Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Thesis for the Degree of Bachelor of Science in Engineering - Computer Network Engineering, 15 credits

ANSIBLE IN DIFFERENT CLOUD ENVIRONMENTS

Axel Witt
awt20002@student.mdu.se

Sebastian Westling
swg20003@student.mdu.se

Examiner: Hossein Fotouhi
Mälardalen University, Västerås, Sweden

Supervisor(s): Václav Struhár
Mälardalen University, Västerås, Sweden

Company Supervisor(s): Martin Mella
Curitiba AB, Stockholm, Sweden

08/06/2023

Abstract

Cloud computing offers higher reliability and lower up-front IT costs than traditional computing environments and is a great way to dynamically scale both resources and capabilities. For further efficiency and consistency, cloud computing tasks can also be automated with tools such as Ansible. In this thesis, we will analyze and compare the abilities of Ansible with the three leading cloud platforms, i.e. Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP). To evaluate this, we cover three different areas of automation with cloud platforms. These areas are performance, user complexity, and missing network functionalities. The performance was evaluated through experiments that revealed a big gap between the platforms, where AWS was the clear winner in all scenarios. Microsoft Azure was slightly faster than GCP when a low number of virtual machines were created but GCP scaled better than Azure when more virtual machines were created. The user complexity was evaluated on the setup process and the creation of playbooks where Azure was the clear winner in both areas. AWS and GCP had similar setup processes, but AWS takes second place through its superior documentation for the creation of playbooks. All three platforms had missing network functionalities, but the majority of the missing functionalities were not relevant as they were not related to deployment which is the main usage of an automation tool like Ansible. However, some deployment functions were missing, for example, the firewall function for AWS was missing in Ansible.

Contents

1. Introduction **1**

2. Background **2**
2.1 Infrastructure as Code (IaC) 2
2.2 Secure Shell (SSH) 2
2.3 Ansible 2
2.3.1 Inventory 3
2.3.2 Dynamic Inventory 3
2.3.3 Playbooks 3
2.4 Virtualization 3
2.5 Cloud Computing 4
2.6 Virtual Private Cloud 4

3. Related Work **5**

4. Problem Formulation **6**

5. Method **7**

6. Ethical and Societal Considerations **8**

7. Overview on Cloud platforms **9**
7.1 Microsoft Azure 9
7.2 Amazon Web Services (AWS) 9
7.3 Google Cloud Platform (GCP) 9

8. Experiment **10**
8.1 Execution of Measurements 11
8.1.1 Microsoft Azure 12
8.1.2 Amazon Web Services 12
8.1.3 Google Cloud Platform 13

9. Results **14**
9.1 Measurements of Configuration Time 14
9.2 Measurements of Transferred Data 17
9.3 Numerical results 21

10.Discussion **22**
10.1 Experimental Results 22
10.2 Overview of User Complexity 23
10.3 Overview of Network Features 24

11.Conclusions **25**

12.Future Work **26**

References **29**

Appendix A Appendix **30**

Appendix B Appendix **32**

Appendix C Appendix **34**

Appendix D Appendix **36**

Appendix E Appendix **38**

Appendix F Appendix **40**

Appendix G Appendix **42**

List of Figures

1 Example of Ansible architecture 3
2 Workflow Chart 7
3 Cloud Network Topology 10
4 Avarage configuration time in Microsoft Azure 14
5 Avarage configuration time in Amazon Web Services 15
6 Average configuration time in Google Cloud Platform 15
7 Average configuration time across all cloud platforms 16
8 Average amount of transferred data when configuring Microsoft Azure 17
9 Average amount of transferred data when configuring Amazon Web Services 18
10 Average amount of transferred data when configuring Google Cloud Platform . . . 18
11 Average amount of transferred data across all cloud platforms 19
12 Average amount of network bandwidth across all cloud platforms 20

List of Tables

1 Specifications for the Ansible controller in each cloud platform 11
2 Average configuration time, transferred data, and network bandwidth of Microsoft Azure 21
3 Average configuration time, transferred data, and network bandwidth of Amazon Web Services 21
4 Average configuration time, transferred data, and network bandwidth of Google Cloud Platform 21
5 Microsoft Azures Network functions 37
6 Amazon Web Services Network functions 39
7 Google Cloud Platform Network functions 41

1. Introduction

Cloud computing has a lot of uses, including virtual server hosting, database management, and much more. Public cloud services, one of the fastest growing technologies, is expected to grow 20.7% in 2023, an increase from 2022 which was forecast to grow by 18.8% [1]. Public cloud services have grown over the years but during the Covid-19 pandemic cloud services surged and grew faster than ever [2].

In traditional computing environments, resources are allocated with peak load requirements in mind which leads to over-provisioning and under-utilization [3]. It has been observed that the average utilization of application servers typically falls within the range of 5% to 20%, indicating that several resources such as CPU and RAM remain unused during non-peak periods. This can easily be solved with one of the most popular cloud computing features, virtualization, which provides a way to dynamically scale both resources and capabilities according to one's application's needs. Cloud computing also offers higher reliability and lower up-front IT costs.

Ansible is a leading Infrastructure as code (IaC) tool for managing infrastructure. With Ansible tasks can be automated, tasks such as updating software, starting a virtual machine, and configuring a network device. With IaC multiple manual configurations can be reduced to a click and execution of a script [4]. With an automation tool such as Ansible, an employee does not need all access to a cloud service web interface but could instead execute a prepared script that does the job [5]. NASA for example lowered their site update of nasa.gov from 1 hour to less than 5 minutes [6] with the use of Ansible. Ansible lowers the update time because multiple servers can be updated at the same time through one Ansible controller. Network infrastructure is an ever-evolving area that entails continuous change, such as a new database requiring a new feature or addressing vulnerabilities across a dozen computers that needs patching [7]. With automation, this type of work can be reduced from hours to minutes.

This thesis aims to thoroughly assess and compare the capabilities of automation with Ansible paired with the three largest cloud platforms, Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP). We will assess the capabilities of the three different cloud platforms with Ansible in three different areas. Performance, user complexity, and missing network functionalities. The performance will be measured in an experiment through the configuration time and amount of transferred data. User complexity will be evaluated through the setup process and creation of playbooks and missing network functionalities will be evaluated by comparing cloud platform network functionalities with their Ansible counterpart. Due to many companies migrating into the cloud, automating their cloud resources would save both time and money.

2. Background

In the following section, the concepts and technologies used in this thesis will be explained to give the reader the basic understanding necessary to follow the thesis.

2.1 Infrastructure as Code (IaC)

IaC is best described as using code to automate infrastructure, improving the scalability, security, and resiliency [8]. IaC is a key tool for network engineers due to a consistent end result in the environment [9]. Instead of manually configuring multiple devices which could lead to inconsistencies, with IaC these devices will all be configured the same as long as it uses the same configurations in an IaC tool. Popular IaC tools include *Ansible*, *Chef*, *Puppet*, and *Terraform* [10]. IaC works consistently and repeatedly to change systems and configurations [7]. IaC is very close to software development with it being seen more as programming code than network configuration. This means that other users can easily see what the code does to a system, and if they want they can use it for their system too, and expect the same result as the original creators' system.

There are two methods for utilizing IaC, agent-based and agentless configuration [11]. Agent-based configurations have their managed system install software as unnwell as the controller device. An agent-based managed system often uses a pull method, which is initialized by the managed systems requesting configuration from the controller device. With an agentless configuration, only the controller device needs the software for the IaC tool to function and instead relies on protocols such as SSH to forward the configuration through the push method. The push method works by the controller device pushing configuration instead of the managed servers requesting it. There are also instances where both the push and pull method is used.

2.2 Secure Shell (SSH)

SSH is a network protocol that gives users a secure and encrypted way to access remote computers over insecure networks [12]. Using SSH provides strong password authentication and authorization with a public and private key pair. SSH is used in Ansible through the connectivity tool OpenSSH.

2.3 Ansible

Ansible is an open-source automation tool written in Python, which was released officially to the public in 2012 [13], [14] and is as of now in version 2.14 which was released in November 2022. Ansible is used to configure systems, install, and update software, and other related tasks [13]. Ansible is a hybrid of imperative and declarative language [15]. In an imperative language, everything goes in the order it was given. A declarative language, on the other hand, means the configuration given will eventually lead to a reliable result with correct restrictions and requirements. In Ansible, you define the state desired for a system or infrastructure by writing playbooks[16]. The playbooks contain tasks that describe the intended configuration for a target system. Rather than listing the exact steps to achieve an intended state, Ansible specifies what should be achieved. Ansible uses the imperative language model for actions inside the playbooks, for example with performing conditional checks or executing specific commands.

Ansible uses an agentless-based configuration, the only software required is on the controller device. It uses OpenSSH as a transport method, all that is needed to communicate with end devices is an SSH key pair. To automate end devices, Ansible uses playbooks and inventories which are written in YAML. YAML is a human-readable data serialization language that is easy to read and write for humans and it is the language used to create playbooks and inventories with Ansible [17]. An example of an Ansible architecture can be seen in figure 1.

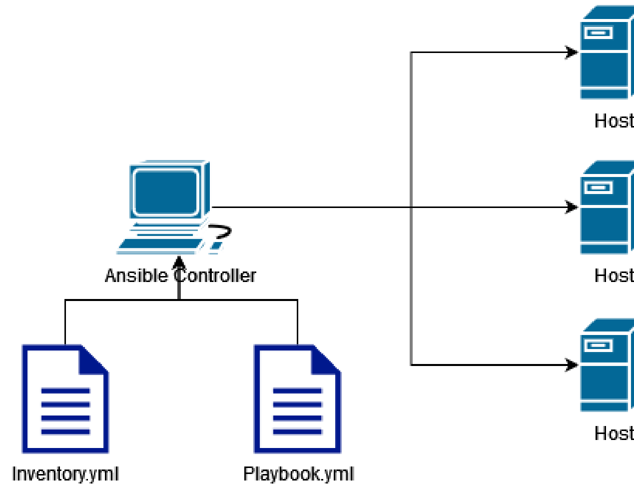

Figure 1: Example of Ansible architecture

Ansible as an automation tool is very lenient when it comes to memory and CPU usage [18], only needing more of both when having multiple forks. A fork is what Ansible uses to decide the maximum simultaneous amount of end devices for each task [19].

2.3.1 Inventory

An inventory is used to categorize the hosts that are managed by Ansible, and an inventory is also used to automate multiple hosts at the same time [20]. These hosts can be categorized by the operating system, application, geographical location, and more. Ansible inventories have several useful features like the parent/child feature which can be used to simplify the management of hosts that are in the same category. Another useful feature is the range feature which is used to define a range of hosts with one block of code instead of manually writing each separately.

2.3.2 Dynamic Inventory

When using a cloud platform, new virtual machines get created and deleted multiple times. Therefore, using a static inventory becomes tedious and Ansible loses its purpose [21]. A dynamic inventory is an inventory that collects the data of the inventory from an already existing source, like a cloud platform [22, pp. 12-15]. Ansible uses dynamic inventories as executables that get called when Ansible runs to collect inventory data in real time. Scripts can be created by anyone to collect data on the inventory, but Ansible also has multiple inventory plugins that can gather the resources.
The playbooks used during the experiments will also create new virtual machines meaning they would have to be added manually to a static inventory. With a dynamic inventory, we will not have to manually add virtual machines and instead, Ansible will read from our cloud platform.

2.3.3 Playbooks

Ansible playbooks offer a simple configuration management and multi-machine deployment system which can be used to deploy both simple and complex applications [23]. A playbook executes instructions for the agents specified in the inventory. As mentioned earlier, these playbooks are written in the language Yaml which makes it easy to understand the syntax and start configuring files.

2.4 Virtualization

Virtualization is an essential concept for this thesis since it is the foundation on which cloud computing is built [24]. Virtualization uses the hardware of a single computer and divides it into

multiple virtual computers, often called virtual machines. The use of virtual machines allows for multiple operating systems to run on a single computer. This can be done through a hypervisor which is software that creates and runs virtual machines [25]. For our thesis, we use virtual machines inside the different cloud platforms as Ansible controllers. Virtualization allows for more efficient utilization of the hardware of a physical computer.

2.5 Cloud Computing

The term cloud computing refers to a model for delivering IT services over the internet [26]. With cloud computing users can access a pool of computing resources such as networks, servers, and more. Users can access these on a pay-per-use basis which means that the price of these services will vary depending on how many resources are created and used. These services typically include computing power, data storage, or applications over the internet. Multiple models are used to define the different types of services that are offered [27]. Infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). IaaS solutions replace the need of having to invest in IT infrastructure components, instead, these resources are available through the cloud. These resources include computing power, networks, virtualization, etc. This solution can lead to lower costs and more flexibility, but the customers are still fully responsible for their applications. PaaS solutions offer development tools and services to help create and deploy applications through the cloud. PaaS builds upon IaaS and still includes the basic infrastructure mentioned earlier but also adds the development tools. SaaS solutions are where cloud platforms deliver ready-to-use applications while also being responsible for maintaining and managing them, including maintenance, updates, scaling, and delivery.

