

Disaster Recovery Analysis of different Cloud Managed Kubernetes Clusters

Sergio Fernández Rubio

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
MSc Data Engineering

In collaboration with Claranet Deutschland GmbH

School of Computing

April 2022

MSc dissertation check list

Learning outcome	The markers will assess	Pages ¹	Hours spent
Learning outcome 1 Conduct a literature search using an appropriate range of information sources and produce a critical review of the findings.	* Range of materials; list of references * The literature review/exposition/background information chapter	11-23, 69-73	180 hours
Learning outcome 2 Demonstrate professional competence by sound project management and (a) by applying appropriate theoretical and practical computing concepts and techniques to a non-trivial problem, <u>or</u> (b) by undertaking an approved project of equivalent standard.	* Evidence of project management (Gantt chart, diary, etc.) * Depending on the topic: chapters on design, implementation, methods, experiments, results, etc.	24-53, 74-97	300 hours
Learning outcome 3 Show a capacity for self-appraisal by analysing the strengths and weakness of the project outcomes with reference to the initial objectives, and to the work of others.	* Chapter on evaluation (assessing your outcomes against the project aims and objectives) * Discussion of your project's output compared to the work of others.	54-64	120 hours
Learning outcome 4 Provide evidence of the meeting learning outcomes 1-3 in the form of a dissertation which complies with the requirements of the School of Computing both in style and content.	* Is the dissertation well-written (academic writing style, grammatical), spell-checked, free of typos, neatly formatted. * Does the dissertation contain all relevant chapters, appendices, title and contents pages, etc. * Style and content of the dissertation.		80 hours
Learning outcome 5 Defend the work orally at a viva voce examination.	* Performance * Confirm authorship		1 hour

Have you previously uploaded your dissertation to Turnitin? Yes

Has your supervisor seen a full draft of the dissertation before submission? Yes

Has your supervisor said that you are ready to submit the dissertation? Yes

¹ Please note the page numbers where evidence of meeting the learning outcome can be found in your dissertation.

Authorship Declaration

I, Sergio Fernández Rubio, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project, I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:

A handwritten signature in black ink, appearing to read 'Sergio', with a stylized flourish underneath.

Date: **25/01/2022**

Matriculation no: **40506379**

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please write your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

Sergio Fernández Rubio

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

Disaster Recovery is nowadays an increasingly important topic of research, as society is depending more and more on technology and communications for every task or process, be it related to business or government. This applies to systems, data, and its links. In recent years, there has been a shift in production systems towards the usage of Kubernetes, which is a piece of software that orchestrates computing resources in a different paradigm. This tool improves, but not solves completely, several aspects of a disaster recovery process, as it has built-in replication and scaling of the applications running within. It also allows easy deployment of load balancers and because of its design, it facilitates the migration process of workloads. However, the literature suggests that disaster recovery addressing specifically Kubernetes is not well studied, while progressively more companies are making heavy use of it. In this dissertation, Disaster Recovery procedures are investigated, leveraging Kubernetes in the cloud. Mainly, the Recovery Time Objective (RTO) and partially, the Recovery Point Objective (RPO) are studied in the context of two cloud providers in this dissertation. These providers include Amazon Web Services and Google Cloud Platform. Two main disasters that a cloud user could suffer are characterised: The first, a software update issue; and the second, a cloud zonal outage. For the given scenarios, it has been found that AWS has a mean noticeable shorter RTO in the first scenario compared to GCP. However, in the second scenario, the RTO was surprisingly longer than GCP, mainly because an OpenID Cloud Identity Provider was set in place in AWS.

Contents

1	INTRODUCTION	11
1.1	Background	12
1.2	Scope of the project.....	14
1.3	Research questions and aims	14
2	LITERATURE REVIEW.....	16
2.1	The Nature of Disaster Recovery & SLAs	16
2.2	The Importance of Disaster Recovery	17
2.3	Disaster Recovery Costs	17
2.4	Approaches to Disaster Recovery	18
2.5	Disaster Recovery on Kubernetes.....	20
2.5.1	Disaster Recovery with Velero.....	21
2.6	Summary	23
3	DESIGN & METHODOLOGY.....	24
3.1	Scenarios.....	24
3.2	Disaster Recovery Software	24
3.2.1	Workload for the tests	25
3.2.2	Kubernetes deployment.....	25
3.3	Benchmarking & results acquisition	27
3.4	Summary	31
4	IMPLEMENTATION	32
4.1	Software stack	32
4.1.1	Velero.....	32
4.1.2	Bash.....	33
4.1.3	Terraform	35
4.1.4	Helmfile / Helm.....	36
4.1.5	Kustomize: Test Application.....	40
4.1.6	Additional Kubernetes utilities.....	43
4.1.7	Git / GitHub.....	43
4.1.8	R language.....	43
4.2	Cloud stack.....	46
4.2.1	AWS.....	46
4.2.2	GCP.....	47
4.3	Summary	47

5	EXPERIMENTS.....	48
5.1	Experimentation	48
5.1.1	Implementation of experiments	50
5.1.2	Experiment run	52
5.2	Summary	53
6	EVALUATION	54
6.1	Results.....	54
6.2	Strengths and Limitations.....	59
6.3	Comparison to Others' Work.....	61
6.4	Overview on research questions and aims	61
7	CONCLUSIONS.....	65
7.1	Research Overview	65
7.2	Main takeaways	66
7.3	Future work & Recommendations	66
8	REFERENCES.....	69
	Project proposal.....	74
	Terraform files and its explanation	86

List of Tables

Table 1 - Timings for Scenario 1 in GCP	54
Table 2 - Timings for Scenario 1 in AWS.....	54
Table 3 - Timings for Scenario 2 in GCP	56
Table 4 - Timings for Scenario 2 in AWS.....	56

List of Figures

Figure 1 - Disaster Recovery with Velero - Creating a Backup.....	22
Figure 2 - Disaster Recovery with Velero – Restoring a previously created backup.....	22
Figure 3 - Architecture of the test application	25
Figure 4 - Architecture of the applications to be deployed in the Kubernetes cluster.....	26
Figure 5 - Time measurement for scenario 1.....	27
Figure 6 - Time measurement for scenario 2.....	29
Figure 7 - Velero recovery process in the scenarios.....	33
Figure 8 - Bash main.sh file for scenario 1	34
Figure 9 - Bash loop.sh file for scenario 1	34
Figure 10 - Bash main.sh file for scenario 2	35
Figure 11 - Bash loop.sh file for scenario 2	35
Figure 12 - Helmfile file for GCP.....	36
Figure 13 - Helmfile values for Velero chart (velero.yaml) for GCP	37
Figure 14 - Helmfile file for AWS	38
Figure 15 - Helmfile values for Velero chart (velero.yaml) for AWS.....	39
Figure 16 - Kustomize file	40
Figure 17 - WordPress deployment file (wordpress-deployment.yaml)	41
Figure 18 - MySQL deployment file (mysql-deployment.yaml)	42
Figure 19 - R code to create the charts	45
Figure 20 – R code to calculate the t-test	46
Figure 21 - Tree folder structure for the experiments	49
Figure 22 - Terraform GKE code being applied on GCP cloud.....	50
Figure 23 - Configured test WordPress site.....	51
Figure 24 - Part of the Helmfile values file which was added later to schedule the backup.....	52
Figure 25 - Density plot for Scenario 1	55
Figure 26 - Density plot for Scenario 2	57
Figure 27 - Density plot for Scenario 2 (subtracting OIDC provider).....	57

Acknowledgements

I would like to give special thanks to my supervisor, Dr. Paul Lapok, for its great amount of help and patience, even when it took much time selecting the right topic for this project. I am also grateful to Dr. Neil Urquhart for its suggestions and insightful questions.

I would also like to give special thanks to my manager at my current job position, Mr. Fabian Dörk, for its continuous support and for the agreeing of the use of resources by Claranet Deutschland GmbH. I am also grateful to Mr. Aitor Zabala, my former manager, who supported and encouraged me by sending the needed recommendation letter to start this MSc.

I would also like to give special thanks to Ms. Raquel Teruel, who kindly proofread this document in a timely manner. I am also very grateful to my mother, Ms. María Dolores Rubio, who supported me financially to successfully conclude this MSc.

Thanks also go to my wonderful wife Annais for her support during all these difficult times, enduring with me throughout time along with my health problems. Without her kind help and understanding, this project would have never come to fulfilment.

1 Introduction

The term “Disaster Recovery” (DR) has become increasingly omnipresent in businesses, and specially on critical business services that need to run constantly, as the mean to articulate the need of a way to restore a previous working state and ensuring business continuity. This active state should be guaranteed in the time it is expected to run, and usually businesses express these constraints through Service Level Agreements (SLAs) and Service Level Objectives (SLOs). In an active Disaster Recovery process, the timings in which applications are inoperative, are commonly measured in terms of Recovery Time Objective (RTO) and Recovery Point Objective (RPO), which directly affects SLAs and SLOs.

Disaster Recovery design depends on the technologies used which need to be functioning. Nowadays, Kubernetes has been progressively gaining traction in all kinds of companies, for deploying software applications. This is due to its scalability, resource optimization, integrated high availability options, workload portability, low overhead, and a big ecosystem in which a big technical talent pool is available. Kubernetes can be deployed in many settings, but the benchmarking and calculation in a managed cloud context of these parameters are the object of this work.

The work focuses on the different approaches to Disaster Recovery of Kubernetes in the cloud. Mainly two clouds were investigated, Google Cloud Platform (GCP) and Amazon Web Services (AWS), which are public clouds widely used in the industry (*As Quarterly Cloud Spending Jumps to Over \$50B, Microsoft Looms Larger in Amazon’s Rear Mirror | Synergy Research Group*, 2022). The implications of each disaster recovery architecture were reviewed and applied to Kubernetes specifically, and the conclusion is that the offerings and suggestions are very similar. This architecture, as per AWS naming (GCP does not name them, but, as already mentioned, it offers similar architectures), are (Eliot, 2021):

1. Backup & Restore.

2. Pilot Light.
3. Warm Standby.
4. Multi-site active/active.

Two main disaster types (system upgrades, power issues) were mapped to the Backup & Restore architecture, which was studied through tests in the different cloud providers. Both disasters are the main drivers of disaster in businesses worldwide (Alhazmi & Malaiya, 2013).

The capabilities of the Kubernetes cloud designs were investigated by deploying key applications into the clusters, such as Velero, a backup and restore tool for Kubernetes resources and persistent volumes (*Velero*, n.d.), as well as designing and deploying disaster recoveries with these capabilities in mind.

1.1 Background

Interest in disaster recovery has been an increasingly researched topic in general. Although there is no single definition of disaster recovery (Marshall & Schrank, 2013), Quarantelli defines it broadly as “reconstruction, restoration, and rehabilitation, bringing the post disaster situation to some level of acceptability” (Quarantelli, 1999). Although this is a term commonly used when discussing in the context of Information Technology (IT) systems, as a matter of fact, disaster recovery can refer to any non-IT area susceptible to disaster, and susceptible to a recovery.

Organizations should carefully decide whether they can afford a disaster recovery plan in place, as it is, most of the times, critical for its survival/success (Fallara, 2004). This usually lies in the context of a business continuity plan (Lam, 2002), given the fact that disaster recovery is a subset of a business continuity plan (Zhang & McMurray, 2012), although those terms can and actually are frequently used interchangeably. Other authors also include disaster recovery planning within the context of a IT contingency planning (Wiboonrat, 2008a).

Disaster recovery measures typically include the following four key steps to its adoption (Rudolph, 1990):

- 1) Plan for the worst-case scenario
- 2) Document strategies for recovering critical business applications
- 3) Develop and implement technologies which aid in the recovery of systems and applications
- 4) Continuous training of employees which directly deal with the systems, to handle future incidents.

In IT, disaster recovery has been progressively gaining importance, as the dependence on business processes and technology are ever increasing (Prazeres & Lopes, 2013). There are numerous examples of study on disaster recovery plans in recent years regarding private datacentres.

For example, Cegieta conducts an analysis of disaster recovery practices at an energy company in Poland (Cegieta, 2006), where current DR practices were assessed mathematically and the solutions in place were studied, discerning five different solutions, each one with its own RPO/RTO values. Or another example is Lozupone, which studied and applied DR techniques on a medical record company (Lozupone, 2017), where possible scenarios were identified, as well as their mitigations, particularly including cybersecurity threats. The result was a creation of a contingency plan.

There are other examples as well in cloud environments in more recent years about DR. Sahi et al. studied cloud DR practices for the eHealth sector oriented to RPO (Sahi et al., 2016), where approaches to harden the security and privacy of the data were suggested. Another example is Tamimi et al. which researches in a theoretic way about the topic of DR in the cloud (Tamimi et al., 2019), and as a result informational tables were developed with details per service on RPO and RTO.

But there are relatively few exploring disaster recovery in Kubernetes, and even then, just explanatory (Minh Bui, 2020), and, as far as known, none about disaster recovery on managed Kubernetes clusters in the cloud.

Therefore, in this project, the concept of disaster recovery within the context of Kubernetes is explored, and more specifically, within the cloud as a managed service. This approach is one of the main three patterns of Kubernetes deployment models (Chandrasekaran, 2020a).

There are many cloud vendors offering Kubernetes as a product, such as Google, Amazon, Microsoft, DigitalOcean, Alibaba and Oracle, but the focus will be on the services offered by Google and Amazon: GKE (Google Kubernetes Engine) and Amazon EKS (Amazon Elastic Kubernetes Service), as they are two widely utilised services in the market.

1.2 Scope of the project

The disasters addressed in this work are defined in this section. This is a key aspect, as everything to be designed will be based on the type of disaster to be faced and this must be explicitly written on the DRP (Disaster Recovery Plan).

Based on a Symantec study from 2010, the two main causes for disasters in a 5-year period are system upgrades, and power outage/failure/issues (Suguna & Suhasini, 2015).

System (or software) upgrades are the leading cause of disaster. This means, routinely planned upgrades by clients on systems can leave the system inoperable, with the impossibility of running a given application. This does not mean the cloud services are unavailable, as it is understood that they are still available.

A second leading cause of disaster is power outage/failure/issues. This could be understood as area-zone disruption in the cloud (as opposed to a regional disruption). As explained later in Chapter 2, this has happened in different clouds at different times, in different zones. For this reason, the preparation work should include a recovery at another zone in the same cloud vendor, and subject to a cross-cloud recovery.

For this project, the focus will be on these two scenarios.

1.3 Research questions and aims

The research questions that this project will address are:

1. How to decrease the RTO and RPO of a fully production Kubernetes cluster after a disaster in the cloud?
2. Where are the limits and consequences on employing the identified technologies?
3. What differences exist in relation to Disaster Recovery between cloud providers?

The aim of this project is to research, explore and evaluate the current disaster recovery solutions available for Kubernetes running in the cloud, as a managed service, comparing and evaluating the offering in the different cloud providers.

To achieve the aims of this project, this dissertation is set out as follows:

- In chapter 2, the literature regarding the topic of Disaster Recovery is reviewed, so as to assess the actual state of the art, identifying key aspects which will be introduced in the project further on.
- In chapter 3, the methodology for this work is set. The scope of the tests is explained and delimited, the foundations for the benchmarking are set, the applications under test will be shown, as well as the underlying software needed for the tests.
- In chapter 4, the software and the cloud stacks are thoroughly explained, along with the code which will be used for the tests.
- In chapter 5, the results are presented and evaluated, in reference to the previous expectations.
- Finally, in chapter 6, the project is summarised. Then conclusions are drawn and suggestions for new avenues of research are considered.

2 Literature review

This work compares disaster recovery methods in the context of managed Kubernetes offerings in the cloud area. Therefore, this chapter will discuss different methods of disaster recovery currently available in both clouds (AWS, GCP), with the aim to compare its target metrics, namely RTO and RPO.

2.1 The Nature of Disaster Recovery & SLAs

The following metrics or KPIs are key to defining the Disaster Recovery efficiency (Suguna & Suhasini, 2015):

- Recovery Time Objective: Measures the duration in which business functions are unavailable, until the finishing time when operations resume (including the time before disaster is declared and the time to perform tasks).
- Recovery Point Objective: Measures the duration between the last backup of the data and the time of the disaster. Normally, in the disaster recovery plan, this is defined as the duration between two successive backups, and thus the maximum amount of data that can be lost when restoration is successful.

In order to measure disaster recovery efficiency in regards to target recovery requirements, these two metrics (RTO and RPO) are commonly tracked, as well as a third, which is the monetary expenses (Gunawan et al., 2013). These metrics are critically important when considering companies which have a set of predefined SLAs (Service Level Agreements) with customers, in order to minimize risk (Oktadini & Surendro, 2014).

These definitions define SLAs and SLOs for the workloads running in a given cluster.

In managed Kubernetes distributions in the cloud, SLAs are publicly available (*Google Kubernetes Engine SLA* | *Google Cloud*, 2021)(*Amazon EKS Service Level Agreement*, 2020), and therefore, the target SLAs should be calculated for a given workload based on these figures. Therefore, the RTO and RPO should be designed according to these.

Disasters also come in various forms. Alhazmi and Malaiya (2013) mentioned the most likely causes of a disaster in a 5-year period including system upgrades, power outage/failure/issues, fire, configuration change management, cyber-attacks... Hence, the focus will remain on the first two causes (system upgrades and power issues) when comparing the offerings in Google GKE and Amazon EKS.

2.2 The Importance of Disaster Recovery

As mentioned earlier, The impact of addressing disaster recovery in a company could be considered critical for its survival/success (Fallara, 2004). Google and Amazon themselves have faced catastrophic events in the past (Andrade et al., 2017) and the project is going to have this into account.

In October 2015, Google Apps were inaccessible, causing widespread disruption from classrooms to businesses and beyond. In September 2015, AWS suffered a tremendous outage which affected key customers such as Netflix, Tinder, and IMDb.

As Andrade et al. also mention, 50% of businesses that have experienced a major data loss due to disasters go out of business within 24 months, and 93% out of business within 5 years. Therefore, disaster recovery planning is not a process of minor importance. It is a business necessity.

2.3 Disaster Recovery Costs

It is also critical for businesses to assess the total costs before considering an approach to disaster recovery. Usually, over a year, the total cost could be calculated as following (Alhazmi & Malaiya, 2012):

$$C_T = C_i + C_o + C_d$$

Where C_T is the sum of the initial cost C_i , recurring cost C_o , plus the expected annual cost of potential disasters C_d .

Similarly, the ongoing (or recurring) cost C_o is the sum of ongoing storage cost C_{os} , data transfer cost C_{ot} , and processing cost C_{op} .

$$C_o = C_{os} + C_{ot} + C_{op}$$

The annual disaster cost is defined as the total expected cost of disaster recoveries C_{ri} plus the unrecoverable disasters C_{ui} , for each type of disaster i and its probability p_i .

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

It should be noted that the recovery cost includes the cost of using the backup after the failover and the cost of lost transactions. The cost of lost transactions is proportional to the RTO duration. Loss of reputation would be another factor to consider.

2.4 Approaches to Disaster Recovery

Different sets of disaster recovery processes can be designed to minimize cost or/and SLA disruption (by means of RTO and RPO), as for example, in the topic of SLA-aware IaaS (Infrastructure-as-a-Service) Cloud networks (de Souza Couto et al., 2014) or by manually prioritising RPO and RTO in business units, within the context of the same organization (Wiboonrat, 2008a).

Cloud approaches to disaster recovery has been thoroughly studied, especially in latest years, where the cloud is now the de-facto standard for workloads, supporting a wide range of business operations. This can be drawn from Gartner estimates and forecasts of past and future market spend, as it is expected to reach \$397 billion in 2022 (Costello & Rimol, 2021).

