National College of Ireland

# Designing a Robust Multi-Cloud DevOps Strategy for Kubernetes Disaster Recovery: An Open Source Approach

MSc Research Project
Cloud Computing


Shashank Arvind Gokhale
Student ID: 21226661

School of Computing
National College of Ireland

Supervisor: Dr. Rashid Mijumbi

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

National College of Ireland

| | |
|---|---|
| **Student Name:** | Shashank Arvind Gokhale |
| **Student ID:** | 21226661 |
| **Programme:** | Cloud Computing |
| **Year:** | 2023 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Dr. Rashid Mijumbi |
| **Submission Due Date:** | 14/08/2023 |
| **Project Title:** | Designing a Robust Multi-Cloud DevOps Strategy for Kubernetes Disaster Recovery: An Open Source Approach |
| **Word Count:** | 5301 |
| **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:** | Shashank Arvind Gokhale |
| **Date:** | 14th August 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): | |

# Designing a Robust Multi-Cloud DevOps Strategy for Kubernetes Disaster Recovery: An Open Source Approach

Shashank Arvind Gokhale
21226661

**Abstract**

This study focuses on designing a DevOps centered strategy for a resilient multi-cloud disaster recovery. It targets managed Kubernetes clusters deployed on Google Cloud Platform (GCP) and Azure, utilizing open-source tools to enhance flexibility, mitigate vendor lock-in, and streamline the disaster recovery process. A CI/CD pipeline is established to increase the efficiency of disaster recovery across cloud providers. The research evaluates Recovery Time Objectives (RTO) and Recovery Point Objectives (RPO) for containerized applications rolled out on GCP & Azure, specifically in scenarios involving software upgrades and area wide power outages/datacenter failures. The aim is to offer an alternative to cloud specific tools, enabling migration and restoration across cloud service providers post disaster. The developed pipeline explores variations in RTO & RPO amongst distinct cloud providers for the same application stack. The findings reveal GCP's efficiency in restoring applications after a system upgrade failure, while Azure demonstrates quicker migration and restoration capabilities in datacenter failure scenarios.

**Keywords -** Kubernetes, Disaster Recovery, DevOps approach, google kubernetes engine, Azure kubernetes service, velero.

# 1 Introduction

In today's dynamic business landscape, the ability to swiftly develop, deploy, and scale applications is paramount to an organization's success. Microservices architecture for applications with Kubernetes (K8s) container orchestration is a proven approach to achieve scalability, agility, and service iteration capabilities to meet these demands (Burns et al. (2022)). Moreover, when this approach is practised on the cloud it can significantly benefit a business by offering unparalleled scalability, resource elasticity, inherited global reach and cost-efficiency by its pay-as-you-go model.

As many organizations are adopting this microservices approach for application deployment there is a need for a robust disaster recovery (DR) process to ensure business continuity in the event of a disruption. These DR scenarios can be caused due to various factors such as natural calamities, underlying system hardware failure, human errors while configuring the systems, and cybersecurity incidents such as ransomware, phishing, or malware attacks. Recovery Time Objective (RTO),

Recovery Point Objective (RPO), and expenses associated are key considerations in DR planning.

Cloud service providers (CSP) offer disaster recovery as a service (DRaaS) to restore the services of an organization during a time of disruption. But these solutions lack flexibility and customization option for DR as these services are designed to work in specific workflow patterns which might not perfectly align with the organization's requirements. Enterprises utilizing this offering increase their dependence on CSPs subjecting themselves to potential risks, as any issues or disruptions on the provider's side may impact the effectiveness and reliability of the DR process. Particularly, this is true for certain specialized industry sectors which are subjected to stringent data privacy regulations and are required to retain their data within specific geographic location. The possible workaround for these companies is to have a multi-cloud DR strategy to ensure their data stays in the same region i.e., migrate all resources to another CSP in the same region. Furthermore, using DRaaS provided by cloud providers enables vendor lock-in making migration difficult to different CSP. Also, there are limitations to the cloud services covered by DRaaS and involve recurring subscription charges resulting in an expensive DR strategy.

Cloud computing is based on shared resource model (Buyya et al. (2009)) where the underlying hardware resources are pooled and allocated dynamically to multiple tenants. Cloud users interact with virtualized resources that are abstracted from the physical infrastructure. However, it's important to note besides CSP's aim to present a consistent and standardized interface to users, there exists difference in physical infrastructure such as variations in processor types, clock speeds, memory configurations, and storage technologies. These variations can impact performance and might lead to subtle performance differences between instances. Studying how these changes in performance impacts a multi-cloud DR process is also important in selecting which cloud provider to be chosen as the primary and secondary sites for DR.

The current state of kubernetes DR on cloud platforms is limited to using cloud provider's tool enabling vendor lock-in or rely on expensive third party DR softwares. Open-source tools offer a compelling solution to this challenge, enabling businesses to achieve tailor made DR for there requirements while avoiding vendor lock-in and financial burden. But these are harder to implement due to their complexity and there is a need to develop a strategy in which these tools can be seamlessly integrated reducing the overall complexity of DR implementation.

## 1.1 Research Problem

How can we create a robust and effective DevOps strategy for multi-cloud disaster recovery of containerized applications on Kubernetes clusters, using open-source tools? Also, what differences in Recovery Time Objectives and Recovery Point Objectives can we observe among different cloud providers?

## 1.2 Motivation

In the realm of cloud computing, we believe this DevOps-centric approach for multi-cloud kubernetes disaster recovery using open-source tooling can bring benefits as outlined below:

1. Provides a comprehensive methodology to implement multi-cloud DR for Kubernetes utilizing platform-agnostic approach.
2. Mitigates risks associated with vendor lock-in.
3. Facilitates the comparison of variations in RTO & RPO among different CSP, assisting researchers and developers in making well-informed decisions when selecting a CSP for their DR requirements.

## 1.3 Structure of the Document

Following is an overview of the work's content, In Section 2, the literature regarding DR is reviewed to assess current state of the art and identify key performance indicators for DR. Section 3, describes research methodology, scope of experimentation, and results acquisition strategy. Section 4, delves into open-source tools utilized and involves high-level architecture of the proposed solution. Section 5, shows the final outputs of the successfully implemented DevOps-centric approach for multi-cloud kubernetes DR. Section 6, provides an in-depth analysis of the results gathered from experiments conducted. Section 7, summarizes all the research outcomes and offers recommendations for future work.

# 2 Related Work

This section delves into the significance of disaster recovery within information systems, aiming to secure business continuity. It examines the endeavours of various researchers in tackling challenges to enhance disaster recovery performance metrics.