2.6 Virtual Private Cloud

Virtual private cloud (VPC) is the concept of isolating networks on cloud platforms [28]. With a VPC, customers can securely share resources between the devices on the cloud. To explain it simply, a VPC is a private network inside the cloud.

3. Related Work

Dutta et al. [29] compare the three largest cloud platforms according to market share. They compared the following six categories of cloud computing: compute services, storage services, database services, networking services, management services, and security services. Instead of elaborating further regarding these individual comparisons, they finish the study by showing a list of the key benefits and drawbacks of each cloud platform. The conclusion of this study is that there is no clear winner when it comes to cloud platforms and that you should use whatever best satisfies your requirements. Their recommendation is to distribute risk by using multiple vendors as cloud platforms and utilizing the best features from each platform. This study focuses on comparing the features between the three largest cloud platforms while our thesis aims to specifically evaluate the configuration and compatibility of these features using the automation tool Ansible.

Santoso et al. [30] compare the time it takes to configure a server infrastructure manually and with Ansible on the cloud platform Alibaba Cloud. The main goal of this study is to prove that using the Ansible framework is superior to configuring numerous servers manually. In their work, they performed the experiment with the same topology every time and instead changed the number of CPU cores and the size of the RAM for each test. After executing each test, both with Ansible and the traditional way, an average setup time was presented. It took an average of 27 minutes to manually configure the server infrastructure on the Alibaba Cloud, while Ansible could configure the same server infrastructure with an average time of 3 minutes and 30 seconds. This study shows the power of using Ansible for cloud configuration compared to manual configuration. Our thesis focuses solely on automated configurations with Ansible and instead wants to compare the difference in configuration time between different cloud platforms. We are also performing the experiments with the same amount of vCPUs on the Ansible controllers across all experiments and are instead changing up the configuration by changing the number of virtual machines for each test.

Wågbrant and Radic [11] compare the automation tools Ansible, Saltstack, and Puppet bolt in their thesis "Automated Network Configuration: A Comparison Between Ansible, Puppet, and SaltStack for Network Configuration". The goal of the thesis was to make a comprehensive overview of each tool and how they compare, included is their capabilities and documentation. The thesis also executes multiple experiments to compare configuration time and transferred data with each tool. The thesis experiments focus on router configuration, both virtual and physical. Their thesis focuses on comparing different automation tools' capabilities and their configuration on routers, while our thesis aims to use only one automation tool, Ansible, and instead compare its integration on different cloud platforms.

4. Problem Formulation

As mentioned in the introduction, cloud computing is growing at a steady pace. In this thesis, we aim to find out how Ansible compares in terms of *performance*, *user complexity*, and *network features* using different cloud platforms (Microsoft Azure, Amazon Web Services, Google Cloud Platform). Because of both network automation and cloud platforms growing popularity, companies might want to know which cloud platform is the most suitable for their company and what works with what platform.

RQ1. What is the difference in performance when using Ansible with the different cloud platforms?

RQ2. What is the difference in user complexity when configuring Ansible with the different cloud platforms?

RQ3. What network cloud features cannot be performed with Ansible?

The different cloud platforms we want to compare have very wide use cases, AWS and GCP for example have different Artificial Intelligence (AI) and Machine learning features available on their platform. In this thesis the computer networking perspective is utilized, therefore the primary focus of this thesis is the network features. We want to focus on network features, more specifically, virtual private clouds, virtual machines, security groups/firewalls, and load balancers. We will also only focus on the official Ansible modules for the cloud platforms even though there exists a lot of community modules since the community modules are less secure and more prone to bugs.

5. Method

The first method that we will be using during this thesis is a literature overview to get an in-depth understanding of the different software that will be used and evaluated. The literature overview will mostly consist of reading and analyzing official documentation from the software developers to get an idea of which functionalities we want to compare between the different cloud platforms for performance evaluation. We also explored literature found in Google Scholar and SCOPUS. This method will also be used to answer our second question about missing functionalities in Ansible by comparing the documentation of the different cloud platforms with the documentation of Ansible. The literature overview will also be used to identify previous work in the field which is relevant and related to our thesis.

During the literature overview, we started planning the experiment [31], the second method used in this thesis. For us to make a fair assessment of the differences in performance between the cloud platforms, we will be executing the same tasks on all three cloud platforms and the tasks will also be configured as equally as possible. Since these tests are performed on the cloud, the different regions and locations can affect the test results. To counteract this, we will be hosting Ansible internally in the cloud on either the built-in cloud shell or a virtual machine inside the cloud. This should minimize the latency that can arise from connecting to the cloud remotely. The measurements used to evaluate the performance are the amount of time passed and the amount of data transferred during the configuration of the tasks. These measurements can be used to figure out the amount of data transferred per second and the network usage. When the experiment is completed, the results will be analyzed to determine if they are relevant to answer the questions raised in our thesis. If this is not the case, the experiment will be altered and redone. A visual representation of our workflow can be seen in Figure 2.

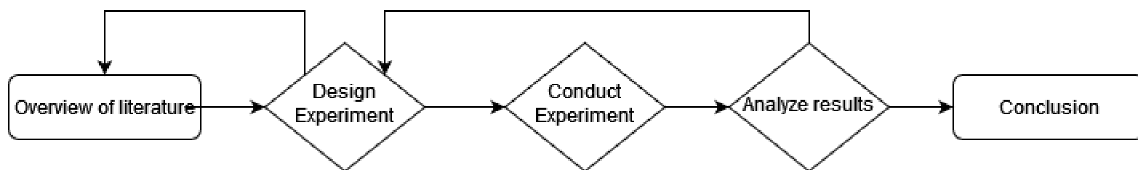

Figure 2: Workflow Chart

6. Ethical and Societal Considerations

Our thesis does not entail any ethical stances as it is carried out in a test environment where no sensitive information is being used. All passwords being used are unique for this thesis and will not be repeated outside it, and all the IP-addresses used are located in the private address space.

As for the societal considerations, in many industries people have a growing fear of losing their jobs to automation tools and software which can be used to replace human manual labor as the technology continues to evolve. This fear also applies to the network industry where people fear that the use of automation software will remove the need for engineers to do their regular daily tasks. This may be true, but it does not mean that all these engineers will be out of jobs. There will still be a need for engineers to configure, oversee and update these automation softwares.

7. Overview on Cloud platforms

A cloud platform can be described as an IT company that provides on-demand scalable computing resources. Different cloud platforms use other pricing models, but the most common is a pay-as-go model where you pay for the services used by the hour or minute. Cloud platforms have 4 different deployment models, private, public, community, and hybrid cloud [32]. The computing technology is in big data centers and all cloud platforms chosen for this thesis also provide IaaS, PaaS, and SaaS.

7.1 Microsoft Azure

Microsoft launched its version of a cloud computing platform in 2010 and is the second largest cloud provider according to the market where Azure controls 23% of the cloud infrastructure market as of the fourth quarter of 2022 [33]. According to their website, the Azure cloud currently as of 2023 spans across 60 geographic regions and 140 countries with a minimum of three separate availability zones in all availability zone-enabled regions [34], [35]. An availability zone is a logical data center in a region.

Microsoft Azure uses their Azure virtual machines as part of their IaaS. With the service, you are able to emulate processors, memory storage, and network resources. The service is able to use both Windows and Linux [36]. Virtual machines in Azure charge hourly, unlike the other cloud platforms.

7.2 Amazon Web Services (AWS)

AWS launched its popular on-demand cloud computing platform in 2006 and is currently the largest cloud provider according to the market where AWS controls 32% of the entire cloud infrastructure market as of the fourth quarter of 2022 [33]. The AWS cloud currently as of 2023 spans 99 availability zones in 31 geographic regions around the world according to their website, with 15 availability zones and 5 regions more on the way [37].

AWS uses their Elastic Compute Cloud (EC2) for all their virtual machines [29]. With EC2 AWS achieves a scalable capacity for their computing cloud. Launching multiple virtual machines, managing security and network configurations, as well as handling storage become effortless with EC2 [38]. Notable features of EC2 are their computing environments, elastic IP addresses (static IPv4 addresses), and virtual networks (VPCs). EC2 uses a pay-per-usage, where the price is set for a minimum of one minute and after that, it is based on a per-second basis.

7.3 Google Cloud Platform (GCP)

Google launched GCP in 2008, their alternative to a cloud computing platform. It is the third largest cloud provider in the world according to the market where GCP controls 10% of the cloud infrastructure market as of the fourth quarter of 2022 [33]. According to their website, as of 2023, GCP is available in 35 different regions with 106 accompanying availability zones and more regions and zones on the way [39].

To emulate servers, GCP uses their Google Compute Engine which is an IaaS [40]. Google Compute Engine provides among other things the use of virtual machines, DNS servers, and load balancers. The compute engine works as a pay-per-usage, setting the price on how many seconds the service is used and has a minimum of one minute. The compute engine also allows for custom compute types, meaning it can be customized for a customer to their needs.

8. Experiment

In this section, we explain the settings of the experiment to evaluate the performance of different cloud platforms. We discuss how we design the topology and why we chose this specific topology. The details on how the experiment is conducted and how everything used in the experiment is configured is explained in Section 8.1. In each cloud platform, a cloud shell is used or a virtual machine is configured to act as the Ansible controller device. The specifications for the device are listed in Table 1.

In the experiment, the performance of Ansible is assessed by automatically creating a predefined topology with playbooks in Azure, AWS, and GCP cloud platforms. During the creation of the topology, both the amount of data sent and the time it took to create the topology will be measured. The topology we chose to create is depicted in Figure 3, it is a virtual private cloud, and inside that network, there is a subnet. The VPC also has a security group or firewall that prevents traffic from going in or out. Pointed towards the subnet that is created there is a load balancer that is listening on port 80, the load balancer also does a health check every second. Lastly, there is a number of virtual machines that are created all having 2 vCPU and 4 Gigabytes of memory. The virtual machines are assigned to the subnet created earlier in the playbook.

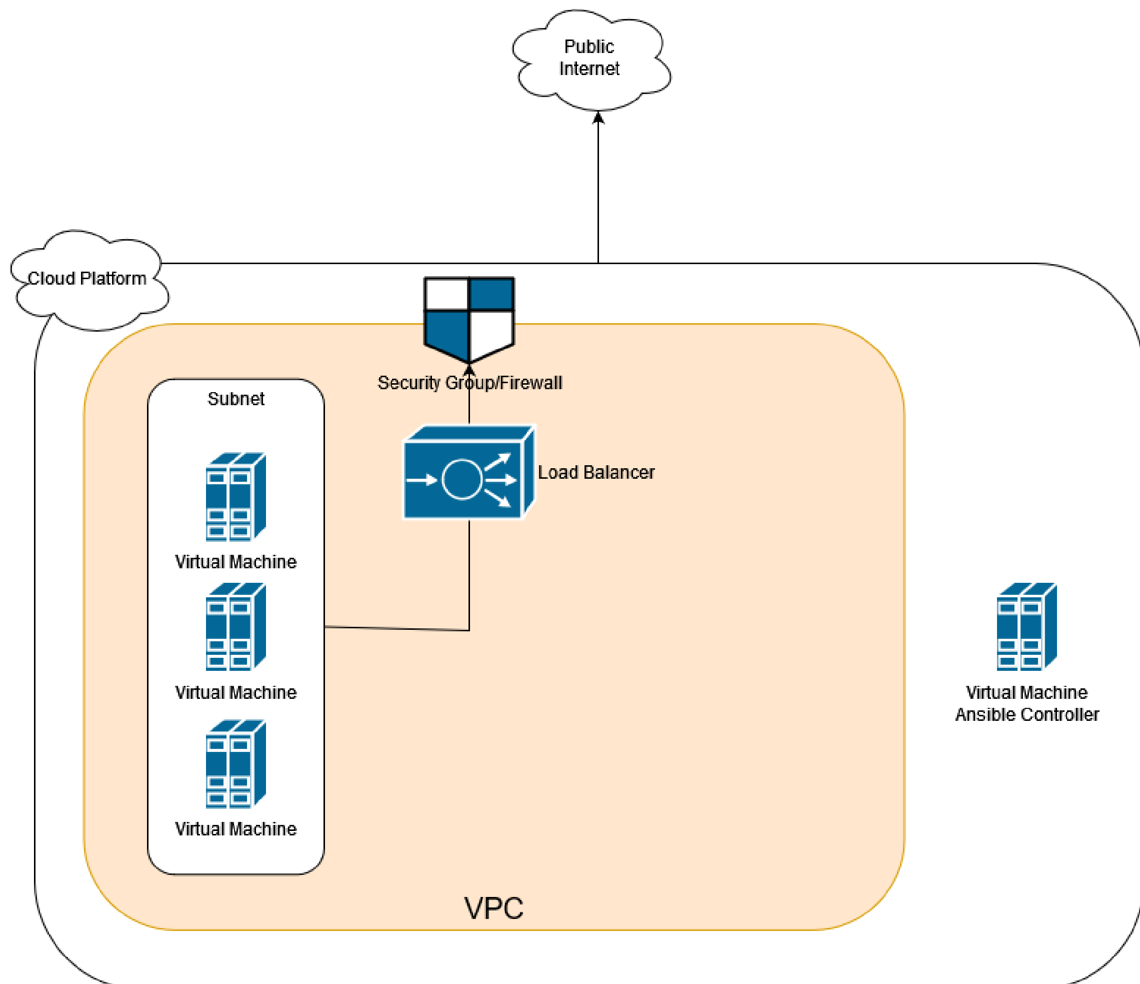

Figure 3: Cloud Network Topology

The topology to evaluate the performance of Ansible is a fixed network containing a VPC, subnet, security group/firewall, and a load balancer. We chose this topology as our baseline because it provides a good foundation, containing all essential network elements present in contemporary networks. These devices were consistently deployed for every test performed. The only variable that was not consistent throughout the tests was the number of virtual machines created. The reason for changing the number of virtual machines across the different tests is that they are one of the most used services on cloud platforms and it is common to create multiple virtual machines with different purposes [1][2][3]. The virtual machines also tested how scalable each cloud platform is with an increasing amount of deployed devices.

