National College of Ireland

Single and Multi-Cloud Disaster Recovery Management using Terraform and Ansible

MSc Research Project
Cloud Computing

Ram Barath Thiyagarajan
Student ID: x18147305

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ram Barath Thiyagarajan
Student ID:	x18147305
Programme:	Cloud Computing
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Muhammad Iqbal
Submission Due Date:	12/12/2019
Project Title:	Single and Multi-Cloud Disaster Recovery Management using Terraform and Ansible
Word Count:	5480
Page Count:	19

Signature:	
Date:	3rd February 2020

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

Single and Multi-Cloud Disaster Recovery Management using Terraform and Ansible

Ram Barath Thiyagarajan
x18147305

Abstract

With Disaster recovery (DR) in cloud computing, services provided by cloud providers are mostly managed by third-party disaster recovery providers. Therefore, it creates the necessity to develop an effective DR method to minimize the total cost of recovery along with the Recovery Point Objective (RPO) and Recovery Time Objective (RTO). This paper proposes an idea of DevOps -Infrastructure as a code (IaaC) disaster recovery management using Terraform and Ansible. In this study, Terraform and Ansible templates are developed to build single and Multi-Cloud Infrastructure based on Amazon Web Services (AWS) and Google Cloud Platform (GCP). Our main aim is to study the process of automated disaster recovery during an emergency, along with the total time taken to bring up the services to regular operation. The study compares the automatic concept with manual actions taken during the recovery period and shows the benefits of using IaaC.

Index Terms – Disaster Recovery, Cloud Computing, Infrastructure as a code, RPO, RTO

1 Introduction

Cloud computing became a top trend for the past ten years in the area of IT software and hardware because of its elasticity, reliability, scalability, service orientation, and availability. Another critical thing in cloud systems is, it supports different SLAs to provide excellent quality of services and a variety of information systems. With the development of cloud computing, computing services made accessible to reach every customer irrespective of the business size. Cloud service providers (CSP) are offering a wide variety of resources under three categorizations as Infrastructure as a Service, Platform as a Service, and Software as a Service. Cloud infrastructure system is the collection of compute hardware's, storage devices, and other dependent resources offered by cloud service owner. Otey (2019) explained widely about the concept of Multi-cloud environment and its need to overcome the drawback faced due to the single cloud model. The multi-cloud approach mainly used to overcome the vendor lock-in issues, interoperability issues, and to manage cost structure.

1.1 Motivation and Background

According to Jim (2019) IBM's official press release, 85 percent of big enterprises using the Multi-cloud model to design their IT architectures. IBM estimates that by 2021, 97 percent of the large and mid-level enterprises, IT will move to multi-cloud. Creating the complex structure for an organization, along with maintenance, will always be a strenuous activity, even there are some risks as well. The main thing is that only the hardware resources maintained by CSP. Cloud service consumers (CSC) should take responsibility for the creation and maintenance of cloud services according to Alshammari et al. (2018). On the other hand, DR management is the most crucial part of IT architecture while designing the cloud model. CSC should think about DR, application security, patch management, and updates while designing architecture in cloud systems.

There are several methods to create and manage cloud infrastructure remotely. But most of them are provided by cloud service providers like command-line interface, graphical interface to access their cloud systems listed in Sun et al. (2016). The main advantage of this manual approach is no profound knowledge or skills needed by the customer side engineers. Gu et al. (2014) highlighted that the only requirement for this method is to understand and operate based on the cloud provider documents. At the same time, this approach is not a suitable one when recovery is in need. When there is an emergency that arises in computing resources, the systems going into the offline state and never come back to operation by itself. It could be the accidental deletion of critical data or computing resources. Sun et al. (2016) explained that the CSP could not handle this form of failure because the CSC has the responsibility to manage the components on top of the cloud infrastructure. In the case of database-based systems, the frequent backup and the replication are part of the day to day activity. But for the other system components, the critical data should be backed up properly to recover anytime.

Already we have seen the main disadvantage of manual infrastructure maintenance; it requires more time to bring the affected resources into regular operation. Also, this period is not constant and increases based on the complexity and number of cloud components. Time taken during the process of DR will directly impact the trust and business of the client. Disasters cannot by any defined set of procedures and may come at any time. Also, planning complex solutions for recovery is not a great idea. These factors motivated me to design an Automated Infrastructure DR model. Here we have taken the concept of Infrastructure as a Code to create a template for building infrastructure in the cloud as explained by Lavriv et al. (2018). It is nothing but code with instructions to cloud providers to act. Chang et al. (2019) explains about the great functionality of reusability, which highly recommended during recovery situations and to build similar architecture again. IaaC concept makes Infrastructure development and deployment much more accessible and provides impeccable ability in automation. It reduces the time taken to build and helps to manage it using the current state of the infrastructure.

The research aims to find out the best method for DR in the cloud system. The single cloud DR method suits best for Mid-level organization, and the Multi-cloud DR method suits best for Complex or high priority IT architecture. To achieve higher availability, Lavriv et al. (2018) compared the IaaC based automated model with an existing manual approach. Terraform, and Ansible is the recognized automation tools to prove the research work decision.

1.2 Research Question

The current issues concerning Cloud DR are the recovery time of the business application in case of an emergency breakdown.

Can the recovery time of cloud infrastructure during crisis be reduced using Terraform and Ansible tools?

1.3 Research Objectives

The research question addresses the following objective: Objective 1: Designing AWS cloud infrastructure using Terraform, Objective 2: Designing AWS cloud infrastructure using Ansible, Objective 3: Designing Multi-Cloud infrastructure in AWS and GCP, Objective 4: calculating the response time taken for the proposed idea, Objective 5: Comparing the recovery time between Manual and Automated operation, Objective 6: Documentation of the research work along with user manual.

2 Related Work

2.1 Different Disaster Recovery studies on ICT:

In line with a popular standard definition, disaster recovery is nothing but a set of policies, software tools, and a set of procedures providing the service for improvement of critical IT infrastructure and its resources following a system failure or Man-made failure. Disaster recovery mechanisms not only designed for Information and communications technology (ICT). It is common for all the business strategies where continuous service is needed. According to Cloud and software-based disaster recovery model, the expectation is the same as principles and highlights from the general workflow. There are various studies on disaster recovery cases explored before; commonly, all the reviews deliver the same set of considerations such as privacy, security, sustainability, and disaster case management itself. In Watson et al. (2014) research paper, there are many models proposed to provide a useful disaster recovery model like "Multi-Site" approach, the main objective of this model is to store the data in two or more separate sites along with multiple methods. Alshammari et al. (2017) proposed many child approaches available under multi-site model such as Logical Separation of business units in the same region using Single Cloud single region (SCSR) DR management, physically separated business units in multiple region using Single cloud Multiple region (SCMR) DR management, and the last one is separation of Infrastructure between two different isolated service providers using Multi-cloud DR system.

In an Organization, the disaster recovery process is one of the most important and highly critical activities to ensure running business operations with higher availability. All DR processes will follow the same set of the process such as replication, maintaining multiple copies, and regular synchronization to match with the current data in the primary site. The main agenda of the DR process lies with how quick the service back online after the catastrophic incident.

Wood Wood et al. (2016) comes up with a cloud disaster recovery mechanism based out of previously proposed recovery best practices, processes, and other factors influencing the disaster recovery model. In this paper, author mainly concentrates on the risks and challenges involved during the process of failover recovery, rollback steps, and the other means associated with it. The main objective of his study is to find out the best possible solution to ensure the affected system back to the regular operation with all defined configurations and dependencies. Wood categorized three different recovery backup strategies, such as Warm, Hot, and cold, to reduce the impact of critical resources across all possible scenarios. The author also defined the step by step procedure for DR starting from network configuration, virtual machine migration, and secured data isolation.