## 2.1 Crucial Role of Disaster Recovery for Business Continuity

Having a robust disaster recovery strategy is of utmost importance, capable of enduring disruptions caused by natural or human-induced events. This forms a fundamental cornerstone that dictates the company's ability to either thrive or falter in the aftermath of a disaster. According to Gartner reports, worldwide public cloud end-user spending is to reach about $600 billion in 2023[1], showing a 21.7% growth from the previous year. It predicts that By 2026, 75% of organizations to adopt a digital transformation model based on the cloud as a fundamental platform.

However, In 2021, several significant cloud outages underscored the vulnerabilities of relying solely on single cloud provider[2]. On December 7th, AWS us-east-1 experienced an outage affecting many crucial services. Google Cloud faced a networking glitch on November 16th. AWS EU-Central region encountered an outage on June 10th due to temperature-related connectivity issues. The largest Europe-based cloud provider, OVHcloud, saw its data centre burn down on March 10th. According to Andrade et al. (2017) Half of the businesses that encounter significant data loss resulting from disasters cease operations within 24 months, while a staggering 93% cease operations within a span of 5 years. This shows that having a robust disaster recovery plan plays a crucial role in business continuity.

[1]Gartner.com
[2]Cast.ai

## 2.2 Disaster Recovery Performance Metrics

RTO and RPO are considered to be the primary performance metrics of disaster recovery efficiency (Suguna and Suhasini (2014)). RTO measures the time required to restore the system to normal order after a disruption. Whereas RPO measures the time between the last successful data backup before the disruption till the disaster, thus providing maximum data which can be lost after the successful restoration. The other important metric is the financial expense involved in performing DR (Alhazmi and Malaiya (2012)). It is very important for a company to declare the SLAs (Service Level Agreements) and SLO (Service level objectives) based on these metrics to their customers to safeguard themselves from risks during the event of a catastrophe. Alhazmi and Malaiya (2012) gave an equation to calculate the cost of DR for an organization for a time period of one year.

The total cost ($C_T$) can be calculated as the sum of initial cost ($C_i$), operating cost ($C_o$), and expected annual costs for potential disasters ($C_d$):

$$C_T = C_i + C_o + C_d \quad (1)$$

The annual disaster cost ($C_d$) is given by:

$$C_d = \sum_i p_i(C_{ri} + C_{ui}) \quad (2)$$

where ($C_{ri}$) is the expected cost of disaster, ($C_{ui}$) is the unrecoverable disaster that is dictated by RTO. for each type of disaster $i$, ($p_i$) is the probability of its occurrence.

Alhazmi and Malaiya (2013) highlighted potential disaster triggers over a 5-year span, encompassing events such as system upgrades, power disruptions, fire, configuration change management, and cyber-attacks. But, System upgrades and power outages accounted for 72% and 70% respectively which were highest when compared with other events. Hence in this study, we will be mainly focusing on these two events to design our DR approach.

## 2.3 Evolution of Disaster Recovery Approaches

### 2.3.1 Traditional Approaches: Hot, Warm, and Cold Standby

Traditionally, DR strategies were categorized into three main approaches: hot standby, warm standby, and cold standby. Hot standby, as discussed by Wood et al. (2010) and Pokharel et al. (2010), involves maintaining mirrored, ready-to-activate servers in a secondary datacenter. This approach ensures near-zero Recovery Time Objective (RTO) and Recovery Point Objective (RPO) but comes at a higher cost due to dual active deployment. Warm standby strikes a balance between recovery speed and cost-effectiveness. It involves keeping applications in a "warm" state with partial deployment, offering flexibility in RPO through synchronous or asynchronous replication. On the other hand, cold standby, characterized by periodic data replication and longer RPO intervals, suits applications without stringent SLA requirements.

### 2.3.2 DRaaS a Cloud-Based Approach

The advent of cloud computing brought about a paradigm shift in DR approaches. DR as a Service (DRaaS), as explored by Wood et al. (2010), offers a cloud-based

solution that provides benefits such as cost savings, scalability, and flexibility. It facilitates seamless migration of systems to the cloud and ensures uninterrupted business operations. However, challenges related to data compliance, security, and vendor lock-in need to be addressed for widespread adoption.

### 2.3.3 Agile & DevOps Integration

Researchers like Baham et al. (2017) proposed agile methodologies for DR, emphasizing real-time updates, flexibility, and adaptability. These principles align with the agile development framework and emphasize continuous improvement. The integration of DevOps culture, as highlighted by Rajkumar et al. (2016), brings collaboration between development and operations teams, enabling efficient software delivery with stability. This cultural shift ensures rapid deployments, reduced failure times, and effective uptime.

### 2.3.4 Automation using Infrastructure as Code

Automation emerged as a key enabler in DR evolution. Lavriv et al. (2018) employed Infrastructure as Code (IaC) approach using tools like Terraform to automate disaster recovery processes. This approach significantly reduced RTO when compared to manual process and demonstrated the potential of automation in enhancing disaster recovery efficiency. but the proposed solution was limited to virtual machine instance recovery and a single cloud provider.

### 2.3.5 Multi-Cloud Strategy

As organizations embrace multi-cloud deployments for enhanced redundancy and availability, new challenges arise. Gallagher and Lennon (2022) advocated for deploying applications across multiple availability zones in a multi-cloud architecture. This approach minimizes disruptions during disasters and emphasizes the importance of deployment pipelines to streamline processes. However this study was limited to only utilizing cloud providers tools for DR which might result in vendor lock-in.

## 2.4 Disaster Recovery for Kubernetes

Kubernetes, an open-source container orchestration tool introduced by Google in 2014, was designed with a strong focus on improving the experience for developers working on cluster applications. Its core aim is to streamline the deployment and management of complex distributed systems while capitalizing on the efficiency benefits of containers. Additionally, it incorporates disaster recovery capabilities at its core (Burns et al. (2016)). Nonetheless, even meticulously designed Kubernetes clusters are susceptible to failures arising from factors like system upgrades or widespread outages. This reality has prompted numerous authors to turn to Velero, an open-source backup and restore tool designed for Kubernetes, to execute efficient disaster recovery procedures (Poniszewska-Marańda and Czechowska (2021)). Sameer et al. (2022) conducted a comparative study of disaster recovery tools for Kubernetes clusters, highlighting the significance of tool selection based on functionality and platform support according to them velero and kasten k10 were best amongst the bunch.

