Linnéuniversitetet
Kalmar Växjö

Linnaeus University
Faculty of Technology – Department of Computer Science

2DV517 - Deployment Infrastructures Examination 2

Group: 6

Students:
Tomas Marx-Raacz von Hidvég (tendn09)
Susanna Persson (sp222xw)
Dennis Demir (dd222gc)
Julia Perlkvist (jp223es)

Contents

1.Introduction

This document aims to record the planning and execution of the course 2dv517 group project. It will record how the plan was devised, how it changed and why. It will also record in which way the work was done and divided throughout the allotted time schedule for the project.

Furthermore, within this document we will also include references to theoretical works regarding patterns that we followed, both by accident and by design.

The repository for the project can be found on the following address: https://github.com/DennisDemir24/2DV517-assignment2

And the Video presentation of the setup can be found on the following address: https://youtu.be/G92s0LrFO48

Individual links to the repo will be presented using IEEE standard and presented in the bibliography.

1.2. Overview

Now after having the initial meeting with the group, it became apparent that none of us had experience with working with any of the automation tools, except Kubernetes which is rather a tool for dividing an infrastructure into containers.

Spurred by a statement in an instructional video in the course 1dv032 minted by the faculty prefect Morgan Ericsson that you could do all pieces of the puzzle with Ansible, the decision was taken to explore if this assignment could be undertaken wholly through Ansible.

As for the general structure of the setup the choice was taken to go for a traditional VM setup instead of a container setup through Kubernetes due to the limited prior experience the group had of automation tools. This decision was to lower the risk of the task becoming too overwhelming. The setup was divided into a network, subnet and router followed by eight virtual machines as seen in Table 1.

Table 1 . Basic setup

VM	Amount
WordPress hosts servers	3
MySQL servers	2
NFS file sharing server	1
NginX external traffic load balancer	1
Prometheus/Grafana monitoring server	1

A repository and project board were set up on GitHub by one of the group members and shared with the rest of the participants in this project.

As far as workflow was concerned the group agreed on working only with the master branch and having clear communication regarding uploads to said repository in a separate Discord channel that was set up for this groups work during the course.
Weekly meetings were planned, initially during Wednesday but after two weeks moved to Monday to recap what had been done properly until the sprint review taking place on Tuesdays.

2.Exploratory work and preparation

The participants started off the project by familiarizing themselves with the tools and the contents of the website as it existed in its current state. For this, half of the group attempted to set up the old version of the website to gain an understanding of its different components and the steps involved in the setup, while the other half of the group began studying Ansible.

Since none of the group members had prior experience with Ansible aside from reading about it in a prior assignment, the decision was taken that some group members would start learning the tool basics while others studied the infrastructure so that we then could share our new knowledge with each other and combine our understanding of the website with knowledge about how to automate the setup.

Of course, manual setup is a good first step to take before automating to figure out the steps involved and ensure that they work as intended. A bash script [1] was created for the initial setup of the old version of the website which gave some idea of what to do as a basis for then expanding and creating the new infrastructure and automating it.

However, this script failed to run the site properly and as to why was discovered much later which will be covered later.

3.Workflow - trial, error and progress

Our initial plan had been to rely on Ansible for automation as it seemed to be capable of achieving everything we needed. Although decent progress with Ansible was made, The work grinded slightly to a halt with the provisioning through Ansible due to some weaknesses in the modules for OpenStack provisioning. The decision was taken to lay aside Ansible as a provisioning tool for this assignment and explore Terraform as a tool for provisioning instead.

The provisioning through Terraform worked a lot better and the planned setup was quickly devised [2]. Ansible was left to configure the servers from an inventory file created through the Terraform provisioning [3].

A decision at this point was taken to use a combination of dynamic configuration and hardcoded configuration through pre-created configuration files for various packages that would run on the virtual machines. This was achieved through pre-determining the internal IP addresses of the VM's through Terraform. This made it possible to construct for example template configuration files for NginX load balancing, both internal (for MySQL read and write separation) as well as easy setup of a configuration for the routing of calls from the public web towards the WordPress servers [4].

Initially the intention was to use a bastion jump-host (The NginX server) as a point of entry to the network and to run both Ansible and Terraform from remote machines. However, we struggled with making this work properly, and eventually decided to drop the idea before pouring too much time and effort into it that might be better used elsewhere. Instead, a separate AC controller machine was constructed through Terraform, bringing the total of machines up to nine and finalizing the setup as can be seen in Figure 1.