According to Togawa and Kanenishi Togawa and Kanenishi (2013), disaster recovery management is vital for e-learning. The education system like a Moodle system. The author has taken one of the sensitive scenarios of frequently occurring Earthquakes across different regions in Japan for the period of the next 30 years and proposed a new Disaster recovery model to backup all education-based data as an emergency process. Authors provided valid proof of his research work and expected results to benefit the intended outcome. But on the other side, the author missed explaining in detail the RPO, RTO, Financial estimation, performance scale, and the triggering mechanism. These factors are playing a vital role in the implementation n of recovery mechanisms across all business domains.

Pokharel and Park Pokharel et al. (2016) discussed the Geographical Redundancy Approach, which is mainly for ensuring High Resource Uptime, Availability, and Sustainability, even after an unexpected failure. The Authors also considered effective cost management for implementing this recovery approach. They have taken the Markov chain model to analyze the proposed DR approach. In this paper, they have made two geographically separated datacentres to store information for better Redundancy. The Authors not discussed the RPO and RTO, which is highly essential while considering the Multi-site model of redundancy.

2.2 Cloud Model for Disaster Recovery Management:

2.2.1 Single-Cloud and Multi-Cloud Model:

Cloud computing is gaining its popularity day to day on both technically and with increasing users. In Alshammari et al. (2017), authors detailed about the Public cloud providers and their services to offer better solutions to customers. Unlike the self-maintained private data center, cloud providers enable different well-structured procedures for storing the data in a highly secured manner. They store the data in logically separated hardware and using API's to interact with all resources. It enables customers to use the services from anywhere irrespective of the data location. Cloud service providers maintain their hardware infrastructure more securely by providing only access level control to consumers. It enables them to incorporate many end-users to utilize their resources more optimally. From a Disaster recovery point of view, the primary vital advantages of cloud computing like High Availability, redundancy, flexibility, and geographically separated adds its valuable contribution.

The authors Sahebjamnia et al. (2015) investigated the business impact due to the disaster. They have analyzed the complexity, which involves predicting failures, reason, time, and extendibility. Authors have come up with a new framework to support organizations against unpredictable disasters. In Khoshkholghi et al. (2014), the authors surveyed from a cloud provider point of view. They also explained about the cloud provider's responsibility to ensure the availability to the customers at any cost. Even though the data center is down, cloud providers should provide continuous service to customers. Authors have investigated all the current approaches with disaster recovery in the cloud system.

According to Groninger, Capretz, Mezghani, and Exposito, Grolinger et al. (2013)], a unified Disaster recovery cloud data (Disaster – CDM) model is still not implemented, and under the testing phase, only a few simulations designed to satisfy the individual business needs. The main objective of this paper is to linguistically integrate extensive Data resources to handle disaster recovery for Data management. The Author's proposal is more of merging the cloud resources to reduce the risk of data loss. But the paper doesn't deal with the suitable approach to find the best solution. Business continuity always concerning the disaster management system; Saquib Saquib et al. (2013) innovates a new disaster recovery plan as a SaaS for database-centric applications. The main motive of this service is to provide Higher Availability, the quick turnaround time for recovery, and Data loss transactions. The Author claims the proposed approach aims for zero RPO and very minimal RTO using Database server and iSCSI model replication. Author also adds more about the other advantages such as automatic failover, rollback, scalability of the proposed service (DRaaS).

Alzain, Eric, and Ben AlZain et al. (2013) introduced the new terminology called Multiple-Cloud or interconnected cloud. CachinCachin et al. (2014) and VukoliVukolic (2015) have taken the nod from the Alzain and discussed the various layers in the multi-cloud architecture. They categorized the bottom and the second layer to place the inner and inter-cloud to reside accordingly. Either one of them would be the primary and another one as secondary. The authors are also discussing the Byzantine faults, which are nothing but the common identification of all acknowledged hardware and software failures in the cloud environment. Designing and implementing an architecture for BFT protocols are highly tricky in a real scenario. So, looking for an alternate option is much needed for multi-cloud design. There are few more protocols identified then advanced to tested against BFT faults such as DepSky-CA protocols. These protocols analyzed by Bessani and Correia Bessani et al. (2013).

However, all these researches are mostly manual approaches and do not deal with automation in the cloud platform. This paper was proposed to prove that the automation solution will be the best solution for disaster recovery management. In this paper the automated solution is compared with manual operation. This paper is the experimental study of disaster recovery using Terraform and Ansible in a real cloud platform Amazon web services and Google Cloud Platform.

3 Methodology

The proposed DevOps concept is chosen to design any complex IT infrastructure as a programmatic code. The mainly discussed area in this section is the complete utilization of its real capability and enables full automation of the required configuration settings and management. This Section explains the components and the details needed to prove that this proposed idea could perform the research. Here the Cloud Service providers (CSP), Cloud Service Consumers (CSC), Terraform, and Ansible are the main entities driving the whole process. As said before, the service level agreement with CSP has its limitations while coming to disaster and followed by recovery. Here the CSC is responsible for managing the infrastructure for their business needs.

3.1 Terraform:

Terraform Documentation (2019b) is a software tool used for creation, modification, deletion, and versioning of cloud infrastructure in a most secured way and efficiently. It provides services to more than 200 cloud and in-house infrastructures. Terraform design infrastructure using High-level syntax popularly called as templates or modules. The main advantage of using terraform it can be re-used and versioned for easy modification. It also provides Resource Graph, Auto Provisioning, Dry run capabilities. In Orzechowski et al. (2018) Lavriv et al. (2018), the authors described terraform as the best automation tool to perform a disaster recovery process.

- **Terraform init:** It initializes the code and downloads required prerequisite packages based on CSP
- **Terraform plan:** This feature is like the dry-run process. After the templates designed, the code can be tested for any errors
- **Terraform apply:** It applies all the programmatic code designed to build infrastructure and perform API calls to CSP for the creation of resources
- **Terraform destroy:** This feature is the handy option to delete all the created infrastructure in case of any wrong configuration. It highly helps to manage the cost of running cloud resources
- **Terraform graph:** Graph feature is used to display the infrastructure in a graphical representation.

3.2 Ansible:

Ansible Documentation (2019a) is another Open Source software tool mainly used for configuration management in the cloud and non-cloud legacy systems. Ansible is popularly known for configuration setup; however, it can manage cloud infrastructure as same as terraform. Singh et al. (2015) It supports all popular CSPs to provide API calls for cloud infrastructure management Masek et al. (2018).Figure 1 is the high-level architecture of the Ansible workflow, Project, Playbook, Role, Task and module described in detail below.

- **Hosts:** The Targeted host details are mentioned here to push the code for cloud infrastructure. In this paper, the localhost is the base location to trigger the cloud infrastructure creation tasks.
- **Playbooks:** It is a file containing all the creation, configuration setup of cloud infrastructure. The format of the file is YAML
- **Roles:** It includes modules, handlers, tasks, and files. Roles are a collection of directories required for complete infrastructure management.
- **Modules:** Modules are the pre-defined API trigger for every creation process. The multiple modules grouped under playbook for consolidated cloud build.