## 2.5 Comparative Analysis of Related work

In Table 1, related works have been compared based on their focus, benefit and challenges

| Papers | Focus | Benefit | Challenges |
|---|---|---|---|
| Wood et al. (2010) | DRaaS | Cost savings, flexibility, & scalability. | Compliance & data security, standardization, vendor lock-in, deployment complexity & expensive. |
| Pokharel et al. (2010) | Multi-site cloud DR | High availability low downtime. | Multi-cloud migration, intera-cloud traffic & deployment complexity. |
| Baham et al. (2017) | Agile methodology for DR | Rapid development, reduced failure times. | Leadership support, Lack of situational awareness. |
| Rajkumar et al. (2016) | DevOps Culture | Extended availability, faster deployments, & increased communication . | limited to cloud providers tools, Vendor lock-in, & multi-cloud deployments. |
| Larviv et al. (2018) | Terraform for DR | RTO improvement from automation. | Multi-cloud deployment not explored, & was limited to virtual machine instances on cloud. |
| D. Gallagher et al. (2022) | Multi-Cloud application DR | High availability, & deployment pipelines. | Utilized cloud providers tooling, vendor lock-in, & deployment complexity |
| Sameer, S et al. (2022) | Kubernetes DR Tools | Available tools feature comparison. | Multi-cloud DR was not explored, & DR was not automated |

Table 1: Comparison of Related Works

# 3 Methodology

From section 2, it is quite evident that there are challenges present such as complexity involved in designing & implimenting, vendor lock-in, multi-cloud data migration, lack of situational awareness & coordination between teams to implement a robust DR process. However, the proposed DevOps-centric approach utilizing open-source toolings for robust disaster recovery for Kubernetes helps mitigate these challenges. The failures caused by system upgrades and power outages/wide area outage is still a big issue and accounted for 72% & 70% respectively over a period of five years in a study conducted by Alhazmi and Malaiya (2013). hence we will be considering these two scenarios to be implemented and evaluated for our proposed solution.

In our research, we focus exclusively on the deployment of Kubernetes through managed clusters offered by cloud providers. This emphasis is aligned with the shared responsibility model, which mandates comprehensive patching and updates of the underlying hardware. Furthermore, the abstraction of hardware failures

and repairs to the end user contributes to the successful realization of our chosen scenarios.

## 3.1 Scenarios

We have selected namely two scenarios which are as follows:

1. Software Upgrade Failure
2. Area Wide Power Outage/ Datacenter Outage in a region.

The approach for both scenarios will be different but both will make use of pipelines for disaster recovery and will be deployed on the managed Kubernetes clusters. Additionally, the consistency of the application stack across both scenarios is paramount, as any deviation within this stack could potentially disrupt the accuracy of the DR timing data.

### 3.1.1 Scenario 1

Disruptions can arise from errors in the updated version of an application stack deployed to the Kubernetes cluster. These disruptions encompass heightened latency, resource shortages, pod malfunctions, conflicts in dependencies, and configuration anomalies. In such instances, a prompt rollback and restoration to the previous version, complete with its persistent data, become pivotal to safeguarding business continuity. In this scenario, the managed Kubernetes cluster remains unaffected and operational. However, the deployed application on nodes necessitates reverting to the previous version. Managing this necessity falls under the responsibility of the backup utility which is driven by a pipeline workflow, which will be elaborated upon in subsequent sections of this discussion.

### 3.1.2 Scenario 2

A datacenter outage within a specific region poses a significant threat to the entire application, causing disruptions in business continuity. This is primarily due to the complete unavailability of the underlying compute resources, which includes our managed Kubernetes cluster. In such circumstances, there are two potential courses of action: firstly, relocating the entire application to an alternate region within the same cloud provider; secondly, migrating to a different cloud provider altogether, this is driven by data privacy regulations necessitating user data to remain within the same geographic jurisdiction. To address this scenario, a meticulously crafted pipeline is required. This pipeline would orchestrate the allocation of infrastructure resources across diverse regions or alternate cloud provider. Once these resources are provisioned, the kubernetes backup utility comes into play recovering the application along with its persistent volumes, effectively restoring normal operational status for the application stack on the managed kubernetes cluster.

## 3.2 Architectural Overview

Figure 1 shows the architectural overview of the proposed approach. In which a continuous delivery pipeline will be created which when triggered performs a set of automated workflows for creating the required managed Kubernetes cluster on the cloud by making use of an infrastructure orchestrator whose configuration files are kept in a distributed version control system. After the managed kubernetes cluster is created it starts with the second stage of deploying the kubernetes backup tool

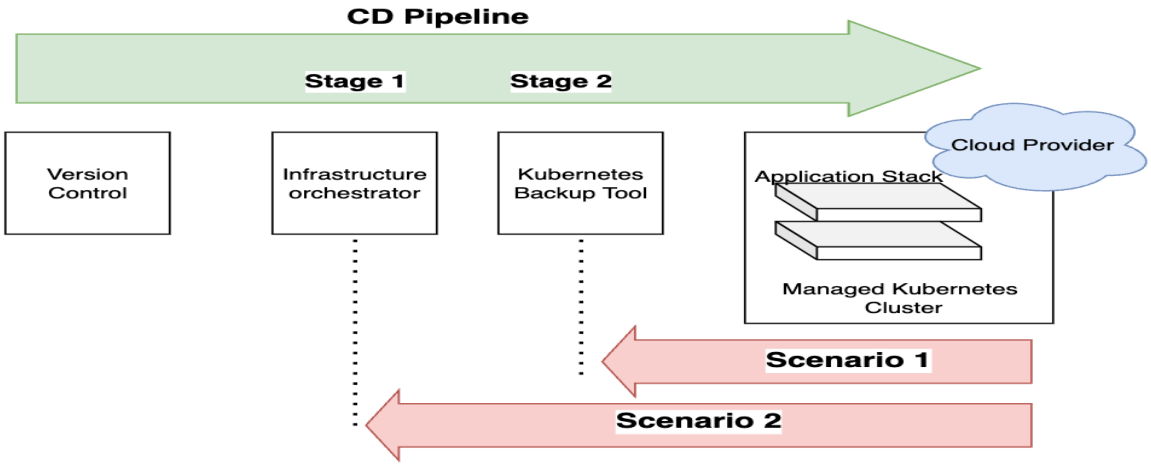



Figure 1: Architectural overview of the approach

on the managed cluster to manage the backup and restoration of the application stack. The version control system acts as the only source of truth for the processes running in the pipeline.

Distinct pipelines have been tailored to cater to the specific demands of scenario 1 and scenario 2. The pipeline dedicated to scenario 1 encompasses an automated workflow executed for the Kubernetes backup tool, solely focusing on restoring the application stack. Conversely, the pipeline designed for scenario 2 is a more comprehensive undertaking. It encompasses two pivotal stages: the orchestration of infrastructure via an infrastructure orchestrator, coupled with the subsequent utilization of the Kubernetes backup tool's workflow. Together, these stages facilitate the seamless migration of the application to an alternate region or a different cloud provider.