Platform	Azure	AWS	GCP
vCPU(s)	2	2	2
RAM	4	1	8
Virtual machine/Cloud shell	Cloud shell	Virtual machine	Virtual machine
Linux distro	Ubuntu(Azures version)	Ubuntu	Ubuntu
Ansible version	2.14.3	2.14.3	2.14.3

Table 1: Specifications for the Ansible controller in each cloud platform

8.1 Execution of Measurements

When evaluating the performance of deployments, we looked at two different measurements. The duration between executing the playbook and finishing the infrastructure change, and the amount of data transferred during this duration. To execute the experiments we created playbooks which can be seen in Appendix A, B, and C. The measurements will focus on the creation of the network, the playbooks created for deleting the network are used to allow us to save time by not deleting the network manually. A Python script has also been created that will automate the creation and deletion of playbooks 20 times. This was done so we can get multiple of the same tests without having to manually start the playbooks repeatably. The Python script can be seen in appendix G. The script creates a variable containing the amount of data transferred before the playbook is started with the network interface in the */proc/net/dev* file. Then the playbook for creating the topology is started, the command used also uses the command grep to grab the string containing the playbooks' runtime in seconds. After this, the variable records how much data were transferred in Bytes after the playbook was completed. The variable that contains the amount of data transferred before we ran the playbook subtracts from the amount of data transferred after it has been completed. This results in the total amount of data transferred during the playbook runtime. All information that is saved in variables is sent into a text file which is used for later analysis.

The task order in Ansible in each playbook is as follows:

- Create a virtual private cloud network
- Create a subnet inside the VPC
- Create a security group/firewall
- Create a load balancer with health checks
- Start Virtual machine(s)

In the following section, the installation of Ansible in each cloud platform will be explained along with differences in modules in Ansible, because of different requirements and existing modules in Ansible, the playbooks differ.

[1]https://intellipaat.com/blog/top-azure-services/
[2]https://mindmajix.com/top-aws-services
[3]https://www.testpreptraining.com/blog/what-are-the-top-google-cloud-platform-services/

8.1.1 Microsoft Azure

In Azure, instead of using a virtual machine as the Ansible controller, we used the built-in Azure cloud shell. The Azure cloud shell already has Ansible pre-installed, making the initial installation and configuration for Ansible minimal on Azure. The only thing we altered is the *ansible.cfg* configuration file where we added *callbacks_ enabled = timer.* This line will configure Ansible to return the amount of time passed to execute the playbook. This configuration setting is also configured on AWS and GCP. This configuration file does not exist in the pre-installed Ansible on Azure so we created it manually.

```
[defaults]

callbacks_enabled=timer
```

It is possible to create a default configuration file containing all configuration options with the command *ansible-config init –disabled -t all > ansible.cfg,* but since we only want to change one setting we decided to create a blank configuration file. Normally the first step in an Ansible deployment is to create the hosts file which specifies individual or groups of targeted hosts which is used by Ansible to establish a connection. Since the playbooks will execute internally from within the cloud, a file to specify the hosts is not necessary, instead, we specify localhost as the target host within the playbook itself. The next step was to create a playbook that will execute the planned tasks. See Appendix A for the full playbook with descriptions of the configured tasks.

The main difference while configuring playbooks for Azure compared to AWS and GCP is that all resources must belong to a resource group which is why the resource group is created first, all the following resources are configured to belong to that created resource group. The resource group is also where the location of the resources is specified instead of manually specifying each individual resource. We decided to use the location Sweden Central since that is closest to our current location.

8.1.2 Amazon Web Services

For Ansible to be able to work with AWS a few prerequisites needed to be installed to work. AWS Virtual machines do not have Ansible pre-installed but it can be downloaded and installed through the following commands:

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install software-properties-common
$ sudo apt-add-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible
```

As with Azure, *callbacks_ enabled = timer* was also configured in the file *ansible.cfg* of the virtual machine. Our virtual machine needs to have Boto3 and Botocore installed, done through the command: *pip install boto3 botocore.* Boto3 and Botocore are Python SDKs (Software Development Kit) containing the tools necessary to create, configure and manage AWS infrastructure. Another thing Ansible requires for it to work with AWS is an access key and a secret key, which can be downloaded from the AWS console. The access key and secret key can then be implemented in a separate file or Ansible vault (which encrypts the keys) to then be called in the playbook. The keys can also be specified in the playbook, but it is less secure and not recommended.

The choice for the region in AWS was EU-NORTH-1 which is located in Sweden. AWS has a cloud shell like all the other cloud platforms, but it is region dependent and some regions are missing it, including Sweden as of the writing of this thesis. Another reason for the choice is that it uses Amazon Linux 2, which we are not familiar with. To solve this issue a virtual machine was instead created to act as the Ansible controller. This should not impact internet speed due to it still being in the region EU-NORTH-1. The virtual machine uses the instance[4] type t3.micro and an installation of Ubuntu 22.04. To connect to the virtual machine, SSH is used from a personal laptop.

[4]Virtual machines and instances are interchangeable, though in the cloud virtual machines are often called instances. We have decided to use the word virtual machine in our thesis due to it being more commonly known.

With AWS, the playbook only required one additional module beyond the planned modules to make it functional. The requirement for the security group module called *ec2_security_group* was for the virtual private cloud to have an internet gateway which is done with the *ec2_vpc_igw* module. This module enables the managing of internet gateways, which means it can be used to create, delete and manage an internet gateway.

Something that AWS has that the other cloud platforms do not have is a count option for creating virtual machines in its *ec2_instance* module. This means that a loop does not manually have to be written in the playbook for Ansible to create multiples of the same virtual machine. This might lead to a faster configuration because Ansible will not have to loop through the creation of virtual machines and can instead create all at the same time.

8.1.3 Google Cloud Platform

GCP has a built-in cloud shell that is available to run playbooks, but because this cloud shell uses 4 vCPU's it was decided that a virtual machine needed to be created similar to AWS to have similar specifications. The region chosen for the GCP was Europe-North1, located in Finland. To be able to use this virtual machine with Ansible in the GCP environment we also need to install requests and google-auth using *pip install requests google-auth*. This virtual machine also does not have Ansible installed, meaning we had to download and install it. Similar to AWS, downloading and installing Ansible is done through the following commands:

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install software-properties-common
$ sudo apt-add-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible
```

As with Azure, *callbacks_enabled = timer* was also configured in the file *ansible.cfg* of the virtual machine. GCP has a few extra modules to make our original configuration work as intended. Google's playbook needed a separate module for a health check that could then be linked to the module for creating the load balancer. Another thing that the Google playbook needed was an *instance group*. Something that differentiates the GCP from the other cloud platforms is that the GCP needs an *instance group* to be created for the load balancer to send traffic. In a realistic topology the *instance group* would contain a group of virtual machines that would act as servers for a website, the load balancer would then decide which server would receive data.

Unlike the other cloud platforms, the GCP does not have a way to create security groups in Ansible. Therefore it was chosen that the GCP would create a firewall for its topology with the same implementation as the security groups from the other platforms.

9. Results

The results collected during the experiment will be displayed in this section. The results will be presented with graphs and tables which show the configuration times, the amount of transferred data, and the megabytes per second for each cloud platform. Each cloud platform will have independent graphs and tables for configuration times and the amount of transferred data. Graphs will also be displayed containing all cloud platforms for configuration time, transferred data, and megabytes per second. To not misrepresent the results, both independent results were shown along with a graph containing all cloud platforms to get a bigger picture of all cloud platforms together.

The experiments topology that was created added a different amount of virtual machines to test the scalability of Ansible with each cloud platform. Something to note is that the x-label shows zero virtual machines, which means that it is only the baseline of configurations (as explained in section 8. the baseline is the VPC, subnet, security group/firewall, and load balancer with health checks). To ensure fair results each experiment was performed 20 times and from that an average was calculated. The reason for testing with a maximum of 5 virtual machines is because of the limits from Azure and GCP. Azure only allowed a maximum of 10 vCPUs being active at the same time which meant that we could only start 5 virtual machines at most. For GCP, there was another problem. They started throttling our scripts when we started and deleted too many devices in a short amount of time.

9.1 Measurements of Configuration Time

This section presents graphs that show our results from measuring the configuration time during the experiment. Configuration time is an important measuring unit that gives a good indicator of differences in the performance of Ansible between the different cloud platforms.

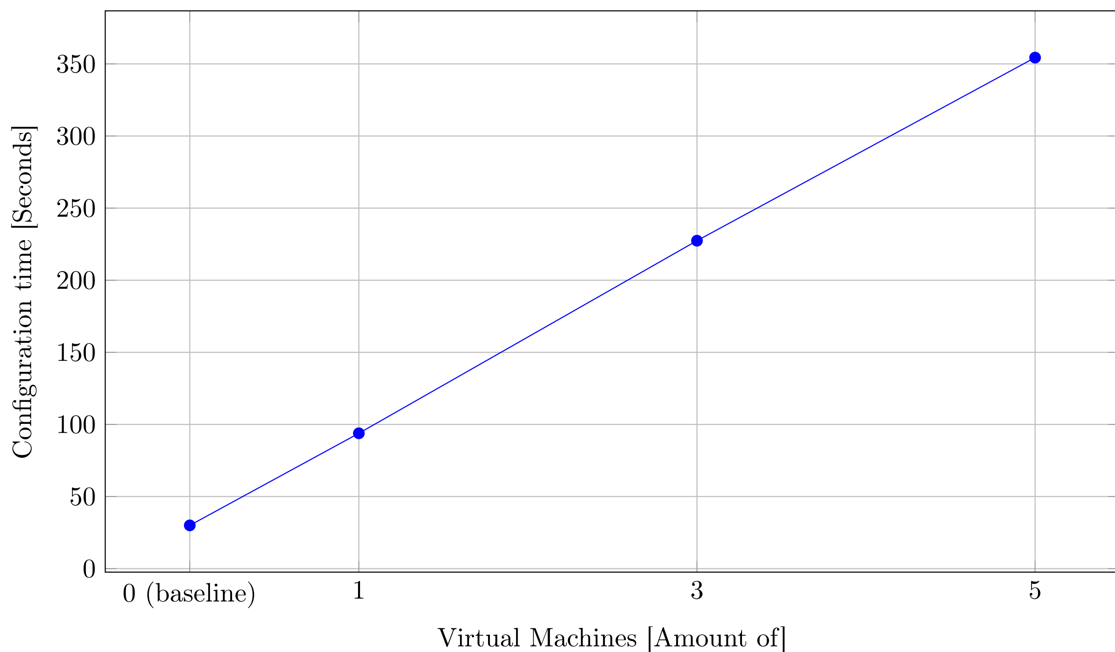

Figure 4: Avarage configuration time in Microsoft Azure

It can be observed in figure 4 that the average configuration time with Azures' playbook is very linear. While the configuration time without any virtual machines only takes 30 seconds the configuration time increases to 93.85 seconds at 1 virtual machine.