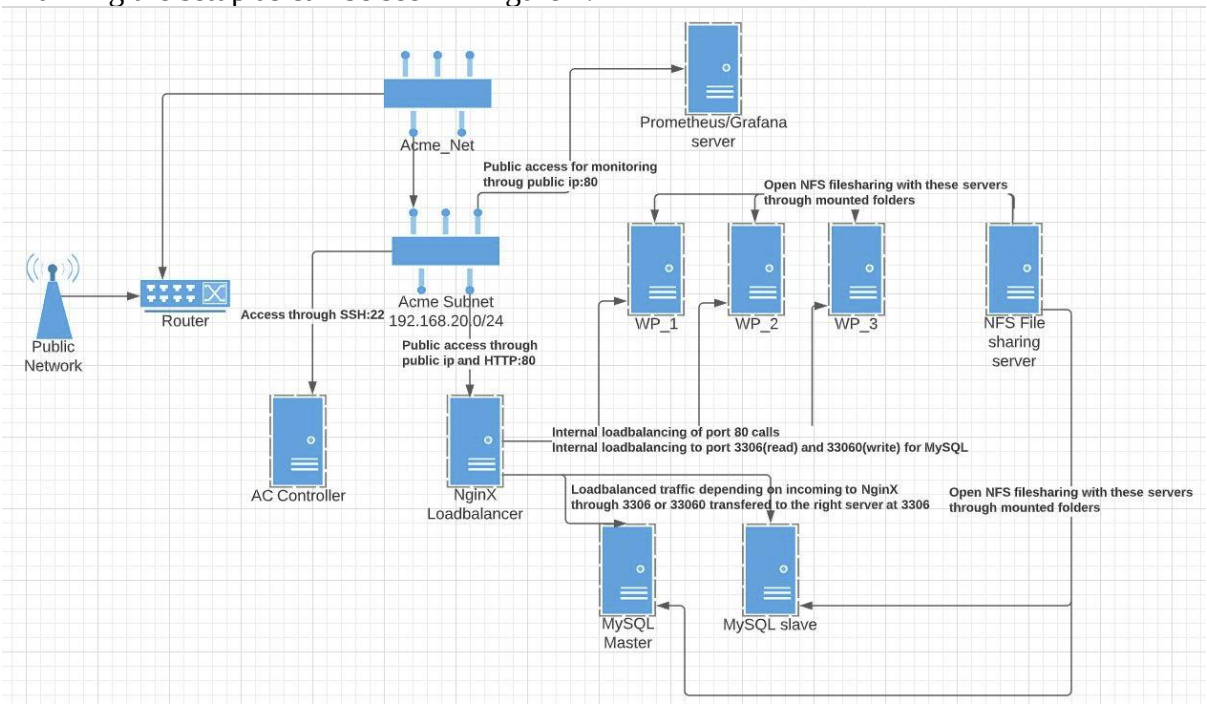

Figure 1. General setup of machines through Terraform.

A Bash script was created to transport the Ansible folder and various template files and backup files to the Ansible Controller machine so one, after accessing it through SSH could run the rest of the configuration process [5].

After making some progress with Ansible and Terraform we started to consider the monitoring aspect. For this we decided that we were going to use Prometheus [6], and we also decided to use Grafana [7] alongside it to present the data from Prometheus. None of the group members had any real experience with said applications but it had been briefly run through in a previous course and as such it became an obvious choice.

When researching the Prometheus/Grafana bundle the Node Exporter

Tool [8] was also included into the mix to act as a metric collector of machine metrics for Prometheus.

4.Specific automation tools and what part they play

4.1. Bash scripts

Bash scripts, being a very early take on automation still plays a part in this project. In this case it was the ad-hoc solution to the otherwise dreary process of moving everything needed for Ansible to the ansible controller VM. What it does is loop through the files in the Ansible directory and send everything that is not marked ".secret" or is named "upload.sh" to the Ansible VM.

4.2. Terraform

Terraform was used for what it was supposed to, to provision the OpenStack VMs, network, subnet, router, and security group as well as assigning said security groups to the individual servers through the module for this cloud.

