benefits including knowledge sharing, continuous feedback, and improvement. In the intricate multi-cloud environment, the ability to leverage the Infrastructure as a Code (IaC) tool to provision the cloud infrastructure automatically becomes crucial. This approach has many benefits, such as code revisioning, static code inspection, reducing repetition, and cost-saving. There are many IaC tools available in the market, Terraform from HashiCorp stands out alongside many others like AWS CloudFormation and Pulumi. Infrastructure as Code can facilitate fast cloud infrastructure provisioning. The ability to create cloud infrastructure quickly can be an advantage for Business Continuity and Disaster Recovery (BC/DR) operation with applications running in multiple cloud environments.

Disaster Recovery (DR) involves strategies and plans on how an organization can recover from unexpected events, for example, natural disasters, deliberate disruption, loss of utilities and services, and hardware or software failures (Abualkishik, A.Z., A., A. and Gulzar, Y. 2020). Measuring the performance of the DR solution involves utilizing metrics such as

Recovery Point Objective (RPO), Recovery Time Objective (RTO), and cost. On the other hand, Business Continuity (BC) involves how an organization can continue operations seamlessly without downtime when disaster strikes.

1.2 Research Question

One of the key challenges associated with leveraging cloud resources is the ability to swiftly recover enterprise's application and maintain business continuity from unforeseen disasters, which can be beyond the awareness and control of the selected CSP.

This challenge motivates the formulation of the following research question: Can the Infrastructure as Code tool, such as Terraform, contribute to the reduction of a cloud-native application's recovery time during a crisis? Furthermore, to what extent this reduction in recovery time can be achieved?

In consideration of the research question, we propose to deploy the cloud-native application in multiple cloud environments using Terraform. If multiple CSPs are utilized, one is designated as the primary infrastructure and the others as secondary fallbacks in the event of disaster. This approach could potentially reduce the cloud-native application's disaster recovery time. The result could be continuous business operation without disruption, also mitigating the risk of vendor lock-in.

1.3 Document Structure

The remainder of this paper is structured as follows. In section 2, we conduct a literature review on the topic of BC/DR in a single or multiple cloud environment. Section 3 and 4 outline the methodologies used for our research and the final specification of the proposed solution. Section 5 explains more details about the implementation phase. Section 6 outlines how the proposed solution is evaluated against varies of test scenarios. Finally, section 7 provides the conclusion and future works.

2 Related Work

This section comprises a comprehensive analysis of the existing literature on the topic of BC/DR, organized in chronological order. Several works have been studied in the area of cloud-native application, as well as the BC/DC solutions for single-cloud and multi-cloud environments. However, from our extensive review, it has come to our attention that only one of the papers provided limited metrics for measuring the performance of BC/DR solutions.

The investigation conducted by Alshammari, M.M. *et al.* (2017) offers a detailed studies of the complexity of achieving disaster recovery in both single-cloud and multi-cloud environments. The study identifies and discusses several challenges that companies may face in the context of disaster recovery, emphasizing the potential security issues that may arise and data loss when all information is stored in a single cloud environment. To mitigate these risks, it proposes to adopt a multi-cloud environment, which has advantages in terms of high

availability, load balancing, and flexibility. Furthermore, the authors express the intention to develop the disaster recovery system in a multi-cloud environment in the future and provide insight of the metrics for RTO and RPO. The study conducted by Lavriv, O. *et al.* (2018) proposed the utilization of IaC tool Terraform for recovering the cloud infrastructure within a local OpenStack cloud instance. The paper presents the suitability of the IaC tool Terraform for disaster recovery in the cloud environment and the procedures on how to implement it. Disaster recovery duration is used for the performance measurement. However, it is worth noting that investigation conducted was confined to a single instance of the OpenStack cloud, it did not carry out the testing in any publicly available cloud environment. Additionally, the authors did not consider the utilization of a multi-cloud architecture design.

The research presented by Harwalkar, S., Sitaram, D. and Jadon, S. (2019) introduces a lightweight OpenStack prototype aimed at addressing disaster recovery challenges within a multi-cloud environment. The proposed open-sourced solution suggests that it is possible to facilitate heterogeneous multi-cloud disaster recovery through the utilization of OpenStack API and third-party plug-ins. However, it is important to note that the solution remains untested in any of the main publicly available cloud environments. Therefore, no performance metrics were ever provided, such as RTO and RPO. Offering a comprehensive study of disaster recovery, Abualkishik, A.Z., A., A. and Gulzar, Y. (2020) discuss in detail the spectrum of disasters organizations may encounter, several types of disaster recovery solutions, and the pros and cons of adopting cloud-based DR approach. The authors extensively highlight issues and challenges associated with cloud-based disaster recovery and propose the utilization of services and resources from multiple CSPs as a better alternative. However, the details of the implementation of the identified better solution are not provided.

The study conducted by Stamenkov, G. (2022) investigates many potential problems associated with the planning of disaster recovery and business continuity, which can be created with various frameworks or institutions. The research systematically compares a set of different BC/DR plans including critical information infrastructure plans, disaster recovery plans, business continuity plans, and many more, to study the pros and cons of each. An innovative layered plan model was proposed in this study. Each layer in the model contains three distinct BC/DR plans to address different levels of operation consists of normal operation, non-disastrous local emergency event, and disastrous event. Notably, this research is beneficial for large enterprise with the capability of allocating resource for selecting the best suitable plan, but the comprehensive information and lack of implementation details approach may overwhelm small and medium-sized businesses who do not have dedicated BC/DR resource. Moreover, no specific details on how the proposed layer plan can be executed. The research carried out by Liu, B., Xin, Y. and Zhang, C. (2022), states that the traditional single-cloud disaster recovery model falls short of meeting the current business need. The authors introduce a new novel system architecture that utilizes both the traditional DR method and blockchain technology. The proposed system runs on top of the multiple cloud environments, providing a third-party service that can potentially help organizations to implement a multi-cloud DR solution. It consists of many features such as distributed storage, security audit, and network transmission optimization. Additionally, it offers a unified multi-cloud resource management

interface and multi-cloud data DR capability. Although the authors put a lot thought into the design of a well-structured Disaster Recovery as a Service (DRaaS) system, the complexity of implementing a such system by businesses remain a considerable challenge. Moreover, the lack of details regarding system implementation, RTO & RPO and associated cost may further complicate the potential adoption of the proposed solution.