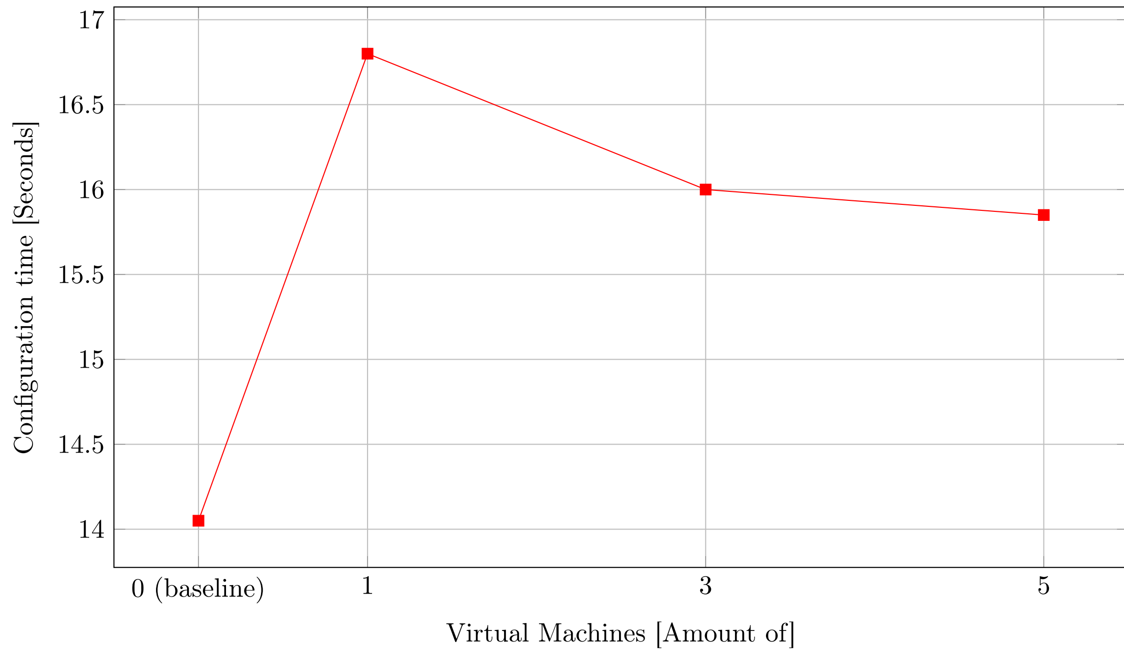

Figure 5: Avarage configuration time in Amazon Web Services

As seen in figure 5, AWS increases in configuration time the most between the baseline configuration and 1 virtual machine. It then drops at 3 and 5 virtual machines with 5 virtual machines having the second lowest configuration time. A logical reason AWS does not increase as much as the others are the *count* option available in the *ec2_ instance* module. Unlike other cloud platforms that need to loop each time to obtain the same virtual machine, AWS eliminates the need for looping and instead determines the appropriate number of virtual machines to create per task with the count option.

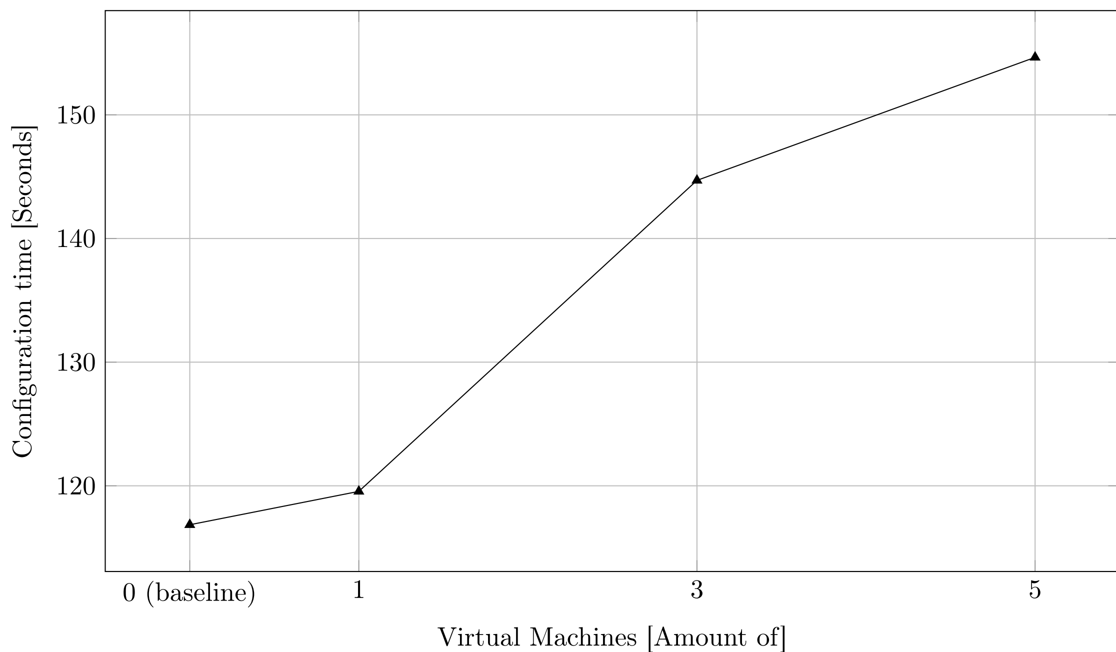

Figure 6: Average configuration time in Google Cloud Platform

The results from GCP shown in figure 6 show a slow increase between 0 and 1 virtual machines in configuration time. The largest increase can be seen between 1 and 3 virtual machines, although the configuration time between 3 and 5 virtual machines shows an increase, it is not as steep as between 1 and 3 virtual machines.

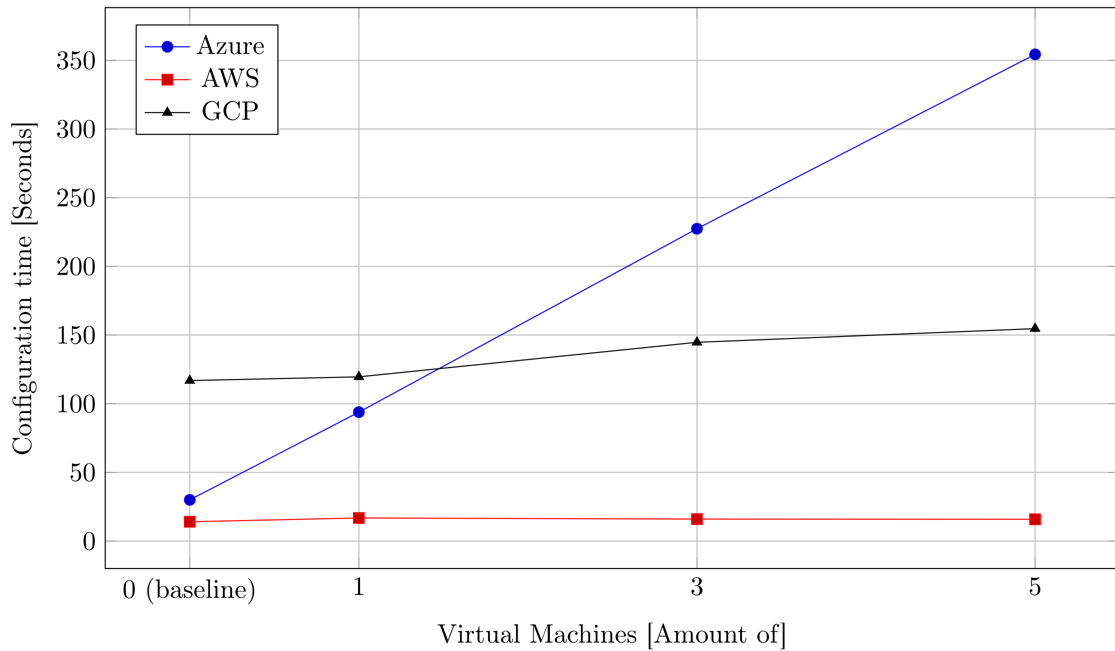

Figure 7: Average configuration time across all cloud platforms

Looking at the results in figure 7 from all cloud platforms it can be observed how they differ from each other in performance, AWS is the fastest with a very consistent graph that does not seem bothered by an increase of virtual machines. Azure on the other hand seems to perform worse the more virtual machines it has, with a high incline. Though Azure is the closest to linear it still is not linear, which might be because the sample size fluctuated. Like AWS, GCP also has a consistent graph that increases a bit but it seems like the initial configuration takes more time than any of the other cloud providers.

9.2 Measurements of Transferred Data

In this section, we present figures that show the amount of transferred data during the experiment. The amount of transferred data is measured for two reasons. One reason is that the amount of data being transferred gives a good indication of how much data is needed for communicating tasks with Ansible to cloud platforms. When we then compare the cloud platforms we get an idea if one cloud platform sends more than the others, it might indicate that the others have more optimized messages. The second reason is that with both the configuration time and the amount of transferred data, we can calculate the bandwidth of Ansible with the different cloud platforms.

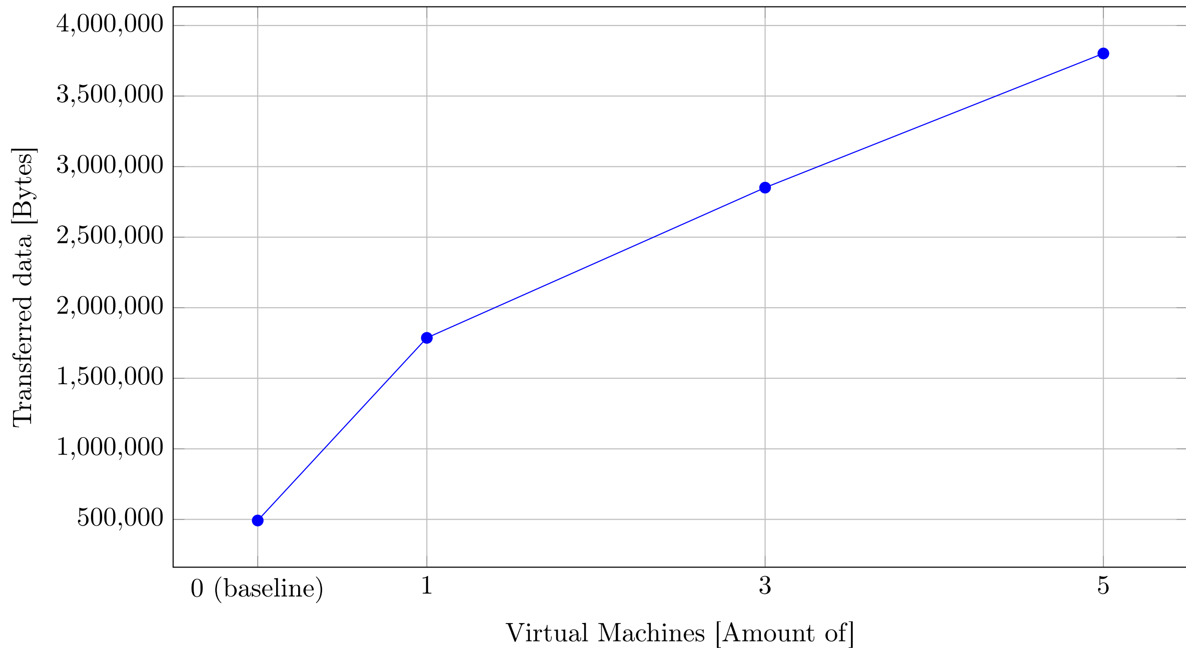

Figure 8: Average amount of transferred data when configuring Microsoft Azure

Figure 8 illustrates the average amount of data being transferred when configuring our topology on Azure. As seen on the graph, when more virtual machines are being created, there is also a larger amount of data being transferred from the Ansible host. Another takeaway as seen from the graph is that the creation of virtual machines is responsible for the most amount of data being transferred. Creating the topology with 0 virtual machines sends an average of 500.000 bytes, but adding just 1 virtual machine added more than 1 million bytes being sent.