4 Design Specification

According to the Gartner report Gartner (2019), DR is not just a process; it involves the collection of policies, best practices, procedures, and the combination of both software and hardware tools. So, designing an architecture for recovery objective is highly essential

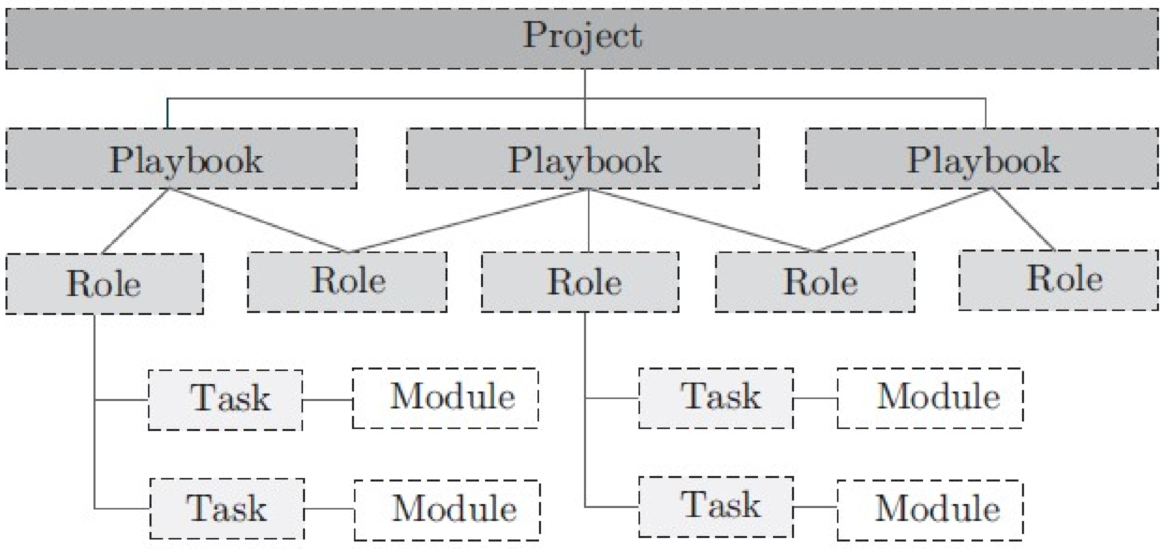


Figure 1: Ansible Script Architecture Documentation (2019a)

and useful. The figure 2 shows the high-level design of the implemented multi-cloud architecture. It describes all the components of the multi-cloud service providers. This paper identifies AWS as Primary CSP and Google Cloud Platform (GCP) as secondary CSP.

4.1 Case Study 1: Creating AWS 2 Tier Architecture using Terraform:

The paper evaluates the technique of managing the software application running on Cloud infrastructure using the IaaC concept. The main aim of using this technique is to reduce the recovery time. This case study experiment is performed using the local machine with the above configuration for Terraform triggers. In this case study, For our study, we have created 1 AWS VPC, 2 Availability zones (AZ's) 4 Subnets, which includes two private and public subnets, 1 Autoscaling group running along with 1 Elastic load balancer (ELB). For security purposes, the auto-scaling group created on private subnets, and it allows only traffic from ELB.

Firewall settings such as ingress and egress configurations constructed using AWS Security Groups (SG). Auto-scaling launch configuration mentioned with a minimum of 2 EC2 instances and a maximum of 4 cases for better scalability and fault tolerance. On the other side, an Internet gateway created to allow all incoming and outgoing internet traffic into the VPC network. NAT gateway designed to allow only egress internet traffic for private subnet instances for better safety. Outside connections cannot directly hit private subnets.

Overall the implementation is the same as the production environment in real-time with all necessary components for providing a more agile and secure environment. Static website is used for testing the cloud level failure and Operations level failure. The Implementation steps have drawn as a flowchart in the figure.

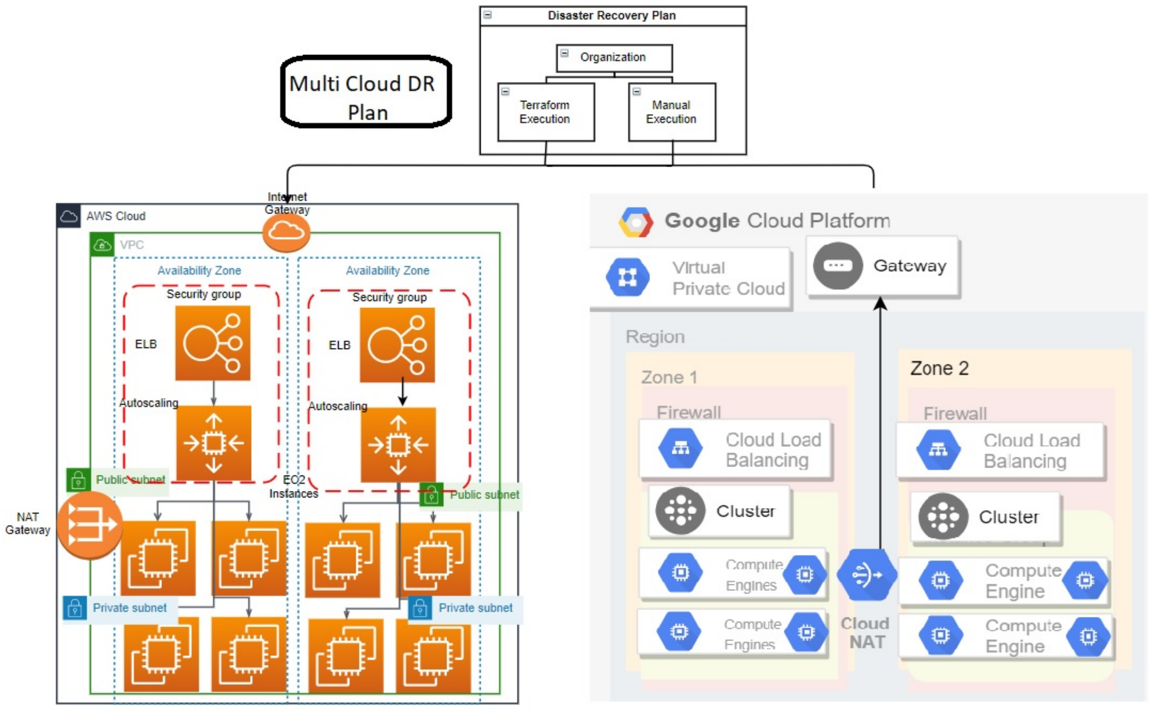


Figure 2: Multi-Cloud Architecture

4.2 Case Study 2: Creating AWS 2 Tier Architecture using Ansible:

This experiment replaces Terraform with the Ansible along with all similar AWS environment setup which includes, VPC, EC2 instances, SG, Subnets, NAT Gateway, Internet Gateway, and an application to check the operation of the cloud. The Implementation steps for the AWS architecture using Ansible are the same as Case Study one; therefore, the figure describes the workflow process for both terraform and Ansible. The only difference is instead of using terraform based HCL scripts, for ansible YAML based playbooks are developed. All AWS components like VPC, Subnets, SG, Autoscaling group, ELB, Nat gateway, Internet gateway are configured separately in multiple Ansible playbooks. Other configurations are the same as case study 1 by keeping security and reusability in mind.

4.3 Case Study 3: Creating AWS and GCP Architecture using Terraform:

In case study 3, the complete cloud infrastructure varies from the other two cases. Here AWS and GCP are considered as Primary and Secondary CSP's respectively. The web-based application hosted in the apache server considered as a core, and designing a downloads DR plan is our research work.

As discussed in the section (4.1) and (4.2), AWS components are designed using the terraform templated file called HCL for this case study's primary CSP. On the other hand, GCP considered as the secondary recovery cloud site in case the entire primary site is shut down due to an emergency.

Our Architecture itself is designed in such a way to perform faster DR process to reduce the application downtime. Local machines are pre-installed with Ansible and other dependencies like python3, boto3. The figure 2 shows the high-level architecture of this case study with multiple cloud design. The local machine in the on-premise network is the parent network for triggering the DR process. GCP architecture includes Compute engines, Cloud load Balancer, Cluster, Firewalls, Zone, VPC, Cloud Nat, and gateway.