Companies need first to assess and select the backup/recovery placement, as each option has its strengths and its shortcomings (Alhazmi & Malaiya, 2012; Suguna & Suhasini, 2015):

- Onsite (Backup and running system in one location): Data can be synchronized easily, high risk of a dual disaster, high initial costs and high post-disaster costs.
- Co-location (Backup is remotely located at another site): Data can be synchronized less easily, depending on the links available. Low risk of dual disaster, medium initial costs, but high post-disaster costs.
- Cloud (Backup located in one of the available cloud vendors): Data can be synchronized less, depending heavily on traffic costs. Low risk of dual disaster, low initial costs, low post-disaster costs.

There are many mechanisms available for disaster recovery, each one with its own particularities, which could be mainly summarised in three main mechanisms (Suguna & Suhasini, 2015; Wood et al., 2010) which are:

- Hot standby/backup site: Second datacentre, which typically provides a set of mirrored, stand-by servers, providing the best KPIs (Key Performance Indicators) in regard to RPO and RTO. This is, certainly, the most expensive option of the three, given the fact that, usually, the entire application (even if it is deployed on different distributed servers) is actively deployed twice, while one of them is in standby. However, 0-seconds RTO and RPO are virtually impossible to achieve.
- Warm standby/backup site: It is a trade-off between a hot and a cold standby/backup site. Regarding to RPO, the definition is flexible, as it can entail synchronous (hot) or asynchronous (cold) replication from the active site. But regarding to RTO, the applications are usually kept on a “warm” state, in which the application is usually partially or totally deployed, but heavily scaled down. This slows recovery, but also significantly reduces cost.
- Cold standby/backup site: In this disaster recovery method, data is replicated on a periodic basis, and the RPO timings are affected. As a result, it is in the range of hours, or even days. Additionally, the backup

site is not readily available to act swiftly when a disaster occurs, as hardware may be needed to be swapped, and operating systems and applications to be installed. This is the only adequate option for applications that do not need strict SLAs.

Additionally, apart from the selection of a hot/warm/cold site implementation, there are practical requirements and decisions to be made when considering the actual implementation of a disaster recovery. Predominantly, three types of mechanisms regarding to the deployment of the actual disaster recovery solution, and especially in relation to cloud disaster recovery could be considered (Wood et al., 2010).

- Network reconfiguration: An approach to network reconfiguration must be selected, notably depending also on the software architecture, but there would be mainly two. Through either modifying DNS records, or updating routes in the main routers (Robinson et al., 2014).
- Security and Isolation: Companies should consider carefully when using the cloud on how to deal with privacy matters in respect of storage, network, and compute resources. Cloud vendors typically offer different levels of security and privacy, but even heterogeneity exists in the level of security treatment each component provides (Takabi et al., 2010).
- Resource migration and cloning: When tasked with a hot/warm site backup approach, appropriate methods of migration and cloning of resources need to be selected. Considering VMs, there are currently approaches to migration and cloning (Al-Kiswany et al., 2011). But there are as well for Kubernetes (Schrettenbrunner, 2020) especially crafted for stateless pods, which do not hold application states.

2.5 Disaster Recovery on Kubernetes

Kubernetes is a widely used container management system which stands out because of its ability to handle application and server-level errors in a distributed manner, incorporating disaster recovery processes in its core. Its

goals are to build on the capabilities of containers to provide significant gains in programmer productivity and ease of both manual and automated system management (Burns et al., 2016).

However, even failures can occur in a well-designed Kubernetes cluster, due to systems upgrades, or due to area-wide outages. This is the reason why many authors consider using Velero, formerly known as Ark (Burns & Tracey, 2018; Minh Bui, 2020; Poniszewska-Marańda & Czechowska, 2021).

Velero mainly is a tool for backing up cluster resource data. Still, Velero is not only concerned with the management of Kubernetes resource data but also serves as a framework for managing application data. Starting from Velero 1.5, it also supports Restic, a backup tool which now works with Kubernetes volumes (Amarnath, 2020). Usually, and before the implementation of Restic, volumes would be backed up through the tools made available by means of the cloud vendors, which provide the underlying storage driver.

However, Velero does not perform application-aware snapshots, as it is not aware of application consistency after a disaster. Additionally, it lacks a standard web UI for easiness.

Nevertheless, Velero is currently the only open-source tool actively used for disaster recovery in Kubernetes, therefore, it is going to be employed for the analysis and tests.

2.5.1 Disaster Recovery with Velero

The backup creation process for resources (not including volumes) is the following – AWS EKS (Elastic Kubernetes Service) used as an example (Miller & Ciuffo, 2021), as in Figure 1:

1. The Velero CLI makes a call to the Kubernetes API server to create a backup CRD object.
2. The backup controller:
 - a. Checks the scope of the backup CRD object, namely if filters are set.
 - b. Queries the API server for the resources that need a backup.

- c. Compresses the retrieved Kubernetes objects into a .tar file and saves it in Amazon S3.

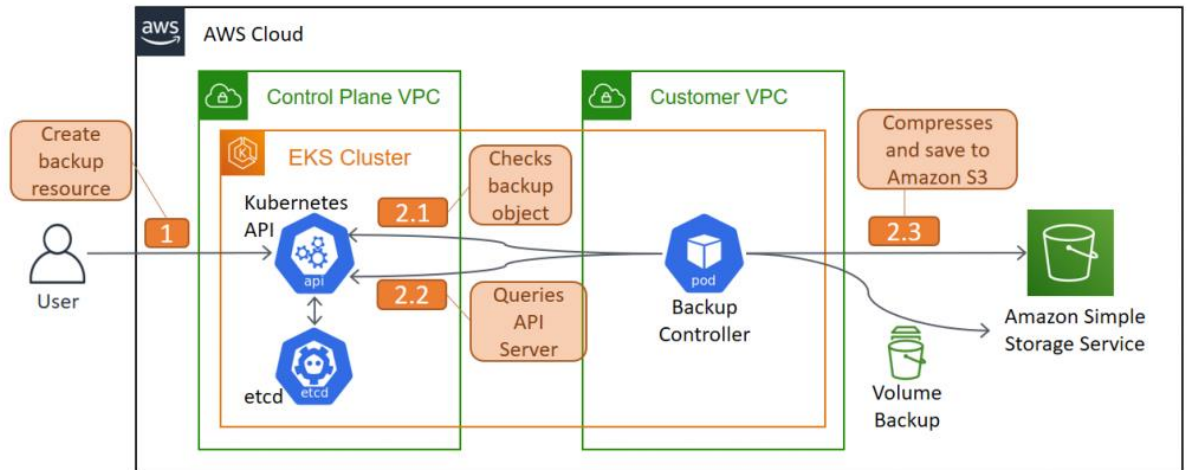


Figure 1 - Disaster Recovery with Velero - Creating a Backup

The restore process explained here below, as in Figure 2:

1. The Velero CLI makes a call to Kubernetes API server to create a restore CRD that will restore from an existing backup.
2. The restore controller:
 - a. Validates the restore CRD object.
 - b. Makes a call to Amazon S3 to retrieve backup files.
 - c. Initiates restore operation.

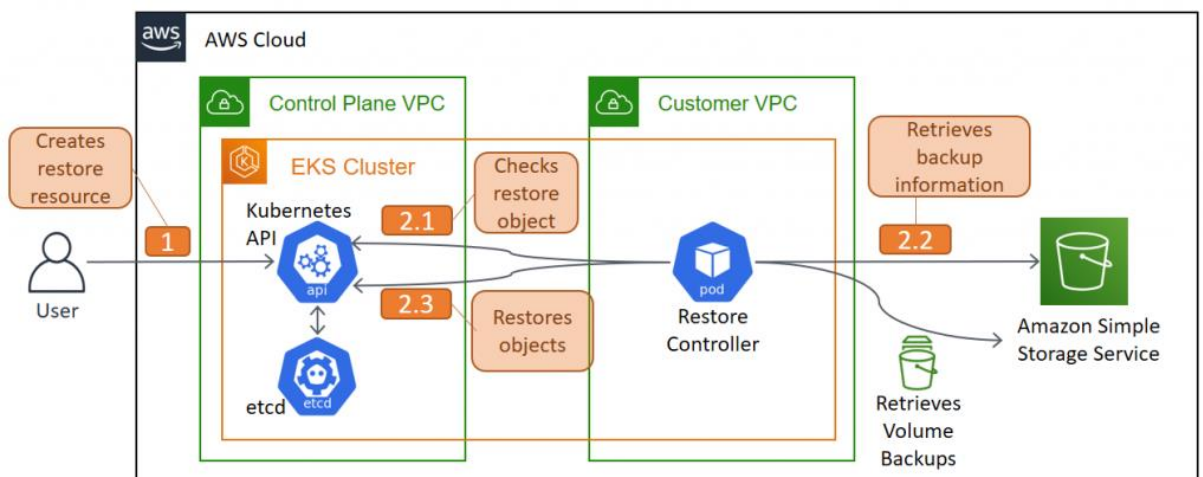


Figure 2 - Disaster Recovery with Velero – Restoring a previously created backup

Velero needs to use the plugins for the snapshots of the volumes, depending on the cloud provider in which is being installed.

2.6 Summary

This chapter gave an overview of what Disaster Recovery is. Different types of approaches to disaster recovery in the literature and in practice were numbered and described; its pros and cons were also discussed. Metrics for the correct definition of a DR were given and explained, such as RTO and RPO. Moreover, they were linked to business SLAs, which are of utmost importance for the success of businesses in a 5-year period. Additionally, costs were considered, showing practical equations to calculate the impact on costs having DR into account. Finally, DR was discussed in the context of Kubernetes, and Velero was presented.

Reflecting on the main questions and aims of this project, it has been found that RTO and RPO can be decreased on practice in virtual machine-based workloads, using different DR approaches, such as cold, warm, or hot standby sites. However, Kubernetes workloads have not been tested this way in the literature. Therefore, in this work RTO and RPO will be studied in Kubernetes in different scenarios, and the limits and consequences of each solution will be considered. Neither differences in DR approaches in cloud providers has been studied in the literature, and this project will address it, both in AWS and GCP.

3 Design & Methodology

This chapter shows the methodology used in this project, to examine and benchmark the different approaches to DR in Kubernetes.

There are numerous ways in which Kubernetes clusters could be deployed in the cloud. The scope is going to be limited to managed Kubernetes clusters already running in the cloud, backups also stored in the managed cloud services provided by the vendor, and recovery strategies following key ideas of the suggestions from cloud providers.

3.1 Scenarios

As mentioned in Chapter 1, the focus of this project will be targeting mainly two types of disasters: System/software upgrade failures, and power outage / failure / issues.

The approach will be different in both cases, as the situation implies and needs a different kind of recovery technique, and in both cases the possible concepts are going to be explored. However, both scenarios will leverage a common software stack.

3.2 Disaster Recovery Software

Both scenarios will leverage Velero, which will oversee the resource backup, as well as the volume backup, where the data is held. Velero offers the possibility to use the storage driver provided by the cloud vendor to make the snapshots, through means of plugins, as explained in the earlier chapter.

Terraform, which is an open source Infrastructure as Code (IaC) tool created by HashiCorp (*What Is Terraform? | IBM*, n.d.) is going to be used for the deployment of the cluster and its dependencies in both cloud providers. An Elastic Kubernetes Service (EKS) in AWS, and a Google Kubernetes Engine (GKE) are going to be provisioned through Terraform.

Helmfile, a declarative specification for deploying helm charts (*Roboll/Helmfile: Deploy Kubernetes Helm Charts*, n.d.), is going to be used to deploy Velero in the cluster. Helm is a Kubernetes application manager which helps define,

install and upgrade apps. In this case, Helmfile is a wrapper of helm, improving it in specific ways, and making the deployment easier. This way, we can deploy Velero each time seamlessly.

Kustomize will also be used, which is a templating language for configuration management of Kubernetes (*Kubernetes-Sigs/Kustomize: Customization of Kubernetes YAML Configurations*, n.d.). It understands Kubernetes objects and applies configuration files to the specified manifests.

Finally, command language interpreter bash will be used to paste everything together, in a single script.

3.2.1 Workload for the tests

The application subject to the tests is depicted in Figure 3. This testing scenario is composed of a simple WordPress blog, which uses MySQL as a database. Both the application and data recovery are going to be tested, simulating the state after a disaster.

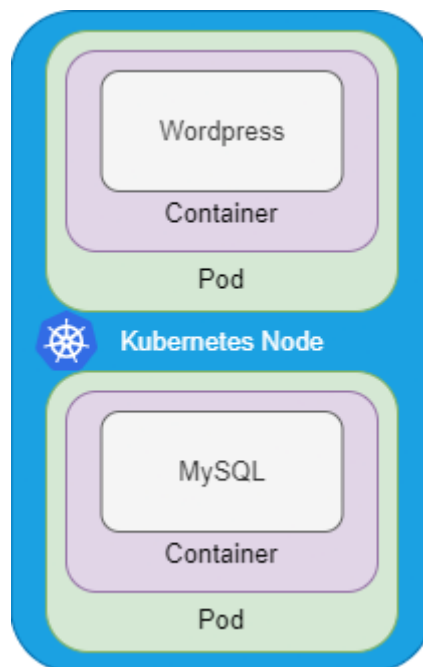


Figure 3 - Architecture of the test application

3.2.2 Kubernetes deployment

The resulting Kubernetes stack is shown in Figure 4. Velero will need to be installed on a different namespace, and another namespace should be created to hold all the application testing environment, which will use deployments, services, and persistent volumes.

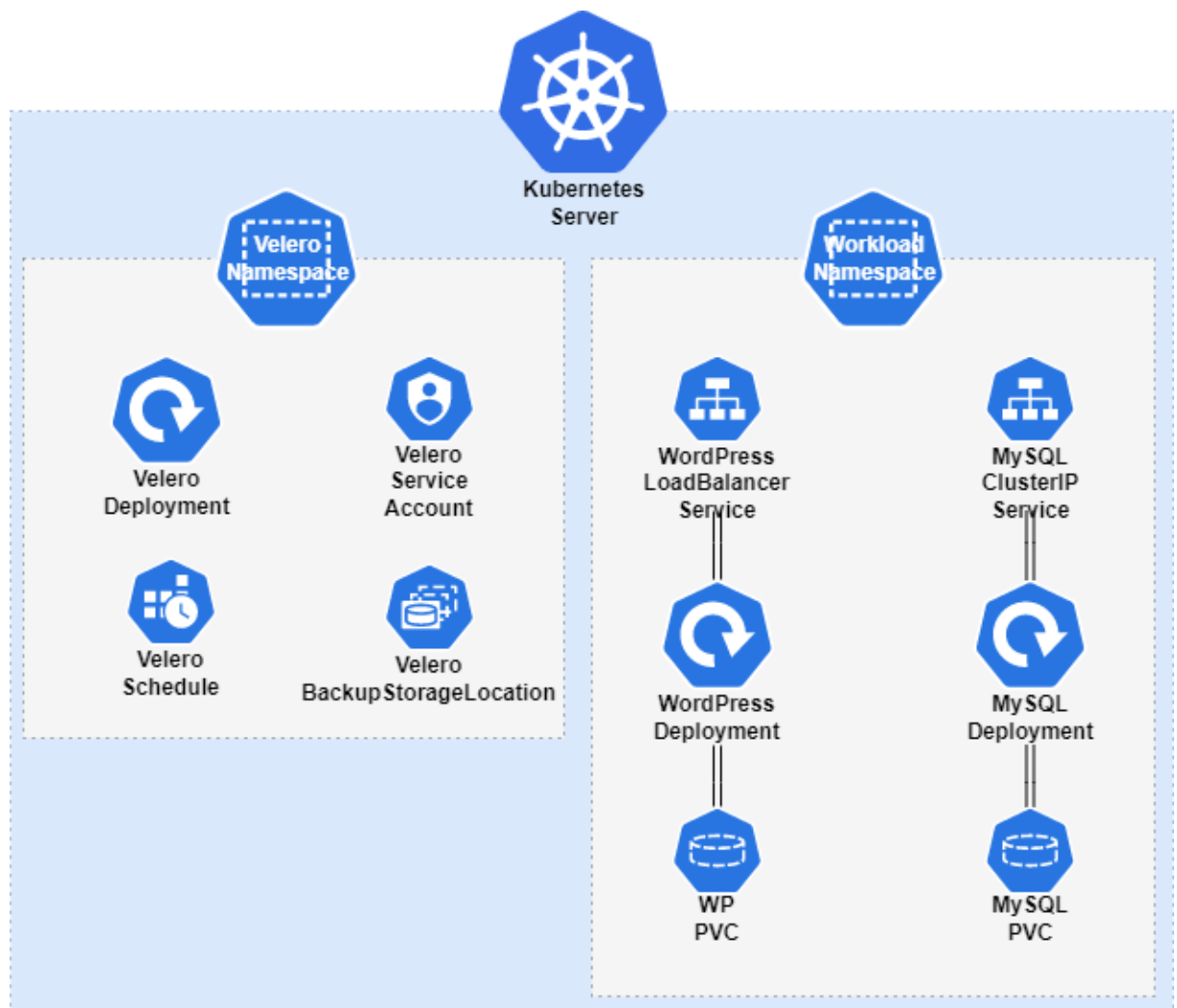


Figure 4 - Architecture of the applications to be deployed in the Kubernetes cluster

Cloud providers usually offer various datacentre locations inside a region, commonly called zones. Kubernetes clusters can leverage this feature to increase SLAs and SLOs.

The Kubernetes deployment is going to be single-zoned. Multi-zone clusters are available at both cloud providers, and they are preferred over single-zone clusters, as disaster recovery is already incorporated this way. However, for the tests, a single-node single-area master node will be leveraged, given the fact

that, although a regional cluster would improve the SLA metric, the focus will be on the steps after the disaster, thus not affecting total RTO and RPO in our case. Additionally, power issues (the second disaster scenario) could affect entire regions too, so regardless of the option, it could be argued that the second scenario would not affect the outcomes in terms of RTO and RPO.

3.3 Benchmarking & results acquisition

Definitions of benchmarking vary. Key themes include measurement, comparison, identification of best practices, implementation, and improvement (Anand & Kodali, 2008).

The aim, similarly, will be to measure and compare both cloud providers through a series of tests, for both scenarios. Cloud vendors provide shared resources, so differences based on the total usage during the test time should be expected. There are references on AWS (Bakshi & Kim, 2021; Miller & Ciuffo, 2021) and on GCP (*Disaster Recovery Planning Guide | Cloud Architecture Center | Google Cloud*, n.d.) about approaches and recommendations in Disaster Recovery.

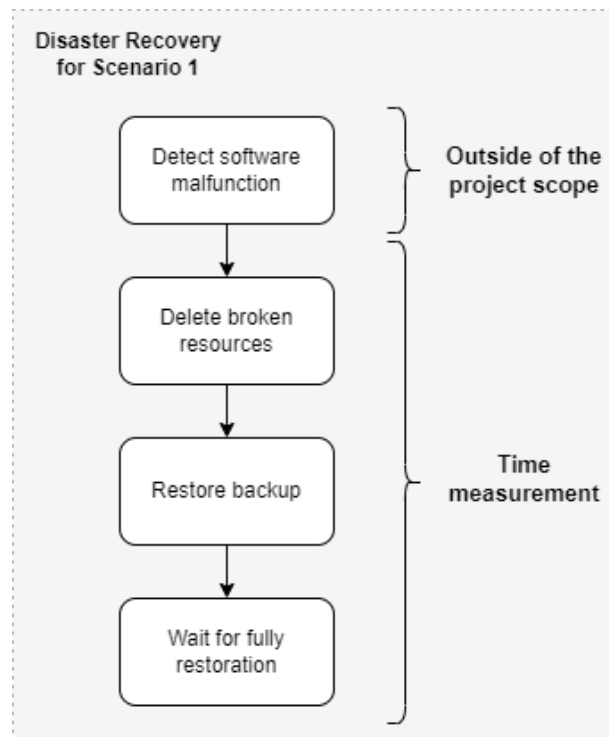


Figure 5 - Time measurement for scenario 1

In figure 5, the first scenario and the approach is showcased to the measurement of time. The detection and triggering of the alert of the disaster will not be part of this study, as access to live clusters for this project is not possible. Software like Prometheus or Nagios can be configured to alert about the disasters, and act automatically or manually, depending on previous decisions on SLA constraints.