## 3.3 Results Acquisition

It is important to acquire consistent output for the designed pipelines for the disaster recovery process and to benchmark the efficiency of the disaster recovery process we will be considering the timing data for each iteration for the scenarios discussed. the number of test samples size required for having a confidence level of 95% can be given by the below equation (Israel (1992))

$$n_0 = \frac{Z^2 \cdot p \cdot q}{e^2} \tag{1}$$

where: $n_0$ is the required sample size, $Z$ is the critical z-score, $p$ is the estimated proportion of attribute present in the population, $q$ is $1 - p$, $e$ is the desired level of precision.

Population in our case is infinite and by assuming p=0.5 with $e \pm 5\%$ we get the total sample size required to be:

$$n_0 = \frac{(1.96^2 \cdot 0.5 \cdot 0.5)}{0.052^2} = 384.16 \tag{2}$$

The necessary sample size is considerably large, surpassing the allocated budget for provisioning cloud resources within the confines of the free trial. Additionally,

deploying these resources would incur significant expenses and time investment. Therefore, we will conduct 20 iterations for each scenario, & utilize this sample size to assess the outcomes. This evaluation will be conducted with a confidence interval of 95% and will be elaborated upon in the evaluation section of this report.

### 3.3.1 Collection of timing data for scenarios

The hardware infrastructure supporting managed Kubernetes clusters varies across regions even within the same cloud provider. Furthermore, different cloud providers might utilize entirely distinct underlying hardware. It's important to note that these hardware resources are shared among multiple tenents to increase resource utilization of cloud. Figure 2 illustrates the sequence of steps in the pipeline and

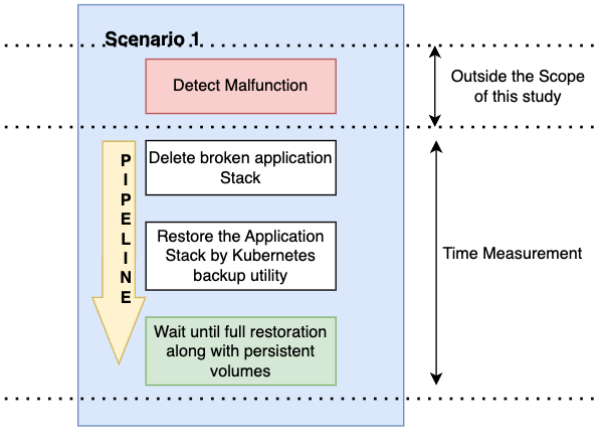



Figure 2: Time Measurement Scenario 1

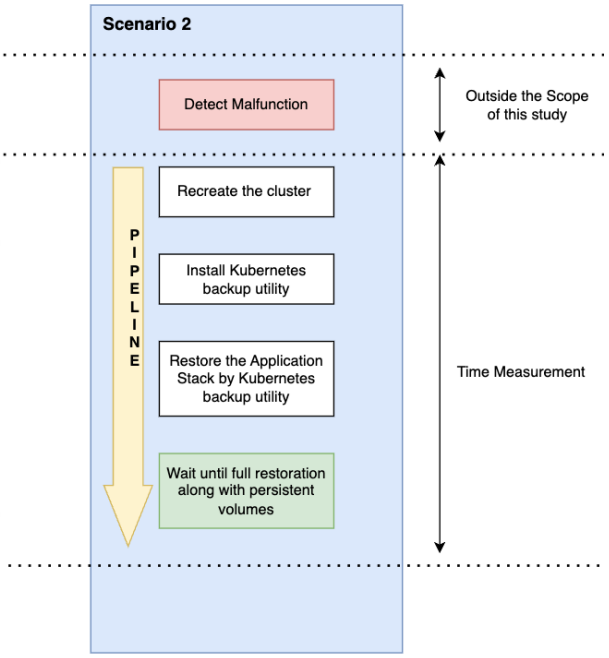



Figure 3: Time Measurement Scenario 2

outlines the timing measurement process for scenario 1. It's important to note that detecting malfunctions in containerized applications falls outside the scope of this project and can be addressed using open-source monitoring tools like Prometheus. The time measurement commences immediately when the pipeline initiates the recovery of the containerized application. The initial step involves removing the faulty application, followed by the restoration of the application to a specific point in time using the backup utility. Lastly, the pipeline awaits the containerized application to become operational and change its status to 'ready' before stopping the timer. This procedure is repeated 20 times to gather the necessary samples across different cloud providers.

Figure 3 illustrates the sequence of steps in the pipeline and outlines the timing measurement process for scenario 2. The time measurement for this pipeline initiates as soon as stage 1 workflow is triggered as shown in Figure 1 involving the infrastructure orchestrator which recreates the managed kubernetes cluster in the same cloud provider but different region or on another cloud provider altogether

depending on the requirement. the next step involves installing the kubernetes backup utility on the provisioned cluster after successful installation the point-in-time backup is restored on the cluster. Lastly, the pipeline awaits the containerized application to become operational and change its status to 'ready' before stopping the timer. similar to scenario 1 this procedure is repeated 20 times to gather the necessary samples to compare the differences between cloud providers.

# 4 Design Specification

## 4.1 Open-source Tools Utilized

### 4.1.1 GitHub Actions

GitHub Actions [3]is an automation platform provided by GitHub. we will make use of it to create DR pipelines to restore the application on the k8s clusters. When the pipeline is triggered the workflow file defined in yaml is executed on a managed runner instance which can be a virtual machine or a container. The steps declared in the workflow file can be run sequentially or parallel which will speed up the deployment time.

### 4.1.2 Terraform

To implement the methodology discussed in section 3, we have used Terraform[4] developed by hashicorp as the infrastructure orchestrator. Terraform is an open-source IaC tool which enables users to provision the infrastructure resources in a declarative and version-controlled manner to a plethora of cloud environments thanks to its plugin support. Moreover, its native support of state management ensures only new and modified resources are created reducing time for deployment.

### 4.1.3 Velero (formerly Heptio Ark)

Velero (formerly Heptio Ark)[5] is an open-source backup and disaster recovery tool designed for Kubernetes clusters. It is a cloud-native tool which takes advantage of K8s custom resource definitions (CRDs) to define backup specifications. It offers a reliable solution for backing up and restoring K8s resources and persistent volumes by storing the backups on object stores. The key benefits this tool offers are multi-cloud integrations by plugins, incremental backups, scheduling, and cluster migration support. Figure 4 shows the overall workflow of the backups in Velero. The steps involved are as follows:

1. Velero client makes a call to the k8s API server to create a backup object.
2. The BackupController notices the newly created backup object and performs validation.
3. The BackupController starts the backup process and collects the required data to backup by querying the API server for resources.
4. The BackupController makes a call to the object store and uploads the backup file.