Nikolovski, S. and Mitrevski, P. (2022) explain the fundamental prerequisites for crafting a successful BC/DR solution in their study. It critically identifies the essential missing parameters in the reviewed literature, such as the infrastructure size, and the data permission, format, and volume. The paper emphasizes the importance of including measurement metrics such as RTO, RPO, and data volume when assessing a DR solution. However, the proposed solution remains primarily theoretical, lacking specifics on how it can be implemented and validated. The investigation conducted by Pruthvi Kumar Reddy, D. *et al.* (2022) focuses on the data recovery part of disaster recovery, by leveraging an already available infrastructure BC/DR plan via a cloud-based architecture. The proposed solution integrates multiple advanced concepts such as data encryption technologies, fog computing, and multi-cloud functionalities. The authors make the argument that the utilization of fog computing can mitigate network latency issues by bringing the computing resources closer to the target data, especially for a large data set. This is only true when the data volume is significantly big for the system to back up, it may cause delays in copying from the source to the target system. The proposal also suggests data segmentation and distributed backup across multiple CSPs for enhanced security, reducing the potential impact of data breaches. Each CSP will hold partial data due to the data partitioning process, therefore the attacker cannot make sense of the information even though one of the CSPs has a data breach. However, the implementation of such system can be extremely complicated. Furthermore, the scenario of one of the CSP experiencing a DR issue of its own while simultaneously the proposed system encounters some sort of difficulties raises concerns about data integrity, because each CSP holds the data partially, none of them has the whole data set. Additionally, the measurement metrics are limited to the data upload and encryption, with no mention of the duration of data restoration.

Alonso, J. *et al.* (2023) conducted a full systematic literature review (SLR) exploring the dynamics of cloud-native applications in multi-cloud environments. The authors argue that the model of traditional single-cloud architecture is becoming obsolete around BC/DR, as organizations increasingly gravitate towards multi-cloud solutions. According to the study, businesses utilizing a multi-cloud architecture model have many advantages, such as cost reduction, enhanced data agility, and accelerated innovation. There are 940 academic papers and research journals reviewed by the authors, among them only 88 are accepted to be studied further to contribute to the final SLR. The research identifies three distinct categories of cloud-native applications: those replicate from one CSP to another, the one fully distributed and leveraging multiple CSPs, and the hybrid model that combines both. Notably, the study states heterogeneity as a significant challenge in the context of application running in multi-cloud environments. Furthermore, the provisioning and management of cloud-native applications in this complex setting can be difficult. The authors found a notable research gap in this domain, which we will conduct further exploration and investigation in this paper.

Table 1 - *Related BC/DR work with performance evaluation*

Authors and Ref.	Research Title	RTO	RPO	Cost
Alshammari, M.M. *et al.* (2017)	Disaster recovery in single-cloud and multi-cloud environments: Issues and challenges	No	No	No
Lavriv, O. *et al.* (2018)	Method of cloud system disaster recovery based on "Infrastructure as a code" concept	Yes	No	No
Harwalkar, S., Sitaram, D. and Jadon, S. (2019)	Multi-cloud DRaaS using OpenStack Keystone Federation	No	No	No
Abualkishik, A.Z., A., A. and Gulzar, Y. (2020)	Disaster Recovery in Cloud Computing Systems: An Overview	No	No	No
Stamenkov, G. (2022)	Layered business continuity and disaster recovery model	No	No	No
Liu, B., Xin, Y. and Zhang, C. (2022)	A Solution for A Disaster Recovery Service System in Multi-cloud Environment	No	No	No
Nikolovski, S. and Mitrevski, P. (2022)	On the Requirements for Successful Business Continuity in the Context of Disaster Recovery	No	No	No
Pruthvi Kumar Reddy, D. *et al.* (2022)	Encrypted Cloud Storage Approach For A Multicloud Environment Using Fog Computing	No	No	No
Alonso, J. *et al.* (2023)	Understanding the challenges and novel architectural models of multi-cloud native applications – a systematic literature review	No	No	No

Upon reviewing **Table 1**, which comprehensively listed all the studies incorporated in the literature review, we find a notable absence of BC/DR performance measurement details that provided by the authors in the previous research. Metrics such as RPO, RTO, and cost are crucial for assessing the performance of the proposed BC/DR solution. The identified research gap is that the previous studies focus more on theoretical validation rather than the concrete implementation of robust systems. Therefore, we propose to design and develop a multi-cloud BC/DR solution and provide a comprehensive list of performance measurements. The proposed solution holds the potential of improving RTO, enabling continuously business operation with less down time. Additionally, it also could mitigate the risk of vendor lock-in, and possibly cost-saving benefits.

3 Research Methodology

The primary research interest centres around the question of "*Can the Infrastructure as Code tool, such as Terraform, contribute to the reduction of a cloud-native application's recovery time during a crisis? Furthermore, to what extent this reduction in recovery time can be achieved?*" To address this question, the proposed solution involves leveraging Terraform to provision and manage multiple cloud infrastructures across different CSPs, providing a robust environment for the cloud-native application to run. In this setup, one cloud infrastructure serves as the primary platform, while others function as fallbacks in the event of disasters. So, what is Terraform?

Terraform is a widely adopted IaC tool developed by HashiCorp, which utilizes its own proprietary HashiCorp Configuration Language (HCL) to define resources, whether in the cloud environment or locally on-premises infrastructure. Leveraging Terraform HCL can have the same benefits as any other programming language, such as versioning control in repositories like GitHub, code reusability for multiple cloud infrastructures, and shareability between different teams. The integration between DevOps workflow tools like GitHub Action and Terraform can be seamless, allowing triggering automated builds from various Terraform code branches. It also facilitates automated deployment and update to different cloud environments. Additionally, Terraform is compatible with all the major CSPs, such as AWS, Azure, and Google Cloud (Terraform 2023).