The time measurement loop will start right before the deletion of the errored resources. As explained earlier, the disaster is a software upgrade malfunction, and the first step for disaster recovery remediation is to delete the resources already in the cluster, as having duplicated resources in the same cluster can create unintended havoc.

This is then followed by the issue of a restore operation, which is a request automatically approved or rejected. Finally, the loop will wait for the actual restoration of resources, not marking it as completed until a successful check of the entire application health status.

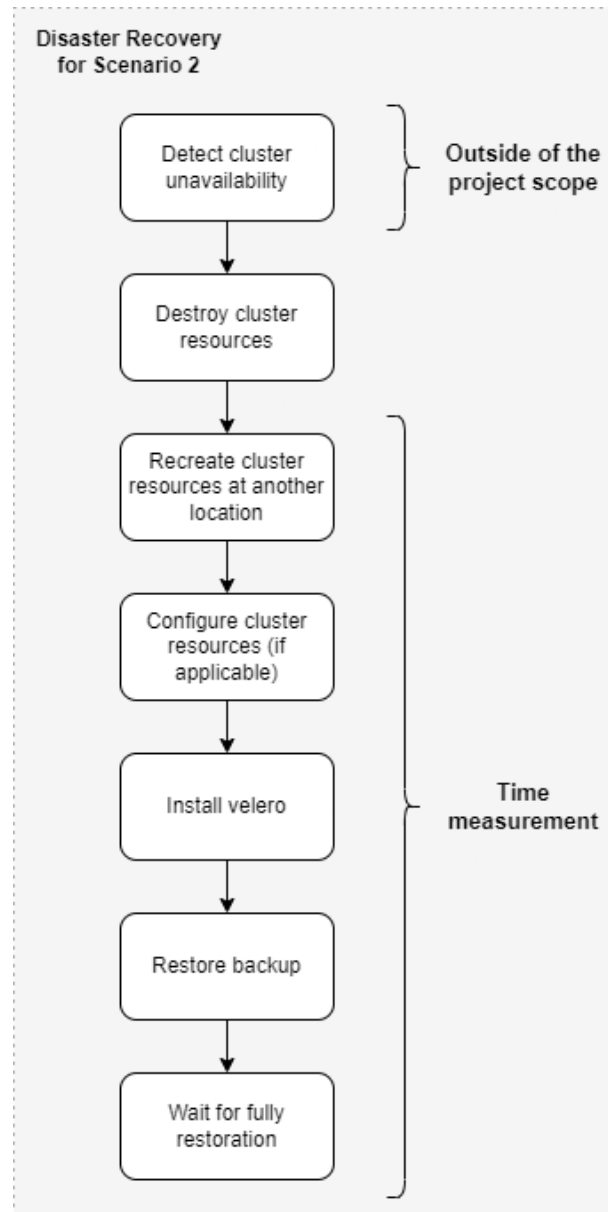


Figure 6 - Time measurement for scenario 2

Similarly, in Figure 6 is showcased the time measurement methodology. Again, disaster detection will be out of scope of this work, and this time, there is one additional step that will not count as total time, that is, the destruction of the cluster and cluster resources.

The rationale behind this decision, is that after a power issue at the zone/region, the cluster would be inaccessible. For the tests, the resources will be destroyed, as well as the cluster to simulate the disaster. But this time will not count towards the benchmark, as it is an artificial step.

Therefore, the first step after a disaster is to create the cluster at another location. This step could be fully automated, but the software should be aware of the location of the incidents, and this could be difficult to guess after a disaster, because the software should be aware of the areas that have not been affected. Otherwise, this step needs to be manual.

Then, after the creation of the cluster, and depending on the cloud provider, the initial configuration for the cluster needs to be applied. This could be, for example, the configuration of a storage provider, monitoring, or logging. Applications usually depend on other pieces of software in a ready state.

Following with the next step, is the installation of Velero. In order to issue a Velero restore process, Velero needs to be installed in the cluster. Then, after all these steps, the same steps as on the first scenario will be performed: The restore operation is issued, and the completion of it will be waited.

Finally, in order to know the number of tests needed to be performed, the formula from Cochran (Israel, 1992) is going to be employed:

$$n_0 = \frac{Z^2 pq}{e^2}$$

Where n_0 is the sample size, Z^2 is the abscissa of the normal curve that cuts off an area α at the tails ($1 - \alpha$ equals the desired confidence level, e.g., 95%), e is the desired level of precision, p is the estimated proportion of an attribute that is present in the population, and q is $1-p$. The value for Z is found in statistical tables which contain the area under the normal curve.

Due to the fact that the population is infinite in our scenarios, the formula can be successfully applied. Assuming a maximum variability $p = 0.5$ (as we do not know the characteristics of the population), and for a 95% confidence interval with $\pm 5\%$ precision, the resulting sample size is the following:

$$n_0 = \frac{1.96^2 * 0.5 * 0.5}{0.05^2} = 384.16$$

Which results in 385 samples, as it is always recommended to round up when calculating sample sizes. Unfortunately, this number of tests is higher than our allowed budget, as cloud costs are expensive. Therefore, the tests to be run on scenario 1 will be 50 for each environment. For scenario 2, the number will be 20 samples, because it is more expensive to run.

Additionally, the two-sample t-test is going to be calculated, to clearly check which cloud is faster in which scenario.

3.4 Summary

In this chapter the methodology for the deployment of the test environment, as well as the design of the benchmarks have been discussed. Kubernetes environments in both clouds and both scenarios have also been explained, as well as the deployment tools, such as Velero, Terraform, Helmfile, Kustomize and Bash, which will serve us as the fundamental DR stack.

The number of tests has also been calculated in this chapter, as there was no other study involving RPO, RTO and Kubernetes. However, the resulting number has resulted extremely high, and our budget is out of scope for that number.

4 Implementation

After implementation tools and scenarios were studied, the need of a fully automated disaster recovery process is paramount, so all the implementation has considered this requirement.

4.1 Software stack

In this section, the software to be used during the tests is going to be presented and described. Only the key/main pieces of software are explained, as the complete software stack that has been used cannot fit under this section. This includes the Disaster Recovery software, the programming languages used, and related Kubernetes tooling. It is worth noting, that some of the used software code is partially inspired and taken from the internal Claranet GmbH repositories.

4.1.1 Velero

Velero is, as already mentioned, the key basic open-source tool for disaster recovery in Kubernetes. It restores the cluster resources as well as the persistent volumes. Velero can be used for more use cases, like for migrations and replications of production clusters, but this project will focus on the DR use case.

Our scenario for disaster recovery will use its backup and restore services. An example restore operation is shown on Figure 7, which exemplifies the restore process in our experiments, which will be used many times for the tests, as opposed to the backup service, which will only be needed once.

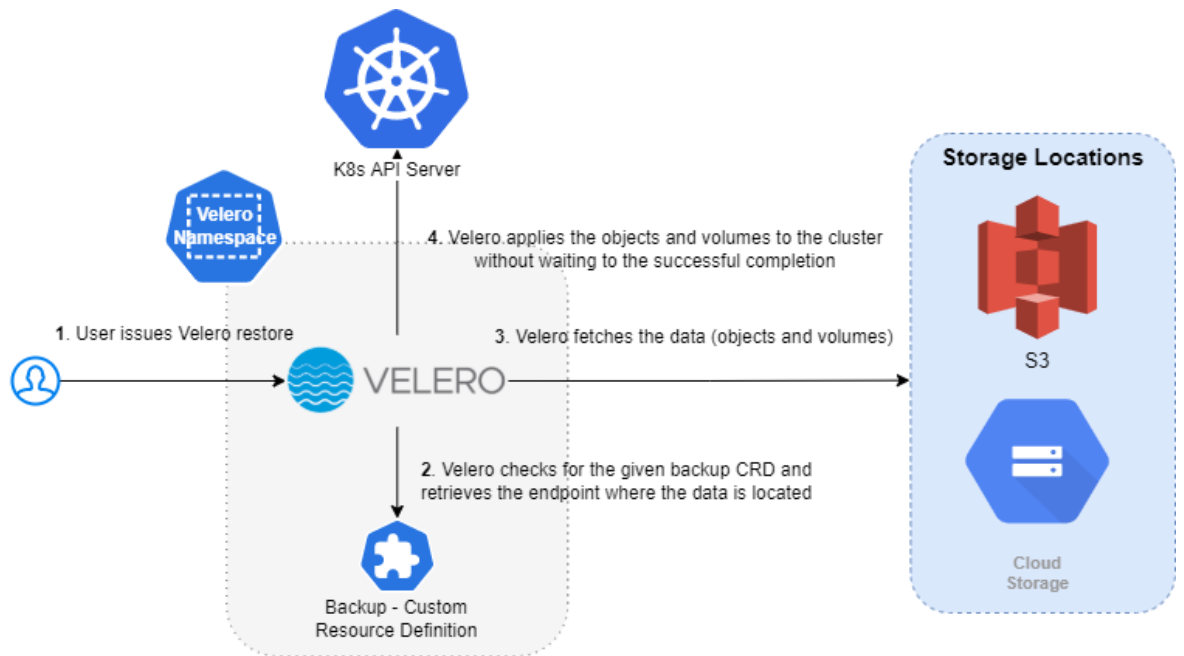


Figure 7 - Velero recovery process in the scenarios

4.1.2 Bash

Since Ubuntu is being used for our development, and the rest of the tools mainly offer Command Line Interface (CLI) tools, it was decided to use this approach. In this context, loops are programmed to replicate the tests for a definite number of times, and a system to execute commands after a successful restore has been designed.

On Figure 8, the main file can be seen for scenario 1. This file destroys the workload objects suffering the simulated DR and reapplies it from a previous backup.

```
#!/bin/bash
kubectl delete -k ./
until velero restore create $RANDOM-my-backup-$RANDOM --from-backup
test;
do
    sleep 1;
done;
until kubectl -n workload wait deploy/wordpress --timeout=300s --
for=condition=available;
do
    sleep 1;
done;
```

Figure 8 - Bash main.sh file for scenario 1

On Figure 9, the control loop is shown. This bash file calls recursively the previous file, timing the full run.

```
#!/bin/bash
TIMEFORMAT=%R;
for i in {1..50}
do
    sleep 5
    time timeout 300s ./main.sh
    sleep 5
done
```

Figure 9 - Bash loop.sh file for scenario 1

On Figure 10, the main file can be seen for scenario 2. This file applies the Terraform files, and then Velero is installed in the cluster, and the backup is restored.

```
#!/bin/bash
terraform -chdir=terraform/cluster/ apply -auto-approve
gcloud container clusters get-credentials sergio-test --zone europe-
    west1-b --project claranet-playground # FOR GKE
aws eks --region eu-west-3 update-kubeconfig --name sergio-test # FOR
    EKS
helmfile -f kubernetes/velero/helmfile.yaml apply
until velero restore create my-backup --from-backup test
do
    sleep 1;
done;
until kubectl -n workload wait deploy/wordpress --timeout=300s --
    for=condition=available;
do
    sleep 1;
done;
```

Figure 10 - Bash main.sh file for scenario 2

On Figure 11, the control loop is shown. This bash file calls recursively the previous file, timing the full run. The first lines are not including in the time tracking, as this are steps done in order to delete the cluster, which is not in scope for this project.

```
#!/bin/bash
TIMEFORMAT=%R;
for i in {1..20}
do
    kubectl delete svc -n workload wordpress
    terraform -chdir=terraform/cluster/ destroy -auto-approve
    kubectl config unset <cluster-name>
    sleep 5
    timeout 2000s time ./main.sh
    sleep 5
done
```

Figure 11 - Bash loop.sh file for scenario 2

4.1.3 Terraform

To interact with the cloud provider, Terraform files have been developed. The state has been saved locally, as the work is undergoing in just one computer. They mainly consist of a storage bucket, a Kubernetes cluster, and its network-related configuration, in addition to policies to access the cluster and the storage bucket.

In Amazon, the storage bucket service is called “S3”, and in GCP it is called “Cloud Storage”. Terraform version 1.1.6 will be used. The files are shown and explained in Appendix 2, since they occupy many pages.

4.1.4 Helmfile / Helm

To deploy Velero we will use Helmfile, which will make the deployment easy and repeatable. Helmfile is a wrapper for Helm to install this stack, using a predefined values file. Helm is a tool which could be considered as a package manager, which deploys software to a running Kubernetes cluster

```
context: gke_claranet-playground_europe-west1-b_sergio-test

repositories:
- name: vmware-tanzu
  url: https://vmware-tanzu.github.io/helm-charts

releases:
- name: velero
  namespace: velero
  chart: vmware-tanzu/velero
  version: 2.27.1
  labels:
    group: backup
    app: velero
    release: velero
  values:
    - velero.yaml
  set:
    - name: credentials.secretContents.cloud
      file: ../../secrets/terraform/service-accounts/velero-backups@claranet-playground.iam.gserviceaccount.com-secret.json
```

Figure 12 - Helmfile file for GCP

On Figure 12, the main Helmfile file is shown. VMware provides the chart of Velero, so it is used as our chart repository. Then, the Velero deployment is defined, using the Velero chart version 2.27.1. Credentials are finally needed in order to grant Velero “write” access to the Google Cloud Storage bucket.

```

image:
  repository: velero/velero
  tag: v1.7.1
  pullPolicy: IfNotPresent

initContainers:
- name: velero-plugin-for-gcp
  image: velero/velero-plugin-for-gcp:v1.3.0
  imagePullPolicy: IfNotPresent
  volumeMounts:
  - mountPath: /target
    name: plugins

metrics:
  enabled: false

kubect1:
  image:
    repository: docker.io/bitnami/kubect1

configuration:
  provider: gcp
  backupStorageLocation:
    name: gcp
    bucket: "velero-backups-sergio-test"

credentials:
  useSecret: true

upgradeCRDs: true
backupsEnabled: true
snapshotsEnabled: true
deployRestic: false

```

Figure 13 - Helmfile values for Velero chart (velero.yaml) for GCP

On Figure 13, the values file can be found, which configures details for the Velero deployment. First of all, Velero version 1.7.1 is going to be employed. Then the Velero plugin for Google Cloud Platform is set. Next, metrics are disabled, otherwise the deployment fails as there is no Prometheus server running in the cluster. Afterwards, the bucket name is specified, and backup and snapshots are enabled.

```
context:  arn:aws:eks:eu-west-3:384894877891:cluster/sergio-test

repositories:
- name: vmware-tanzu
  url: https://vmware-tanzu.github.io/helm-charts

releases:
- name: velero
  namespace: velero
  chart: vmware-tanzu/velero
  version: 2.27.1
  labels:
    group: backup
    app: velero
    release: velero
  values:
    - velero.yaml
```

Figure 14 - Helmfile file for AWS

On Figure 14, the Helmfile is shown. It is really similar to the GCP one, but it lacks the secret configuration in the end. This is due to AWS incorporating an OpenID Connect (OIDC) Identity Provider, in which a trust relationship can be established between a Velero Service Account and the S3 bucket to be configured.

```

image:
  repository: velero/velero
  tag: v1.7.1
  pullPolicy: IfNotPresent

initContainers:
- name: velero-plugin-for-aws
  image: velero/velero-plugin-for-aws:v1.3.0
  imagePullPolicy: IfNotPresent
  volumeMounts:
  - mountPath: /target
    name: plugins

podSecurityContext:
  fsGroup: 1337

metrics:
  enabled: false

kubect1:
  image:
    repository: docker.io/bitnami/kubect1

configuration:
  provider: aws
  backupStorageLocation:
    name: aws
    bucket: "sergio-tests-velero-backups"
    config:
      region: eu-west-3

serviceAccount:
  server:
    create: true
    name: velero
    annotations:
      eks.amazonaws.com/role-arn:
"arn:aws:iam::384894877891:role/velero-role"

credentials:
  useSecret: true

backupsEnabled: true
snapshotsEnabled: true
upgradeCRDs: true
deployRestic: false

```

Figure 15 - Helmfile values for Velero chart (velero.yaml) for AWS

On Figure 15, the values file can be seen, showing many similarities to the GCP one, but being more verbose due to the OIDC provider being configured. It includes a service account, which is created with the assigned role to communicate with the S3 bucket.

4.1.5 Kustomize: Test Application

To deploy the workload, Kustomize is used, which will ease the deployment and repeatability. In this case, the password is introduced in a shared space, to be distributed by both the WordPress client and the MySQL database, as shown in Figure 16. This example has been partially extracted from the official sample guide available from the Kubernetes webpage (*Example: Deploying WordPress and MySQL with Persistent Volumes* | Kubernetes, n.d.)

```
secretGenerator:  
- name: mysql-pass  
  namespace: workload  
  literals:  
  - password=testing123  
resources:  
- mysql-deployment.yaml  
- wordpress-deployment.yaml
```

Figure 16 - Kustomize file

On Figure 17 the WordPress deployment is shown. Firstly, the Service which will expose the deployment is developed. This will expose the application through a load balancer in port 80. This is not strictly required, but in order to check that the backup is actually in place, access to the UI is needed.

Then, the PersistentVolumeClaim object is defined, with 20Gi of capacity, with the read and write capabilities added. Finally, the Deployment object is shown. WordPress version 4.8 will be used, setting the database URL to the MySQL to be deployed. Also, the secrets are imported thanks to the previous Kustomize file.

On Figure 18, the MySQL deployment is shown. After defining the Kubernetes namespace in which the application will run, again a similar Service object is defined to expose MySQL for the WordPress deployment in port 3306. Likewise, there is a PersistentVolumeClaim configured, as well as a MySQL deployment with version 5.6 deployed.


```

apiVersion: v1
kind: Service
metadata:
  namespace: workload
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
  selector:
    app: wordpress
    tier: frontend
  type: LoadBalancer
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  namespace: workload
  name: wp-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: workload
  name: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
    spec:
      containers:
        - image: wordpress:4.8-apache
          name: wordpress
          env:
            - name: WORDPRESS_DB_HOST
              value: wordpress-mysql
            - name: WORDPRESS_DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-pass
                  key: password
          ports:
            - containerPort: 80
              name: wordpress
          volumeMounts:
            - name: wordpress-persistent-storage
              mountPath: /var/www/html
      volumes:
        - name: wordpress-persistent-storage
          persistentVolumeClaim:
            claimName: wp-pv-claim

```

Figure 17 - WordPress deployment file (wordpress-deployment.yaml)

```

apiVersion: v1
kind: Namespace
metadata:
  name: workload
---
apiVersion: v1
kind: Service
metadata:
  namespace: workload
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
  clusterIP: None
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  namespace: workload
  name: mysql-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: workload
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
        - image: mysql:5.6
          name: mysql
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-pass
                  key: password
          ports:
            - containerPort: 3306
              name: mysql
          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql
      volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pv-claim

```

Figure 18 - MySQL deployment file (mysql-deployment.yaml)

4.1.6 Additional Kubernetes utilities

To test, modify and work with the Kubernetes environment, there exist auxiliary tools which help to debug the deployments and troubleshoot issues.

The first tool to be mentioned is a tool to access the UI from the local environment, called kubefwd (*Kubefwd - Kubernetes Service Forwarding*, n.d.). Kubefwd is going to be employed several times during testing, to simplify the access and test the Persistent Volumes are correctly downloaded.