[3]GitHub.com
[4]terraform.io
[5]velero.io

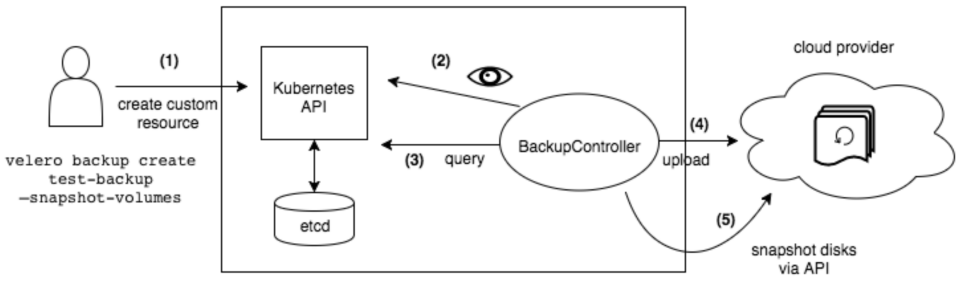



Figure 4: velero backup workflow (courtesy velero.io)

### 4.1.4 Azcopy

AzCopy[6]is an open-source command line utility developed by Microsoft. It is an ideal solution for a range of data management tasks, including cloud-to-cloud data migration, backup, content distribution, and synchronization. The newer version of AzCopy v10 supports direct transfer of data from cloud to cloud, not utilizing the mediators' bandwidth, thus saving on costs. In our case, AzCopy is utilized to seamlessly transfer restic backup data generated by velero which is stored on Google Cloud Storage (GCS) bucket to an Azure storage account. This process is orchestrated by a cron job that runs on GitHub Actions, ensuring regular and automated transfers.

## 4.2 High Level Overview of Proposed Architecture

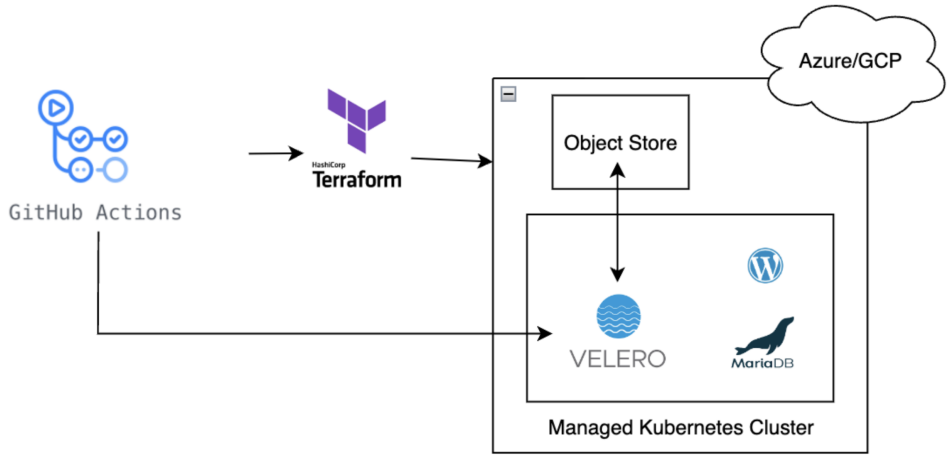



Figure 5: High level overview of setup architecture

In the context of our study, we have selected Azure and GCP as our designated cloud providers. Our strategy entails the utilization of GitHub actions to create a DR pipeline the first stage of the pipeline makes use of Terraform for the provisioning of K8s cluster resources on the cloud provider(Azure/GCP). Subsequent to the successful deployment of the cluster, the second stage of the pipeline starts the installation of Velero onto this cluster. This installation will be followed by

[6]Microsoft.com

the execution of the recovery procedure for the application stack. The restoration process and backup will make use of restic plugin i.e, the backup created by velero will be stored as a repository in the object storage infrastructure of the cloud provider (Google cloud storage[7] & Azure storage accounts[8]) rather than taking a snapshot of the application using a CSI driver which limits the restoration to only one cloud provider. The pipeline will not terminate until the application restored on the cluster changes its status to 'ready'.

### 4.2.1 Application Stack

The application stack comprises a straightforward WordPress[9] blog, incorporating MariaDB as its database management system. Notably, persistent volume claims are leveraged to preserve data on persistent volumes. This application serves as the subject of comprehensive testing across both scenarios simulating the process of restoration after a disruption.

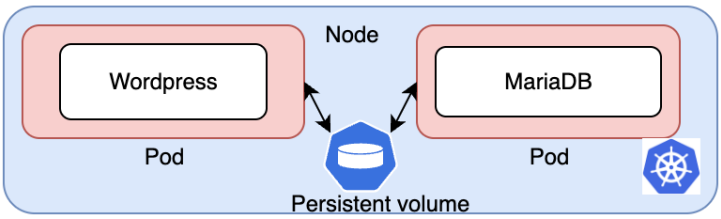



Figure 6: Data migration between cloud Providers

### 4.2.2 Data Migration Between Cloud Providers

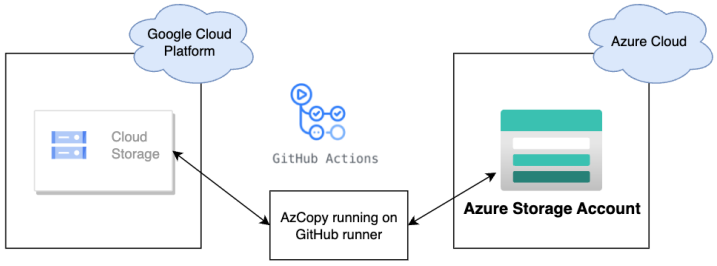



Figure 7: Data migration between cloud Providers

Figure 7 Illustrates the synchronization process between the GCS bucket and the Azure storage account. This synchronization is facilitated through the utilization of AzCopy, executed within the GitHub runner environment. The execution of AzCopy is orchestrated as a scheduled task through GitHub Actions, employing a cron job configuration. This data mobility forms a critical component of a multi-cloud disaster recovery strategy, ensuring the continuity of the application stack. In the event of unavailability from one cloud provider, seamless migration to an alternative cloud provider is assured.

[7] cloud.google.com
[8] microsoft.com
[9] wordpress.com

# 5 Implementation

Following the careful selection of the discussed open-source tools and disaster recovery architecture in the preceding section, our next step involves the implementation of the disaster recovery pipeline, aligning with a DevOps-centric approach.

## 5.1 Cloud Configuration

To ensure programmatic access to cloud resources, we created service accounts. Security keys are then generated for these accounts & securely stored within GitHub secrets, serving as environmental variables accessible to the DR pipeline enabling seamless authentication with the cloud provider when required. To ensure the unbiased acquisition of test results, we have deliberately selected resources with comparable specifications from both cloud providers. Specifically, for GCP, the managed K8s cluster is configured with "e2-standard-2" instances, featuring 2 vCPUs, 8 GB of memory, and includes the deployment of such two nodes. Likewise, for the Azure cluster, we have selected the "Standard-D2-v2" instance, also equipped with 2 vCPUs, 7 GB of memory, and two nodes deployed. Furthermore, the chosen regions for both cloud stacks are within North America, Azure resources are deployed in "East US," while GCP resources are in "us-central1."