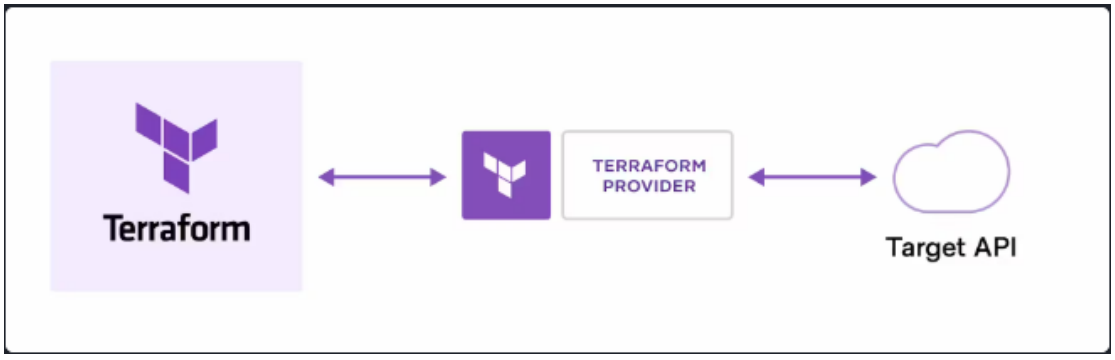

Figure 2. Terraform Provider. Source: Terraform 2023

For provisioning cloud resources, Terraform makes the call to the target CSP's Application Programming Interface (API), with its core components called Terraform providers, which serve as the communication bridge to the CSP API, see Figure 2. There are extensive number of Terraform providers available, contributed by both HashiCorp and the Terraform community (Terraform 2023).

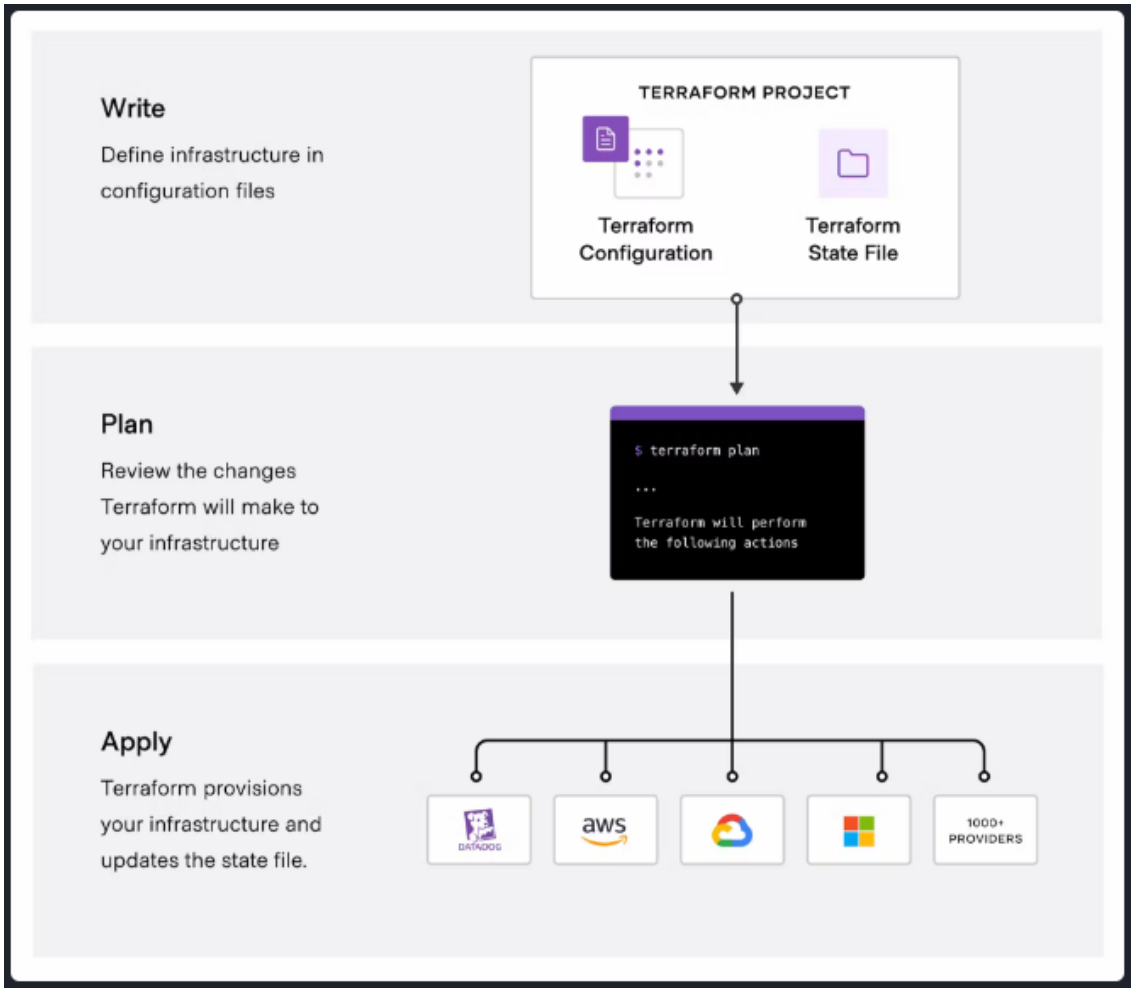

Figure 3. Terraform Workflow. Source: Terraform 2023

To provision infrastructure with Terraform, there are three main steps, illustrated in Figure 3:

- Write: Define cloud resources using Terraform HCL configuration file.
- Plan: Run "terraform plan" command to generate a detailed plan, to add/delete/update to the cloud infrastructure.
- Apply: Use the plan created in the previous step and run "terraform apply" command, Terraform will perform the execution of instructions and make changes to the cloud infrastructure.

While Terraform orchestrates the cloud infrastructure, the cloud-native application itself is deployed using a pre-built docker image. Docker container is the main component of the cloud-native application architecture. It allows the application run seamlessly across different CSP platforms without any additional configuration.