A second tool which will aid in our case is kubectl (*GitHub - Ahmetb/Kubectl: Faster Way to Switch between Clusters and Namespaces in Kubectl*, n.d.). This tool will assist in the process of switching between clusters by modifying the local kubeconfig file. This utility also comes with the kubens command, which allows the switching of Kubernetes namespaces, as well by modifying the local kubeconfig file.

4.1.7 Git / GitHub

The distributed version control system Git has been used to track the files of this project, as a backup system with GitHub, and as a way to transfer data from the different workstations where this project has been realised.

4.1.8 R language

In order to show the timings as visually attractive charts, and to calculate sample sizes, R language is to be employed, with mainly its “ggplot2” package, which eases the plotting of the resulting data.

On Figure 19, data is loaded from the timings data per environment and per scenario. Then, they are plotted for reference but without saving the output. Then, the data is extracted into a xlsx file, which is needed to be imported in this dissertation.

Then, the data is combined and plotted three times, using density plots. This time, the charts will be saved, to import then into this dissertation.

Finally, a t-test is going to be performed, to confirm or reject if the timings are significantly different from each other in the test setting. The file is shown in Figure 20.

```

library("stringr"); library("ggplot2"); library("dplyr"); library("writexl")

### GCP ###
gcp_cluster <- read.table("../gke/timings.txt", header = F, sep = " ", colClasses =
"numeric")
colnames(gcp_cluster) <- "time_elapsed"
ggplot(gcp_cluster, aes(x=time_elapsed)) +
  geom_histogram(bins=7)
write_xlsx(gcp_cluster, "gcp_cluster.xlsx")

gcp_workload <- read.table("../gke/kubernetes/workload/timings.txt", header = F, sep = " ",
colClasses = "numeric")
colnames(gcp_workload) <- "time_elapsed"
ggplot(gcp_workload, aes(x=time_elapsed)) +
  geom_histogram(bins = 20)
write_xlsx(gcp_workload, "gcp_workload.xlsx")

### AWS ###
aws_cluster <- read.table("../eks/timings.txt", header = F, sep = " ", colClasses =
"numeric")
colnames(aws_cluster) <- c("time_elapsed", "time_elapsed_oidc")
ggplot(aws_cluster, aes(x=time_elapsed)) +
  geom_histogram(bins=10)
ggplot(aws_cluster, aes(x=time_elapsed_oidc)) +
  geom_histogram(bins=10)
write_xlsx(aws_cluster, "aws_cluster.xlsx")

aws_workload <- read.table("../eks/kubernetes/workload/timings.txt", header = F, sep = " ",
colClasses = "numeric")
colnames(aws_workload) <- "time_elapsed"
ggplot(aws_workload, aes(x=time_elapsed)) +
  geom_histogram(bins = 20, color="darkblue", fill="lightblue") +
  #geom_vline(aes(xintercept=mean(time_elapsed), color="blue", linetype="dashed")) +
  geom_density(alpha=.92, fill="#FF6666")
write_xlsx(aws_workload, "aws_workload.xlsx")

### COMBINED ###
# Cluster
aws_cluster$group <- "aws"
aws_cluster_main <- aws_cluster %>% select(-time_elapsed_oidc)
gcp_cluster$group <- "gcp"
cluster_combined <- rbind(aws_cluster_main, gcp_cluster)
cluster <- ggplot(cluster_combined, aes(x=time_elapsed, fill=group)) +
  geom_density() +
  ggtitle("Time elapsed for DR in scenario 2 (with OIDC provider)") + labs(fill = "Vendor",
x="Time elapsed (s)", y="Density")
ggsave("cluster.png", plot=cluster)

# Cluster without OIDC
aws_cluster$no_oidc <- aws_cluster$time_elapsed - aws_cluster$time_elapsed_oidc
aws_cluster_no_oidc <- aws_cluster %>% select(c(-time_elapsed, -time_elapsed_oidc))
colnames(aws_cluster_no_oidc) <- c("group", "time_elapsed")
cluster_combined2 <- rbind(aws_cluster_no_oidc, gcp_cluster)
cluster_no_oidc <- ggplot(cluster_combined2, aes(x=time_elapsed, fill=group)) +
  geom_density(alpha=.5) +
  ggtitle("Time elapsed for DR in scenario 2 (without OIDC provider)") + labs(fill =
"Vendor", x="Time elapsed (s)", y="Density")
ggsave("cluster_no_oidc.png", plot=cluster_no_oidc)

# Workload
aws_workload$group <- "aws"
gcp_workload$group <- "gcp"
workload_combine <- rbind(aws_workload, gcp_workload)
workload <- ggplot(workload_combine, aes(x=time_elapsed, fill=group)) +
  geom_density(alpha=.5) +
  ggtitle("Time elapsed for recovery in scenario 1") + labs(fill = "Vendor", x="Time elapsed
(s)", y="Density")
ggsave("workload.png", plot=workload)

```

Figure 19 - R code to create the charts

```

gcp_cluster <- read.table("../gke/timings.txt", header = F, sep = " ", colClasses =
"numeric")
colnames(gcp_cluster) <- "time_elapsed"

gcp_workload <- read.table("../gke/kubernetes/workload/timings.txt", header = F, sep = " ",
colClasses = "numeric")
colnames(gcp_workload) <- "time_elapsed"

aws_cluster <- read.table("../eks/timings.txt", header = F, sep = " ", colClasses =
"numeric")
colnames(aws_cluster) <- c("time_elapsed", "time_elapsed_oidc")

aws_workload <- read.table("../eks/kubernetes/workload/timings.txt", header = F, sep = " ",
colClasses = "numeric")
colnames(aws_workload) <- "time_elapsed"

t.test(gcp_cluster$time_elapsed, aws_cluster$time_elapsed)
t.test(gcp_cluster$time_elapsed, aws_cluster$time_elapsed-aws_cluster$time_elapsed_oidc)
t.test(gcp_workload$time_elapsed, aws_workload$time_elapsed)

```

Figure 20 – R code to calculate the t-test

4.2 Cloud stack

In this section, cloud provider technologies are described, with the particularities of each one being considered, explaining the values to be applied in both test scenarios.

4.2.1 AWS

An EKS cluster is central to our deployment. The cluster to be deployed has a single, zonal master node, and a single worker node, with the instance type set to “c6i.large”, which provides two 3rd generation Intel Xeon Scalable CPUs and 4GBs of memory.

The Kubernetes version is going to be 1.21, and Spot instances will be used to reduce the cost of the continuous deployments. 20GB of disk storage will be requested for each worker node. An OIDC Identity Provider will also be requested to allow services to authenticate themselves to other AWS services. Region “eu-west-3” (Paris) will be used for the tests, as it is the closest option to where the test workstation is placed. The S3 bucket will be as well placed in the same region.

In order to download the kubeconfig file locally (to access the cluster from local) the command-line utility “aws” is going to be employed. This utility is able to perform many common and uncommon tasks in AWS.

4.2.2 GCP

On GCP, the GKE service is offered as a cloud-managed Kubernetes cluster. The to-be deployed cluster will be allocated in a single zone, with a single master node, with the instance type set to “n2-standard-2”, which provides two 2nd/3rd generation Intel Xeon Scalable CPUs (Depending on availability and location) and 8GBs of memory.

The Kubernetes version is going to be set as well to 1.21, and Preemptible instances will be used to reduce the cost of the continuous deployments. 20GB of disk storage will be requested for each worker node. Region “europe-west-1” (Belgium) will be used for the tests, as it is the closest option to where the test workstation is placed. The storage bucket will be as well placed in the same region.

4.3 Summary

In summary, there is a wide variety of software being used to perform the tests, a complex stack which uses complex pieces of software.

Velero is going to perform the actual backups and restores, Terraform interacts and commands cloud resources, Helm installs software in the cluster, Kustomize will install the test application in the cluster too, and Bash will be used to glue everything together. R language is utilised for plotting the charts, and both clouds (AWS, GCP) are going to be similarly used as our resource provider.

5 Experiments

In this chapter, the results of the experiments are described. They have been conducted following the study and design presented in previous chapters, employing the software mentioned in the last one.

5.1 Experimentation

In this section, the experiments are described as initially planned. To run the experiments, some preliminary work was needed to be performed, as the written software is expecting a fully set up cluster with backups and an application successfully configured. Therefore, firstly the preparation steps are **explained**, regardless of the cloud provider being put to test.

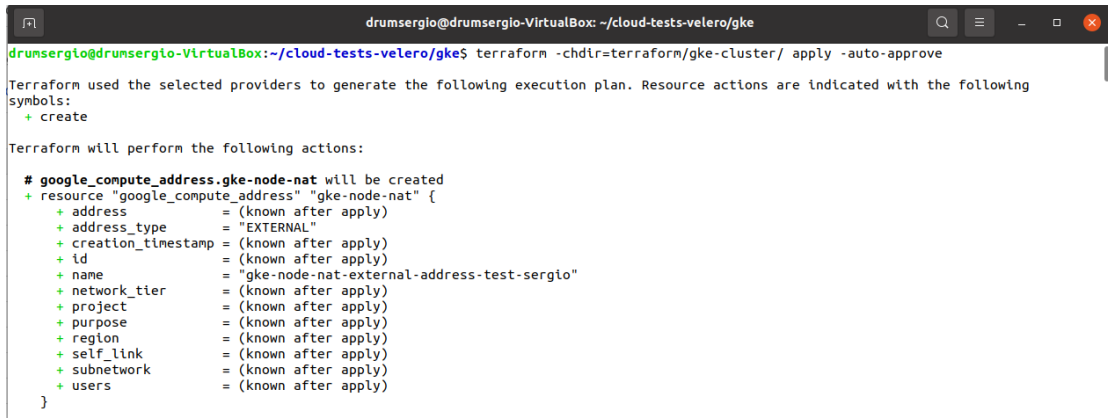
In this chapter, the experiments are described jointly by environment. In order to differentiate commands applied to the AWS and GCP environments, the scripts will be coloured in orange for AWS, and in blue for GCP. Also, to understand all the commands and its relative location. The tree folder structure is shown in Figure 21.



Figure 21 - Tree folder structure for the experiments

5.1.1 Implementation of experiments

In order to prepare the environments, the cluster should be completely configured, so some preliminary steps are defined. First, the Terraform code should be applied in both environments. This is showcased on Figure 22.



```
drumsergio@drumsergio-VirtualBox: ~/cloud-tests-velero/gke
drumsergio@drumsergio-VirtualBox:~/cloud-tests-velero/gke$ terraform -chdir=terraform/gke-cluster/ apply -auto-approve

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following
symbols:
+ create

Terraform will perform the following actions:

# google_compute_address.gke-node-nat will be created
+ resource "google_compute_address" "gke-node-nat" {
+   address           = (known after apply)
+   address_type      = "EXTERNAL"
+   creation_timestamp = (known after apply)
+   id                = (known after apply)
+   name              = "gke-node-nat-external-address-test-sergio"
+   network_tier      = (known after apply)
+   project           = (known after apply)
+   purpose            = (known after apply)
+   region            = (known after apply)
+   self_link         = (known after apply)
+   subnetwork        = (known after apply)
+   users             = (known after apply)
}
```

Figure 22 - Terraform GKE code being applied on GCP cloud

Similarly, the command to execute it in EKS would be the following.

```
terraform -chdir=terraform/eks-cluster/ apply -auto-approve
```

Also, on a different folder in which platform services are defined, terraform manifests must be applied. This is the place where the configuration of the bucket takes place. The terraform code can be found in Appendix 2.

```
terraform -chdir=gke/terraform/platform-services/ apply -auto-approve
```

```
terraform -chdir=eks/terraform/platform-services/ apply -auto-approve
```

After the cluster is up and running, the kubeconfig files need to be set that one may interact with the Kubernetes cluster. The commands to fetch the kubeconfig files are the following:

```
gcloud container clusters get-credentials sergio-test --zone europe-west1-b --project claranet-playground
```

```
aws eks --region eu-west-3 update-kubeconfig --name sergio-test
```

When the clusters are added locally, the application is next deployed through Kustomize.

```
kubectl apply -k ./gke/Kubernetes/workload/
```

```
kubectl apply -k ./eks/Kubernetes/workload/
```

Then, using kubefwd utility, a proxy is created for the Service object (The same command is used in both environments).

```
sudo -E kubefwd svc -n workload
```

Therefore, the local workstation can access the WordPress UI and configure a test blog, as seen in Figure 23.

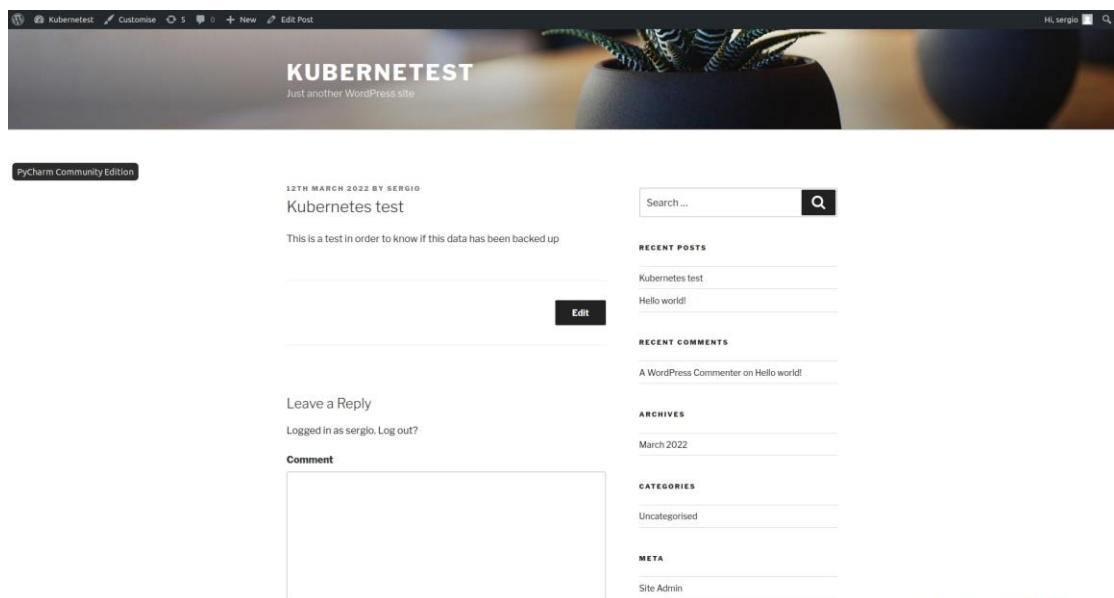


Figure 23 - Configured test WordPress site

Then, Velero must be installed to back up the running workload. The installation is done through Helmfile with the following command:

```
helmfile -f gke/kubernetes/velero/helmfile.yaml apply

helmfile -f eks/kubernetes/velero/helmfile.yaml apply
```

The Velero Helmfile includes a default preconfigured schedule as follows, which will be used to create the backup in the bucket, as seen on Figure 24.

```
schedules:
  my-schedule:
    schedule: "30 1 1 1 *"
    template:
      storageLocation: <storage-location> # "gcp" or "aws"
      ttl: "9999h0m0s"
      includedNamespaces:
        - workload
      includedResources:
        - '*'
```

Figure 24 - Part of the Helmfile values file which was added later to schedule the backup

Finally, when connected to the cluster, the schedule was triggered with the following command:

```
velero create backup my-test --from-schedule velero-my-schedule
```

Consequently, a backup is created called “my-test”, using the scheduled backup as a template.

Lastly, the clusters are teared down. However, this instruction is already included in the loop.sh file, which will be in charge of running the tests.

5.1.2 Experiment run

To undertake the tests for the first scenario (Software / system upgrades disaster), a simple command is needed, as follows:

```
./gke/kuberentes/workload/loop.sh
```

```
./eks/Kubernetes/workload/loop.sh
```

This will make the test run 50 times for each environment. The test time is relatively short, being less than 1h30min long for both environments, approximately. However, this has taken longer in Google because there were some random errors for unknown reasons while restoring the backup.

On the other hand, the tests for the second scenario (Area outage, by any means, like power outages) are the following:

```
./gke/loop.sh
```

```
./eks/loop.sh
```

The tests have been run 20 times in each environment. The test time is long, especially in AWS, where it has taken more than 30 hrs. in total, approximately. In GCP the time has been considerably less, amounting to less than 7 hrs., approximately. The tests have been run at different times of the day and in different days, until the target number of samples have been gathered.

For AWS, the test included the collection of two timings: Total time and the time spent configuring the OIDC Identity Provider. The configuration of this service is very time-consuming. However, this is the recommended way of accessing AWS-managed services like S3. This metric was collected to be later subtracted it from the total time, with the aim of having a fairer comparison between the two cloud providers.

5.2 Summary

In this chapter, the preparation of the experiments and the test runs were carried out, both in AWS and GCP. This paved the way to analyse the results in the next chapter.

6 Evaluation

In this chapter, and having completed the experiments, it is time to evaluate the work being carried out and extracting the value from it. In this chapter, a critical evaluation of the strengths and limitations will be accomplished.

6.1 Results

Table 1 - Timings for Scenario 1 in GCP

Test Number	Total time (s)
1	97.169
2	88.369
3	87.561
4	88.075
5	95.675
6	79.261
7	97.773
8	92.474
9	81.676
10	90.333
11	99.42
12	82.195
13	88.256
14	96.287
15	90.197
16	103.238
17	98.242
18	90.221
19	90.462
20	90.84
21	90.038
22	92.484
23	91.895
24	90.92
25	85.278
26	99.36
27	87.096
28	82.384
29	113.442
30	99.893
31	91.831
32	99.5
33	87.458
34	82.034
35	98.622
36	102.407
37	95.122
38	86.494
39	80.501
40	100.771
41	93.897
42	82.844
43	81.582
44	84.625
45	99.888
46	84.646
47	76.075
48	84.587
49	84.069
50	95.563

Table 2 - Timings for Scenario 1 in AWS

Test Number	Total time (s)
1	61.282
2	83.576
3	37.157
4	50.238
5	47.2
6	40.205
7	49.232
8	71.498
9	49.453
10	44.9
11	48.39
12	73.4
13	53.499
14	42.074
15	76.576
16	45.243
17	34.415
18	62.404
19	65.798
20	32.411
21	43.032
22	47.234
23	72.357
24	50.451
25	41.484
26	49.435
27	58.307
28	52.263
29	86.458
30	44.192
31	39.266
32	37.196
33	184.721
34	62.301
35	52.272
36	67.6
37	45.246
38	58.305
39	73.557
40	49.226
41	37.184
42	70.587
43	60.282
44	54.298
45	73.797
46	43.2
47	86.668
48	45.199
49	32.252
50	63.325

Beginning with the first scenario, where 50 tests were arranged and executed, the results have been the following, being Table 1 for GCP and Table 2 for AWS

In order to understand the data, a chart has been developed with R (Figure 25). This draws a kernel density estimate, which is a smoothed version of the histogram (*Geom_density Function - RDocumentation*, n.d.).