## 5.2 GitHub

GitHub repositories have been utilized for storing the configuration files needed by the DR pipeline to create infrastructure resources and serve as the only source of truth for configuration files eliminating the ambiguity of configuration drift. The figure 8(b)figure depicts a successfully executed DR pipeline, encompassing a sequence of steps that include the terraform apply, the installation of Velero, the restoration of the application via Velero, and the subsequent resource destruction. These steps are iterated over 20 times to gather timing data for the evaluation process.

## 5.3 Terraform

Terraform was successful in creating the required infrastructure resources on the cloud providers after initialization of the DR pipeline. Figure 8(a)shows the code snippet for the deployment of managed k8s clusters on Azure. Terraform has been configured to use a remote backend to store the state file i.e., on the object store of the cloud provider which prevents multiple pipelines from getting triggered as the state file would be locked until the first process completes attaining the desired state.

## 5.4 Velero

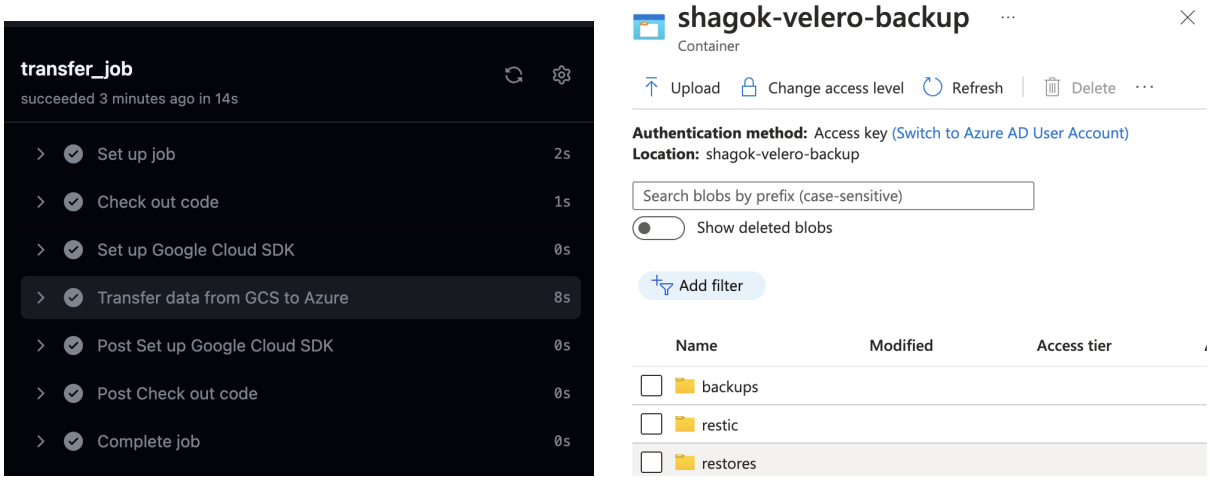
Velero is configured on the cluster and uses the object store to store the backups which are created using restic plugin providing us with a file system backup to restore. The Figure 9 shows the configured storage location for velero on google cloud named "shagok-velero-backup" and also confirms the application is successfully restored as the pods subjected to restoration attained status "ready".


(a) Terraform Code Snippet (b) Github Action Pipeline

Figure 8:

```
Running loop 1

NAME      PROVIDER   BUCKET/PREFIX          PHASE       LAST VALIDATED                 ACCESS MODE   DEFAULT
default   gcp        shagok-velero-backup   Available   2023-08-02 21:54:53 +0000 UTC   ReadWrite     true

Restore request "fs-backup-20230802215456" submitted successfully.
Run `velero restore describe fs-backup-20230802215456` or `velero restore logs fs-backup-20230802215456` for more details.

pod/my-release-mariadb-0 condition met

pod/my-release-wordpress-7cf75458b8-mndvf condition met
```

Figure 9: Pipeline output for velero successful restoration




(a) Azcopy workflow with completed status

(b) Newly created container for the inbound data to azure

Figure 10:

## 5.5 Azcopy

Azcopy used to migrate the data from GCP to Azure has been successfully achieved as shown in the figure. new storage container has been created with the same name as that of GCP storage bucket "shagok-velero-backup".

# 6 Evaluation

## 6.1 Scenario 1

The application stack encounters errors as a result of a system upgrade, necessitating a restoration to its previous operational state, inclusive of its persistent volumes. In this case, solely the application stack experiences disruption, while the Kubernetes cluster remains unaffected. Consequently, only the application stack requires redeployment which is achieved through a point-in-time restore procedure utilizing Velero. This restoration process has been executed on both GCP and Azure across a span of 20 iterations. from the table 2 & table 3 it is quite clear that GCP is faster than Azure by 40.85 seconds w.r.t their mean time to restore. Also, from the figure 11 illustrating density plot for 95% confidence interval, it is quite evident that there is a sizable difference between both providers suggesting GCP is faster then azure and also has high probability of faster restoration in the 95% interval of the distribution.

## 6.2 Scenario 2

Scenario 2 involves migrating the application completely to a different region or cloud provider using pipelines designed to automate the workflow.

### 6.2.1 Fail-over to different cloud provider

Table shows the timing data for 20 iterations for both migration directions i.e., from GCP to Azure and Azure to GCP. Figure 12 depicting 95% confidence interval using t-distribution reveals that GCP to Azure migration is faster than Azure to GCP

| Loop | Duration (s) |
|---|---|
| 1 | 71 |
| 2 | 80 |
| 3 | 73 |
| 4 | 71 |
| 5 | 105 |
| 6 | 67 |
| 7 | 71 |
| 8 | 67 |
| 9 | 84 |
| 10 | 87 |
| 11 | 66 |
| 12 | 128 |
| 13 | 75 |
| 14 | 67 |
| 15 | 69 |
| 16 | 74 |
| 17 | 76 |
| 18 | 76 |
| 19 | 67 |
| 20 | 67 |
| Mean Time | 74 |

Table 2: GCP timing for scenario 1

| Loop | Duration (s) |
|---|---|
| 1 | 140 |
| 2 | 111 |
| 3 | 96 |
| 4 | 99 |
| 5 | 89 |
| 6 | 88 |
| 7 | 127 |
| 8 | 260 |
| 9 | 136 |
| 10 | 131 |
| 11 | 108 |
| 12 | 98 |
| 13 | 83 |
| 14 | 135 |
| 15 | 100 |
| 16 | 97 |
| 17 | 105 |
| 18 | 99 |
| 19 | 98 |
| 20 | 97 |
| Mean Time | 114.85 |