5 Implementation

Terraform, and Ansible is the identified tool for our study. The ability of these tools enables the infrastructure design as an easily manageable code and applies the changes whenever the need arises, which is the core theme for our study approach. Installing Terraform and Ansible in one or more instances in our local machine. Multiple instances help in improving fault tolerance.

For the implementation Intel i7 core processor is used along with Windows 10 home edition. The other software components used in our research work are Virtual Box for VM creation and putty for local ssh. The configurations are as follows,

5.1 Implementation of Single Cloud DR plan using Case Study 1 and 2:

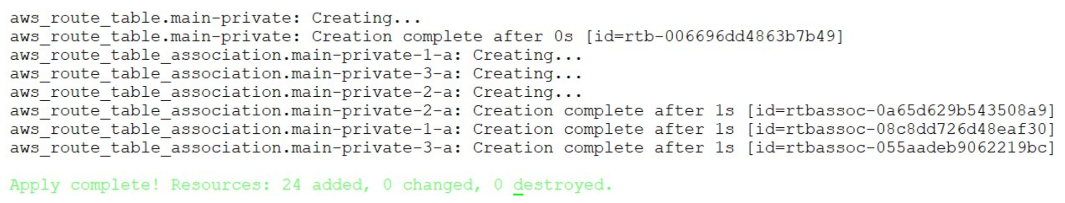
Figure 5 shows the typical flowchart implemented for a single cloud and multi-cloud DR management. Here the first half of the workflow represents the implemented single cloud DR architecture. Local Machine named as In-house instances, periodic checks will perform, and the automatic sanity checks scheduled on the running application. If there are any changes identified in the operations side of the application having issues in connection. The Next process is to determine whether it is an application-side issue or Infrastructure side issue. If it is an infrastructure side issue, terraform will execute the state comparison between the existing state file and current state to identify where it went wrong and perform the recovery operation immediately to bring back the system into regular service.

Terraform project for Single cloud DR process represented in the figure. This high-level process-oriented architecture represents the implemented workflow. It consists of terraform based Hashicorp configuration language (HCL) scripts describing Amazon Web Services (AWS) resources using pre-defined terraform modules. For a fundamental understanding of the written code, all the AWS modules defined separately. vpc.tf is the collection of user-defined variables for AWS Virtual Private Cloud (VPC) and subnets. autoscaling.tf and elb.tf files are having the autoscaling setup for the EC2 instances and Elastic Load balancer (ELB) configurations. Similarly, all the necessary AWS resources are structured using .tf extensions.

After scripting, all the required files for entire AWS architecture, execute terraform commands to create all the defined resources. Terraform would generate a JSON file describing the infrastructure architecture. This file later used to compare the current and expected state of the cloud infrastructure to trigger a disaster recovery plan.

Below is the trigger command for the terraform to look for all .tf files for managing the cloud resources. In figure 3 represents the AWS infrastructure creation using terraform modules.

terraform apply



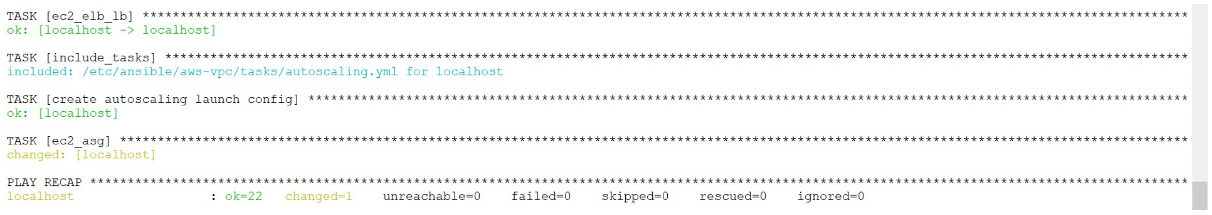
```
aws_route_table.main-private: Creating...
aws_route_table.main-private: Creation complete after 0s [id=rtb-006696dd4863b7b49]
aws_route_table_association.main-private-1-a: Creating...
aws_route_table_association.main-private-3-a: Creating...
aws_route_table_association.main-private-2-a: Creating...
aws_route_table_association.main-private-2-a: Creation complete after 1s [id=rtbassoc-0a65d629b543508a9]
aws_route_table_association.main-private-1-a: Creation complete after 1s [id=rtbassoc-08c8dd726d48eaf30]
aws_route_table_association.main-private-3-a: Creation complete after 1s [id=rtbassoc-055aadeb9062219bc]

Apply complete! Resources: 24 added, 0 changed, 0 destroyed.
```

Figure 3: Terraform AWS Creation

As discussed in section 5.1, Figure 5 workflow is the same for case study 2; the only difference is using ansible instead of terraform. For case study 2, a similar set of AWS architecture designed using Ansible playbooks. Figure 4 represents the AWS infrastructure creation using ansible playbooks.