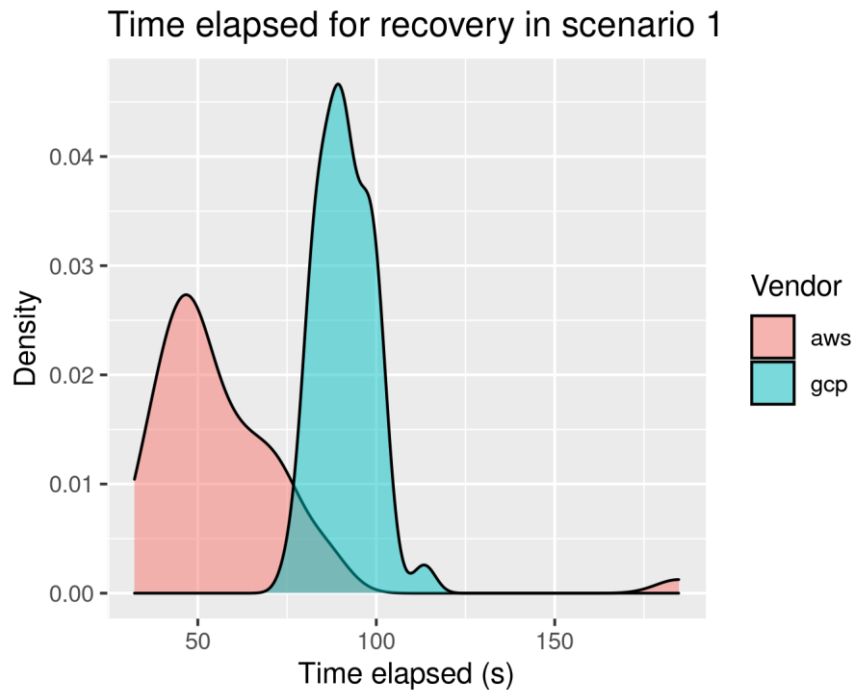


Figure 25 - Density plot for Scenario 1

After executing the t-test, the conclusion is that the p-value of the tests is less than the significance level $\alpha = 0.05$ (actually, $p = 4.908 * 10^{-14}$). Consequently, the conclusion is that the GCP timings average is significantly different from AWS. Thus, EKS is considerably faster when recovering from a failed software or a failed system, with the following means:

```
mean_aws_workload <- mean(aws_workload$time_elapsed)
mean_aws_workload
57.01292

mean_gcp_workload <- mean(gcp_workload$time_elapsed)
mean_gcp_workload
91.0606

mean_gcp_workload - mean_aws_workload
```

34.04768

Therefore, the disaster recovery scenario in EKS is 34.04768 seconds faster in mean than the GKE cluster recovery.

Now addressing the second scenario, where 20 tests were arranged and executed, the results are found below, being Table 3 for GCP and Table 4 for AWS:

Table 3 - Timings for Scenario 2 in GCP

Test Number	Total time (s)
1	759.945
2	760.224
3	790.448
4	744.852
5	754.157
6	750.142
7	723.589
8	753.339
9	822.367
10	762.845
11	762.151
12	736.85
13	756.257
14	762.556
15	736.402
16	706.114
17	703.242
18	752.527
19	735.453
20	766.101

Table 4 - Timings for Scenario 2 in AWS

Test Number	Total time (s)	Time spent configuring OIDC (s)
1	2,594.727	1,805
2	2,885.034	2,133
3	2,766.108	1,820
4	2,570.61	1,821
5	2,629.164	1,871
6	2,611.574	1,827
7	2,593.397	1,820
8	2,911.612	2,075
9	2,559.397	1,793
10	2,675.619	1,900
11	2,679.319	1,806
12	2,746.628	1,739
13	2,566.481	1,798
14	2,558.722	1,827
15	2,576.981	1,813
16	2,620.607	1,807
17	2,501.998	1,746
18	2,420.686	1,692
19	2,348.576	1,655
20	2,701.98	1,942

In order to understand the data, another two charts have been developed with R, which also will draw a kernel density estimate. This time, the second chart will have the OIDC Identity Provider timing removed, to compare the providers in a different setting. Both are represented in Figure 26 and Figure 27.

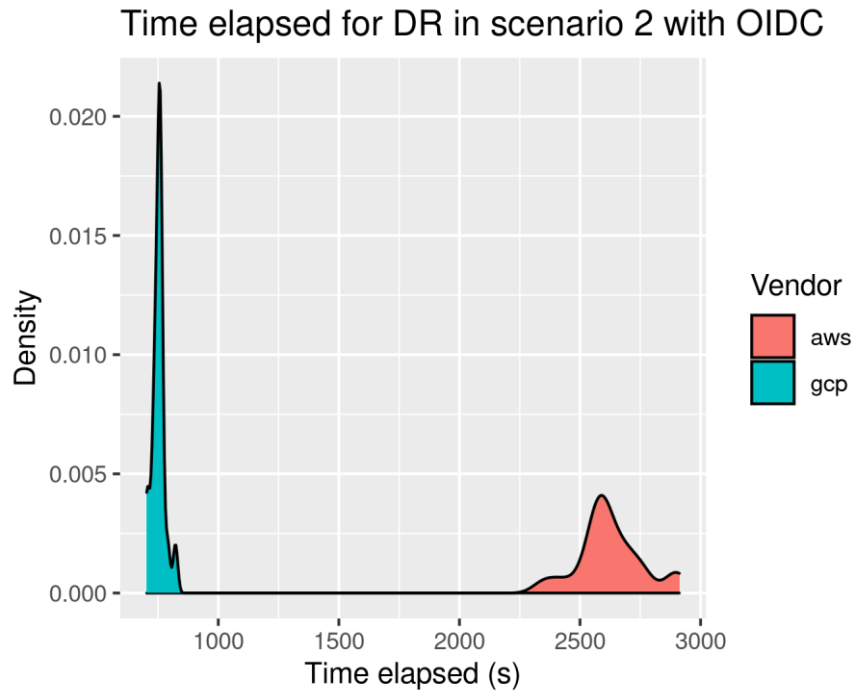


Figure 26 - Density plot for Scenario 2

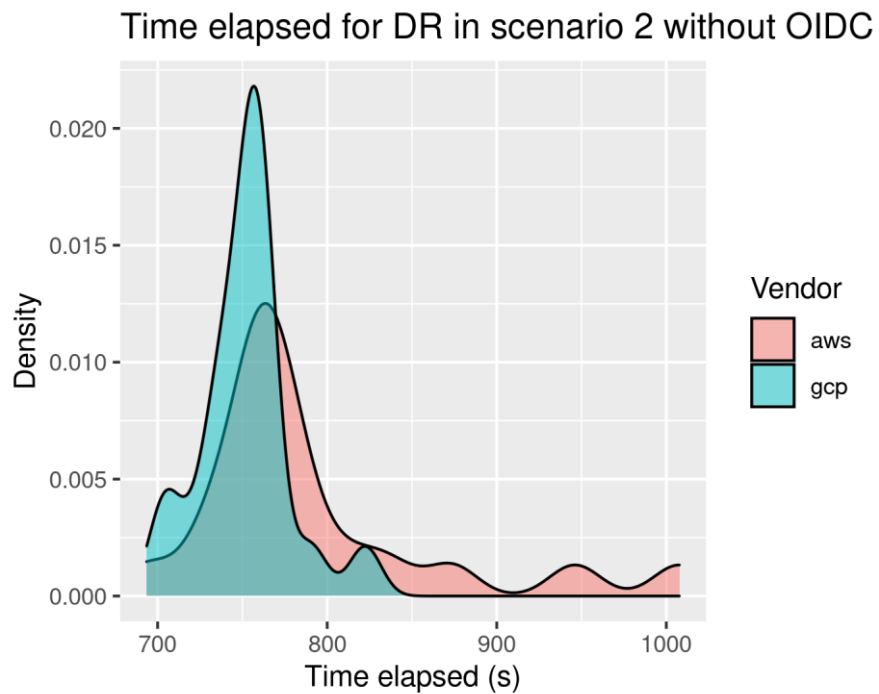


Figure 27 - Density plot for Scenario 2 (subtracting OIDC provider)

After executing the t-test, for both settings (with and without the OIDC Identity Provider) the conclusions are as shown:

The p-value for the first assumption is clearly below the significance level $\alpha = 0.05$ (actually, $p = 2.2 * 10^{-16}$). Consequently, the first conclusion is that the GCP timings average is significantly different from AWS. Thus, EKS is excessively slower when recovering from an area-cluster disaster, with the ensuing means:

```
mean_aws_cluster <- mean(aws_cluster$time_elapsed)
mean_aws_cluster
2625.961
mean_gcp_cluster <- mean(gcp_cluster$time_elapsed)
mean_gcp_cluster
751.9781
mean_aws_cluster - mean_gcp_cluster
1873.983
```

Therefore, the disaster recovery scenario in the GKE cluster is 1873.983 seconds faster in the calculated mean than the EKS cluster recovery.

However, this value is biased due to the fact that AWS employs the OIDC identity provider, which is not available in GCP. If this time is discounted, the p-value for the second assumption is remarkably below the significance level $\alpha = 0.05$ (actually, $p = 0.03$). Consequently, the first conclusion is that the GCP timings average is different from AWS. Thus, EKS is slightly slower when recovering from an area-cluster disaster without OIDC provider in place, with the following means:

```
mean_aws_cluster_no_oidc <- mean(aws_cluster$time_elapsed -
aws_cluster$time_elapsed_oidc)
mean_aws_cluster_no_oidc
791.461
mean_gcp_cluster <- mean(gcp_cluster$time_elapsed)
mean_gcp_cluster
751.9781
mean_aws_cluster_no_oidc - mean_gcp_cluster
39.48295
```

Therefore, the disaster recovery scenario in the GKE cluster, even when subtracting AWS OIDC, is 39.48295 seconds faster in mean than the EKS cluster recovery.

6.2 Strengths and Limitations

In this section, a subjective compilation of the strengths and limitations of this research is given.

Strengths

- i. One strength of this project is that the studied scenarios are practical, and they can cover a wide range of disasters, not only those which were studied in this context. The results from the 2010 Symantec study (*Symantec 2010 Disaster Recovery Study*, 2010) about incidence on disasters are arguably still valid today, and companies are still facing several types of disasters.
- ii. Also, all the software used for the testing is being actively used in a wide variety of companies and industries. Sharing this knowledge and employing it to perform individual tests in the research, development, and innovation activities of these companies is beneficial for all the community.
- iii. Another strength is the number of tests achieved. Usually, the cloud provider costs hamper the testing and prevents the research to come to fruition. This research is only commonly available at the same cloud companies for internal Key Performance Indicators (KPIs), and this effort has been made to make public the efficiencies, to guide a prospect client when deciding provider and calculating its risks.

Limitations

- i. This research is limited to the studied disasters, as well as some technical parts of it, for instance, leaving out alerting, or the simulation of a disaster through chaos engineering. Furthermore, the topic should be a cross-departmental, joint effort in a business context (Kirvan, 2022).
- ii. The breadth of different configurations in the cloud providers is astonishing. And there is no exact match from one cloud provider to another. In this research, common/suggested values and resources have been applied, and similar resources have been deployed. However, they are not the same.
- iii. Study of all Disaster Recovery methods. In GCP and especially in AWS, there is much advice on how to perform disaster recovery based on budget. In this project, the focus has been on low-budget scenarios. Scenarios in which the cluster is recovered in a couple of seconds have not been studied. Additionally, a specific managed service for Disaster Recovery for GKE has not been included in the analysis (*Backup for GKE* | Google Cloud, n.d.)
- iv. There are many cloud providers which were not tested, especially Azure, commonly regarded as the third player in cloud providers. This project focused mainly in AWS and GCP.
- v. In connection with the budget and the number of experiments, they have been cautiously spent, and the number is fine when compared to the inexistence of these tests on literature. However, the number of experiments were meant to achieve the results from the sample size formula.
- vi. In regard to the use of Spot/Preemptible instances, to drive the costs down, these types of instances have been instantiated. Cloud resources are shared, and this is the reason why there is variability in the charts, but by using Spot/Preemptible instances the variability is a higher risk. This is maybe the reason of the appearance of a few outliers in the

charts, or the failure of a recovery process which was experienced seldomly.

6.3 Comparison to Others' Work

This project has used in several ways the foundations of the work of Suguna & Suhasini (2015) concerning disaster occurrence, such as electing the two most recurring disasters mentioned in their work, and techniques in DR, like the cold-site backup, and applied it to Kubernetes. In a sense, this project could be considered as a continuation and as an extension of their work, this time including Kubernetes and adding a comparison between different cloud providers—not just via estimates on RTO and RPO—which is the case in their work in Table 2. Furthermore, these estimates are effectively outdated, and as seen in the results of this research, now the recovery level achieved in terms of RTO in this project is arguably the same as the top recovery level set in their work (being a mirrored data with failover). In this project, nothing has been mirrored. These fast timings can be achieved thanks to the public cloud providers.

Wood et al. (2010) approaches the topic from a cost perspective, comparing the Public Cloud and Colocation in Figures 2 and 3. In our project, cost has not been studied, but a similar approach has been taken to measure the differences between AWS and GCP in terms of timings, using two distinct scenarios to compare the big differences that can emerge from two rather similar scenarios.

Minh Bui (2020) uses Velero to design a Disaster Recovery solution for a particular company. In this scenario, Kubernetes objects are backed up to AWS. However, it does not include any backup of Kubernetes volumes. In the current dissertation, the AWS and GCP plugins for the storage of volumes were also incorporated to the research, bringing it closer to reality.

6.4 Overview on research questions and aims

The research questions were the following:

1. How to decrease the RTO and RPO of a fully production Kubernetes cluster after a disaster in the cloud?
2. Where are the limits and consequences on employing the identified technologies?
3. What differences exist in relation to Disaster Recovery between cloud providers?

With reference to the first research question, in this study, the methodology used for the DR of both scenarios has been similar. The study could have included a comparison between DR in a cold standby, and a DR of a mirrored scenario, but as explained earlier on, the costs and the time to complete it have been a restriction. However, as explained in chapters 1 and 2, there are mainly three (Suguna & Suhasini, 2015) or perhaps four (Eliot, 2021) approaches to DR. As per Suguna & Suhasini, namely, hot, warm, and cold standby, and as per Eliot, Backup & Restore, Pilot Light, Warm Standby and Multi-site-active/active. The concepts and the terminology are slightly outdated because they apply to a virtual machine-based scenario rather than a container-based scenario, yet they help in defining the different service levels that a Kubernetes cluster could achieve. A basic, first level would be a backup & restore scenario, which was the case in our study. In higher tiers, a second disaster recovery site should be provisioned, scaled down or fully scaled, contributing towards RTO, and near or real-time data transfer, contributing to RPO.

Regarding the second research question, the limits of applying DR technology are found to be costs, engineer-hours, and privacy concerns. The consequences faced for the complete definition and execution of a DRP is primarily cost. Costs generally increase when targeting for a lower RTO and RPO, as shown by Wiboonrat (2008). In our experiments, RTOs and RPOs were relatively and deliberately slow due to the tight budget. However, data loss could almost completely be avoided with close to 0 second RPOs. Secondly, another consequence of DRPs is that engineer-hours are scarce. They can be conceived with a different meaning from simply costs, as human resources are

valued elements which are not only finite, but difficultly replaceable. Thirdly, and a repercussion which at first sight could go unnoticed, has to do with data storage location and protection, privacy, and security (Takabi et al., 2010). Under this presented research, this does not imply any sort of problem. However, when using cloud-backed global storage, data could reside in countries or continents not previously agreed with clients, or not compliant with local regulations.

Another way of answering to the second question, is to review the limits and consequences of the precise technology employed in this project. Starting with Terraform, this supposes that the company is already using Terraform to deploy its DR. Transitioning from manually created cloud resources could be challenging depending on the scenario, although Terraform includes a functionality to import resources. Another limitation is that, as of now, it does not offer Graphical User Interface (GUI). Other solutions which could serve as alternatives to Terraform include Pulumi or Cloudify. Following now with Velero, one of its limitations, is that objects do not get overwritten if they exist already. Another consequence is that the Velero version, Kubernetes version and Helm version in the recovery cluster must be the same of the original backup, and this information may be unknown. This applies as well for the number of nodes, which should be greater or at least equal as of when creating the backup.

As to the third research question, it could be argued that there are no fundamental differences about DR architecturally, as shown in this work. In fact, AWS and GCP offerings are surprisingly similar. For instance, both offer regional clusters, which, in a way, are built-in DR methods (Alvarez-Parmar, 2020) (*Regional Clusters | Kubernetes Engine Documentation | Google Cloud*, n.d.) and similar architectures could be designed and implemented in both cloud providers. On the other hand, as an example, GCP offers a managed service for backing up the GKE cluster, called “Backup for GKE” (*Backup for GKE | Google Cloud*, n.d.), which is an alternative method to Velero for doing basic backups with a low pricing.

Regarding the main output from this study, which is the timings between GKE and EKS services, it could be argued that there are differences, especially on a zonal disaster scenario, in which an EKS cluster would take more than triples the time to recover. Additionally, in the first scenario of a software update issue, EKS would be slightly faster. Although this information is unlikely to change the decision on what cloud provider to select, it is a critical piece of information linked to SLA calculations, and similarly, RPO and RTO.

7 Conclusions

There are many Disaster Recovery types nowadays, based on the disaster, technology used, recovery type, and most important, business needs. As discussed in Chapters 1 and 2, businesses need to critically assess their risks, and to design a DRP that suits their needs. Therefore, this project has aimed to clearly differentiate the performance of DR in the Kubernetes clusters of distinct cloud providers. Additionally, in this work the different possible disasters have been evaluated, as well as different DR methods.

7.1 Research Overview

This project aimed to compare RTOs and RPOs of a set of given disasters within the context of cloud managed Kubernetes clusters. The selected cloud providers were AWS and GCP, and the selected disasters were two: Software/System update failure, and power issues, area wide.

The project aimed to track RPO, as this was mentioned in the project proposal. Unfortunately, after studying RPO in the cloud, the only aspect worth extracting out of our project is that it depends entirely on the schedule configuration of Velero. It is true that there are solutions involving mirrored PVs in Kubernetes, but the cloud providers do not offer such service, instead it would rely on a third-party software, and this would be out of scope for this project, since RPO cannot be benchmarked, but set accordingly to predefined times. Nevertheless, the only found open-source data mirroring solution for Kubernetes is merely on its inception (Tomlinson, n.d.). Application-based replication is not considered (*Run a Replicated Stateful Application | Kubernetes*, n.d.).

Even benchmarking Velero for a quick PV snapshotting in both cloud providers would not give back useful results, as this is clearly not used in practice, given the fact that if a backup or snapshot is made every couple of seconds. In this situation, then, a mirrored backups would be the to-go solution, perhaps additionally combined with some classical DR backup, with a longer time between backups.

Overall, the expectations and goals have been met. A thorough review of the available software and configuration of cloud providers was done prior to the start of the experimentation and before the writing of this document. The experiments and benchmarks helped characterize RTO timings in both scenarios, although there is more avenues which could be investigated in future research.

7.2 Main takeaways

The takeaways from this dissertation have been several, since even the whole strategy and concept to finish this dissertation has been one. Focusing on the main ideas, these would be the key points:

- Disaster Recovery is a loosely defined topic, which involves many different subjects into one single goal: Recovery.
- The breadth of software involved to recover from a disaster is considerable. Technological stack knowledge is critical for a successful recovery.
- There are many avenues for every budget regarding Disaster Recovery. RTO and RPO, and its target KPIs SLAs and SLOs are directly correlated with budget, if wisely spent.
- The degree of variability in cloud resources is something to have into account not only for disaster recovery, but when planning and identifying risks.
- The importance of directives from management to create a successful Disaster Recovery Plan.
- RTO varies with cloud and scenario. As it has been found, EKS was faster on a software/system issue, whereas GKE was noticeable faster.

7.3 Future work & Recommendations

The project has been centred on very concrete scenarios, for unlikely real workloads, with specific cloud/software configurations. A real-world project, paired with mirror environments to estimate timings would be a perfect way to characterise a given scenario for a given company, perfecting the estimates given in this research project. However, the cost would be particularly high. Not only as per cloud spendings, but because of engineer man-hours. Nevertheless, the insights could be priceless.

Testing by using crossed storage domains might as well be remarkably interesting. That is, using S3 storage service while on GKE, and using Cloud Storage on an EKS cluster. Egress traffic is expensive, but little data could be transferred to lower its costs. Using this approach, the characterisation would be more in-depth, as a comparison could be performed, in order to address which combination is the fastest, or which component is lagging down the recovery.