Table 3: Azure timing scenario 1

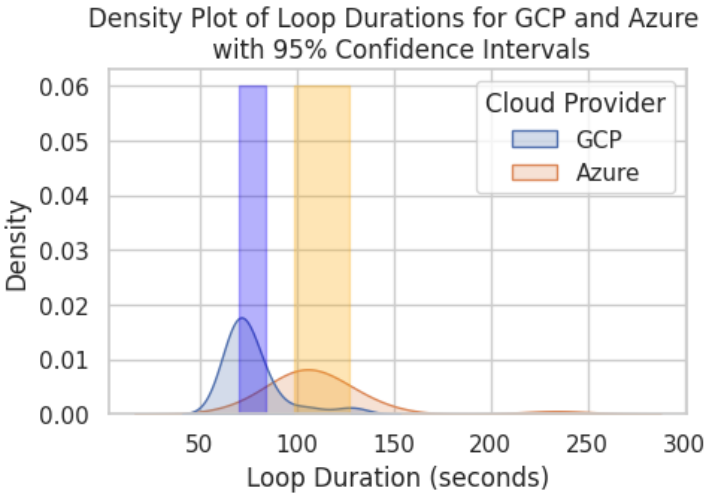



Figure 11: Density plot scenario 1

cloud and w.r.t mean time of restore from table 4 & table 5 we can say that GCP to Azure is 73.75 seconds faster than Azure to GCP. Hence, we can use GCP as our primary site & Azure as secondary site.

| Loop | Total Duration (s) |
|---|---|
| 1 | 593 |
| 2 | 595 |
| 3 | 583 |
| 4 | 635 |
| 5 | 588 |
| 6 | 617 |
| 7 | 600 |
| 8 | 605 |
| 9 | 594 |
| 10 | 614 |
| 11 | 582 |
| 12 | 593 |
| 13 | 628 |
| 14 | 632 |
| 15 | 570 |
| 16 | 595 |
| 17 | 620 |
| 18 | 562 |
| 19 | 580 |
| 20 | 598 |
| Mean Time | 599.2 |

Table 4: Azure to GCP Failover

| Loop | Total Duration (s) |
|---|---|
| 1 | 491 |
| 2 | 621 |
| 3 | 451 |
| 4 | 412 |
| 5 | 489 |
| 6 | 680 |
| 7 | 775 |
| 8 | 432 |
| 9 | 542 |
| 10 | 474 |
| 11 | 517 |
| 12 | 462 |
| 13 | 461 |
| 14 | 510 |
| 15 | 521 |
| 16 | 534 |
| 17 | 537 |
| 18 | 638 |
| 19 | 468 |
| 20 | 494 |
| Mean Time | 525.45 |

Table 5: GCP to Azure Failover

### 6.2.2 Migration to Different Region

The pipeline was triggered to move the application to secondary region which involved provisioning of GKE cluster and restoring the application using velero. similarly, this was triggered on Azure to migrate the application to secondary region. This process was iterated for 20 iterations as shown in the table. from the figure 13 it is evident that Azure was faster by 161.7 seconds than GCP when migrating the application to new region.

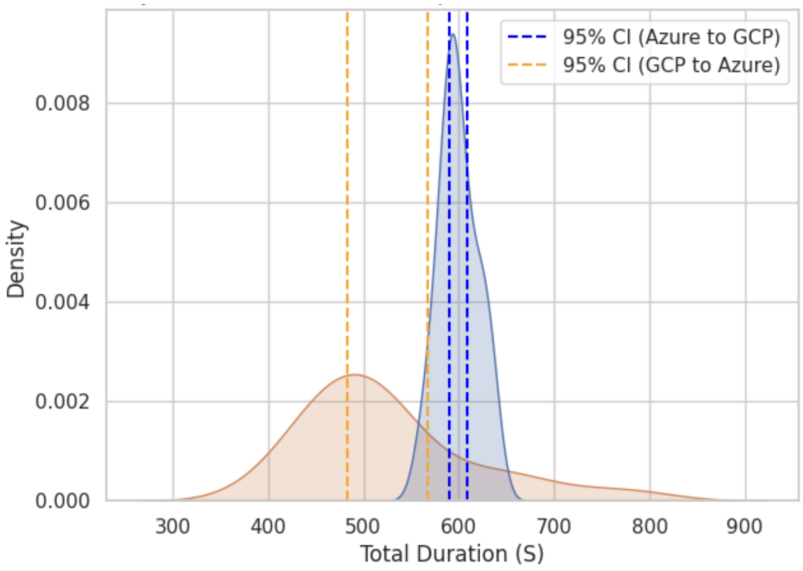



Figure 12: Density plot scenario 2 failover to other cloud provider

| loop | Total Duration (s) |
|---|---|
| 1 | 622 |
| 2 | 621 |
| 3 | 610 |
| 4 | 602 |
| 5 | 589 |
| 6 | 585 |
| 7 | 605 |
| 8 | 618 |
| 9 | 579 |
| 10 | 627 |
| 11 | 587 |
| 12 | 610 |
| 13 | 593 |
| 14 | 607 |
| 15 | 612 |
| 16 | 587 |
| 17 | 576 |
| 18 | 614 |
| 19 | 602 |
| 20 | 591 |
| Mean Time | 601.85 |

Table 6: GCP migration to different region

| loop | Total Duration (s) |
|---|---|
| 1 | 415 |
| 2 | 410 |
| 3 | 443 |
| 4 | 456 |
| 5 | 433 |
| 6 | 421 |
| 7 | 423 |
| 8 | 444 |
| 9 | 412 |
| 10 | 444 |
| 11 | 412 |
| 12 | 420 |
| 13 | 522 |
| 14 | 442 |
| 15 | 417 |
| 16 | 465 |
| 17 | 430 |
| 18 | 450 |
| 19 | 492 |
| 20 | 452 |
| Mean Time | 440.15 |

Table 7: Azure migration to different region

## 6.3 Discussion

Having conducted all the experiments for both scenarios and obtained the results, we can affirm the successful implementation of our DevOps-centric approach to restoring the containerized application on a Kubernetes cluster. The comprehens-

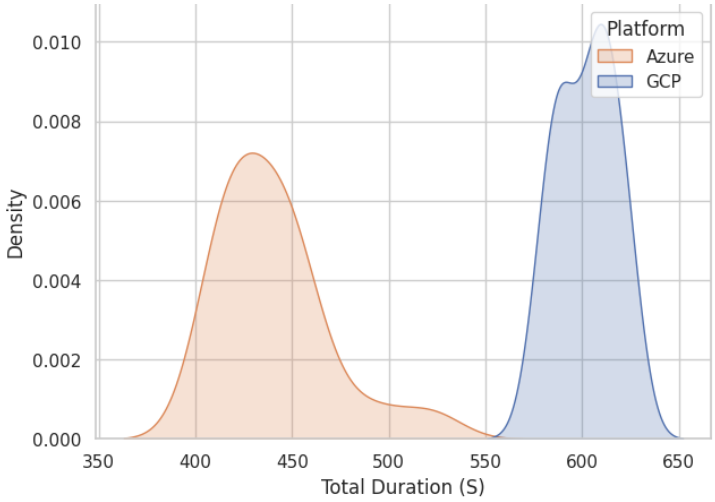



Figure 13: Density plot scenario 2 migration to different region

ive analysis of experimental outcomes from first scenarios indicates that GCP was faster in recovering when system failures occur due to system upgrades and for the second scenario of area wide outage for cloud providers Azure was faster in migrating to a different region and also failover from GCP.