ansible-playbook main.yml



```
TASK [ec2_elb_lb] *******************************************************************************************************************************************
ok: [localhost -> localhost]

TASK [include_tasks] ****************************************************************************************************************************************
included: /etc/ansible/aws-vpc/tasks/autoscaling.yml for localhost

TASK [create autoscaling launch config] *********************************************************************************************************************
ok: [localhost]

TASK [ec2_asg] **********************************************************************************************************************************************
changed: [localhost]

PLAY RECAP **************************************************************************************************************************************************
localhost                  : ok=22   changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Figure 4: Ansible AWS Creation

5.2 Implementation of Multi Cloud DR plan using Case Study 3:

Figure 5 shows the consolidated case study flowchart implemented for multi-cloud DR management. The implemented solution for this experiment is designing AWS, GCP, Local machine for initial trigger and terraform. In-House instances play a vital role in deciding which cloud model is useful for the situation where DR is needed. If there is an issue with one or more cloud resources in the primary cloud infrastructure, a single cloud DR method explained in section (5.1) flow model will be invoked. In case if there is a complete outage of VPC or the entire region in any one of the cloud networks, manual execution or automated scripts used to trigger the second half of the flowchart involving secondary Cloud service provider provisioning.

Here it is a straightaway process, there is no need to compare the expected state of the infrastructure with the current state. After understanding the primary cloud resources behavior, secondary cloud creation with the pre-defined template triggered to ensure the web application is up and running in another CSP network.

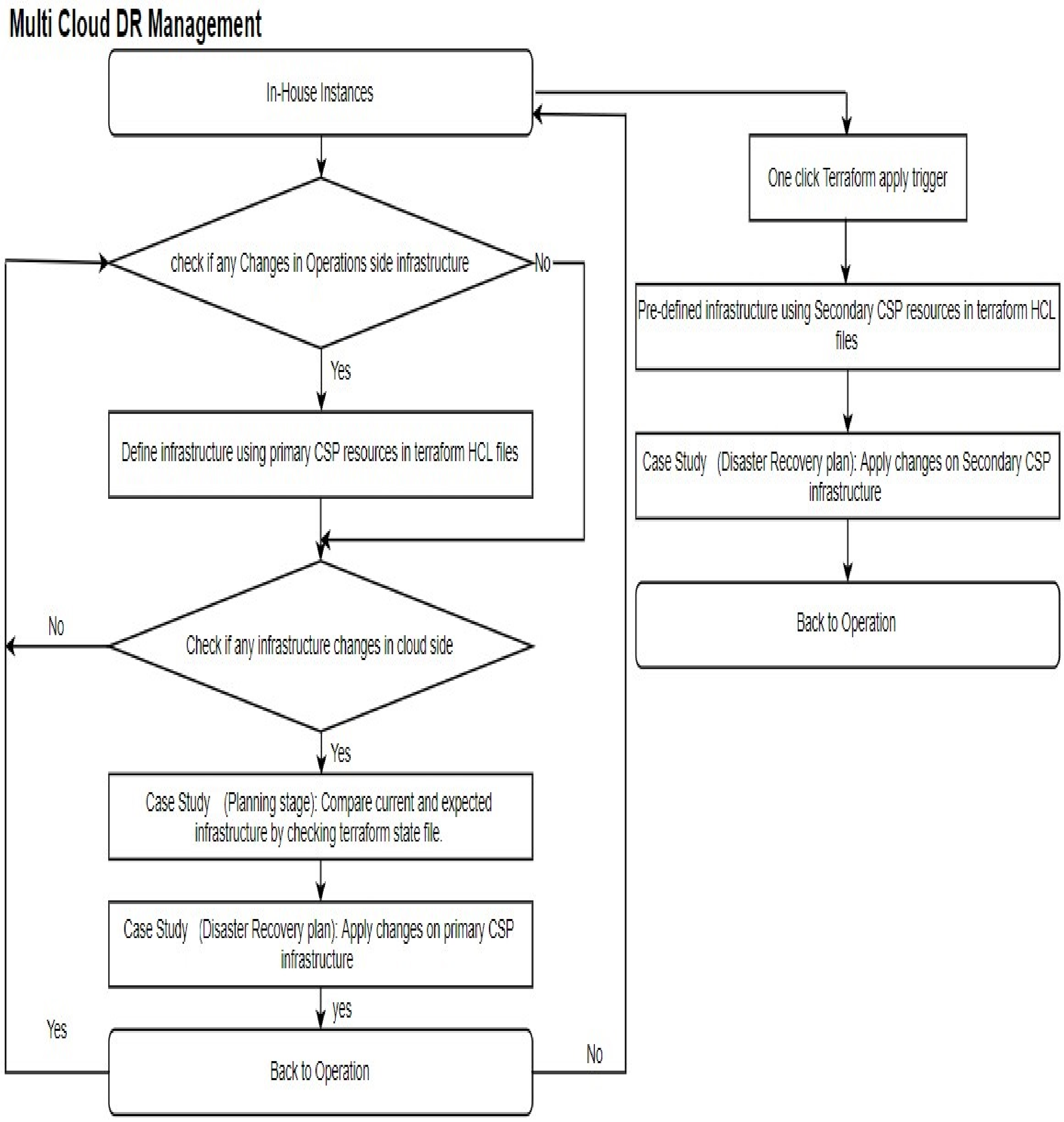

Multi Cloud DR Management
In-House Instances
check if any Changes in Operations side infrastructure
No
Yes
Define infrastructure using primary CSP resources in terraform HCL files
No
Check if any infrastructure changes in cloud side
Yes
Case Study (Planning stage): Compare current and expected infrastructure by checking terraform state file.
Case Study (Disaster Recovery plan): Apply changes on primary CSP infrastructure
yes
Yes
Back to Operation
No
One click Terraform apply trigger
Pre-defined infrastructure using Secondary CSP resources in terraform HCL files
Case Study (Disaster Recovery plan): Apply changes on Secondary CSP infrastructure
Back to Operation

Figure 5: Multi-Cloud workflow

6 Evaluation

This Section is to provide the experimental analysis of the proposed model and the main findings of the research work. As part of the research project, my idea is to run a production-grade web application into a single and multi-cloud environment with the help of both manual and automated approaches. Finally, to perform the comparison between the manual and automatic operation during the disaster.

The Experimental plan includes the public cloud platform AWS and GCP. Both cloud environments used to build and manage the cloud infrastructure using our concept of automation. For the evaluation purpose, basic AWS Free tier account and GCP free accounts used for configuring the changes. In both cloud environments, ubuntu 18.04 taken as the default AMI option to create a computing engine.

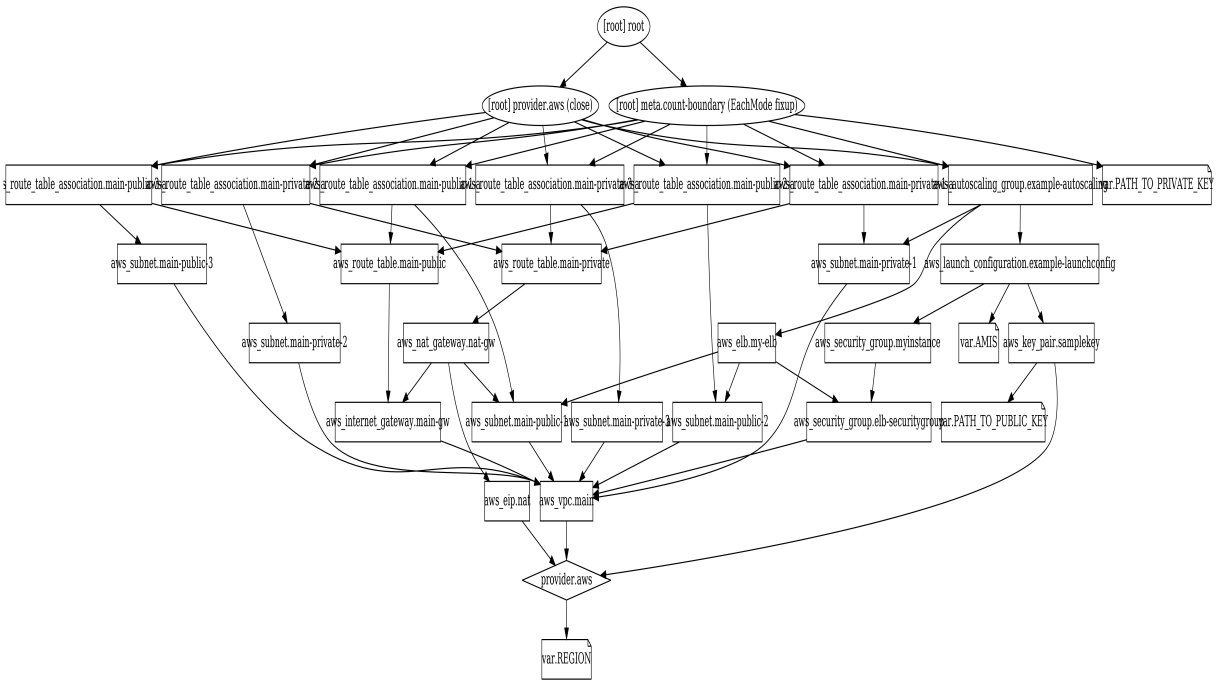

[root] root
[root] provider.aws (close)
[root] meta.count-boundary (EachMode fixup)
aws_route_table_association.main-public-3
aws_route_table_association.main-private-2
aws_route_table_association.main-public-1
aws_route_table_association.main-private-3
aws_route_table_association.main-public-2
aws_route_table_association.main-private-1
aws_autoscaling_group.example-autoscaling
var.PATH_TO_PRIVATE_KEY
aws_subnet.main-public-3
aws_route_table.main-public
aws_route_table.main-private
aws_subnet.main-private-1
aws_launch_configuration.example-launchconfig
aws_subnet.main-private-2
aws_nat_gateway.nat-gw
aws_elb.my-elb
aws_security_group.myinstance
var.AMIS
aws_key_pair.samplekey
aws_internet_gateway.main-gw
aws_subnet.main-public-1
aws_subnet.main-private-3
aws_subnet.main-public-2
aws_security_group.elb-securitygroup
var.PATH_TO_PUBLIC_KEY