A docker container is a standalone unit of stripped-down version of the operating system and the software application combined. It includes all software dependencies required inside the container and can run in any computing environments support docker. See Figure 4

Docker Architecture. All major cloud CSPs support running docker as a service, including AWS Elastic Container Service, Azure Container Instance and Google Cloud Run.

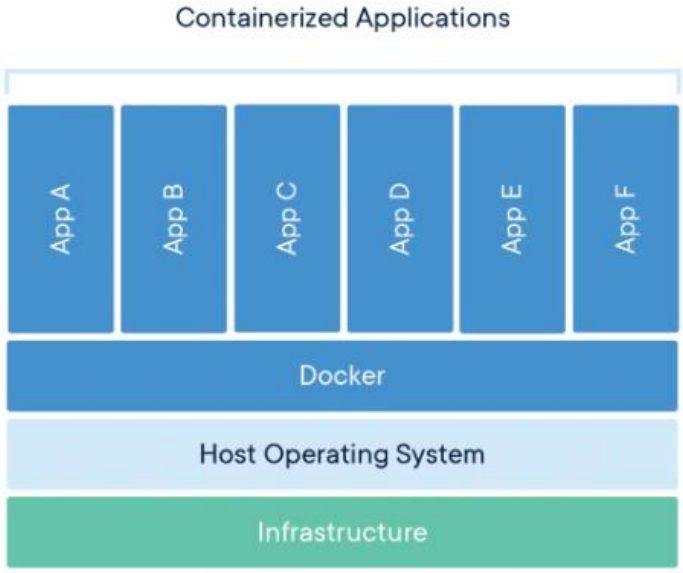

Figure 4. Docker Architecture. Source: Docker 2023

4 Design Specification

The design adopted in this research project is different from the conventional multi-cloud architecture model. While many multi-cloud applications typically leverage the distributed architecture, where various cloud resources from different CSPs are provisioned and utilized simultaneously, this project takes another approach. Typically, only one CSP environment is active at any given time, with the others remaining in a fallback state. The fallback state means no resources are provisioned in that environment, but are ready to be created when required, as illustrated in Figure 5. Within the active live AWS environment, many components are provisioned in the cloud infrastructure, such as a Virtual Private Cloud (VPC), three public subnets in different Availability Zones (AZ), three Elastic Container Service (ECS) clusters, an application load balancer, a Route 53 Domain Name Service (DNS), and possibly a Relational Database Service (RDS) database. In the fallback environment, Azure or Google Cloud, there are no resources provisioned as demonstrated in Figure 5, except for the presence of a replicated database.

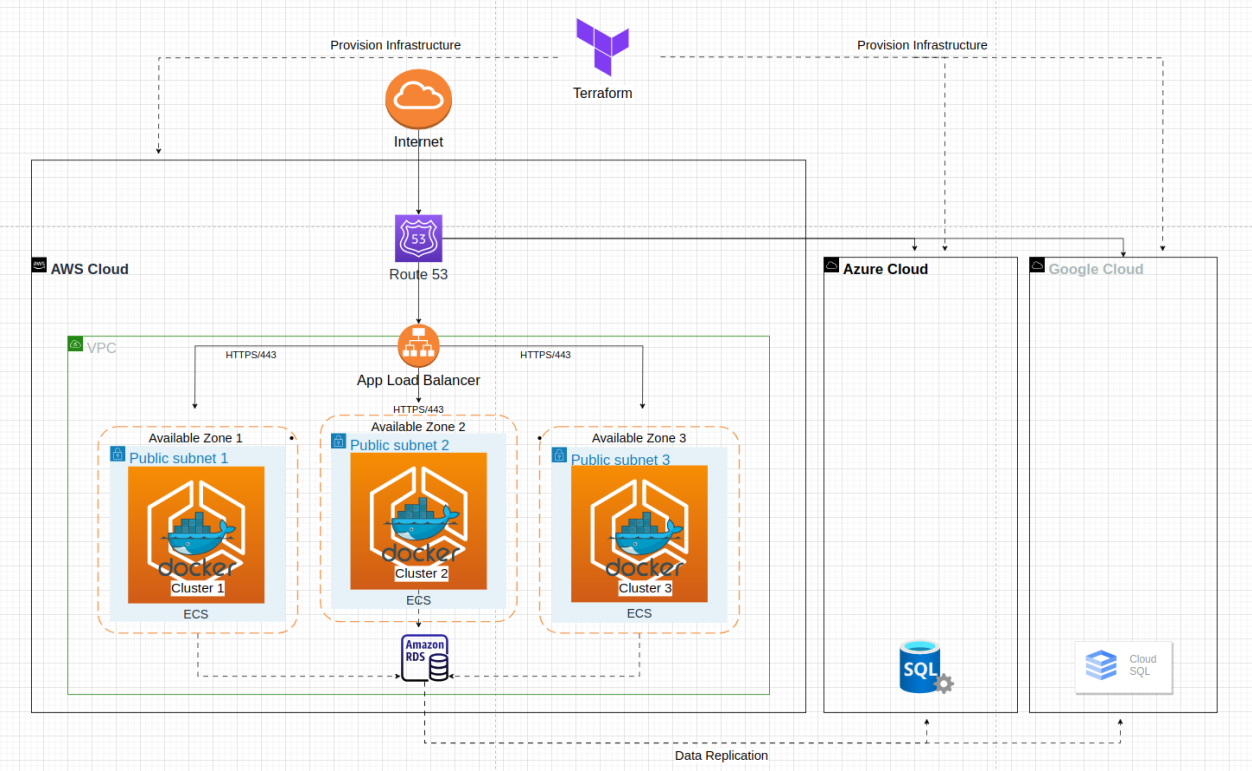

Figure 5. Multi-Cloud Architecture Diagram

In the event of errors being detected within the live AWS infrastructure through monitoring and alerting software, another cloud environment can be provisioned inside Azure or Google Cloud by Terraform. The switch of Route 53 DNS to the newly deployed application environment is carried out quickly to facilitate a seamless transition. Once the new environment is live, Terraform can perform the deletion of the old infrastructure to optimize cost. This design specification ensures a swift response process to potential cloud infrastructure failures in the active CSP, reducing the time required for the disaster recovery procedure, and mitigating the risk of the system having any downtime.