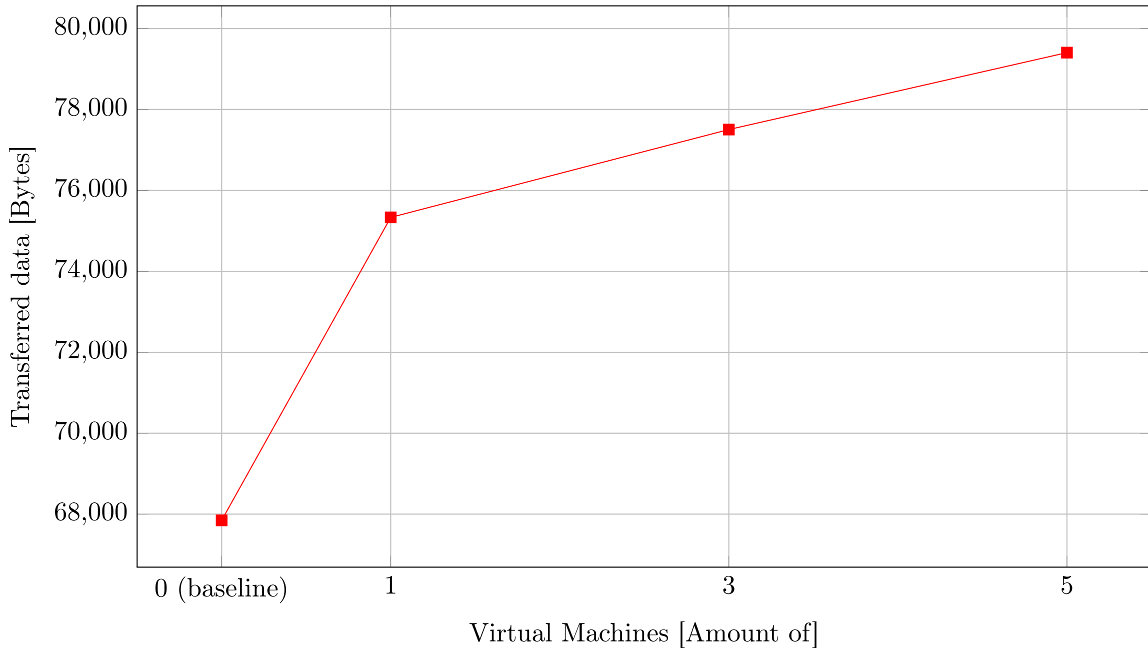

Figure 9: Average amount of transferred data when configuring Amazon Web Services

Figure 9 illustrates the average amount of data being transferred when configuring our topology on AWS. Compared to the Azure measurements, the AWS topology transferred the most amount of data during the initial configuration of the VPC, subnet, security group, and load balancer. As seen from the graph, the virtual machines being created are not adding much to the total amount of data being transferred.

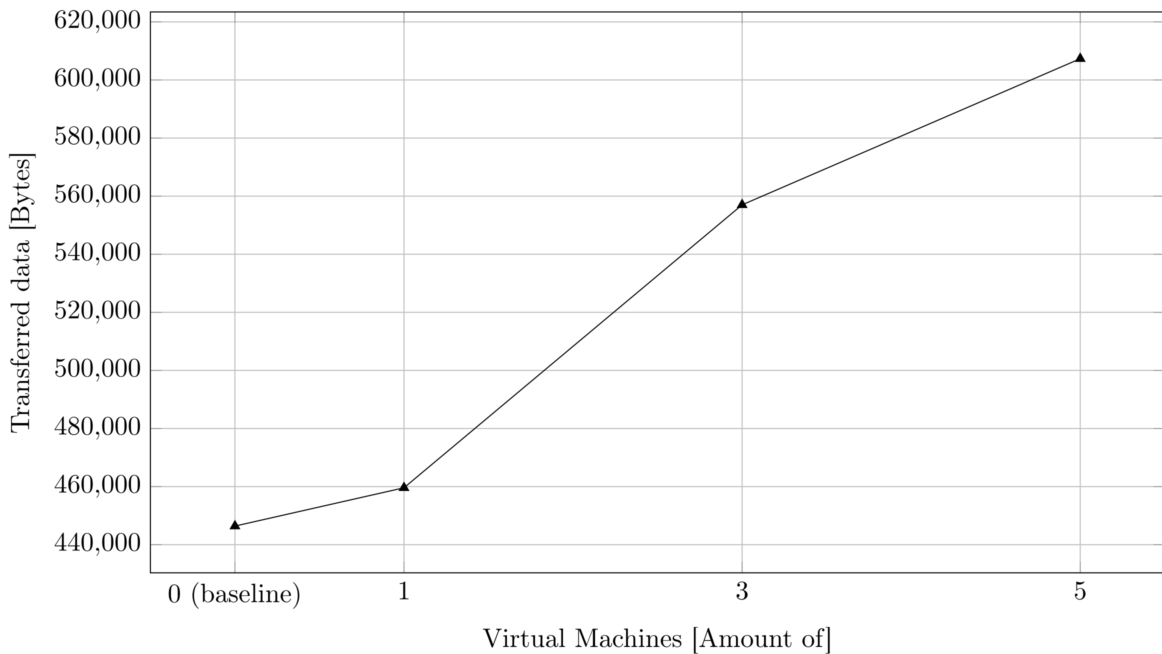

Figure 10: Average amount of transferred data when configuring Google Cloud Platform

Figure 10 illustrates the average amount of data being transferred when configuring our topology on GCP. GCP behaves more like AWS than Azure, where the most amount of data being transferred is during the initial configuration of the VPC, subnet, firewall, and load balancer. The measurements show that creating 1 virtual machine barely increased the data being transferred at all. However, the creation of 3 and 5 virtual machines showed a substantial increase in the amount of data being transferred.

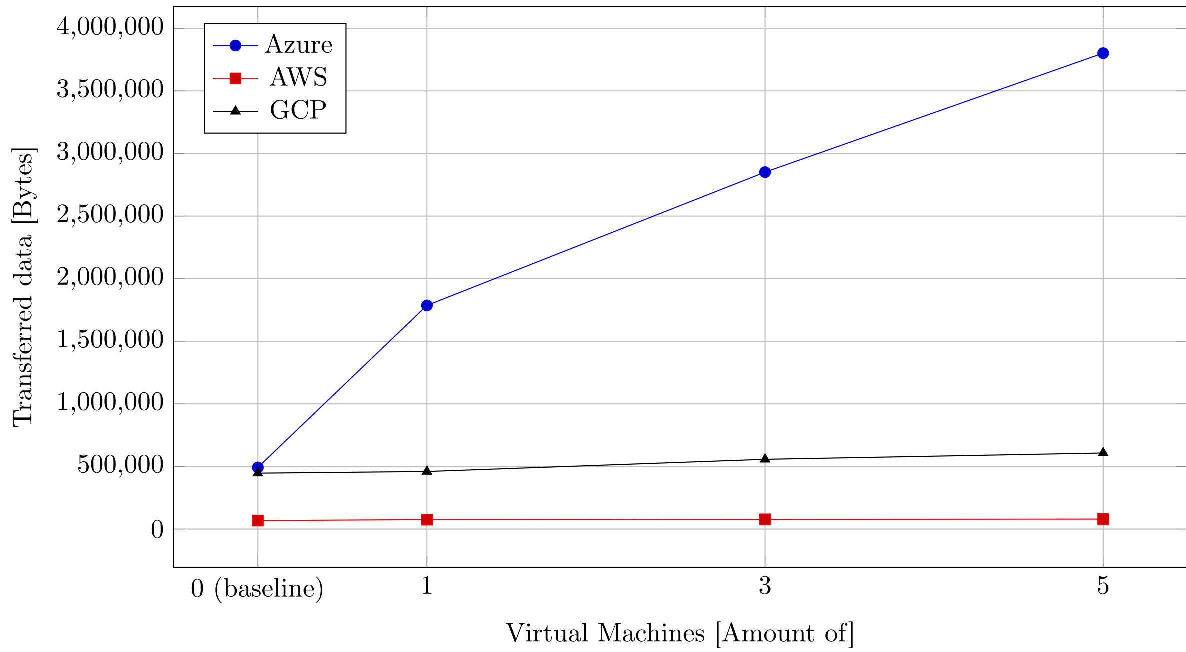

Figure 11: Average amount of transferred data across all cloud platforms

Figure 11 illustrates the average amount of data being transferred when creating our chosen topology throughout all cloud platforms.

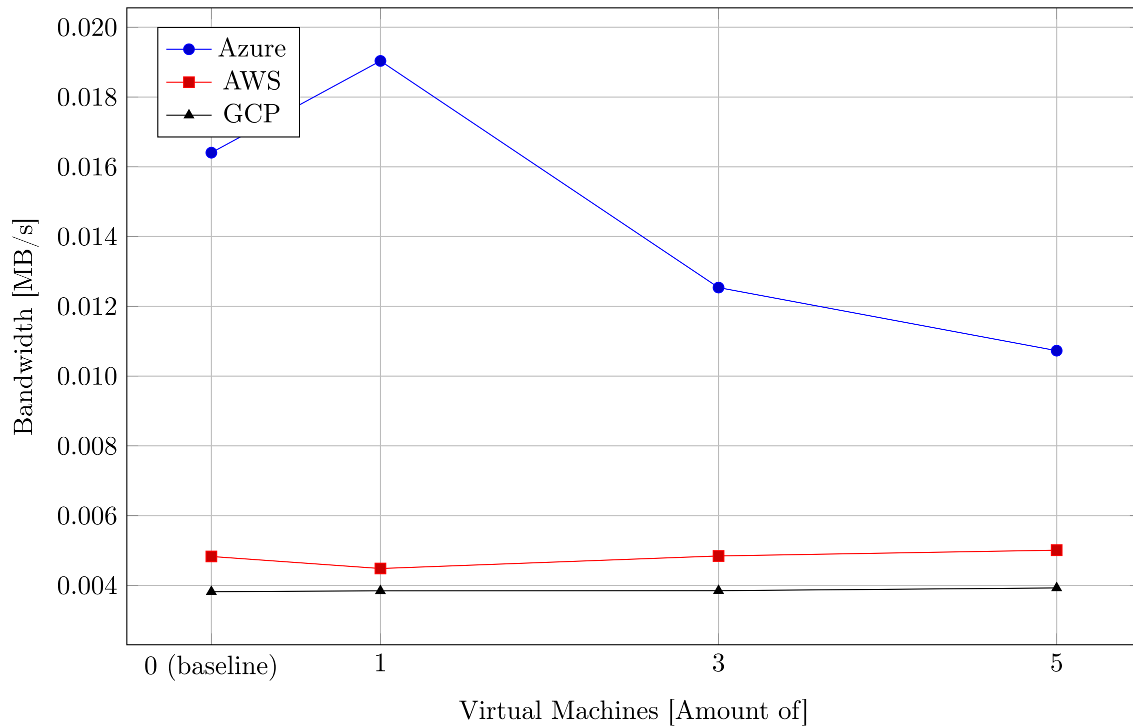

Figure 12: Average amount of network bandwidth across all cloud platforms

Looking at figure 12, Azure has the highest bandwidth out of all the cloud platforms although it does have the most variation. GCP has the lowest bandwidth although it does have the most consistency and AWS is slightly less linear than GCP but is faster.

9.3 Numerical results

In this section, everything presented in the graphs is displayed in tables. This gives a more detailed view of each result.

Analyzing table 2 it can be noted that when Azure starts up virtual machines, transferred data increases by about 1.2 MB (megabytes). Table 3 shows the averages of AWS, here it can be observed that the creation of 5 virtual machines has a lower configuration time than the creation of both 1 and 3 virtual machines while still transferring more data than both of them. In table 4, GCP has the most consistent bandwidth out of all the services, which means that the increase in configuration time and data transferred seems the most consistent out of all cloud platforms.

Virtual Machines	Configuration Time (s)	Transferred Data (B)	Bandwidth (MB/s)
0	30	492137.45	0.016405
1	93.85	1786415.5	0.019035
3	227.4	2851061.35	0.012538
5	354.35	3801813.8	0.010729

Table 2: Average configuration time, transferred data, and network bandwidth of Microsoft Azure

Virtual Machines	Configuration Time (s)	Transferred Data (B)	Bandwidth (MB/s)
0	14.05	67845.55	0.004829
1	16.8	75333.75	0.004484
3	16	77504.65	0.004844
5	15.85	79405.5	0.005010

Table 3: Average configuration time, transferred data, and network bandwidth of Amazon Web Services

Virtual Machines	Configuration Time (s)	Transferred Data (B)	Bandwidth (MB/s)
0	116.85	446404.85	0.003820
1	119.55	459592.1	0.003844
3	144.7	557007.65	0.003849
5	154.65	607327.4	0.003927

Table 4: Average configuration time, transferred data, and network bandwidth of Google Cloud Platform

10. Discussion

In this section, we will discuss the results we have achieved through our research into the capabilities of the tool with the different cloud platforms and during our experiments. We will also discuss the two other research questions presented. The difference in user complexity while using Ansible on different cloud platforms, and also what network features exist on these cloud platforms but cannot be configured with Ansibles standard modules.