aws_eip.nat
aws_vpc.main
provider.aws
var.REGION

Figure 6: Terraform Representation of Cloud resources generated by Terraform Graph functionality

Figure 6 represented our AWS cloud services created by the Terraform software tool based on infrastructure as code. This Graph designed using terraform Graph functionality, which mentioned already in section (3.1). It shows the clear workflow and its dependencies build using infrastructure as a code concept. The flow of the chart contains AWS services, which interact with each other for the designed web application. Here we can view the created VPC, 4 Subnets including two private and two public, Key pair for SSH connection, Security Group. Please refer to section (4.1) for the list of services used for terraform based cloud infrastructure creation.

6.1 Experiment 1/ Comparing Single cloud Disaster recovery duration between Terraform based Infrastructure build vs. Manual:

As discussed in the Implementation section, AWS cloud resources created using Terraform script. To perform the experiment, the disaster was triggered manually by deleting a few of the cloud resources. After the deletion, the recovery process has triggered by comparing the current state file with the existing state file. Finally, the deleted resources are identified and initiate the request to bring cloud resources back to the operation. During the second iteration, all the cloud resources destroyed using terraform destroy command and perform the same experiment to bring back the web application into normal operation.

An investigation between the Manual and Automated recovery options identified and the cloud resources created separately considering the different number of cloud resources involved in the disaster, the total number of instances involved, and the dependencies of every situation — the result of this experiment presented in figure 7. The chart shows the experiment results and its index. Figure 7 shows the relation between the Recovery duration of the system and the complexity of the cloud system. The complexity is considered based on the number of instances. The comparison results presented the benefit of using Terraform over a manual approach for disaster recovery in cloud infrastructure. The Recovery time in seconds ranges from 96 seconds to 148 seconds for the terraform based recovery. On the other hand Manual recovery time ranges from 3600 to 7000 seconds approximately.

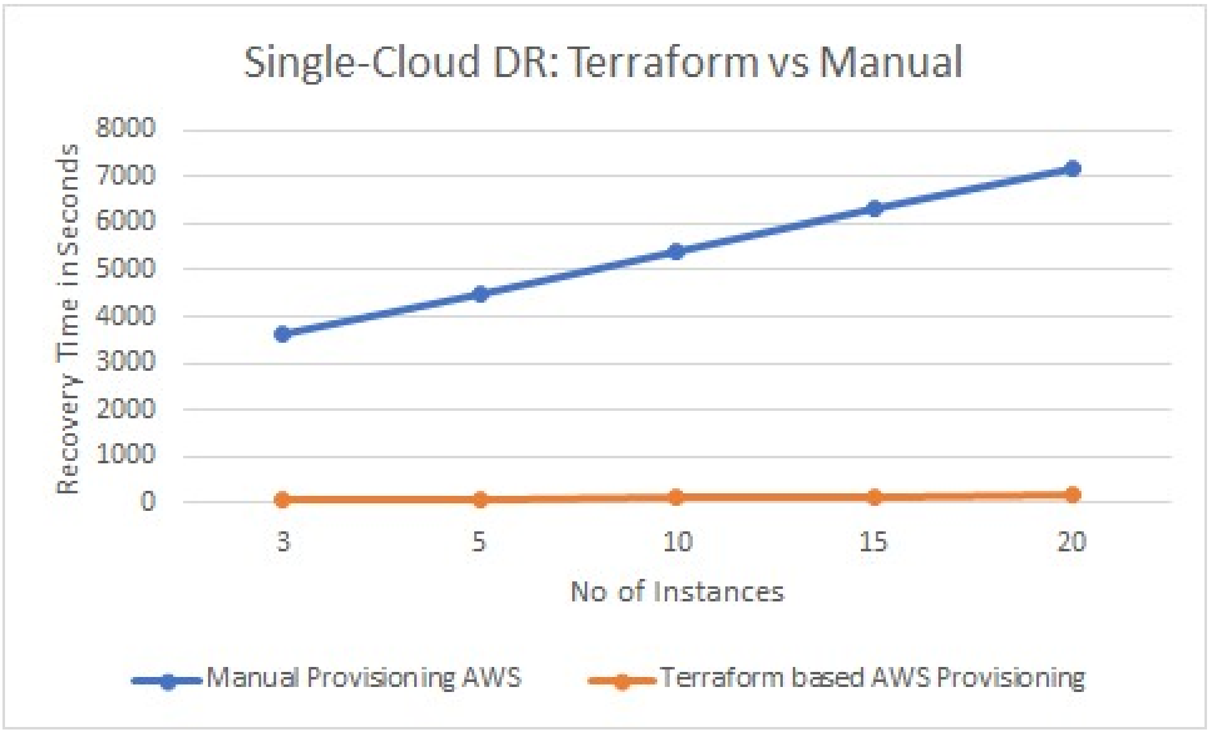


Figure 7: Terraform Vs Manual DR Comparison

6.2 Experiment 2/ Comparing Single cloud Disaster recovery duration between Ansible based Infrastructure build vs. Manual:

In analysis 2, the disaster was triggered manually by deleting computing resources using the AWS console. After the deletion, as part of the recovery process, the ansible-playbook command is executed to perform the creation of stopped services in AWS. An investigation between the manual and ansible based automated disaster recovery model identified. Figure 9 displays the experimental results of a comparison between the recovery duration and complexity of the cloud infrastructure. As same as experiment 1, the complexity and the dependencies considered. Unlike the Terraform trigger, ansible needs to initiate manually. While executing the Ansible playbooks, the creation modules implemented only when the recovery expected, and there are no duplicate cloud resources will create. The comparison chart clearly shows the difference between the time taken to recover from a cloud disaster using both manual and ansible way of DR plan. Again, automation tool based DR is highly effective and quick in response to the disaster. Here the Ansible based AWS creation for recovery takes around 220 to 560 seconds depends on number of instances varies from 3 to 20.

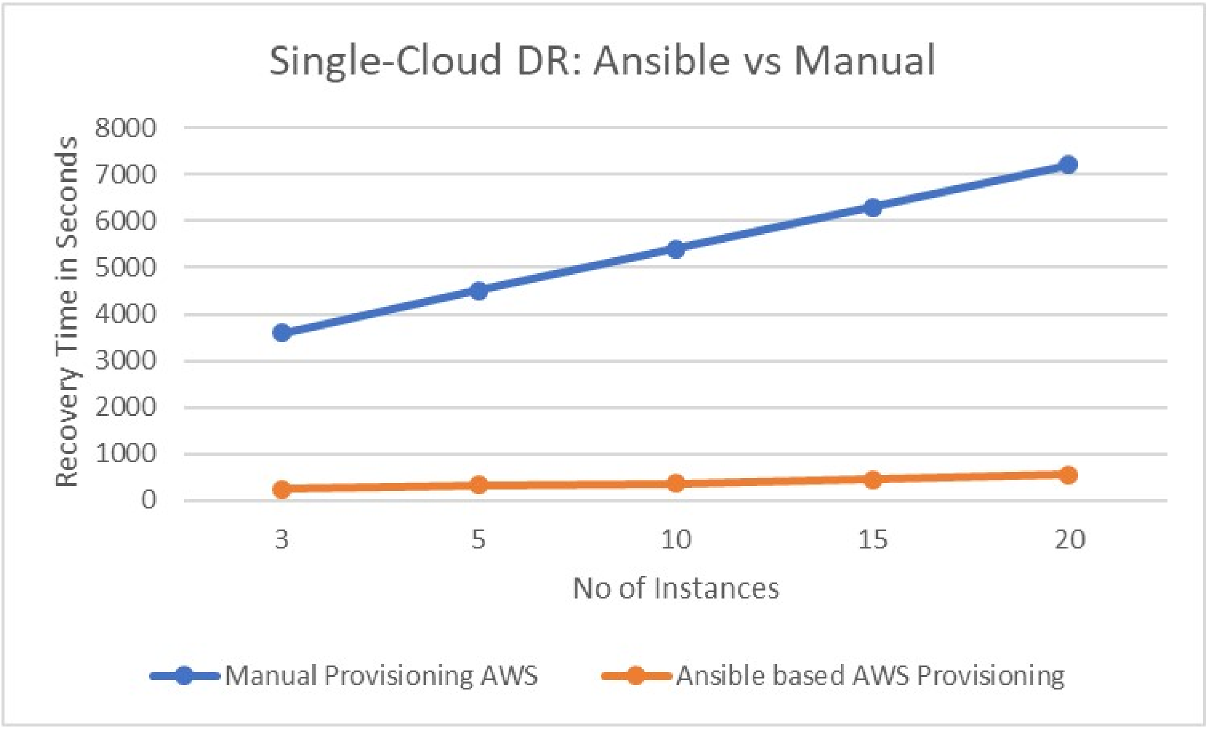


Figure 8: Ansible Vs Manual DR Comparison

6.3 Experiment 3 / Comparing Single cloud Disaster recovery duration between Terraform based infrastructure build vs. Ansible:

This experiment not executed separately; the results are the combination of section (6.1) and (6.2). The recovery time details from the terraform based DR model and the ansible based DR model compared to find out the best automation tool in cloud DR management. The Results show that terraform is the best IaaC tool while compared to ansible. The recovery time and the number of instances comparison values are plotted and displayed in figure 9. This result clarifies the best among the two in the latest IaaC tools. Even though there are not many differences in recovery time duration. Terraform outperforms Ansible in all other aspects like Good Community support for cloud infrastructure build, functionalities specific for DR management, Automated State comparison feature, and easy deletion process.