From the acquired results we can state that it would be better for organizations to consider GCP as their primary site to host their application as it supports faster RTO for the failed application stack after a software revision and use Azure cloud as a standby site if an area wide outage would occur as it is faster in building up required infrastructure resources to host the containerized application than GCP.

The RPO considerations are subjected to organizations preference. velero can schedule backups as low as every 5 minutes and also in our experimentation we have scheduled it to run every 5 mins and the data is synchronized between cloud providers every 5 mins using azcopy. which results to be maximum of 10 minutes for asynchronous initiation and 5 minutes for syncronous i.e., the migration job starts as soon as the application is backed up. Reducing the backup interval even further would result in impacting cluster performance and huge backup data hoarding in object stores driving the costs up.

An important point to note is that the designed pipeline must authenticate with the cloud provider, granting programmatic access to execute tasks on the cloud platform. The time required for authentication is not uniform, as it varies based on the number of requests being handled by the CSP, introducing a latency factor. this can be optimized by testing different types of authorizers for cloud provider in further studies to improve performance of the designed pipeline.

# 7 Conclusion and Future Work

In conclusion, we were able to successfully develop a DevOps-centric strategy for the multi-cloud DR process by creating CI/CD pipelines by making use of open-source tooling to restore the containerized application running on managed Kubernetes cluster as efficiently as possible on both the chosen cloud providers and compare

the RTO & RPO between the cloud providers for the scenarios of system upgrade failures & area wide outage requiring application to be migrated to different region or cloud provider. This platform-agnostic approach to DR mitigated the risks associated with vendor lock-in, and allows organizations to maintain flexibility and freedom in their cloud service provider selection.

From the evaluation section it can be stated that organizations should consider GCP as their primary site to host their application as it supports faster RTO for the failed application stack after a software revision and use Azure cloud as a standby site if an area wide outage would occur as it is faster in building up required infrastructure resources to host the containerized application than GCP.

The process of implementing such a pipeline is notably intricate and demands the professional expertise to make well-informed decisions during the tool selection phase. The range of software available for disaster recovery is extensive when building a solution of this nature, and this study can serve as an effective initial reference point. This study has investigated GCP and Azure; however, there is potential for further exploration into other cloud platforms and on-premise data centers. Another potential avenue for research is to analyze disparities based on the underlying hardware type. In this case, we focused on standard free-tier offerings, but it is also worthwhile to examine systems with enhanced performance capabilities across various cloud providers. A study can be conducted to minimize the time taken for pipeline initialization which directly depends upon the mechanism used for authentication with the cloud provider by testing different types of authorizers.

# References


Alhazmi, O. H. and Malaiya, Y. K. (2012). Assessing disaster recovery alternatives: On-site, colocation or cloud, *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*, pp. 19–20.

Alhazmi, O. H. and Malaiya, Y. K. (2013). Evaluating disaster recovery plans using the cloud, *2013 proceedings annual reliability and maintainability symposium (rams)*, IEEE, pp. 1–6.

Andrade, E., Nogueira, B., Matos, R., Callou, G. and Maciel, P. (2017). Availability modeling and analysis of a disaster-recovery-as-a-service solution, *Computing* **99**: 929–954.

Baham, C., Hirschheim, R., Calderon, A. A. and Kisekka, V. (2017). An agile methodology for the disaster recovery of information systems under catastrophic scenarios, *Journal of Management Information Systems* **34**(3): 633–663.

Burns, B., Beda, J., Hightower, K. and Evenson, L. (2022). *Kubernetes: up and running*, " O'Reilly Media, Inc.".

Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J. (2016). Borg, omega, and kubernetes: Lessons learned from three container-management systems over a decade, *Queue* **14**(1): 70–93.
**URL:** *https://doi.org/10.1145/2898442.2898444*

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J. and Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering

computing as the 5th utility, *Future Generation Computer Systems* **25**(6): 599–616.
**URL:** *https://www.sciencedirect.com/science/article/pii/S0167739X08001957*

Gallagher, D. and Lennon, R. G. (2022). Architecting multi-cloud applications for high availability using devops, *2022 IEEE International Conference on e-Business Engineering (ICEBE)*, pp. 112–118.

Israel, G. D. (1992). Determining sample size.

Lavriv, O., Klymash, M., Grynkevych, G., Tkachenko, O. and Vasylenko, V. (2018). Method of cloud system disaster recovery based on" infrastructure as a code" concept, *2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET)*, IEEE, pp. 1139–1142.

Pokharel, M., Lee, S. and Park, J. S. (2010). Disaster recovery for system architecture using cloud computing, *2010 10th IEEE/IPSJ International Symposium on Applications and the Internet*, IEEE, pp. 304–307.

Poniszewska-Marańda, A. and Czechowska, E. (2021). Kubernetes cluster for automating software production environment, *Sensors* **21**(5).
**URL:** *https://www.mdpi.com/1424-8220/21/5/1910*

Rajkumar, M., Pole, A. K., Adige, V. S. and Mahanta, P. (2016). Devops culture and its impact on cloud delivery and software development, *2016 International Conference on Advances in Computing, Communication, Automation (ICACCA) (Spring)*, pp. 1–6.

Sameer, De, S. and Prashant Singh, R. (2022). Selective analogy of mechanisms and tools in kubernetes lifecycle for disaster recovery, *2022 IEEE 2nd International Conference on Mobile Networks and Wireless Communications (ICMNWC)*, pp. 1–6.

Suguna, S. and Suhasini, A. (2014). Overview of data backup and disaster recovery in cloud, *International Conference on Information Communication and Embedded Systems (ICICES2014)*, IEEE, pp. 1–7.

Wood, T., Cecchet, E., Ramakrishnan, K. K., Shenoy, P., Van der Merwe, J. and Venkataramani, A. (2010). Disaster recovery as a cloud service: Economic benefits & deployment challenges, *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*.