5 Implementation

The implementation focuses on the orchestration of the infrastructures using Terraform within the multi-cloud environments. Terraform HCL code is written for each of the cloud infrastructures, tailored to the specific requirements of the selected CSPs, such as AWS and Azure. AWS hosts the live primary platform, while Azure serves as a redundant fallback infrastructure. All Terraform HCL code is committed to the GitHub repository, which has many benefits, such as version control, change tracking, easy collaboration, and automation.

Figure 6. Terraform graph for AWS resources

Figure 6 illustrates the Terraform configuration and execution graph for the implemented AWS infrastructure. The visual representation highlights the dependencies for each of the AWS cloud components in the live environment. During the execution of the generated plan, Terraform traverses this graph from the top to the bottom and ensures all dependencies are met before provisioning any resources.

The following Terraform modules are implemented for AWS environment:

- main.tf - The main Terraform file, defines the `required_providers` and `provider` configurations.

- vpc.tf - The network configuration file, defines the `aws_vpc`, 3 `aws_subnet`, network `aws_security_group`, `aws_internet_gateway`, `aws_route_table`, and `aws_route_table_association`.
- alb.tf - The application load balancer configuration file, defines the `aws_alb`, `aws_alb_target_group`, `aws_alb_listener`, and the load balancer `aws_security_group`.
- ecs.tf - The ECS container cluster service configuration file, defines the `aws_ecs_cluster`, `aws_ecs_task_definition`, `aws_ecs_service`, ECS Identify Access Management (IAM) role `aws_iam_role`, and `aws_iam_policy_attachment`.
- variable.tf - The Terraform variable file, defines a list of hard-coded values, which can be used when running Terraform code.
- output.tf - The Terraform output file, defines a list of calculated output that can be used for testing and further automation.

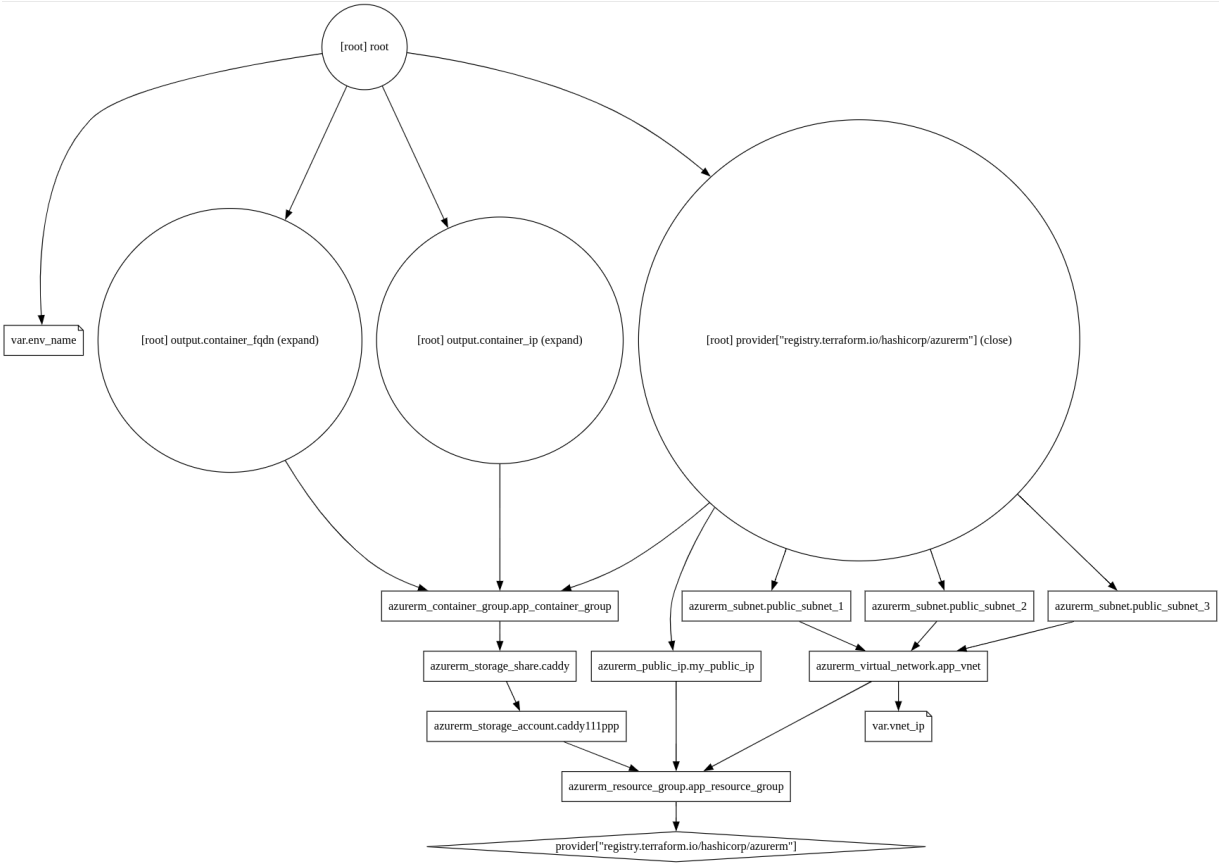

Figure 7. Terraform Graph for Azure Resources

Figure 7 illustrates the Terraform configuration and execution graph for the implemented Azure infrastructure. The visual representation highlights the dependencies for each of the Azure components in the fallback environment.

The following Terraform modules are implemented for Azure:

- main.tf - The main Terraform file, defines the `required_providers` and `provider` configurations.