Even though ansible is not designed for Cloud infrastructure build, it has the capability of performing cloud management like terraform. It shows excellent results when compared to manual DR management in experiment 2. Choosing Ansible over terraform may not be a great decision, but still, it is an option for the cloud consumers who hold Ansible Enterprise license for configuration management.

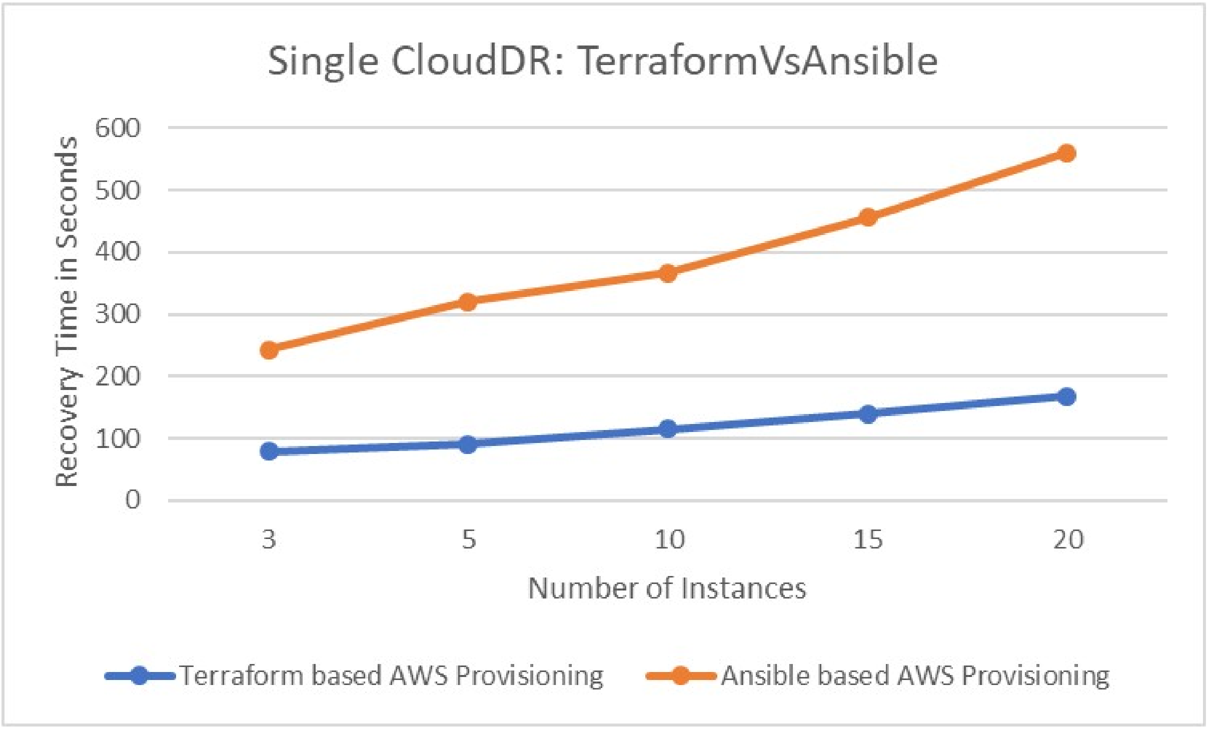


Figure 9: Terraform Vs Ansible DR Comparison

6.4 Experiment 4 / Comparing Multiple cloud Disaster recovery duration between Terraform based infrastructure build vs. Manual:

This experiment is carried out between two different cloud infrastructures, which logically and physically separated across different regions. We have identified AWS and GCP cloud infrastructures for this analysis. AWS as primary cloud resources where the designed web application is running in a secured VPC. Using terraform, GCP intended to replicate the same cloud infrastructure as same as AWS. The recovery process has been initiated by applying to destroy command in primary resources. The main difference from experiment

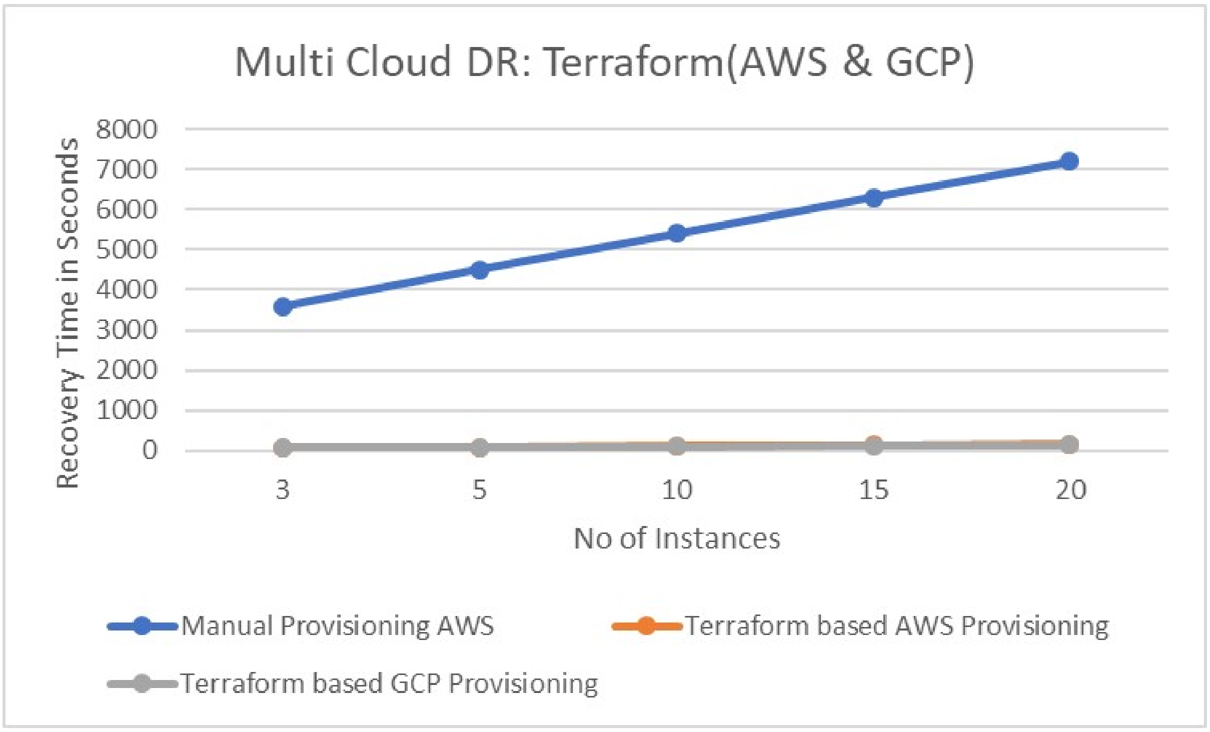


Figure 10: Terraform Vs Manual in Multi Cloud DR Comparison

1 is, here we are not going to compare the current state file with the expected state file. The Manual trigger will be initiated to bring the web application in GCP by running apply command in secondary cloud space.

For this evaluation, the complexity and dependencies are considered as same as experiment 1 and 2. The graph shows the relationship between the disaster recovery response time and the number of instances along with the dependencies. Figure 11 showcases all DR plan executed for the Multi-Cloud DR process using Manual recovery, Terraform based recovery in AWS, and GCP. The Disaster initiated by running destroy command to remove all cloud resources in AWS. After deleting all dependencies, the replica of Web application architecture, which builds on GCP, is launched by using apply command. The web application is back to operation in a fraction of seconds. The results displayed in figure 10.

6.5 Discussion

Overall, the difference between the disaster recovery response time performed with manual and automation. The results increase drastically with the increasing complexity of the architecture, here we have added more instances for multiple iterations to understand the variations. The number of cloud instances count ranges from 3 to 20. The results clearly show the advantages of using Automated tools-based disaster recovery using Teraform and Ansible in both single and multi-cloud infrastructure. The benefit of using an automated solution increases with increasing the number of instances. RTO and RPO calculations not made separately, but they aligned with the recovery time.

7 Conclusion and Future Work

In recent times, we have witnessed a massive increase in using an automated solution in ICT. Improvements in DevOps tools have created a lot of options in disaster recovery management. Terraform, and Ansible tools are proving to be popular automation tools for cloud disaster recovery. This paper has proposed Disaster recovery as a service by using the IaaC concept. We have studied the process of disaster recovery using traditional manual and automated solutions. The tools can extract the best results based on the IaaC concept. It was successful in overcoming the conventional manual method of DR management. The proposed plan provides quick recovery in minimum time. In this paper recovery response time is considered as the most crucial factor in finding the best option for the recovery operation. Our study evaluated the recovery time during the disaster in both manual and automated recovery using Terraform and Ansible. The results clearly show the benefits of using tool-based disaster recovery management over the manual procedure. The benefit depends on the complexity of the cloud infrastructure, and in our condition, it reached 500 times.

As future work, the entire IT stack, including Portal, CRM, Middleware, and Billing system, can be incorporated in the DR process to find the best possible solution for recovery. Also, other possible automation tools for disaster recovery can be tested as a further related research topic of this paper. There are many available solutions for taking data backup and data lossless recovery procedures to compare. Thus, in the future, automation tools have the potential of executing the quick DR and Data Backup with the best-expected results.