Another avenue would be to test using different kinds of resources, to characterise the different variables for the timings. Also, avoiding the use of spot/preemptible instances, to compare the appearance of outliers or the timings when bootstrapping a new instance.

Testing the velocity of Velero snapshots could be another way of continuing this research. As explained before, this would not be practical for use at a company, given the fact that there are other better solutions for replicating data in a rapid way. However, by doing so, a comparison between velocities in different cloud providers could shed some light into how capable they are at transferring data at high throughputs from two different services are.

Another clear avenue would be to continue the path this project has taken and expand its scope to other clouds or private data centres, especially having into account the fact that the cloud provider Azure was not included. For it to be tested inside a same cloud vendor, Velero should contain a plugin for the storage side. Velero officially supports plugins currently for Alibaba, VMware

vSphere, DigitalOcean, HPE Storage, OpenEBS, OpenShift, Portworkx, Storj, Container Storage Interface (CSI) and OpenStack (*Velero Plugins*, n.d.).

Another option would be to include and compare costs of setting up DR in different cloud providers, following the course of action of Wood et al. (2010), extending the analysis in this way, focusing just on the cloud offering, as well as in the different DR methods in each cloud provider.

Lastly, another possibility would be to also test the newly available service “Backup for GKE”, which would compete against our tests. A characterisation could be made, as well as a comparison with the tests done in the GKE cluster. At the present time, nevertheless, there was no similar software available for EKS.

8 References

- Al-Kiswany, S., Subhraveti, D., Sarkar, P., & Ripeanu, M. (2011). VMFlock: Virtual machine co-migration for the cloud. *Proceedings of the IEEE International Symposium on High Performance Distributed Computing*, 159–170. <https://doi.org/10.1145/1996130.1996153>
- Alhazmi, O. H., & Malaiya, Y. K. (2012). Assessing disaster recovery alternatives: On-site, colocation or cloud. *Proceedings - 23rd IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW 2012*, 19–20. <https://doi.org/10.1109/ISSREW.2012.20>
- Alhazmi, O. H., & Malaiya, Y. K. (2013). Evaluating disaster recovery plans using the cloud. *Proceedings - Annual Reliability and Maintainability Symposium*. <https://doi.org/10.1109/RAMS.2013.6517700>
- Alvarez-Parmar, R. (2020, December 7). *Operating a multi-regional stateless application using Amazon EKS | Containers*. <https://aws.amazon.com/blogs/containers/operating-a-multi-regional-stateless-application-using-amazon-eks/>
- Amarnath, A. (2020, September 16). *Velero 1.5: Auto volume backup with restic, DeleteItemAction plugins, Restore Hooks, and much more!* <https://velero.io/blog/velero-1.5-for-and-by-community/>
- Amazon EKS Service Level Agreement*. (2020). <https://aws.amazon.com/eks/sla/>
- Anand, G., & Kodali, R. (2008). Benchmarking the benchmarking models. *Benchmarking*, 15(3), 257–291. <https://doi.org/10.1108/14635770810876593/FULL/PDF>
- Andrade, E., Nogueira, B., Matos, R., Callou, G., & Maciel, P. (2017). Availability modeling and analysis of a disaster-recovery-as-a-service solution. *Computing*, 99(10), 929–954. <https://doi.org/10.1007/S00607-017-0539-8/FIGURES/9>
- As Quarterly Cloud Spending Jumps to Over \$50B, Microsoft Looms Larger in Amazon's Rear Mirror | Synergy Research Group*. (2022, February 3). <https://www.srgresearch.com/articles/as-quarterly-cloud-spending-jumps-to-over-50b-microsoft-looms-larger-in-amazons-rear-mirror>
- Backup and Restore with Velero | Administration Guide | SUSE CaaS Platform 4.2.4*. (n.d.). Retrieved April 2, 2022, from <https://documentation.suse.com/suse-caasp/4.2/html/caasp-admin/id-backup-and-restore-with-velero.html>
- Backup for GKE | Google Cloud*. (n.d.). Retrieved March 16, 2022, from <https://cloud.google.com/kubernetes-engine/docs/add-on/backup-for-gke/concepts/backup-for-gke>
- Baginda, Y. P., Affandi, A., & Pratomo, I. (2018). Analysis of RTO and RPO of a service stored on Amazon Web Service (AWS) and Google Cloud Engine (GCE). *Proceedings of 2018 10th International Conference on Information Technology and Electrical Engineering: Smart Technology for Better Society, ICITEE 2018*, 418–422. <https://doi.org/10.1109/ICITEED.2018.8534758>
- Bakshi, D., & Kim, B. (2021, November 12). *Disaster Recovery with AWS Managed Services, Part I: Single Region | AWS Architecture Blog*. <https://aws.amazon.com/es/blogs/architecture/disaster-recovery-with->

- aws-managed-services-part-i-single-region/
- Balla, D., Simon, C., & Maliosz, M. (2020). Adaptive scaling of Kubernetes pods. *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2020: Management in the Age of Softwarization and Artificial Intelligence, NOMS 2020*.
<https://doi.org/10.1109/NOMS47738.2020.9110428>
- Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, omega, and kubernetes. *Communications of the ACM*, 59(5), 50–57.
<https://doi.org/10.1145/2890784>
- Burns, B., & Tracey, C. (2018). Managing Kubernetes: Operating Kubernetes Clusters in the Real World. In *Managing Kubernetes: Operating Kubernetes Clusters in the Real World*.
- Cegieta, R. (2006). Selecting technology for disaster recovery. *Proceedings of International Conference on Dependability of Computer Systems, DepCoS-RELCOMEX 2006*, 160–167. <https://doi.org/10.1109/DEPCOS-RELCOMEX.2006.49>
- Chandrasekaran, A. (2020a). *Best Practices for Running Containers and Kubernetes in Production*.
- Chandrasekaran, A. (2020b). Best Practices for Running Containers and Kubernetes in Production. *Gartner, August*, 1–14.
<https://www.gartner.com/document/3902966>
- Costello, K., & Rimol, M. (2021, April 21). *Gartner Forecasts Worldwide Public Cloud End-User Spending to Grow 23% in 2021*.
<https://www.gartner.com/en/newsroom/press-releases/2021-04-21-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-23-percent-in-2021>
- de Souza Couto, R., Secci, S., Elias Mitre Campista, M., & Henrique Maciel Kosmowski Costa, L. (2014). *Network Design Requirements for Disaster Resilience in IaaS Clouds*. <http://aws.amazon.com/ec2-sla>.
- Disaster recovery planning guide | Cloud Architecture Center | Google Cloud*. (n.d.). Retrieved January 29, 2022, from
<https://cloud.google.com/architecture/dr-scenarios-planning-guide>
- Eliot, S. (2021, May 14). *Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby | AWS Architecture Blog*. AWS Architecture Blog. <https://aws.amazon.com/blogs/architecture/disaster-recovery-dr-architecture-on-aws-part-iii-pilot-light-and-warm-standby/>
- Example: Deploying WordPress and MySQL with Persistent Volumes | Kubernetes*. (n.d.). Retrieved March 10, 2022, from
<https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/>
- Fallara, P. (2004). Disaster Recovery Planning. *IEEE Potentials*, 23(5), 42–44. <https://doi.org/10.1109/MP.2004.1301248>
- geom_density function - RDocumentation*. (n.d.). Retrieved March 15, 2022, from
https://www.rdocumentation.org/packages/ggplot2/versions/3.3.5/topics/geom_density
- GitHub - ahmetb/kubectx: Faster way to switch between clusters and namespaces in kubectl*. (n.d.). Retrieved March 14, 2022, from
<https://github.com/ahmetb/kubectx>
- Google Kubernetes Engine SLA | Google Cloud*. (2021).

- <https://cloud.google.com/kubernetes-engine/sla>
- Gunawan, A. S., Allen, M., Alhazmi, O. H., & Malaiya, Y. K. (2013). *Disaster Recovery Plans Cloud Evaluating Disaster Recovery Plans Using the Cloud*.
- Israel, G. D. (1992). Determining Sample Size 1 The Level of Precision. *University of Florida*.
- Kirvan, P. (2022, March 9). *Cloud-era disaster recovery planning: Maintenance and continuous improvement*.
<https://www.computerweekly.com/feature/Cloud-era-disaster-recovery-planning-Maintenance-and-continuous-improvement>
- kubefwd - Kubernetes Service Forwarding. (n.d.). Retrieved March 14, 2022, from <https://kubefwd.com/>
- kubernetes-sigs/kustomize: Customization of kubernetes YAML configurations. (n.d.). Retrieved February 23, 2022, from <https://github.com/kubernetes-sigs/kustomize>
- Kubernetes. (n.d.). Retrieved October 30, 2021, from <https://kubernetes.io/>
- Lam, W. (2002). Ensuring business continuity. *IT Professional*, 4(3), 19–25.
<https://doi.org/10.1109/MITP.2002.1008533>
- Lamouchi, N. (2021). Adding Anti-Disaster Layers. *Pro Java Microservices with Quarkus and Kubernetes*, 187–251. https://doi.org/10.1007/978-1-4842-7170-4_6
- Larsson, L., Tärneberg, W., Klein, C., Elmroth, E., & Kihl, M. (2020). Impact of etcd deployment on Kubernetes, Istio, and application performance. *Software: Practice and Experience*, 50(10), 1986–2007.
<https://doi.org/10.1002/SPE.2885>
- Lozupone, V. (2017). Disaster recovery plan for medical records company. *International Journal of Information Management*, 37(6), 622–626.
<https://doi.org/10.1016/J.IJINFOMGT.2017.05.015>
- Marshall, M. I., & Schrank, H. L. (2013). *Small business disaster recovery: a research framework*. <https://doi.org/10.1007/s11069-013-1025-z>
- Menouer, T. (2020). KCSS: Kubernetes container scheduling strategy. *The Journal of Supercomputing* 2020 77:5, 77(5), 4267–4293.
<https://doi.org/10.1007/S11227-020-03427-3>
- Miller, A., & Ciuffo, F. (2021, December 1). *Backup and restore your Amazon EKS cluster resources using Velero | Containers*.
<https://aws.amazon.com/blogs/containers/backup-and-restore-your-amazon-eks-cluster-resources-using-velero/>
- Minh Bui, D. (2020). *Implementing cluster backup solution to build resilient cloud architecture 2020 Laurea*.
- Oktadini, N. R., & Surendro, K. (2014). SLA in cloud computing: Improving SLA's life cycle applying six sigma. *2014 International Conference on Information Technology Systems and Innovation, ICITSI 2014 - Proceedings*. <https://doi.org/10.1109/ICITSI.2014.7048278>
- Pereira Ferreira, A., & Sinnott, R. (2019). A performance evaluation of containers running on managed kubernetes services. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom, 2019-Decem*, 199–208.
<https://doi.org/10.1109/CLOUDCOM.2019.00038>
- Poniszewska-Marańda, A., & Czechowska, E. (2021). Kubernetes Cluster for Automating Software Production Environment. *Sensors 2021, Vol. 21*,

- Page 1910, 21(5), 1910. <https://doi.org/10.3390/S21051910>
- Prazeres, A., & Lopes, E. (2013). Disaster Recovery – A Project Planning Case Study in Portugal. *Procedia Technology*, 9, 795–805. <https://doi.org/10.1016/J.PROTCY.2013.12.088>
- Quarantelli, E. L. (1999). *The Disaster Recovery Process: What We Know And Do Not Know From Research*. <https://udspace.udel.edu/handle/19716/309>
- Regional clusters | Kubernetes Engine Documentation | Google Cloud*. (n.d.). Retrieved March 24, 2022, from <https://cloud.google.com/kubernetes-engine/docs/concepts/regional-clusters>
- Robinson, G., Narin, A., & Elleman, C. (2014). *Amazon Web Services-Using AWS for Disaster Recovery Using Amazon Web Services for Disaster Recovery*.
- roboll/helmfile: Deploy Kubernetes Helm Charts*. (n.d.). Retrieved February 13, 2022, from <https://github.com/roboll/helmfile>
- Rudolph, C. G. (1990). Business Continuation Planning/Disaster Recovery: A Marketing Perspective. *IEEE Communications Magazine*, 28(6), 25–28. <https://doi.org/10.1109/35.56224>
- Run a Replicated Stateful Application | Kubernetes*. (n.d.). Retrieved March 24, 2022, from <https://kubernetes.io/docs/tasks/run-application/run-replicated-stateful-application/>
- Sahi, A., Lai, D., & Li, Y. (2016). Security and privacy preserving approaches in the eHealth clouds with disaster recovery plan. *Computers in Biology and Medicine*, 78, 1–8. <https://doi.org/10.1016/J.COMPBIOMED.2016.09.003>
- Schrettenbrunner, J. (2020). *Migrating Pods in Kubernetes*. https://www.researchgate.net/publication/349662156_Migrating_Pods_in_Kubernetes
- Suguna, S., & Suhasini, A. (2015). Overview of data backup and disaster recovery in cloud. *2014 International Conference on Information Communication and Embedded Systems, ICICES 2014*. <https://doi.org/10.1109/ICICES.2014.7033804>
- Symantec 2010 Disaster Recovery Study*. (2010).
- Takabi, H., Joshi, J. B. D., & Ahn, G. J. (2010). Security and privacy challenges in cloud computing environments. *IEEE Security and Privacy*, 8(6), 24–31. <https://doi.org/10.1109/MSP.2010.186>
- Tamimi, A. A., Dawood, R., & Sadaqa, L. (2019). Disaster recovery techniques in cloud computing. *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology, JEEIT 2019 - Proceedings*, 845–850. <https://doi.org/10.1109/JEEIT.2019.8717450>
- Tomlinson, B. (n.d.). *darthlukan/rhacm-mirror: An OpenShift/Kubernetes operator which orchestrates automatic failover of one RHACM hub to another*. Retrieved March 16, 2022, from <https://github.com/darthlukan/rhacm-mirror>
- Velero*. (n.d.). Retrieved February 12, 2022, from <https://velero.io/>
- Velero Plugins*. (n.d.). Retrieved March 17, 2022, from <https://velero.io/plugins/>
- What is Terraform? | IBM*. (n.d.). Retrieved February 13, 2022, from

- <https://www.ibm.com/cloud/learn/terraform>
- Wiboonrat, M. (2008a). An empirical IT contingency planning model for disaster recovery strategy selection. *IEMC-Europe 2008 - 2008 IEEE International Engineering Management Conference, Europe: Managing Engineering, Technology and Innovation for Growth*.
<https://doi.org/10.1109/IEMCE.2008.4617953>
- Wiboonrat, M. (2008b). An empirical IT contingency planning model for disaster recovery strategy selection. *IEMC-Europe 2008 - 2008 IEEE International Engineering Management Conference, Europe: Managing Engineering, Technology and Innovation for Growth*.
<https://doi.org/10.1109/IEMCE.2008.4617953>
- Wood, T., Cecchet, E., Ramakrishnan, K. K., Shenoy, P., Merwe, J. van der, & Venkataramani, A. (2010). Disaster recovery as a cloud service: economic benefits & deployment challenges. *HotCloud'10: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 8.
- Xu, C., Rajamani, K., & Felter, W. (2018). NBWGuard: Realizing network QoS for kubernetes. *Middleware Industry 2018 - Proceedings of the 2018 ACM/IFIP/USENIX Middleware Conference (Industrial Track)*, 32–38.
<https://doi.org/10.1145/3284028.3284033>
- Zhang, X., & McMurray, A. J. (2012). Embedding Business Continuity and Disaster Recovery within Risk Management. *SSRN Electronic Journal*.
<https://doi.org/10.2139/ssrn.2174785>

Appendix 1

Project proposal

- **Brief description of the research area – background**

Kubernetes (*Kubernetes*, n.d.) is a well-known open-source container-orchestration system widely used in industrial and academic fields (Menouer, 2020). This software tool has become widely used for container orchestration because it simplifies deployment of containerized applications across a cluster of machines in a cloud provider-agnostic way (Larsson et al., 2020). It also has become the de facto standard because of its robustness and maturity (Pereira Ferreira & Sinnott, 2019).

Due to these facts, and according to Gartner estimations, it is foreseen that by 2025, more than 50% of enterprises will adopt a centralized platform engineering and operations approach to facilitate DevOps self-service and scaling, which is a significant increase from less than 20% in 2020. Another prediction is that by 2025, more than 85% of global organizations will be running containerized applications in production, which is a significant increase from fewer than 35% in 2019 (Chandrasekaran, 2020b).

There has been a special focus in literature on improving different areas of Kubernetes such as scaling of resources (Balla et al., 2020), networking (Xu et al., 2018) and many more. Still, considering the aforementioned estimations, we can argue there is also a need on improving the bootstrapping and provisioning process for spinning up new Kubernetes clusters, and especially in the case of disaster recovery in Kubernetes cloud environments.

Disaster recovery can be understood as the process which an organization has to undergo after a service disruption happened in order to resume normal services. The set of predefined procedures is known as Disaster Recovery Plan, DRP. Disaster recovery is an essential requirement of any business where continuity matters. (Poniszewska-Marańda & Czechowska, 2021))

Security and monitoring are anti-disaster layers for Kubernetes clusters, but even with the application of these concepts, the application still faces many risks, such as high availability (Lamouchi, 2021). This is the reason why Disaster Recovery is critical for guaranteeing the agreed Service-Level Agreements (SLAs) and Service-Level Objectives (SLOs).

The aim of the project is to research, explore and evaluate the disaster recovery solutions available for Kubernetes running in the cloud, as a managed service. This will involve measuring the RTO (Recovery Time Objective) and the RPO (Recovery Point Objective) as our objective metrics (Baginda et al., 2018).

The target will be set on identification and testing of current technologies, benchmarking different solutions in the cloud, and discussing about the benefits and shortcomings of the identified solutions.

- **Project outline for the work that you propose to complete**

The idea for this research arose from:

My supervisor Paul Lapok, which suggested me to focus on a research problem related to my current job.

The aims of the project are as follows:

The aim of the project is to research, explore and evaluate the disaster recovery solutions available for Kubernetes running in the cloud, as a managed service, by means of two main metrics: RTO (Recovery Time Objective) and RPO (Recovery Point Objective)

The focus will be on identification and testing of current technologies, benchmarking different solutions in the cloud, and discussing about the benefits and shortcomings of the solutions.

The main research questions that this work will address include:

How to decrease the RTO and RPO of a fully production Kubernetes cluster after a disaster in the cloud? Where are the limits and consequences on employing the identified technologies?

The software development/design work/other deliverable of the project will be:

The project will deliver a comparison based on RTO and RPO in the different clouds, and suggestions on the design of a disaster recovery approach.

The project will involve the following research/field work/experimentation/ evaluation:

The first part will be an evaluation of the state of the art linked to Disaster Recovery. Different approaches to disaster recovery will be studied and evaluated, developing a working DRP in Kubernetes in the cloud.

This work will require the use of specialist software:

GKE (Google Kubernetes Engine), EKS (Amazon Elastic Kubernetes Service), and other open-source software tools.

This work will require the use of specialist hardware:

Cloud resources.

The project is being undertaken in collaboration with:

Cloud resources provided by Claranet Deutschland GmbH.