- vnet.tf - The network configuration file, defines the `azurerm_resource_group`, `azurerm_virtual_network`, and 3 `azurerm_subnet`.
- container.tf - The docker container configuration file, defines `azurerm_container_group`, `azurerm_public_ip`, `azurerm_storage_account`, and `azurerm_storage_share`.
- variable.tf - The Terraform variable file, defines a list of hard-coded values, which can be used when running Terraform code.
- output.tf - The Terraform output file, defines a list of calculated output that can be used for testing and further automation.

6 Evaluation

In this section, we outline more details about the evaluation process of the proposed BC/DR solution. As previously discussed, the assessment will centre around one key performance indicator – RTO. This primary indicator will be validated in the context of two different scenarios:

- Test Scenario 1 – AWS single-cloud DR manual process vs automated Terraform script. In this scenario, it involves testing the BC/DR solution's performance in a single-cloud environment within AWS. A comprehensive analysis between the manual DR process and the automated Terraform script will be conducted.
- Test Scenario 2 – AWS and Azure multi-cloud DR manual process vs automated Terraform script. In this scenario, it extends the evaluation to a multi-cloud environment, including both AWS and Azure. Like the first test, a comprehensive assessment between the manual DR process and the automated Terraform script will be conducted.

These test scenarios have been designed to comprehensively assess the effectiveness of the proposed BC/DR solution and try to address the research questions accordingly.

6.1 Test Scenario 1 – Manual vs Terraform Automatic DR in Single-Cloud AWS Environment

In test scenario 1, we conduct a comparison between manual process and automated script using Terraform for the orchestration of a single-cloud AWS infrastructure. To carry out this test, the assumption of a typical DR scenario is made, such as the entire infrastructure needs to be recreated due to errors. To ensure the same starting point across all tests, all resources within the AWS cloud infrastructure are manually deleted or destroyed through the Terraform destroy command before each experiment.

The manual test is carried out by using the AWS console to create each resource individually. On the other hand, the automated experiment involves the execution of Terraform

HCL configuration code for orchestrating the whole infrastructure. After each test, essential metrics such as RTO is recorded for comparison later. Furthermore, the test complexity is increased gradually by adding the requirement of provisioning more cloud-native applications to form a highly available and fault tolerant Docker cluster.

The primary objective of this test is to evaluate the effectiveness of IaC tool such as Terraform, in reducing disaster recovery time within a single-cloud environment and calculate how much time it can save. The findings are demonstrated in Figure 8.

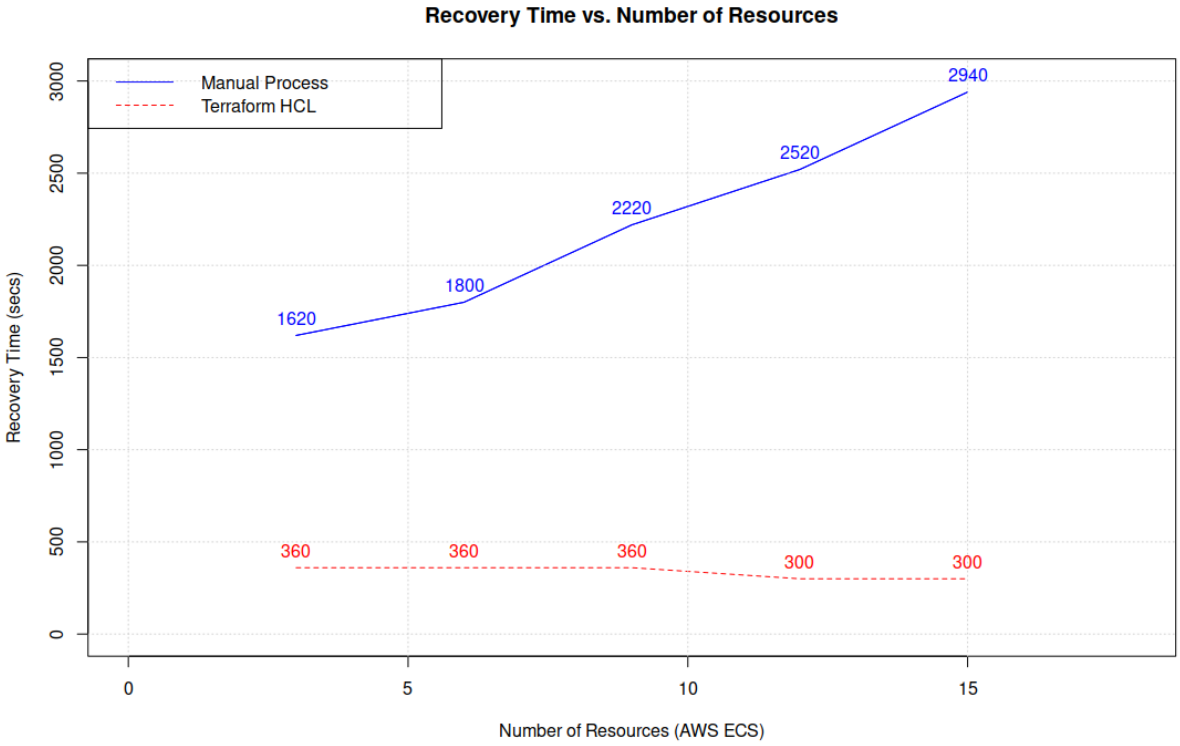

Figure 8. Single-Cloud DR in AWS

Figure 8 illustrates the correlation between the complexity of the cloud infrastructure to be recovered and the duration the recovery process. The complexity is represented by the total number of resources requiring recovery. The blue line represents the manual recovery process using the AWS console, it shows a significant increase in recovery time as the number of resources grows. In contrast, the red line represents the automated infrastructure orchestration process using Terraform, it shows a more consistent and modest decrease in recovery time as the number of resources increase. Therefore, Terraform not only proves efficient in reducing DR time in a single-cloud environment for scenarios with small number of resource but also demonstrates even more time saving for larger amounts of resources. On average, Terraform could potentially save up to 85% of the time required for manual infrastructure configuration.