References

Alshammari, M. M., Alwan, A. A., Nordin, A. and Abualkishik, A. Z. (2018). Disaster recovery with minimum replica plan for reliability checking in multi-cloud, *Procedia computer science* **130**: 247–254.

Alshammari, M. M., Alwan, A. A., Nordin, A. and Al-Shaikhli, I. F. (2017). Disaster recovery in single-cloud and multi-cloud environments: Issues and challenges, *2017 4th IEEE International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, IEEE, pp. 1–7.

AlZain, M. A., Soh, B. and Pardede, E. (2013). A survey on data security issues in cloud computing: From single to multi-clouds, *Journal of Software* **8**(5): 1068–1078.

Bessani, A., Correia, M., Quaresma, B., André, F. and Sousa, P. (2013). Depsky: dependable and secure storage in a cloud-of-clouds, *ACM Transactions on Storage (TOS)* **9**(4): 12.

Cachin, C., Keidar, I., Shraer, A. et al. (2014). Trusting the cloud, *Acm Sigact News* **40**(2): 81–86.

Chang, D. W., Patra, A. and Bagepalli, N. A. (2019). System and method for scaling multiclouds in a hybrid cloud architecture. US Patent 10,462,072.

Documentation (2019a). Ansible documentation and architecture.
URL: *https://www.ansible.com/use-cases/configuration-management*

Documentation (2019b). Terraform usecases and architecture.
URL: *https://www.terraform.io/docs/index.html*

Gartner (2019). Gartner report-dr as a service.
URL: *https://www.gartner.com/doc/reprints?id=1-6SQE4Z2ct=190610st=sb*

Grolinger, K., Capretz, M. A., Mezghani, E. and Exposito, E. (2013). Knowledge as a service framework for disaster data management, *2013 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, IEEE, pp. 313–318.

Gu, Y., Wang, D. and Liu, C. (2014). Dr-cloud: Multi-cloud based disaster recovery service, *Tsinghua Science and Technology* **19**(1): 13–23.

Jim, C. (2019). Ibm managing hybrid cloud architecture.
URL: *https://www.ibm.com/blogs/cloud-computing/2019/02/12/new-ibm-services-multicloud-world/*

Khoshkholghi, M. A., Abdullah, A., Latip, R. and Subramaniam, S. (2014). Disaster recovery in cloud computing: A survey.

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

Masek, P., Stusek, M., Krejci, J., Zeman, K., Pokorny, J. and Kudlacek, M. (2018). Unleashing full potential of ansible framework: University labs administration, *2018 22nd Conference of Open Innovations Association (FRUCT)*, IEEE, pp. 144–150.

Orzechowski, M., Balis, B., Pawlik, K., Pawlik, M. and Malawski, M. (2018). Transparent deployment of scientific workflows across clouds-kubernetes approach, *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, IEEE, pp. 9–10.

Otey, M. (2019). Multi-cloud disaster recovery benefits and challenges.
URL: *https://www.petri.com/multi-cloud-disaster-recovery-benefits-and-challenges*

Pokharel, M., Lee, S. and Park, J. S. (2016). Disaster recovery for system architecture using cloud computing, *2016 10th IEEE/IPSJ International Symposium on Applications and the Internet*, IEEE, pp. 304–307.

Sahebjamnia, N., Torabi, S. A. and Mansouri, S. A. (2015). Integrated business continuity and disaster recovery planning: Towards organizational resilience, *European Journal of Operational Research* **242**(1): 261–273.

Saquib, Z., Tyagi, V., Bokare, S., Dongawe, S., Dwivedi, M. and Dwivedi, J. (2013). A new approach to disaster recovery as a service over cloud for database system, *2013 15th International Conference on Advanced Computing Technologies (ICACT)*, IEEE, pp. 1–6.

Singh, N. K., Thakur, S., Chaurasiya, H. and Nagdev, H. (2015). Automated provisioning of application in iaas cloud using ansible configuration management, *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*, IEEE, pp. 81–85.

Sun, A., Ji, T., Yue, Q. and Xiong, F. (2016). Iaas public cloud computing platform scheduling model and optimization analysis, *International Journal of Communications, Network and System Sciences* **4**(12): 803.

Togawa, S. and Kanenishi, K. (2013). Private cloud cooperation framework of e-learning environment for disaster recovery, *2013 IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, pp. 4104–4109.

Vukolic, M. (2015). The byzantine empire in the intercloud., *SIGACT News* **41**(3): 105–111.

Watson, E. B., Farhangi, A. and Iyer, K. C. (2014). Cross-cloud computing for capacity management and disaster recovery. US Patent 8,719,627.

Wood, T., Cecchet, E., Ramakrishnan, K. K., Shenoy, P. J., van der Merwe, J. E. and Venkataramani, A. (2016). Disaster recovery as a cloud service: economic benefits & deployment challenges., *HotCloud* **10**: 8–15.