Furthermore, the provisioning was configured to create an inventory file for Ansible to use for the configuration of the individual servers as well as the ansible.cfg to replicate some parameters that the individual administrator needs to input into a terraform.tfvars file. It was our ambition to automate the creation of all files that contained environment- or user-specific information (such as floating IP addresses or paths to SSH keys), but because of time constraints this was not done to its fullest potential. During the project it became apparent how important this step is, as there were several times where forgotten manual configuration the cause of errors in the Acme infrastructure was. A cornerstone of DevOps is the automation of tasks just like this, and this project highlighted this in practise for all team members.

Other configurations files were also created and put into the Ansible folder this way including various vars files so that an administrator would have to change as little as possible.

Some pre installations of needed packages was made through including cloud-init (https://github.com/DennisDemir24/2DV517-assignment2/tree/main/cloud_init_ansible) scripts with the terraforming, especially for the Ansible controller that had various dependencies to be able to run both Ansible and some of its modules necessary for this project. Other general cloud-init scripts was also added into the mix for databases and the rest of the machines to preinstall some dependencies as well as updating and upgrading packages during spawning instead of doing it during configuration. This last effort was done so that the ansible configurations wouldn´t take so long to finish once they ran the same checks for updates.

4.3. Ansible

Now Ansible is the real workhorse in this project, and by far the one that needed the most research. Using a combination of dynamically created variables from Terraform as well as, as mentioned previously, pre-constructed conf files of some variables that will never change, for example private network IP addresses of hosts, these playbooks were designed to configure everything that was needed to have the services running on the VMs.

4.3.1. NginX Load balancer

The NginX load balancer was created through installing dependencies, utilizing pre-constructed .conf files and placing them in the right locations as well as starting the service. This was made possible by prior legwork done in terraform with the fixed IP addresses internally. As NginX only deals with this type of proxy commands it was a decently easy task to setup the HTTP load balancing towards WordPress servers as well as the internal TCP stream of the communication between the WordPress servers and the MySQL servers (port 33060 was pointed towards the master, 3306 divided between master and slave).

4.3.2. NFS File sharing

This was our interpretation of a location where administrators could upload files. This was a very straight forward installation of needed dependencies as well as inclusion of a preconfigured exports file that tells the program which IPs and what permissions said IPs would have towards the shared folders. The NFS-commons package also needed to be installed on all servers needed to access said files or in any way needed persistent storage for backups (MySQL and WordPress servers) as well as directories mounted that.

In its current form administrators cannot upload to it, but the filesharing between servers is partly implemented. However, the infrastructure could for example be expanded with an FTP server on this specific VM with proxy through the NginX that was presented in section 4.3.1.

4.3.3. MySQL Master/Slave setup

The MySQL servers were hard issues to deal with, mainly because MySQL out of the box is made to be run on the same server as the application that is supposed to interact with it. To change this fact a combination of pre-constructed configuration files as well as a bit of queries through Ansibles community MySQL module to force it to accept connections from outside localhost. Furthermore, the database backup itself held problems of its own with parameters in the database still pointing towards the old domain name(acme.example.org) and had to be changed. This too was done through Ansible's query system.

Through Ansible's module for MySQL replication a master and slave system were devised. However, due to problems in getting the main WordPress site up and running properly the replication was not properly tested.

4.3.4. WordPress servers

The original idea was to make use of an old bash script for installing LAMP and WordPress and converting it to Ansible playbook(s). However, we mostly used different resources online on Ansible to create the file wordpress.yml which contains the playbook that sets up Apache, PHP and WordPress. This playbook is furthermore added to site_deploy.yml which runs all the playbooks necessary to set up the site.

The playbook was configured to also handle the backup of the WordPress site contents that was provided in the project description. There is room for improvement here when considering the integrity and scalability of the code - if the WordPress servers need to be recreated or upscaled to more servers, the solution ideally should also handle automatic backup or at least deployment of the most current backup. This is however not implemented, and the playbook in its current format reads and adds the contents from whichever backup is in the correct file path.

After much trial and error, we ran into a problem that we as of writing this we could not solve while the site itself is up and running, all the links on the site gives a “Not found” error, despite the GET request showing the correct address when inspected.

4.3.5. Prometheus/Grafana monitoring system

The way we created our setup for the monitoring, we used roles for both Prometheus and Grafana.
The tasks/main.yml are holding the tasks that are being executed to setup the tools for monitoring, the defaults/main.yml are holding the variables that are being used in the tasks/main.yml for Prometheus. The plan was later to call these roles in a Ansible playbook located in the Ansible folder. We also created a role for node-exporter, we planned to use this plugin to produce the metrics for the CPU and pull the data to insert in Prometheus.