- **References**

Al-Kiswany, S., Subhraveti, D., Sarkar, P., & Ripeanu, M. (2011). VMFlock: Virtual machine co-migration for the cloud. *Proceedings of the IEEE International Symposium on High Performance Distributed Computing*, 159–170.
<https://doi.org/10.1145/1996130.1996153>

Alhazmi, O. H., & Malaiya, Y. K. (2012). Assessing disaster recovery alternatives: On-site, colocation or cloud. *Proceedings - 23rd IEEE International Symposium on*

- Software Reliability Engineering Workshops, ISSREW 2012*, 19–20.
<https://doi.org/10.1109/ISSREW.2012.20>
- Alhazmi, O. H., & Malaiya, Y. K. (2013). Evaluating disaster recovery plans using the cloud. *Proceedings - Annual Reliability and Maintainability Symposium*.
<https://doi.org/10.1109/RAMS.2013.6517700>
- Alvarez-Parmar, R. (2020, December 7). *Operating a multi-regional stateless application using Amazon EKS | Containers*.
<https://aws.amazon.com/blogs/containers/operating-a-multi-regional-stateless-application-using-amazon-eks/>
- Amarnath, A. (2020, September 16). *Velero 1.5: Auto volume backup with restic, DeletionAction plugins, Restore Hooks, and much more!*
<https://velero.io/blog/velero-1.5-for-and-by-community/>
- Amazon EKS Service Level Agreement*. (2020). <https://aws.amazon.com/eks/sla/>
- Anand, G., & Kodali, R. (2008). Benchmarking the benchmarking models. *Benchmarking*, 15(3), 257–291.
<https://doi.org/10.1108/14635770810876593/FULL/PDF>
- Andrade, E., Nogueira, B., Matos, R., Callou, G., & Maciel, P. (2017). Availability modeling and analysis of a disaster-recovery-as-a-service solution. *Computing*, 99(10), 929–954. <https://doi.org/10.1007/S00607-017-0539-8/FIGURES/9>
- As Quarterly Cloud Spending Jumps to Over \$50B, Microsoft Looms Larger in Amazon's Rear Mirror | Synergy Research Group*. (2022, February 3).
<https://www.srgresearch.com/articles/as-quarterly-cloud-spending-jumps-to-over-50b-microsoft-looms-larger-in-amazons-rear-mirror>
- Backup and Restore with Velero | Administration Guide | SUSE CaaS Platform 4.2.4*. (n.d.). Retrieved April 2, 2022, from <https://documentation.suse.com/suse-caasp/4.2/html/caasp-admin/id-backup-and-restore-with-velero.html>
- Backup for GKE | Google Cloud*. (n.d.). Retrieved March 16, 2022, from <https://cloud.google.com/kubernetes-engine/docs/add-on/backup-for-gke/concepts/backup-for-gke>
- Baginda, Y. P., Affandi, A., & Pratomo, I. (2018). Analysis of RTO and RPO of a service stored on Amazon Web Service (AWS) and Google Cloud Engine (GCE). *Proceedings of 2018 10th International Conference on Information Technology and Electrical Engineering: Smart Technology for Better Society, ICITEE 2018*, 418–422. <https://doi.org/10.1109/ICITEED.2018.8534758>
- Bakshi, D., & Kim, B. (2021, November 12). *Disaster Recovery with AWS Managed Services, Part I: Single Region | AWS Architecture Blog*.
<https://aws.amazon.com/es/blogs/architecture/disaster-recovery-with-aws-managed-services-part-i-single-region/>
- Balla, D., Simon, C., & Maliosz, M. (2020). Adaptive scaling of Kubernetes pods. *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2020: Management in the Age of Softwarization and Artificial Intelligence, NOMS 2020*. <https://doi.org/10.1109/NOMS47738.2020.9110428>
- Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, omega, and kubernetes. *Communications of the ACM*, 59(5), 50–57.
<https://doi.org/10.1145/2890784>
- Burns, B., & Tracey, C. (2018). Managing Kubernetes: Operating Kubernetes Clusters in the Real World. In *Managing Kubernetes: Operating Kubernetes Clusters in*

the Real World.

- Cegieta, R. (2006). Selecting technology for disaster recovery. *Proceedings of International Conference on Dependability of Computer Systems, DepCoS-RELCOMEX 2006*, 160–167. <https://doi.org/10.1109/DEPCOS-RELCOMEX.2006.49>
- Chandrasekaran, A. (2020a). *Best Practices for Running Containers and Kubernetes in Production.*
- Chandrasekaran, A. (2020b). Best Practices for Running Containers and Kubernetes in Production. *Gartner, August*, 1–14.
<https://www.gartner.com/document/3902966>
- Costello, K., & Rimol, M. (2021, April 21). *Gartner Forecasts Worldwide Public Cloud End-User Spending to Grow 23% in 2021.*
<https://www.gartner.com/en/newsroom/press-releases/2021-04-21-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-23-percent-in-2021>
- de Souza Couto, R., Secci, S., Elias Mitre Campista, M., & Henrique Maciel Kosmowski Costa, L. (2014). *Network Design Requirements for Disaster Resilience in IaaS Clouds.* <http://aws.amazon.com/ec2-sla>.
- Disaster recovery planning guide | Cloud Architecture Center | Google Cloud.* (n.d.). Retrieved January 29, 2022, from
<https://cloud.google.com/architecture/dr-scenarios-planning-guide>
- Eliot, S. (2021, May 14). *Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby | AWS Architecture Blog.* AWS Architecture Blog.
<https://aws.amazon.com/blogs/architecture/disaster-recovery-dr-architecture-on-aws-part-iii-pilot-light-and-warm-standby/>
- Example: Deploying WordPress and MySQL with Persistent Volumes | Kubernetes.* (n.d.). Retrieved March 10, 2022, from
<https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/>
- Fallara, P. (2004). Disaster Recovery Planning. *IEEE Potentials*, 23(5), 42–44.
<https://doi.org/10.1109/MP.2004.1301248>
- geom_density function - RDocumentation.* (n.d.). Retrieved March 15, 2022, from
https://www.rdocumentation.org/packages/ggplot2/versions/3.3.5/topics/geom_density
- GitHub - ahmetb/kubectx: Faster way to switch between clusters and namespaces in kubectl.* (n.d.). Retrieved March 14, 2022, from
<https://github.com/ahmetb/kubectx>
- Google Kubernetes Engine SLA | Google Cloud.* (2021).
<https://cloud.google.com/kubernetes-engine/sla>
- Gunawan, A. S., Allen, M., Alhazmi, O. H., & Malaiya, Y. K. (2013). *Disaster Recovery Plans Cloud Evaluating Disaster Recovery Plans Using the Cloud.*
- Israel, G. D. (1992). Determining Sample Size 1 The Level of Precision. *University of Florida.*
- Kirvan, P. (2022, March 9). *Cloud-era disaster recovery planning: Maintenance and continuous improvement.* <https://www.computerweekly.com/feature/Cloud-era-disaster-recovery-planning-Maintenance-and-continuous-improvement>
- kubefwd - Kubernetes Service Forwarding.* (n.d.). Retrieved March 14, 2022, from

- <https://kubefwd.com/>
kubernetes-sigs/kustomize: Customization of kubernetes YAML configurations. (n.d.). Retrieved February 23, 2022, from <https://github.com/kubernetes-sigs/kustomize>
- Kubernetes.* (n.d.). Retrieved October 30, 2021, from <https://kubernetes.io/>
- Lam, W. (2002). Ensuring business continuity. *IT Professional*, 4(3), 19–25.
<https://doi.org/10.1109/MITP.2002.1008533>
- Lamouchi, N. (2021). Adding Anti-Disaster Layers. *Pro Java Microservices with Quarkus and Kubernetes*, 187–251. https://doi.org/10.1007/978-1-4842-7170-4_6
- Larsson, L., Tärneberg, W., Klein, C., Elmroth, E., & Kihl, M. (2020). Impact of etcd deployment on Kubernetes, Istio, and application performance. *Software: Practice and Experience*, 50(10), 1986–2007. <https://doi.org/10.1002/SPE.2885>
- Lozupone, V. (2017). Disaster recovery plan for medical records company. *International Journal of Information Management*, 37(6), 622–626.
<https://doi.org/10.1016/J.IJINFOMGT.2017.05.015>
- Marshall, M. I., & Schrank, H. L. (2013). *Small business disaster recovery: a research framework*. <https://doi.org/10.1007/s11069-013-1025-z>
- Menouer, T. (2020). KCSS: Kubernetes container scheduling strategy. *The Journal of Supercomputing* 2020 77:5, 77(5), 4267–4293. <https://doi.org/10.1007/S11227-020-03427-3>
- Miller, A., & Ciuffo, F. (2021, December 1). *Backup and restore your Amazon EKS cluster resources using Velero | Containers*.
<https://aws.amazon.com/blogs/containers/backup-and-restore-your-amazon-eks-cluster-resources-using-velero/>
- Minh Bui, D. (2020). *Implementing cluster backup solution to build resilient cloud architecture 2020 Laurea*.
- Oktadini, N. R., & Surendro, K. (2014). SLA in cloud computing: Improving SLA's life cycle applying six sigma. *2014 International Conference on Information Technology Systems and Innovation, ICITSI 2014 - Proceedings*.
<https://doi.org/10.1109/ICITSI.2014.7048278>
- Pereira Ferreira, A., & Sinnott, R. (2019). A performance evaluation of containers running on managed kubernetes services. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom, 2019-Decem*, 199–208. <https://doi.org/10.1109/CLOUDCOM.2019.00038>
- Poniszewska-Marańda, A., & Czechowska, E. (2021). Kubernetes Cluster for Automating Software Production Environment. *Sensors* 2021, Vol. 21, Page 1910, 21(5), 1910. <https://doi.org/10.3390/S21051910>
- Prazeres, A., & Lopes, E. (2013). Disaster Recovery – A Project Planning Case Study in Portugal. *Procedia Technology*, 9, 795–805.
<https://doi.org/10.1016/J.PROTCY.2013.12.088>
- Quarantelli, E. L. (1999). *The Disaster Recovery Process: What We Know And Do Not Know From Research*. <https://udspace.udel.edu/handle/19716/309>
- Regional clusters | Kubernetes Engine Documentation | Google Cloud.* (n.d.). Retrieved March 24, 2022, from <https://cloud.google.com/kubernetes-engine/docs/concepts/regional-clusters>
- Robinson, G., Narin, A., & Elleman, C. (2014). *Amazon Web Services-Using AWS for*

- Disaster Recovery Using Amazon Web Services for Disaster Recovery.*
 roboll/helmfile: Deploy Kubernetes Helm Charts. (n.d.). Retrieved February 13, 2022, from <https://github.com/roboll/helmfile>
- Rudolph, C. G. (1990). Business Continuation Planning/Disaster Recovery: A Marketing Perspective. *IEEE Communications Magazine*, 28(6), 25–28.
<https://doi.org/10.1109/35.56224>
- Run a Replicated Stateful Application | Kubernetes.* (n.d.). Retrieved March 24, 2022, from <https://kubernetes.io/docs/tasks/run-application/run-replicated-stateful-application/>
- Sahi, A., Lai, D., & Li, Y. (2016). Security and privacy preserving approaches in the eHealth clouds with disaster recovery plan. *Computers in Biology and Medicine*, 78, 1–8. <https://doi.org/10.1016/J.COMPBIOMED.2016.09.003>
- Schrettenbrunner, J. (2020). *Migrating Pods in Kubernetes.*
https://www.researchgate.net/publication/349662156_Migrating_Pods_in_Kubernetes
- Suguna, S., & Suhasini, A. (2015). Overview of data backup and disaster recovery in cloud. *2014 International Conference on Information Communication and Embedded Systems, ICICES 2014.* <https://doi.org/10.1109/ICICES.2014.7033804>
- Symantec 2010 Disaster Recovery Study.* (2010).
- Takabi, H., Joshi, J. B. D., & Ahn, G. J. (2010). Security and privacy challenges in cloud computing environments. *IEEE Security and Privacy*, 8(6), 24–31.
<https://doi.org/10.1109/MSP.2010.186>
- Tamimi, A. A., Dawood, R., & Sadaqa, L. (2019). Disaster recovery techniques in cloud computing. *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology, JEEIT 2019 - Proceedings*, 845–850.
<https://doi.org/10.1109/JEEIT.2019.8717450>
- Tomlinson, B. (n.d.). *darthlukan/rhacm-mirror: An OpenShift/Kubernetes operator which orchestrates automatic failover of one RHACM hub to another.* Retrieved March 16, 2022, from <https://github.com/darthlukan/rhacm-mirror>
- Velero.* (n.d.). Retrieved February 12, 2022, from <https://velero.io/>
- Velero Plugins.* (n.d.). Retrieved March 17, 2022, from <https://velero.io/plugins/>
- What is Terraform? | IBM.* (n.d.). Retrieved February 13, 2022, from <https://www.ibm.com/cloud/learn/terraform>
- Wiboonrat, M. (2008a). An empirical IT contingency planning model for disaster recovery strategy selection. *IEMC-Europe 2008 - 2008 IEEE International Engineering Management Conference, Europe: Managing Engineering, Technology and Innovation for Growth.*
<https://doi.org/10.1109/IEMCE.2008.4617953>
- Wiboonrat, M. (2008b). An empirical IT contingency planning model for disaster recovery strategy selection. *IEMC-Europe 2008 - 2008 IEEE International Engineering Management Conference, Europe: Managing Engineering, Technology and Innovation for Growth.*
<https://doi.org/10.1109/IEMCE.2008.4617953>
- Wood, T., Cecchet, E., Ramakrishnan, K. K., Shenoy, P., Merwe, J. van der, & Venkataramani, A. (2010). Disaster recovery as a cloud service: economic benefits & deployment challenges. *HotCloud'10: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 8.

- Xu, C., Rajamani, K., & Felter, W. (2018). NBWGuard: Realizing network QoS for kubernetes. *Middleware Industry 2018 - Proceedings of the 2018 ACM/IFIP/USENIX Middleware Conference (Industrial Track)*, 32–38.
<https://doi.org/10.1145/3284028.3284033>
- Zhang, X., & McMurray, A. J. (2012). Embedding Business Continuity and Disaster Recovery within Risk Management. *SSRN Electronic Journal*.
<https://doi.org/10.2139/ssrn.2174785>

Appendix 2

Project Planning

- Project plan
- Project Diary

Meeting with Paul Lapok on Thu 14/10/21

Objectives:

TASK	% DONE	(EXPECTED) START DATE	(EXPECTED) END DATE
Literature Review	100%	End November	Early December
Methodology	<u>100%</u>	Early December	End December
Initial Report	100%	End November	End December
Cloud approaches research	100%	Early January	Mid January
Testing and Implementation	100%	End January	End February
Write-up	100%	Early March	End March
Dissertation	100%	Early April	8 April

Select a topic for the research project and finish the proposal. The idea is to build an R Shiny web application from several sources of data of the National Institute of Statistics in Spain, perhaps implementing some dynamic visualizations, and perhaps applying Machine Learning to predict future population.

Supervisor's Comments:

Mindset completely changed about the project idea, as this is not really a topic for Research per se. This would involve asking potential users about functionality and applying just what the people wanted. This has lots of risks, and it would be wiser to change topic. He suggested me to focus just on one

topic that can be researchable, not going through the whole project of Market research + R Shiny development + Shiny deployment. Perhaps a good topic is exploring how Shiny and DevOps glue together, as perhaps it could be interesting for companies.

Meeting with Paul Lapok on Fri 19/11/21

Objectives:

Develop the ideas needed for a successful Initial Report

I need guide on how to conduct it. Define more than just the initial research proposal topic.

Request more MSc dissertations from the computing area to check how it is really done.

Progress:

Felt a little overwhelmed with the many possible research topics, and selecting one adequate to me, where I could feel thrilled.

Tried to test some research topic regarding minikube and some disaster recovery plan. I talked with my company manager to select a topic and we could think of some interesting for the company, around disaster recovery.

Supervisor's Comments:

Maybe focus on just this, Disaster Recovery, with a given disaster. For example, find in the literature that 80% of companies suffered one type of disaster. Focus on what are the risks. Also focus on the mitigation steps and how to mitigate prematurely the risks. Could a user interface be useful? Start by studying the background area, literature, and understand the problems that the area is facing.

Meeting with Paul Lapok on Thu 27/01/22

Objectives:

Ask Paul:

- How many words does the dissertation have to be?
- Any suggestion on how to write the methodology section?
- Should I make the results public before the end of the dissertation?

- How to manage the fact that currently there is single AZ cluster... some of these problems would go away with the regional clusters.

Tell Paul that I ditched Azure because of different problems regarding budget and complexity for the access.

Progress:

Initial report finished and sent; I wrote a little bit of the introduction for the Dissertation and started with the chapter 3.

Supervisor's Comments:

Regarding the questions:

- About length of the dissertation: 10K or 20K. Not exact number. The real importance is about reasoning, not length.
- In the methodology part I should propose my experiment, design it. Design the methods by which the research is succeeded. Design the scenarios, how I implemented it, screenshots and diagrams
- Publishing the results in GitHub is a bad idea. Better after presenting the Dissertation.
- The problem of multi-AZ vs single-AZ in disaster recovery, obviously multi-AZ is better, but I have to focus on the steps after and how to improve it, whatever the setting.

Paul gave me more suggestions regarding:

- Comparing values in research papers for RTO and RPO
- Do things similarly as in other papers to know the number of tests, to explain why 10 or 100 samples.
- Maybe investigate with different workload, and different cloud settings.
- Focus on literature more, rather on web pages or other sources. Compare everything to papers, to see changes in performance
- Only focus, for example, on 2 scenarios, and explain why.
- Results if bell charts, research on how to characterise results.

Meeting with Paul Lapok on Thu 10/02/22

Objectives:

Bring up that after so much research, there is no related paper calculating actual timings

Ask if I need some kind of approval for mentioning Claranet in the dissertation.

Progress:

The chapter 3 about methodology was completely finished and reviewed by Paul

Supervisor's Comments:

Try to describe the process of setting up the cluster in both chapters, 3 and 4, as I have not explained it.

Using statistical methods to calculate sample size, given the fact that there is no paper mentioning such a benchmark. And therefore we can argue why have we chosen such number of experiments.

One via could be C-SCORE [https://www.geopoll.com/blog/sample-size-research/](https://www.geopoll.com/blog/sample-size-research/population-size) population size

Regarding mentioning Claranet in the dissertation: I would just need verbal justification from the company.

Focus on the conclusion and ask these questions: What can you improve? What are the main findings?

Meeting with Paul Lapok on Thu 24/02/22

Objectives:

Bring up and talk about the problem of the equations for calculating sample size, my standard deviation is higher than 0.5, which makes the function negative

Ask Paul into how to argue that we don't have that many money to make more tests.

Where should the scripts be mentioned? In the main line or as an Appendix?

How to tackle chapter 5?

Progress:

Finished chapter 3 and addressed all the comments from Paul. I'm halfway through chapter 4 now, describing the cluster deployment

Supervisor's Comments:

Regarding sample size: Increase scenario 2 tests at least with a few more samples. Let's try to use another formula to calculate the sample size.

If the scripts are long, better to put that text in the appendix, otherwise explain in chapter 4.

Include please pictures in chapter 4.

Evaluation in chapter 5: Compare to other Disaster Recovery solutions that I found.

Meeting with Paul Lapok on Thu 10/03/22

Objectives:

Still having issues with the calculations of the sample size. Is normalizing okay?

When should I send the dissertation? When is it really going to be the VIVA?

Progress:

Almost with chapter 4 finished, and writing Appendix 3.

Supervisor's Comments:

Explain the sample size calculations in evaluation. Bring it up in methodologies too and explain that it is not feasible to do this number of tests.

Use median instead of mean in the charts

Perform t-tests in the scenarios presented, comparing AWS and GCP, to know if there are significant differences in both collections

Regarding the sample size: Test using minutes or hours instead of seconds, because it should not change.

Appendix 3

Terraform files and its explanation

Two main directories have been developed for both environments (AWS and GCP).

Inside each directory, there are likewise two other directories. The first one takes care about setting up the Kubernetes cluster, and the second one consists of auxiliary services.