10.1 Experimental Results

The experiments have provided us with a lot of different results. Looking at figure 7 it can be observed that the configuration time of Azure has the highest variation of all cloud platforms. Azure has a very low setup time for the baseline topology, but with the deployment of just one virtual machine, Azure increases the time by more than 1 minute. We noticed that when the playbook was running and configuring everything, Azure also creates multiple network devices that are connected to the virtual machines. These include:

- A network interface
- A network security group
- A public IP address
- A storage account

We have not configured this in the playbook (as seen in appendix A) but it seems to be mandatory for Azure to automatically create. We also tested manually configuring a virtual machine and we found that Azure creates these extra devices there as well.

Looking at figure 7 it is very clear which cloud platform is the fastest, AWS which uses one notable configuration option not available on the other platforms. looking at appendix B it can be observed that in the module for starting virtual machines, a *count* option is available. With this option, AWS is able to start multiple virtual machines at the same time with the same settings without having to loop through the module multiple times. analyzing figure 5 it can be observed that while the configuration time is increased by around 3 seconds, with 3 and 5 virtual machines it decreases. This might be circumstantial but it can also be that because of Ansible working as a hybrid of imperative and declarative infrastructure as a code language. In declarative language, the automation will eventually lead to the expected results in different means. That means that if Ansible has to create 5 virtual machines, with the last virtual machines it should be faster and more efficient.

Analyzing figure 12 it can be seen that Azure has the highest bandwidth, meaning it sends the most megabytes per second, which is very promising but analyzing each of the tables it can be observed in table 2 that it transfers an excessive amount of data, unlike the other cloud platforms. We believe that this is due to Azure having more required configurations for each task, a solution for Azure could be to reduce the amount it sends to start devices. This would lower both the amount of transferred data and configuration time.

Among the cloud platforms, GCP has been observed to exhibit lower bandwidth, and it is in between AWS and Azure in terms of the amount of data transferred, as indicated in the provided tables and figures 10. It also starts with the highest configuration time out of all platforms until Azure overtakes it after 1 virtual machine. We believe that the results we get are because a device needs to be fully started before a new task can begin on Azure and GCP. Ansible treats every platform as one end device, which means that it cannot start and configure every new resource at the same time. We believe this is the reason why Azure is slower than AWS and GCP. AWS on the other hand is much faster in configuration time for everything. A reason for this could be that instead of waiting for everything to start AWS just waits for a device to be initialized instead of

fully started, something that we observed in the web interface of AWS.

As seen in table 1 in Experiment all our virtual machines/cloud shell use the same amount of vCPUs and a different amount of RAM. With a different amount of RAM, it could be seen as an unequal comparison between each cloud platform. But as explained in the background of Ansible, high amounts of RAM are only needed when configuring multiple end devices called forks. Because each cloud platform is only treated as one end device Ansible should not need an exorbitant amount of RAM. Even though we have the same vCPUs for each platform the same thought for forks is applicable to vCPUs. A good example of high amounts of RAM not being needed is seen in AWS, due to it being the fastest configuration time out of all cloud platforms.

The location of this thesis is in the middle of Sweden, therefore the location where the experiments were performed was for each cloud platform to be as close to Sweden as possible. Both Azure and AWS have data centers in Sweden, Azure in Gävle[5], and AWS in Stockholm[6]. GCP on the other hand does not have a data center in Sweden, the data center was therefore chosen to be in Hamina which is located in Finland[7]. Because the virtual machines was inside each cloud platform, we believe this would give as high bandwidth as possible for each platform and as little traffic during the experiments as possible. Without being in the data centers it is impossible to eliminate circulating data traffic from other users. The decision to have data centers close to our location is because it might be of use to Swedish corporations. Because we are writing this thesis with help from Curitiba AB, it is of value for this thesis to inform Swedish corporations which cloud platform is most suitable for automation.

10.2 Overview of User Complexity

User complexity is something we have evaluated while working and configuring Ansible on the different cloud platforms. We decided to divide user complexity into two subcategories, one is user complexity for installing and configuring Ansible on the different cloud platforms, and one is the user complexity of creating the Ansible playbooks on the different cloud platforms.

For the installation and configuration of Ansible, user complexity comes from any extra configuration and installations that must be done besides the base installation of Ansible. The cloud platform with the lowest level of user complexity is Azure. Azure is basically plug and play, since Azure has a built-in cloud shell there is no need for creating a virtual machine with the purpose of acting as the Ansible controller. The built-in shell already has a version of Ansible installed which means that there are no installations or configurations needed to start automating tasks in Azure. Azure is also the only cloud platform out of the three we are comparing that does not need to configure security keys when using Ansible because Ansible is already integrated into the cloud shell. In our thesis, we had to perform one extra configuration to enable the timer module in Ansible, which was used for measurement purposes. However, this timer is optional and not needed to automate tasks in Azure with Ansible.

The process of installing Ansible on both AWS and GCP was similar in terms of user complexity. The first step was to create a virtual machine that would act as the Ansible controller, and then install Ansible on it. To use Ansible with AWS, two extra Python packages are needed, “boto3” and “botocore”. Another step that is needed is to generate a key pair including an access key and a secret key which is needed to control AWS through the Ansible controller. These extra steps that are needed to configure Ansible on AWS put AWS higher in terms of user complexity compared to Azures plug and play characteristics. As previously mentioned, GCP had a similar setup process as AWS. Like AWS, there were two additional Python packages required besides the base installation of Ansible, “requests” and “google-auth”. GCP also needed credentials in the form of a public/private key pair before being able to perform tasks with Ansible playbooks. Since

[5]https://azure.microsoft.com/en-us/explore/global-infrastructure/geographies/#choose-your-region
[6]https://aws.amazon.com/local/nordics/stockholm/
[7]https://cloud.google.com/compute/docs/regions-zones/

GCP has the same amount of extra configuration needed which includes two additional Python packages and a key pair it lands right beside AWS in terms of user complexity regarding the setup process of Ansible. With all these things considered, our conclusion is that Azure is the easiest cloud platform to start using for automating tasks in the cloud.

As previously mentioned, the second category which we have evaluated regarding the user complexity of Ansible in these different environments is the creation of playbooks. User complexity in this category mainly comes from how detailed the official Ansible documentation is regarding the different modules[8][9][10]. We realized early that Azure and AWS had far superior Ansible documentation compared to GCP. The Ansible documentation for Azure and AWS includes many examples covering most variations and options for each module while the GCP documentation only includes one example per module displaying a standard configuration. This makes it harder to learn how to create playbooks for GCP compared to Azure and AWS if you want to configure functions in any other way than the standard configuration, which is often the case. One thing that AWS and GCP have in common regarding the configuration of playbooks is that every device configured is independent of one another and for every task you need to specify both the region and the earlier mentioned key pair. The difference when creating playbooks for Azure is that every device needs to belong to a resource group, making it easier to manage all devices as a unit. The resource group is also where the region is specified. Another thing you do not have to worry about in Azure is specifying any key pairs for each task as the setup process of Ansible through the cloud shell does not require any.

One thing to keep in mind is that complexity can vary depending on the users' experience and expertise with each platform. Disregarding any previous experience, we conclude that Azure is the most user-friendly cloud platform to use while automating tasks with Ansible, based on both the setup process and the creation of playbooks.

10.3 Overview of Network Features

When looking through the network features on Azure, AWS, and GCP, we realized that there are many features that do not have a corresponding module in Ansible. In appendix D, E, F, we have summarized all network functions that exist within the different cloud platforms and if they can be performed with a corresponding Ansible module. By looking at the tables it may seem that Ansible is missing a lot of network functionalities but there are two things to keep in mind. One, the Ansible tool is mainly used for deployment and configuration, which means that many of the network functionalities that are missing in Ansible fall into other categories, monitoring for example. Two, since Ansible is open-source and has a large community, a lot of the functionalities that are missing from the official Ansible modules might exist as community modules. However, for this thesis, we decided to just focus on the official modules since modules that are made by the community are less secure and more prone to bugs.

The limitations on comparing only network functions and official modules were made because of Ansibles' large library of modules. Azure for example has over 200 modules on Ansible, comparing all of them is too large for this thesis.

The most notable missing feature for the cloud platforms is a firewall module in AWS. While AWS does have numerous firewall features, it cannot be found as a module in Ansible. A feature that is not missing but needs configuration before use in Ansible for GCP is the ability to use public IP addresses. The feature is technically available on Ansible but the address must first be reserved before it can be used.

[8] https://docs.ansible.com/ansible/latest/collections/azure/azcollection/index.html
[9] https://docs.ansible.com/ansible/latest/collections/amazon/aws/index.html
[10] https://docs.ansible.com/ansible/latest/collections/google/cloud/index.html

11. Conclusions

To answer our first research question we conducted multiple experiments to evaluate the performance of Azure, AWS, and GCP with Ansible. Performance was measured by analyzing configuration time and transferred data during the automated configuration. All cloud platforms' configuration times can be seen together in figure 7. AWS is the fastest and most time-efficient out of the three, taking a closer look at figure 5 shows that the graph does not vary by a lot and instead keeps a consistent configuration time. As mentioned in the discussion this might be because of the *"count"* configuration in the AWS playbook which the other cloud platforms do not have. Figure 11 shows the average amount of transferred data, Azure has the highest out of all of them. We believe this is because of the additional network devices that Azure creates with virtual machines.

We found that Azure uses the highest bandwidth, which can be observed in figure 12. This could be seen as something good but as seen in table 2 and comparing it to the other tables, Azure has both the highest transferred data and configuration time. Lowering the configuration time would yield even higher bandwidth but lowering the configuration time might be solved if less data is transferred. GCP was the least efficient of all the cloud platforms, while it does transfer less data than Azure the configuration time is still very long. GCP has the lowest bandwidth out of all the cloud platforms we compared. Based on our evaluation of performance, we conclude that AWS is the fastest and the most efficient since it transfers the least data.

The second research question is answered through evaluation and documentation during the work process. Based on our evaluation of user complexity while working and configuring Ansible on different cloud platforms, we found that Azure is the easiest cloud platform to use for automating tasks with Ansible. The setup process for Ansible on Azure is plug-and-play since Azure has a built-in cloud shell with Ansible already installed. In contrast, both AWS and GCP require additional configurations, such as generating a key pair and installing extra Python packages, which increases the user complexity.

Furthermore, we found that creating Ansible playbooks also varies in user complexity between the different cloud platforms. While AWS and GCP require the playbooks to specify the region and key pairs for each task, Azure simplifies the management of devices by grouping them into a resource group. This eliminates the need to specify the region and key pairs for each task individually. Additionally, the quality of official Ansible documentation for Azure and AWS is far superior compared to GCP. This makes it easier to learn how to create playbooks and understand the different settings when using Ansible with Azure and AWS compared to GCP.

It is important to keep in mind that user complexity can vary depending on the users' experience and expertise with the different platforms. However, based on our evaluation as first-time users of each platform, we conclude that Azure is the most user-friendly cloud platform to use while automating tasks with Ansible, based on both the setup process and the creation of playbooks.

For our third research question, we went through all of the cloud platforms' available features in the network categories and sorted them into tables. We found that while the different cloud platforms might be missing some features, the majority of the features missing are not devices that are deployable and are instead applications like a monitoring central hub. One important deployable feature that AWS is missing in Ansible is the ability to deploy a firewall. Both Azure and GCP have a firewall module available for them in Ansible. GCP is also missing the ability to easily deploy a public IP address in Ansible, you have to reserve it before you are able to deploy it, while it is not missing you have to reserve it first.

We found this question very hard to have a conclusion for, one big reason for this is that all cloud platforms have similar features but they also have their own naming scheme for those features meaning it can become very disorienting to navigate their interface. We reason that Azure has the upper hand in this question because AWS is missing a firewall module and GCP is missing the ability to easily deploy a public IP address.

12. Future Work

For future work, experiments can be done with Ansible on cloud platforms in multiple directions. One direction could be to test more modules. Azure for example has over 200 modules, meaning our experiments were a bit limited with testing different modules, but this was done because modules needed to be relatively similar to get fair testing. Because these cloud platforms also use different names for similar things, using their set of naming schemes makes it hard to know which modules deploy similar functions and may confuse some users. This might be the point though, if a customer of one cloud platform has certified themselves and trained for that platform, it might be too much of a hassle to reeducate them in a platform that offers similar services.

Another experiment that can be done with Ansible on cloud platforms for future work is to test the deployment of more virtual machines. We limited ourselves to a maximum of 5 virtual machines, this was because some platforms have limitations due to potential throttling. We encountered in our experiments numerous times where GCP stopped our running playbooks because it thought our requests tried to throttle the service. This might be solved by communicating with Google, but this could extend the experiment's duration, resulting in an unacceptable timeframe for our thesis.