6.2 Test Scenario 2 – Manual vs Terraform Automatic DR in Multi-Cloud Environments (AWS & Azure)

In test scenario 2, we carry out a comparison between manual processes and automated Terraform scripts for orchestrating infrastructure in a multi-cloud environment (AWS & Azure). In contrast to the previous single-cloud disaster recovery test scenario, the main difference is to provision a new DR infrastructure in Azure when the primary AWS environment encounters issues. The same as test scenario 1, the assumption is made that the entire infrastructure requires to be recreated due to unforeseen errors. To ensure consistency across all tests, the resources within the Azure cloud infrastructure are manually deleted or destroyed using the Terraform destroy command before each experiment.

Like test scenario 1, the manual experiment is carried out by using the Microsoft Azure portal to create each resource individually. On the other hand, the automated test involves executing Terraform HCL configuration code for provisioning the whole infrastructure. After each test, key metrics such as RTO is recorded for later comparison. Furthermore, like the single-cloud experiment, the test complexity is increased gradually by adding the requirement of provisioning more cloud-native applications to form a highly available and fault tolerant Docker cluster.

Similar to the previous single-cloud scenario, the key objective of this test is to assess whether an IaC tool like Terraform can effectively reduce disaster recovery time in a multi-cloud environment and calculate how much time it can save. The findings are presented in Figure 9.

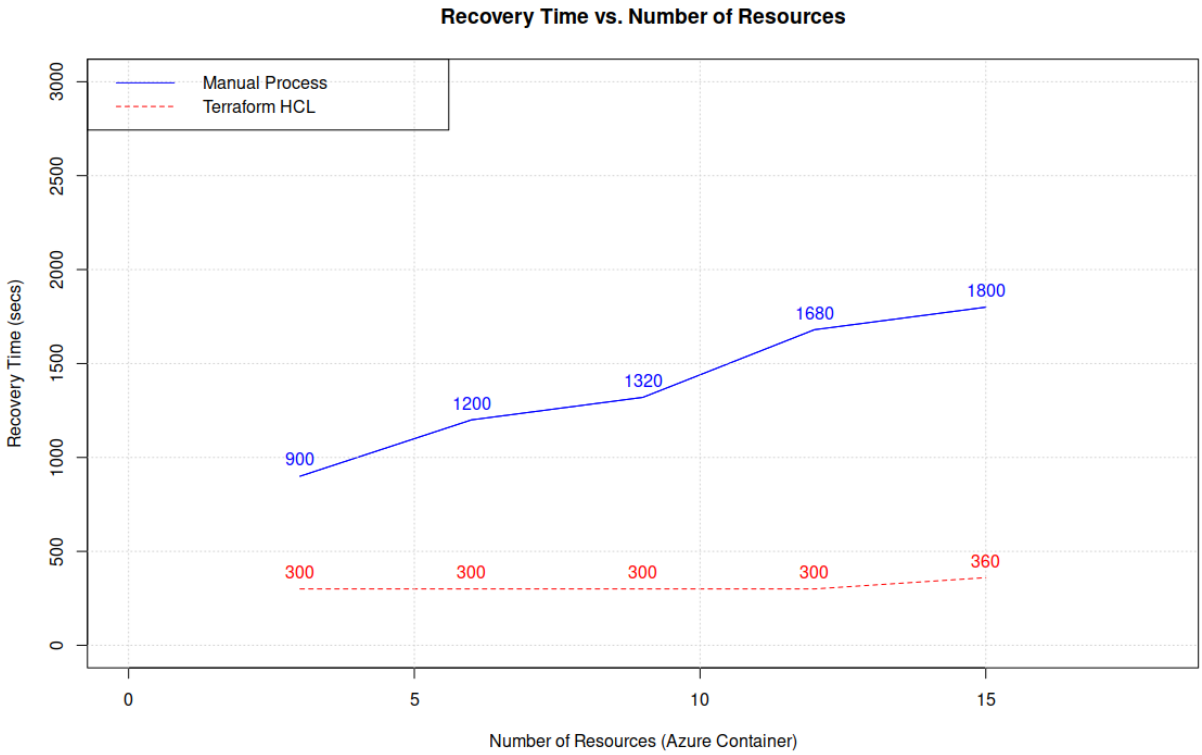

Figure 9. Multi-Cloud DR in AWS & Azure

Figure 9 illustrates the relationship between the complexity of the cloud infrastructure to be recovered and the duration of the recovery process. The complexity is measured by the total number of resources requiring recovery. The blue line represents the manual recovery process utilizing the Microsoft Azure portal, it demonstrates a significant increase in recovery time when the number of resources grows. In contrast, the red line represents the automated infrastructure orchestration process using Terraform, it indicates a more consistent and modest increase in recovery time as the number of resources grows. Therefore, Terraform demonstrates its efficiency not only in reducing DR time in a multi-cloud environment for test scenarios with less resource but also achieving even more time saving for larger infrastructure. On average, Terraform could potentially save up to 75% of the time required for manual infrastructure configuration.

6.3 Discussion

The primary objective of the evaluation of the test scenarios is to assess the effectiveness of the proposed BC/DR solution. In particular, the tests focus on the orchestration of single-cloud AWS infrastructure and multi-cloud AWS & Azure environments using Terraform. The findings provide good insights into the performance of the IaC tool Terraform and offer an opportunity for critical analysis on the experimental design.

The results from test scenario 1 demonstrate the significant advantages of using Terraform for disaster recovery in a single-cloud environment. With the number of cloud-native applications increase, Figure 8 shows a noticeable increase in recovery time with the manual process but a consistent and almost flat line for the automated process with Terraform. The average time saving of 85% using Terraform compared with the manual process is a great achievement. However, the design of test scenario 1 isolates the impact of Terraform only in a controlled environment. The test assumes the entire AWS infrastructure requires recreation due to some errors. In a real-world scenario, to replace the whole infrastructure because of errors from some components is not feasible in a single-cloud architecture. Most likely, the failed components are replaced rather than the entire environment. Furthermore, simply replacing the failed components may not be enough to fix the unforeseen DR issue, in situation such as a Route 53 DNS problem with AWS. Test scenario 1 highlights the advantages of using Terraform, but not enough to address the research questions.