- **EKS files**

Beginning with the Kubernetes cluster in AWS, known as EKS, the following files have been developed:

The following file is the “GENERAL.tf” file, which configures basic information like the AWS region to be used, and where the Terraform state is to be saved.

```
provider "aws" {
  region = var.region
}

provider "aws" {
  alias = "paris"
  region = "eu-west-3"
}

terraform {
  backend "local" {
    path = "terraform.tfstate"
  }
}
```

The previous file is accompanied by this second file, called “GENERAL.var.tf”, which sets the variables which will be used throughout the entire project like the region, availability zone and account ID.

```
variable "project_account_id" {
  default = "REMOVED DUE TO PRIVACY CONCERNS"
}

variable "region" {
  default = "eu-west-3"
}

variable "availability_zone" {
  default = {
```

```

    "0" = "eu-west-3a"
    # "1" = "eu-west-3b"
    # "2" = "eu-west-3c"
  }
}

```

The following file is the “VPC.tf” file, which creates the Virtual Private Cloud (VPC). This will hold the network connectivity of the cluster. It configures the public and private subnetworks and makes possible through a NAT and an Internet Gateway the access from external sources. It also sets routing tables, including the endpoint needed for the S3 bucket.

```

resource "aws_vpc" "default" {
  cidr_block      = var.vpc_cidr
  enable_dns_hostnames = true
  enable_dns_support = true
  tags = {
    Name = var.vpc_name
  }
}

### SUBNETS
resource "aws_subnet" "public" {
  for_each = var.public_cidr
  vpc_id    = aws_vpc.default.id
  cidr_block = each.value.cidr_block
  map_public_ip_on_launch = "true"
  availability_zone_id     = each.value.availability_zone_id
  tags = {
    Name = var.subnet_public
  }
}

resource "aws_subnet" "private" {
  for_each = var.private_cidr
  vpc_id    = aws_vpc.default.id
  cidr_block = each.value.cidr_block
  map_public_ip_on_launch = "false"
  availability_zone_id     = each.value.availability_zone_id
  tags = {
    Name = var.subnet_private
  }
}

### For NAT-GW
resource "aws_eip" "nat" {
  for_each = var.private_cidr
  vpc      = true
}

### NAT GATEWAY
resource "aws_nat_gateway" "default" {
  for_each = var.private_cidr
  allocation_id = aws_eip.nat[each.key].id
  subnet_id     = aws_subnet.public[each.key].id
  depends_on = [
    aws_eip.nat,
    aws_subnet.public,
  ]
}

### INTERNET GATEWAY
resource "aws_internet_gateway" "default" {
  vpc_id = aws_vpc.default.id
}

```

```

### ROUTE TABLES
resource "aws_route_table" "public" {
  vpc_id = aws_vpc.default.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.default.id
  }
  tags = {
    Name = "Public"
  }
}

resource "aws_route_table" "private" {
  for_each = aws_nat_gateway.default
  vpc_id = aws_vpc.default.id

  route {
    cidr_block      = "0.0.0.0/0"
    nat_gateway_id = each.value.id
  }
  tags = {
    Name = "Private"
  }
}

resource "aws_route_table_association" "public" {
  for_each = aws_subnet.public
  subnet_id      = each.value.id
  route_table_id = aws_route_table.public.id
}

resource "aws_route_table_association" "private" {
  for_each = aws_subnet.private
  subnet_id      = each.value.id
  route_table_id = aws_route_table.private[each.value.availability_zone].id
}

resource "aws_vpc_endpoint" "s3" {
  vpc_id      = aws_vpc.default.id
  service_name = "com.amazonaws.${var.region}.s3"
}

resource "aws_vpc_endpoint_route_table_association" "s3-private" {
  for_each = aws_subnet.private
  route_table_id = aws_route_table.private[each.value.availability_zone].id
  vpc_endpoint_id = aws_vpc_endpoint.s3.id
}

```

Along the previous file, this “VPC.var.tf” is the file which holds the values used in the VPC.tf file. It sets values like naming, IP ranges, and the zone to be used.

```

#### VPC
variable "vpc_name" {
  default = "sergio-test"
}
variable "vpc_cidr" {
  default = "172.20.0.0/21"
}

#### SUBNETS
variable "subnet_public" {
  default = "public"
}
variable "public_cidr" {
  default = {
    "eu-west-3a" = {
      availability_zone = "eu-west-3a"
      cidr_block        = "172.20.4.0/24"
      availability_zone_id = "euw3-az1"
    }
  }
}

```



```

}

variable "subnet_private" {
  default = "private"
}

variable "private_cidr" {
  default = {
    "eu-west-3a" = {
      availability_zone = "eu-west-3a"
      cidr_block        = "172.20.6.0/24"
      availability_zone_id = "euw3-az1"
    }
  }
}

```

Then, the next file is `IAM.tf`, which will configure resources related to principals and its permissions. Firstly, the role for the master node is created and it is assigned the required permissions to run the cluster. The role for the worker node is created as well, with more restricted permissions. Then, the role for Velero is created, and the OIDC identity provider is set up for it to access S3 through the service account present in the cluster.

```

# cluster control plane
resource "aws_iam_role" "eks-cluster-role" {
  name = "eks-cluster-role"

  assume_role_policy = <<POLICY
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
POLICY
}

resource "aws_iam_role_policy_attachment" "eks-cluster" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
  role = aws_iam_role.eks-cluster-role.name
}

resource "aws_iam_role_policy_attachment" "eks-cluster_vpc" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController"
  role = aws_iam_role.eks-cluster-role.name
}

# worker nodes
resource "aws_iam_role" "eks-worker-role" {
  name = "eks-worker-role"

  assume_role_policy = <<POLICY
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}

```

```

    ]
}
POLICY
}

resource "aws_iam_role_policy_attachment" "eks-worker" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
  role = aws_iam_role.eks-worker-role.name
}

resource "aws_iam_role_policy_attachment" "eks-worker_cni" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
  role = aws_iam_role.eks-worker-role.name
}

resource "aws_iam_role_policy_attachment" "eks-worker_registry" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
  role = aws_iam_role.eks-worker-role.name
}

resource "aws_iam_role_policy_attachment" "eks-worker_cloudwatchlogs" {
  policy_arn = "arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy"
  role = aws_iam_role.eks-worker-role.name
}

# ----- Velero ----- #
resource "aws_iam_policy" "velero-policy" {
  name = "velero-policy"
  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeVolumes",
        "ec2:DescribeSnapshots",
        "ec2:CreateTags",
        "ec2:CreateVolume",
        "ec2:CreateSnapshot",
        "ec2>DeleteSnapshot"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3>DeleteObject",
        "s3:PutObject",
        "s3:AbortMultipartUpload",
        "s3:ListMultipartUploadParts"
      ],
      "Resource": "arn:aws:s3::sergio-tests-velero-backups/*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3::sergio-tests-velero-backups"
    }
  ]
}
EOF
}

resource "aws_iam_role" "velero-role" {
  name = "velero-role"
  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "${aws_iam_openid_connect_provider.sergio-test.arn}"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {

```

```

        "StringEquals": {
            "${trimprefix(aws_eks_cluster.default.identity.0.oidc.0.issuer,
"https://")}:sub": "system:serviceaccount:velero:velero"
        }
    }
}
]
}
EOF
}

resource "aws_iam_role_policy_attachment" "velero" {
    role      = aws_iam_role.velero-role.name
    policy_arn = aws_iam_policy.velero-policy.arn
}

```

Next, the main file of the EKS cluster is defined, which is called EKS.tf. The EKS master node is first of all defined, associated to the VPC, and assigned the role defined earlier. Then, the worker node configuration is set, and finally, the OIDC Identity Provider is referenced, and associated.

```

### EKS Cluster ###
# Default
resource "aws_eks_cluster" "default" {
    version = "1.21"
    name = "sergio-test"
    role_arn = aws_iam_role.eks-cluster-role.arn
    enabled_cluster_log_types = ["api", "audit", "authenticator", "controllerManager",
"scheduler"]
    vpc_config {
        endpoint_public_access = true
        endpoint_private_access = true
        subnet_ids = [aws_subnet.private["eu-west-3a"].id, aws_subnet.private["eu-west-
3b"].id]
        public_access_cidrs = ["0.0.0.0/0", "212.82.224.202/32", "212.82.224.205/32"]
    }
    kubernetes_network_config {
        service_ipv4_cidr = "10.160.85.0/24"
    }
    depends_on = [
        aws_iam_role_policy_attachment.eks-cluster,
    ]
}

output "endpoint" {
    value = aws_eks_cluster.default.endpoint
}

### EKS Node Group (Workers) ###
resource "aws_eks_node_group" "private" {
    cluster_name = aws_eks_cluster.default.name
    node_role_arn = aws_iam_role.eks-worker-role.arn
    subnet_ids = [aws_subnet.private["eu-west-3a"].id]
    node_group_name = "sergio-private-group"
    instance_types = ["c6i.large"]
    capacity_type = "SPOT"
    disk_size = 20
    scaling_config {
        desired_size = 1
        max_size = 1
        min_size = 0
    }
    version = "1.21"
}

### EKS Identity Provider ###
# EKS cluster OpenID identity provider association
resource "aws_eks_identity_provider_config" "sergio-test" {
    cluster_name = aws_eks_cluster.default.name
    oidc {
        client_id = "sts.amazonaws.com"
        identity_provider_config_name = "sergio-test"
    }
}

```

```

    issuer_url = aws_eks_cluster.default.identity.0.oidc.0.issuer
  }
}

# OpenID Provider
resource "aws_iam_openid_connect_provider" "sergio-test" {
  client_id_list = ["sts.amazonaws.com"]
  thumbprint_list = ["9e99a48a9960b14926bb7f3b02e22da2b0ab7280"]
#CAB073498D7558FEC3B2C414C006ACBA30805431
  url = aws_eks_cluster.default.identity.0.oidc.0.issuer
}

```

Finally, in another folder, the S3 bucket which is used by Velero is configured with a default configuration. The file is called “S3.tf”

```

resource "aws_s3_bucket" "velero_tests_backups" {
  bucket = "sergio-tests-velero-backups"
  versioning {
    enabled = false
  }
  server_side_encryption_configuration {
    rule {
      apply_server_side_encryption_by_default {
        sse_algorithm = "AES256"
      }
    }
  }
  tags = {
    Name = "S3 Bucket for velero EKS backups"
  }
  force_destroy = true
}

```

- **GKE files**

Continuing with the Kubernetes cluster in GCP, known as GKE, the following files have been developed:

The following file has been named “CONFIG.tf”, and configures basic information like the GCP region to be used, and the main variables to be set throughout the folder.

```

provider "google" {
  project = var.gcp_project_id
  region  = var.gcp_default_region
  zone    = var.gcp_default_zone
}

variable "gcp_project_id" {
  default = "claranet-playground"
}

variable "gcp_default_region" {
  default = "europe-west1"
}

variable "gcp_default_zone" {
  default = "europe-west1-b"
}

```

```

}

locals {
  vpc = {
    node_subnet_ip_range   = "10.162.0.0/22"
    pod_subnet_ip_range    = "10.42.0.0/16"
    service_subnet_ip_range = "10.143.0.0/23"
    pod_subnet_name        = "gke-pod-alias-ips"
  }
  node_nat_ports_per_vm   = 1024
  pod_nat_ports_per_vm    = 64
  fw_mgmt_sources         = ["0.0.0.0/0"]
  cluster = {
    name                = "sergio-test"
    min_master_version  = "1.21.6-gke.1500"
    master_ipv4_cidr_block = "172.16.0.32/28"
    daily_maintenance   = "03:00"
    tags                = ["gke-test"]
    node_disk_size_gb   = 20
    node_disk_type       = "pd-standard"
    node_image_type      = "COS"
    node_permissions    = [
      "https://www.googleapis.com/auth/compute",
      "https://www.googleapis.com/auth/devstorage.read_only",
      "https://www.googleapis.com/auth/logging.write",
      "https://www.googleapis.com/auth/monitoring",
    ]
  }
  preemptible = {
    node_version      = "1.21.5-gke.1302"
    initial_node_count = 1
    machine_type       = "n2-standard-2"
  }
}

```

The next file, called “VERSIONS.tf” file, also configures basic information, mainly, where the Terraform state is to be saved, which will be locally.

```

terraform {
  required_version = ">= 0.14"
  backend "local" {
    path = "terraform.tfstate"
  }
  required_providers {
    local = {
      source = "hashicorp/local"
      version = "~> 2.1.0"
    }
    google = {
      source = "hashicorp/google"
      version = "~> 3.59.0"
    }
  }
}

```

In this case, all the required configurations for the setup of the GKE cluster have been joined in a single file, called “GKE-ZONAL.tf”. This file will configure the VPC, the subnetwork for the cluster. Also, a router is set for the node network, and a public IP address is reserved. Then, a NAT is configured to allow outgoing traffic with the previous public IP. Once again, a router and an IP address is reserved for the pod network, and another NAT is configured. Later, a firewall

is set to only allow traffic to the required ports. Lastly, the master node and the worker node are configured.

```
resource "google_compute_network" "gke-vpc" {
  name          = "gke-cluster-test-sergio"
  description    = "GKE Cluster Network"
  auto_create_subnetworks = false
}

resource "google_compute_subnetwork" "gke-vpc-subnet" {
  name          = "gke-worker-nodes"
  description    = "Subnet for worker nodes from GKE"
  ip_cidr_range  = local.vpc.node_subnet_ip_range
  network        = google_compute_network.gke-vpc.self_link
  private_ip_google_access = true
  secondary_ip_range {
    range_name    = local.vpc.pod_subnet_name
    ip_cidr_range = local.vpc.pod_subnet_ip_range
  }
  secondary_ip_range {
    range_name    = "gke-services"
    ip_cidr_range = local.vpc.service_subnet_ip_range
  }
}

resource "google_compute_router" "gke-node-nat" {
  name     = "gke-node-nat"
  network = google_compute_network.gke-vpc.self_link
}

resource "google_compute_address" "gke-node-nat" {
  name = "gke-node-nat-external-address-test-sergio"
}

resource "google_compute_router_nat" "gke-node-nat" {
  name          = "gke-node-nat"
  router         = google_compute_router.gke-node-nat.name
  nat_ip_allocate_option = "MANUAL_ONLY"
  nat_ips        = [google_compute_address.gke-node-nat.self_link]
  min_ports_per_vm = local.node_nat_ports_per_vm
  source_subnetwork_ip_ranges_to_nat = "LIST_OF_SUBNETWORKS"
  subnetwork {
    name          = google_compute_subnetwork.gke-vpc-subnet.self_link
    source_ip_ranges_to_nat = ["PRIMARY_IP_RANGE"]
  }
  log_config {
    enable = true
    filter = "ERRORS_ONLY"
  }
}

resource "google_compute_router" "gke-pod-nat" {
  name     = "gke-pod-nat"
  network = google_compute_network.gke-vpc.self_link
}

resource "google_compute_address" "gke-pod-nat" {
  name = "gke-pod-nat-external-address-test-sergio"
}

resource "google_compute_router_nat" "gke-pod-nat" {
  name          = "gke-pod-nat"
  router         = google_compute_router.gke-pod-nat.name
  nat_ip_allocate_option = "MANUAL_ONLY"
  nat_ips        = [google_compute_address.gke-pod-nat.self_link]
  min_ports_per_vm = local.pod_nat_ports_per_vm
  source_subnetwork_ip_ranges_to_nat = "LIST_OF_SUBNETWORKS"
  subnetwork {
    name          = google_compute_subnetwork.gke-vpc-subnet.self_link
    source_ip_ranges_to_nat = ["LIST_OF_SECONDARY_IP_RANGES"]
    secondary_ip_range_names = [local.vpc.pod_subnet_name]
  }
  log_config {
```

```

        enable = true
        filter = "ERRORS_ONLY"
    }
}

resource "google_compute_firewall" "mgmt" {
    name      = "${google_compute_network.gke-vpc.name}-allow-mgmt-access"
    network    = google_compute_network.gke-vpc.name
    description = "Restrict management access to source prefixes"

    allow {
        protocol = "icmp"
    }

    allow {
        protocol = "tcp"
        ports    = ["22", "80", "443"]
    }

    source_ranges = local.fw_mgmt_sources
}

resource "google_container_cluster" "gke-cluster" {
    name                = local.cluster.name
    min_master_version = local.cluster.min_master_version
    location            = var.gcp_default_zone
    initial_node_count  = "1"
    remove_default_node_pool = "true"
    enable_legacy_abac   = "false"
    network             = google_compute_network.gke-vpc.name
    subnetwork          = google_compute_subnetwork.gke-vpc-subnet.name
    private_cluster_config {
        enable_private_nodes = true
        master_ipv4_cidr_block = local.cluster.master_ipv4_cidr_block
        enable_private_endpoint = false
    }
    master_authorized_networks_config {
        cidr_blocks {
            cidr_block = "0.0.0.0/0"
            display_name = "Internet-facing"
        }
    }
    # Set at least empty to activate aliasIP needed by VPC native cluster
    ip_allocation_policy {
        cluster_secondary_range_name = google_compute_subnetwork.gke-vpc-subnet.secondary_ip_range[0].range_name
        services_secondary_range_name = google_compute_subnetwork.gke-vpc-subnet.secondary_ip_range[1].range_name
    }
    maintenance_policy {
        daily_maintenance_window {
            start_time = local.cluster.daily_maintenance
        }
    }
    # disable basic auth
    master_auth {
        username = ""
        password = ""
        client_certificate_config {
            issue_client_certificate = true
        }
    }
    addons_config {
        http_load_balancing {
            disabled = false
        }
        network_policy_config {
            disabled = false
        }
    }
    network_policy {
        enabled = true
        provider = "CALICO"
    }
}

resource "google_container_node_pool" "preemptible_nodes" {

```

```

name          = "${local.cluster.name}-stateful-1-21-5-gke-1302"
cluster       = google_container_cluster.gke-cluster.name
version       = local.preemptible.node_version
initial_node_count = local.preemptible.initial_node_count
management {
  auto_repair = true
  auto_upgrade = false
}
node_config {
  machine_type = local.preemptible.machine_type
  disk_size_gb = local.cluster.node_disk_size_gb
  disk_type    = local.cluster.node_disk_type
  image_type   = local.cluster.node_image_type
  oauth_scopes = local.cluster.node_permissions
  preemptible = true
  labels = {
    node_pool = "preemptible"
  }
}
}
}

```

Last but not least, the Storage Bucket for Velero is set up, by creating a custom role for it, the bucket itself, a service account and its binding.

```

# =====
# Create custom role for disk/snapshots
# =====
resource "google_project_iam_custom_role" "velero-role" {
  role_id    = "velero"
  title      = "Velero Role"
  description = "A Role to get disks and create/delete Snapshots"
  permissions = [
    "compute.disks.get",
    "compute.disks.create",
    "compute.disks.createSnapshot",
    "compute.snapshots.get",
    "compute.snapshots.create",
    "compute.snapshots.useReadOnly",
    "compute.snapshots.delete",
    "compute.zones.get",
  ]
}

# =====
# The bucket where to save k8s backups
# =====
resource "google_storage_bucket" "velero-backups-bucket" {
  name          = "velero-backups-sergio-test"
  location      = var.gcp_default_region
  storage_class = "REGIONAL"
}

# =====
# The Service Account
# =====

```



```

module "velero-service-account" {
  source = "git::https://git.eu.clara.net/de-tf-modules/gcp/service-
account.git?ref=0.7.0"

  id          = "velero-backups"
  display_name = "Velero Backups"
  dump_iam_json_key = true
  roles       = ["projects/claranet-playground/roles/velero"]
}

# =====
# Bind SA to the bucket
# =====
resource "google_storage_bucket_iam_member" "velero-backups" {
  bucket = google_storage_bucket.velero-backups-bucket.name
  role   = "roles/storage.objectAdmin"
  member = "serviceAccount:${module.velero-service-account.email}"
}

```