References

[1] M. R. DeLisi. 'Gartner forecasts worldwide public cloud end-user spending to reach nearly $600 billion in 2023.' (), [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2022-10-31-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-600-billion-in-2023 (visited on 02/02/2023).

[2] G. Aggarwal. 'How the pandemic has accelerated cloud adoption.' (), [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2021/01/15/how-the-pandemic-has-accelerated-cloud-adoption/?sh=65e1e6906621 (visited on 02/02/2023).

[3] S. V. Nandgaonkar and A. Raut, 'A comprehensive study on cloud computing,' *International Journal of Computer Science and Mobile Computing, a Monthly Journal of Computer Science and Information Technology*, 2014.

[4] Y. Jiang and B. Adams, 'Co-evolution of infrastructure and source code - an empirical study,' in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015. DOI: 10.1109/MSR.2015.12.

[5] A. Patawari and V. Aggarwal, *Ansible 2 cloud automation cookbook: write Ansible playbooks for AWS, Google Cloud, Microsoft Azure, and OpenStack*, eng, 1st ed. Birmingham: PACKT Publishing, 2018, ISBN: 178829582X.

[6] Nasa. 'Nasa: Increasing cloud efficiency with ansible and ansible tower.' (2022), [Online]. Available: https://www.ansible.com/hubfs/pdf/Ansible-Case-Study-NASA.pdf (visited on 07/02/2023).

[7] K. Morris, *Infrastructure as Code: Dynamic Systems for the Cloud Age*, 2nd ed. Sebastopol, California: O'Reilly, 2020, ISBN: 978-1-098-11467-1.

[8] R. Wang, *Infrastructure as Code, Patterns and Practices: With examples in Python and Terraform*, eng. Place of publication not identified: Manning Publications, 2022.

[9] P. Masek, M. Stusek, J. Krejci, K. Zeman, J. Pokorny and M. Kudlacek, 'Unleashing full potential of ansible framework: University labs administration,' in *2018 22nd Conference of Open Innovations Association (FRUCT)*, 2018. DOI: 10.23919/FRUCT.2018.8468270.

[10] J. Schwarz, A. Steffens and H. Lichter, 'Code smells in infrastructure as code,' in *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, 2018. DOI: 10.1109/QUATIC.2018.00040.

[11] S. Wagbrant and V. Dahlen Radic, *Automated network configuration : A comparison between ansible, puppet, and saltstack for network configuration*, 2022.

[12] P. Loshin. 'Secure shell(ssh).' (), [Online]. Available: https://www.techtarget.com/searchsecurity/definition/Secure-Shell (visited on 26/01/2023).

[13] Ansible project contributors. 'How ansible works.' (), [Online]. Available: https://www.ansible.com/overview/how-ansible-works (visited on 26/01/2023).

[14] R. McKendrick, *Learn ansible : automate cloud, security, and network infrastructure using ansible 2.x*, eng, 1st edition. Birmingham, London ; Packt, 2018, ISBN: 1-78899-932-0.

[15] E. Segura. 'Declarative vs imperative in iac.' (2023), [Online]. Available: https://www.linode.com/blog/devops/declarative-vs-imperative-in-iac/ (visited on 24/05/2023).

[16] L. Hochstein and R. Moser, *Ansible: Up and Running : Automating Configuration Management and Deployment the Easy Way.* O'Reilly Media, 2017, ISBN: 9781491979808. [Online]. Available: https://books.google.se/books?id=PcJPMQAACAAJ.

[17] Redhat. 'What is yaml.' (2023), [Online]. Available: https://www.redhat.com/en/topics/automation/what-is-yaml (visited on 15/03/2023).

[18] G. Madapparambat. '8 ways to speed up your ansible playbooks.' (2022), [Online]. Available: https://www.redhat.com/sysadmin/faster-ansible-playbook-execution (visited on 13/04/2023).

[19] H. Dharmasena. 'Difference between forks and serial in ansible.' (2020), [Online]. Available: https://medium.com/devops-srilanka/difference-between-forks-and-serial-in-ansible-48677ebe3f36 (visited on 13/04/2023).

[20] Ansible project contributors. 'How to build your inventory.' (), [Online]. Available: https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html (visited on 26/01/2023).

[21] Ansible project contributors. 'How ansible works.' (), [Online]. Available: https://docs.ansible.com/ansible/latest/inventory_guide/intro_dynamic_inventory.html (visited on 22/02/2023).

[22] *Mastering Ansible - Fourth Edition*, eng. Packt Publishing, 2021, ISBN: 1-80181-878-9.

[23] Ansible project contributors. 'How ansible works.' (), [Online]. Available: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html (visited on 26/01/2023).

[24] IBM. 'What is virtualization?' (), [Online]. Available: https://www.ibm.com/topics/virtualization (visited on 26/01/2023).

[25] VMware, Inc. 'What is a hypervisor?' (), [Online]. Available: https://www.vmware.com/topics/glossary/content/hypervisor.html (visited on 26/01/2023).

[26] VMware, Inc. 'What is cloud computing?' (), [Online]. Available: https://www.ibm.com/topics/cloud-computing (visited on 26/01/2023).

[27] Google. 'What is a cloud service provider.' (2023), [Online]. Available: https://cloud.google.com/learn/what-is-a-cloud-service-provider (visited on 21/03/2023).

[28] S. Rajasoundaran, A. Prabu, S. Routray *et al.*, 'Machine learning based deep job exploration and secure transactions in virtual private cloud systems,' *Computers and Security*, 2021. DOI: 10.1016/j.cose.2021.102379. [Online]. Available: https://doi.org/10.1016/j.cose.2021.102379.

[29] P. Dutta and P. Dutta, 'Comparative study of cloud services offered by amazon, microsoft and google,' *International Journal of Trend in Scientific Research and Development*, 2019. DOI: 10.31142/ijtsrd23170. [Online]. Available: https://doi.org/10.31142/ijtsrd23170.

[30] B. Santoso and M. W. Sari, 'Improvement of setup time on server infrastructure automation using ansible framework,' *Journal of Engineering Science and Technology*, 2022. [Online]. Available: https://jestec.taylors.edu.my/Vol%5C%2017%5C%20Issue%5C%205%5C%20October%5C%202022/17_5_43.pdf.

[31] K. Säfsten and M. Gustavsson, *Forskningsmetodik för ingenjörer och andra problemlösare*, 1st ed. Studentlitteratur AB, Lund, 2019.

[32] P. Wankhede, M. Talati and R. Chinchamalatpure, 'Comparative study of cloud platforms -microsoft azure, google cloud platform and amazon ec2,' *Journal of Research in Engineering and Applied Sciences*, 2020. DOI: 10.46565/jreas.2020.v05i02.004.

[33] L. S. Vailshery. 'Cloud infrastructure services vendor market share worldwide from 4th quarter 2017 to 4th quarter 2022.' (2023), [Online]. Available: https://www.statista.com/statistics/967365/worldwide-cloud-infrastructure-services-market-share-vendor/ (visited on 17/03/2023).

[34] Microsoft. 'Availability zones overview.' (2023), [Online]. Available: https://learn.microsoft.com/en-us/azure/reliability/availability-zones-overview#availability-zones (visited on 21/03/2023).

[35] Microsoft. 'Azure vs aws.' (2023), [Online]. Available: https://azure.microsoft.com/en-us/pricing/azure-vs-aws/ (visited on 21/03/2023).

[36] M. Azure. 'Overview of azure compute services.' (2023), [Online]. Available: https://learn.microsoft.com/en-us/training/modules/azure-compute-fundamentals/overview (visited on 16/05/2023).

[37] A. W. Services. 'Global network of aws regions.' (2023), [Online]. Available: https://aws.amazon.com/ (visited on 17/03/2023).

[38] A. W. Services. 'What is amazon ec2.' (2023), [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html (visited on 15/05/2023).

[39] Google. 'Cloud locations.' (2023), [Online]. Available: https://cloud.google.com/about/locations (visited on 17/03/2023).

[40] B. Lutkevich. 'What is google compute engine?' (2022), [Online]. Available: https://www.techtarget.com/searchaws/definition/Google-Compute-Engine (visited on 16/05/2023).

A Appendix

This is the playbook for Azure that was used during the experimentation. Something that is different from the other playbooks is the additional task used, the task named *Create resource group*. Resource groups are used as a container for Azure solutions resources. The playbook creates 3 virtual machines and it is rather easy to switch to any number of virtual machines. Either delete the *with_ sequence* line to get 1 virtual machine or increase the number after *end*.

```
- hosts: localhost
  connection: local
  tasks:
    - name: Create resource group
      azure_rm_resourcegroup:
        name: testRG
        location: northeurope

    - name: Create Virtual network
      azure_rm_virtualnetwork:
        resource_group: testRG
        name: testVN
        address_prefixes: "192.168.0.0/16"

    - name: Add subnet
      azure_rm_subnet:
        resource_group: testRG
        name: testSN
        virtual_network_name: "testVN"
        address_prefix: "192.168.1.0/24"
      register: subnet

    - name: Create security group
      azure_rm_securitygroup:
        resource_group: testRG
        name: testSG
        rules:
          - name: AllowSSH
            protocol: Tcp
            destination_port_range: 22
            access: Allow
            direction: Inbound
            priority: 1000

    - name: Create Load Balancer
      azure_rm_loadbalancer:
        resource_group: testRG
        name: testLB
        frontend_ip_configurations:
          - name: myFrontEndIP
            subnet: "{{ subnet.state.id }}"
        backend_address_pools:
          - name: myBackendPool
        probes:
          - name: prob0
            port: 80
            protocol: Tcp
        load_balancing_rules:
          - name: myHTTPRule
            protocol: Tcp
            frontend_port: 80
            backend_port: 80
            interval: 1

            enable_floating_ip: false
            load_distribution: Default
            frontend_ip_configuration: myFrontEndIP
            backend_address_pool: myBackendPool
            probe: prob0
```

```
- name: Create VM
  azure_rm_virtualmachine:
    resource_group: testRG
    name: "instance-{{ item }}"
    vm_size: Standard_B1ls
    admin_username: "netadm123"
    admin_password: "Netadm123!"
    image:
      offer: 0001-com-ubuntu-server-jammy
      publisher: canonical
      sku: 22_04-lts
      version: latest
  with_sequence: start=1 end=2
```

Listing 1: The playbook used for Microsoft Azure, specifically 3 virtual machines

B Appendix

The playbook used for AWS during experiments. The *access key* and *secret key* are both redacted because they would enable access to the AWS system used. A notable variable used in the AWS playbook is the *count* variable. This variable is used in the playbook for deploying virtual machines and can be seen in the count option of the *ec2_instance* task. AWS also has to have a *VPC gateway* for the security groups, which was mentioned earlier in the description of work where we discuss more in detail how the playbooks were created.

```
- name: creation of it all
  hosts: localhost
  vars:
    access_key: ################
    secret_key: ################
    region: eu-north-1
    tag: ansible_Slave
    firewaller_Name: firewall_Ansible
    count: 3



  tasks:
  - name: Create AWS VPC
    ec2_vpc_net:
      name: test_Network
      aws_access_key: "{{ access_key }}"
      aws_secret_key: "{{ secret_key }}"
      cidr_block: 192.168.1.0/24
      region: "{{ region }}"
      state: present
      tags:
        created: "{{ tag }}"
    register: vpc_result


  - name: start a subnet in the vpc
    ec2_vpc_subnet:
      vpc_id: "{{ vpc_result.vpc.id }}"
      aws_access_key: "{{ access_key }}"
      aws_secret_key: "{{ secret_key }}"
      cidr: 192.168.1.0/24
      region: "{{ region }}"
      tags:
        Name: ansible_created
        created: "{{ tag }}"
    register: subnet_result

  - name: create VPC gateway
    ec2_vpc_igw:
      region: "{{ region }}"
      aws_access_key: "{{ access_key }}"
      aws_secret_key: "{{ secret_key }}"
      vpc_id: "{{ vpc_result.vpc.id }}"
      state: present
      tags:
        Name: ansible_Gateway
        created: "{{ tag }} "



  - name: Starts a security group/firewall
    amazon.aws.ec2_security_group:
      aws_access_key: "{{ access_key }}"
      aws_secret_key: "{{ secret_key }}"
      name: "{{ firewaller_Name }}"
      region: "{{ region }}"
      vpc_id: "{{ vpc_result.vpc.id }}"
      description: This is for Ansible
```

```
      tags:
        created: "{{ tag }}"
      rules_egress:
        - proto: all
          cidr_ip: 0.0.0.0/0
      rules:
        - proto: tcp
          from_port: 22
          to_port: 22
          cidr_ip: 0.0.0.0/0
        - proto: tcp
          from_port: 80
          to_port: 80
          cidr_ip: 0.0.0.0/0
      state: present
    register: sec_group

  - name: starts load balancer
    elb_classic_lb:
      aws_access_key: "{{ access_key }}"
      aws_secret_key: "{{ secret_key }}"
      name: LoadbalancedAnsible
      state: present
      region: "{{ region }}"
      tags:
        created: "{{ tag }}"
      health_check:
        - ping_protocol: TCP
          ping_port: 80
          interval: 1
      listeners:
        - protocol: http
          load_balancer_port: 80
          instance_protocol: http
          instance_port: 80
      subnets:
        - "{{ subnet_result.subnet.id }}"
      security_group_names: "{{ firewaller_Name }}"


  - name: Start vm
    ec2_instance:
      name: "Ansible_instance-{{ count }}"
      aws_access_key: "{{ access_key }}"
      aws_secret_key: "{{ secret_key }}"
      vpc_subnet_id: "{{ subnet_result.subnet.id }}"
      instance_type: t3.micro
      image_id: ami-09e1162c87f73958b
      region: "{{ region }}"
      count: "{{ count }}"
      tags:
        created: "{{ tag }}"
```