Test scenario 2 introduces the complexity of orchestrating DR infrastructures across multi-cloud environments. Similar to test scenario 1, Figure 9 reveals the noticeable advantages of using Terraform in a multi-cloud DR situation, especially in scenarios with a large number of cloud-native applications. It shows the more resources required for recovery, the more time Terraform can save compared to manual configuration. While the test design assumes the while infrastructure requires recovery, but in a real-world scenario, maybe only a small component of the environment needs to be replaced. This test should be treated as the last defence, and only triggering the DR process while the primary cloud infrastructure cannot be recovered.

Both test scenarios focus on the cloud-native application's disaster recovery time – RTO. The associated cost of the recovered environment (AWS or Azure) and RPO are not recorded due to the time constraints of this project.

7 Conclusion and Future Work

The ability to recover from disasters and continue business operations is critical for any organisation. Finding the quickest and the most reliable solution to recover the failed system can be challenging. To overcome these challenges, we conducted a comprehensive literature review of the existing research papers on the topic of BC/DR and proposed a more efficient solution utilizing the IaC tool Terraform. In this paper, we tried to address the research questions: Can the IaC tool such as Terraform contribute to the reduction of disaster recovery time for cloud-native applications? We evaluated the proposed solution in two test scenarios, including single-cloud AWS infrastructure and multi-cloud (AWS & Azure) environments. Our study showed a significant advantage of leveraging the IaC tool Terraform in both single-cloud and multi-cloud environments. In the conducted tests, Terraform demonstrated a potential reduction in disaster recovery time of 75% or more compared to the manual process. The test results were consistent and had a modest increase or decrease in DR time when more resources require recovery. Therefore, these key findings successfully answered the research question and proved that Terraform can be used as a robust and efficient DR solution in varies of cloud configurations and environments. However, it is important to knowledge the limitations of the experimental design. The assumption of the need to recreate the whole infrastructure after each test may oversimplify the real-world scenarios. Additionally, only focus on the DR recovery time RTO, while other critical metrics such as RPO and cost are not measured due to time constraints.

More time could be spent in future studies on the cost implications of leveraging multi-cloud environments. Comparing the associated cost of identical infrastructure within different CSPs can provide valuable insights for businesses. Understanding the cost of cloud resources from different cloud providers can be crucial for enterprises to decide on resource allocation and cost-efficient DR solutions. RPO is a key performance indicator for the BC/DR solution, future work should explore different options to improve RPO in multi-cloud environments, especially in minimizing data loss for the cloud-native application during the disaster recovery process. Additionally, more real-world scenarios should be considered, rather than assuming the entire infrastructure needs replacement.

References

Abualkishik, A.Z., A., A. and Gulzar, Y. (2020) 'Disaster Recovery in Cloud Computing Systems: An Overview', *International Journal of Advanced Computer Science and Applications*, 11(9). Available at: https://doi.org/10.14569/IJACSA.2020.0110984.

Alonso, J. *et al.* (2023) 'Understanding the challenges and novel architectural models of multi-cloud native applications – a systematic literature review', *Journal of Cloud Computing*, 12(1), p. 6. Available at: https://doi.org/10.1186/s13677-022-00367-6.

Alshammari, M.M. *et al.* (2017) 'Disaster recovery in single-cloud and multi-cloud environments: Issues and challenges', in *2017 4th IEEE International Conference on Engineering Technologies and Applied Sciences (ICETAS)*. *2017 4th IEEE International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, Salmabad: IEEE, pp. 1–7. Available at: https://doi.org/10.1109/ICETAS.2017.8277868.

CNCF Survey. (2020) 'Use of containers in production has increased by 300% since 2016', Available at: https://www.cncf.io/reports/cloud-native-survey-2020/ [Accessed 16 April 2023].

Docker. (2023) 'Use containers to Build, Share and Run your applications', Available at: https://www.docker.com/resources/what-container/ [Accessed 07 December 2023].

Harwalkar, S., Sitaram, D. and Jadon, S. (2019) 'Multi-cloud DRaaS using OpenStack Keystone Federation', in *2019 International Conference on Advances in Computing and Communication Engineering (ICACCE)*. *2019 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, Sathyamangalam, Tamil Nadu, India: IEEE, pp. 1–6. Available at: https://doi.org/10.1109/ICACCE46606.2019.9080005.

Lavriv, O. *et al.* (2018) 'Method of cloud system disaster recovery based on "Infrastructure as a code" concept', in *2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET)*. *2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET)*, Lviv-Slavske, Ukraine: IEEE, pp. 1139–1142. Available at: https://doi.org/10.1109/TCSET.2018.8336395.

Liu, B., Xin, Y. and Zhang, C. (2022) 'A Solution for A Disaster Recovery Service System in Multi-cloud Environment', in *2022 International Applied Computational Electromagnetics Society Symposium (ACES-China)*. *2022 International Applied Computational Electromagnetics Society Symposium (ACES-China)*, Xuzhou, China: IEEE, pp. 1–4. Available at: https://doi.org/10.1109/ACES-China56081.2022.10064903.

Nikolovski, S. and Mitrevski, P. (2022) 'On the Requirements for Successful Business Continuity in the Context of Disaster Recovery', in *2022 57th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*. *2022 57th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*, Ohrid, North Macedonia: IEEE, pp. 1–4. Available at: https://doi.org/10.1109/ICEST55168.2022.9828701.

Pruthvi Kumar Reddy, D. *et al.* (2022) 'Encrypted Cloud Storage Approach For A Multicloud Environment Using Fog Computing', in *2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA)*. *2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA)*, Gunupur, India: IEEE, pp. 1–6. Available at: https://doi.org/10.1109/ICCSEA54677.2022.9936544.

Stamenkov, G. (2022) 'Layered business continuity and disaster recovery model', *Continuity & Resilience Review*, 4(3), pp. 267–279. Available at: https://doi.org/10.1108/CRR-05-2022-0008.

Terraform. (2023) 'What is Terraform?', Available at: https://developer.hashicorp.com/terraform/intro [Accessed 16 April 2023].