Listing 2: The playbook used for Amazon Web Services, specifically with creating 3 virtual machines

C Appendix

The playbook for GCP has one big difference from the other cloud platform playbooks, the GCP deploys a firewall instead of a security group. This is because GCP does not have security groups, we therefore decided that a firewall is the next best thing. While it could be seen as unfair. We believe though that this is not unfair, firewalls and security groups are very similar in that they both provide blocking mechanisms to a network. GCP also requires disk creation in their virtual machines, a thing that is relatively easy to configure but took a while to figure out due to their documentation not mentioning that it was required[11].

```
- name: work
  hosts: localhost
  vars:
    aut_file: "##############################"
    region: europe-north1
    project: ansible-project-379215

  tasks:
  - name: create a gsp network
    google.cloud.gcp_compute_network:
      auth_kind: serviceaccount
      service_account_file: "{{ aut_file }}"
      name: test-network
      auto_create_subnetworks: 'true'
      project: "{{ project }}"
      state: present
    register: network


  - name: create a subnetwork
    google.cloud.gcp_compute_subnetwork:
      name: ansiblenet
      region: "{{ region }}"
      network: "{{ network }}"
      ip_cidr_range: 192.168.1.0/24
      project: "{{ project }}"
      auth_kind: serviceaccount
      service_account_file: "{{ aut_file }}"
      state: present
    register: sub_network

  - name: Create firewall
    google.cloud.gcp_compute_firewall:
      auth_kind: serviceaccount
      allowed:
        - ip_protocol: tcp
          ports:
          - '22'
      service_account_file: "{{ aut_file }}"
      state: present
      name: firewall-ansible
      project: "{{ project }}"
      network: "{{ network }}"



  - name: instance group
    google.cloud.gcp_compute_instance_group:
      name: group-for-loadbalancers
      zone: "{{ region }}-a"
      project: "{{ project }}"
      auth_kind: serviceaccount
      service_account_file: "{{ aut_file }}"
      state: present
```

[11] https://docs.ansible.com/ansible/latest/collections/google/cloud/gcp_compute_instance_module.html#ansible-collections-google-cloud-gcp-compute-instance-module

```
    register: instancegroup

- name: Health check for load balancer
  google.cloud.gcp_compute_health_check:
    name: healthycheck
    type: TCP
    tcp_health_check:
      port: 80
    check_interval_sec: 1
    project: "{{ project }}"
    state: present
    auth_kind: serviceaccount
    service_account_file: "{{ aut_file }}"
  register: healthcheck


- name: Create load balancer
  google.cloud.gcp_compute_backend_service:
    auth_kind: serviceaccount
    service_account_file: "{{ aut_file }}"
    name: testobject
    backends:
    - group: "{{ instancegroup.selfLink }}"
    project: "{{ project }}"
    health_checks:
      - "{{ healthcheck.selfLink }}"
    connection_draining:
      draining_timeout_sec: 10
    state: present

- name: Create instances
  google.cloud.gcp_compute_instance:
    name: "instance_ansible-{{ item }}"
    auth_kind: serviceaccount
    service_account_file: "{{ aut_file }}"
    project: "{{ project }}"
    state: present
    zone: "{{ region }}-a"
    network_interfaces:
    - network: "{{ network }}"
      subnetwork: "{{ sub_network }}"
    machine_type: e2-micro
    disks:
    - auto_delete: true
      boot: true
      initialize_params:
        source_image: projects/ubuntu-os-cloud/global/images/family/ubuntu-2204-lts
        disk_size_gb: 10
    tags:
      created: Ansible-created
  with_sequence: start=1 end=2
```

Listing 3: The playbook used for Google Cloud Platform, specifically 3 virtual machines

D Appendix

This table contains network functions available on the Azures platform and then correlates them to available modules on the Azure module page. The ones that are **bolded** and have in the row of *On Ansible Category* are categories in Azures platform.

Network Function	On Ansible	Name of module
Network Foundation	Category	
Bastion	Yes	azure_rm_bastionhost
Custom IP Prefixes	No	
DNS Private resolvers	No	
DNS Zones	Yes	azure_rm_dnszone
NAT Gateways	Yes	azure_rm_natgateway
Network Managers	No	
Private DNS Zone	Yes	azure_rm_privatednszone
Private Link	Yes	azure_rm_privatelinkservice
Public IP address	Yes	azure_rm_publicipaddress
Public IP Prefix	Yes	azure_rm_publicipaddress
Virtual Networks	Yes	azure_rm_virtualnetwork
Hybrid Connectivity	Category	
Communication Gateways	No	
Connections	No	
ExpressRoute Circuits	Yes	azure_rm_expressroute
ExpressRoute traffic collector	No	
Local network gateway	No	
Mobile Network	No	
Peering services	No	
Peerings	No	
Virtual network gateways	Yes	azure_rm_virtualnetworkgateway
Virtual WANs	Yes	azure_rm_virtualwan
Network Security	Category	
DDOS protection plan	Yes	azure_rm_ddosprotectionplan
Firewall manager	No	
Firewall	Yes	azure_rm_azurefirewall
Firewall policies	Yes	azure_rm_firewallpolicy
Network Security Group	Yes	azure_rm_securitygroup
Network Security Perimeters	No	
Web Application Firewall	Yes	azure_rm_webappaccessrestriction
Load Balancing	Category	
Application Gateway	Yes	azure_rm_appgateway
Load Balancer	Yes	azure_rm_loadbalancer
NGINXaaS	No	
Content Delivery	Category	
Connected Cache Resources	No	
Connected Cache ispCustomer Resources	No	
front door and CDN profiles	No	
Monitoring	Category	
Monitor	No	
Network Watcher	No	

Table 5: Microsoft Azures Network functions

E Appendix

An appendix containing the Amazon Web services network functions. These network functions are grouped by how AWS has them grouped, but we also added a **Miscellaneous** category due to load balancers' availability as network functions in both Azure and GCP. Also worth noting is that AWS does not have firewalls as a module in Ansible, which the other platforms do have.

Network function	On Ansible	Name of module
API Gateway	Category	
HTTP API	No	
WebSocket API	No	
REST API	No	
Private REST API	No	
VPC Links	No	
AWS App Mesh	No	
AWS cloud map	No	
CloudFront	No	
Direct Connect	No	
Global Accelerator	No	
Route 53	Category	
Route 53	Yes	route53
Route 53 healthcheck	Yes	route_53_health_check
Route 53 zone	Yes	route_53_zone
VPC	Category	
VPCs	Yes	ec2_vpc_net
Subnet	Yes	ec2_vpc_subnet
Route tables	Yes	ec2_vpc_route_table
Internet gateways	Yes	ec2_vpc_igw
egress only internet gateways	No	
DHCP option sets	Yes	ec2_vpc_dhcp_option
Elastic IP	Yes	ec2_eip
Manage prefix list	No	
Endpoints	Yes	ec2_vpc_endpoint
Enpoints services	Yes	ec2_vpc_endpoint_service_info
NAT gateways	Yes	ec2_vpc_nat_gateway
Peering connections	No	
Security	Category	
Network ACLs	No	
Security groups	Yes	ec2_security_group
DNS firewall	Category	
Rule groups	No	
Domain lists	No	
Network firewall	Category	
Firewalls	No	
Firewall policies	No	
Network firewall rule groups	No	
Network firewall resource groups	No	
Virtual Private Network	Category	

Customer gateways	No	
Virtual private gateways	No	
Site-to-site VPN connections	No	
Client VPN endpoints	No	
Transit Gateways	Category	
Transit Gateways	No	
Transit Gateways attachments	No	
Transit Gateways policy tables	No	
Transit Gateways route tables	No	
Transit Gateways multicast	No	
Traffic mirroring	Category	
Mirror sessions	No	
Mirror targets	No	
Mirror filters	No	
Miscellaneous	Category	
Application load balancer	Yes	elb_application_lb
Classic load balancer	Yes	elb_classic_lb

Table 6: Amazon Web Services Network functions

F Appendix

The Appendix contains GCPs' network functions. Like the other appendices, categories are **bolded** and contain a *Category* mark in the *On Ansible* row to specify GCP's network function categories. Something to note is that some columns are empty in both their *network funtion* row and *On Ansible* but have a module in *Name of module*, this is because some network functions have 2 or more modules in Ansible that do different things for one network function.

Network function	**On Ansible**	**Name of module**
VPC Network	Category	
VPC network	Yes	gcp_compute_network
Subnet	Yes	gcp_compute_subnetwork
Ip addresses	Yes	gcp_compute_address
Internal IP address	Yes	gcp_compute_address
External IP address[12]	Yes	gcp_compute_address
		gcp_compute_global_address
Ipv4	Yes	
Ipv6	No	
Bring your own IP	No	
Firewall	Yes	gcp_firewall
Firewall rules	Yes	gcp_appengine_firewall_rule
Firewall Policies	No	
Routes	Yes	gcp_compute_route
VPC network peering	No	
Shared VPC	No	
Serverless VPC access	No	
Packet mirroring	No	
Network Services	Category	
Load balancing	Yes	
Frontends	No	
Backends	Yes	gcp_compute_backend_service
Cloud CDN	No	
Cloud DNS	Yes, 2 modules	gcp_dns_managed_zone
		gcp_dns_resource_record_set
Cloud NAT	No	
Traffic director	No	
Routing rule map	Yes	gcp_compute_region_url_map
Service directory	No	
Cloud domains	No	
Private service connect	No	
Network Security	Category	
Cloud armor	No	
Cloud IDS	No	
SSL policies	Yes	gcp_compute_ssl_policy
Hybrid connectivity	Category	
VPN	Yes, 3 modules	gcp_compute_external_vpn_gateway
		gcp_compute_target_vpn_gateway
		gcp_compute_vpn_tunnel
Interconnectivity	Yes	gcp_compute_interconnect_attachement
Cloud router	Yes	gcp_compute_router

[12] Address must be reserved before use

Network connectivity center	No	

Table 7: Google Cloud Platform Network functions

G Appendix

This is the script used in each Ansible Controller to automate the playbooks so we did not have to start everything repeatedly. The script opens a text file to save the results, it then starts a loop and loops 20 times. the variable *bytestart* contains the number of transferred data sent through the interface connected to the internet. The *runtime* variable starts the creation playbook, what the variable then saves with *awk* is the time the playbook takes to finish. With the variable *byteend* the script saves how much data has been transferred. To summarise the rest of the script, it converts the time so it is only in seconds and saves that and transferred data, and in the end, starts the playbook to delete everything in the Cloud platform that was created in this script.

```
import os
import time

fil = open("result.txt", "a")

for i in range(20):
    bytestart = os.popen("cat /proc/net/dev | grep ens5 | awk '{print$10}'").read()
    runtime = os.popen("ansible-playbook create_playbook.yml | grep run
    | awk '{print$8, $9, $10, $11}'").read()
    byteend = os.popen("cat /proc/net/dev | grep ens5 | awk '{print$10}'").read()
    runtime = runtime.strip()
    tid = runtime.strip(' ')
    print(tid)
    minutes = tid[0]
    seconds = int(minutes) * 60 + int(tid[1])
    sent = int(byteend) - int(bytestart)
    fil.write(f"Test {i}\n")
    fil.write(f"Runtime: {seconds} seconds\n")
    fil.write(f"Bytes: {sent}\n")
    time.sleep(5)
    os.system('ansible-playbook delete_playbook.yml')

fil.close()
```

Listing 4: The script used to automate creation and deletion