5.Design considerations

ACME had a list of requirements of their new solution. Here we list these and how we intended to handle the requirements.

5.1. Scalability

ACME wanted at least 3 instances of the web server but also to easily be able to add more. This extends to the database which should be split into a master-slave replication which would also enable dynamic scaling by increasing the number of database servers. To achieve this, we envisioned an infrastructure as code solution where Terraform builds the instances that are defined in code. To add or remove instances of a specific type, we only change the number of that type of resource in the code. Infrastructure as Code also allows for easy

deployment and redeployment of instances. Also connected to scaling is the use of Ansible as a server configuration tool. Ansible deploys configurations on multiple instances at once regardless of how many of a specific resource there is. With Terraform and Ansible in unison we achieve an automated process where we can setup and (if needed) destroy and setup again the entire infrastructure up to the content of Acme Corporation's website, back to the latest backup.
Furthermore, configuration files and playbooks were created so that with a few small alterations some parts of the infrastructure could be scaled up or down with relative ease.

5.2. File server

ACME wanted, if able, to have a separate file storage for uploads from administrators. The Network File Sharing system was set up with this in mind, but the entire infrastructure of it as explained in the section about the NFS servers was not fully realized. The shared folders had yet to be connected to the correct server counterparts but the infrastructure for it was there. Furthermore, the ability to upload was not implemented because most of the efforts were focused on getting the site up. An FTP server on the NFS machine with access to shared folders would have been a suitable choice for this infrastructure.

5.3. Monitoring

ACME wants monitoring of their new site, and this should include 1) a visual overview and 2) an alarm if something unexpected happens, such as a server going down.
We had plans on setting up monitoring for metrics to measure CPU usage, and alert management whenever an instance goes down, but due to lack of time and errors in our setup when we tried setting up a scalable infrastructure these features were cut out. We managed to create a boilerplate with Prometheus and Grafana that would be ready to use whenever we finished the setup of the infrastructure. But unfortunately, we skipped setting it all up due to the errors and time restrictions.

5.4. Theoretical connections.

5.4.1. The starting point

Now, in its original form this application was what is referred to in Morris' book as a Snowflake system. It was a single server application running everything it needed, the only way to scale it up was vertically [9, p. 15]. Furthermore, the structure was highly monolithic and not separated. Everything was run on one machine and without any separation of concerns [9, p.56].

5.4.2. The result

Although not fully implemented or successful some progress has been made in terms of structure.
By dividing the structure into VMs with separation for storage, database, and WordPress servers we have reduced the monolithic structure somewhat, but not fully. To do this one would have to delve into the codebase itself and it is not even certain if this is possible with a WordPress site.

However, the infrastructure has been changed to minimize variation in terms of OS, everything is run on the same type of machines, and efforts have been taken to research the requirements of each application running on them in terms of sizing. Considerations regarding the OS, PHP and WordPress versions have also been researched in the matter which led to the choice of PHP version as WordPress of older versions did not support PHP 8 which could have impacted older php code. [9, p.17]
Furthermore, usage of configuration files and environmental variable files made sure that all servers received the same values where they needed it and raised the human failure tolerance of the deployment procedure [9, p.87].
As for the creation of the servers our case cannot be counted to be either fully continuous configuration synchronization or immutable servers, to be fully towards any of these would go against the combination of tools that we had. However, due to some quirks in the MySQL setup with replication, where once run, the playbook could not be run again (an already started replication cannot be started again) these servers were more in the way of the immutable, where you had to replace them entirely. However, the rest of the systems were, due to Ansible being the tool that managed the configuration, more towards the first [9, p. 194-197].
Ansible as a tool uses push configuration, since nothing is installed on the managed nodes, and everything is pushed to the nodes from a control node. Likewise, can be said about Terraform in its communication with OpenStack cloud to provision the servers, and in that way, we can safely say that the pattern Push Server Configuration has been followed to its fullest extent [9, p. 198-200]

People who helped us

Daniel Johansson and Ryan Bui, Group 3, instrumental in our efforts to correct the database as well as Ryan, who made a last-ditch effort to help us correct the malfunctioning links.
Caesar Lennartsson and Fredrik Ljungner, Group 4, Helped us deal with the issue that MySQL by default does not permit communication from outside its own server.
Delfi Sehidic, Group 7, Tried valiantly to help us get a bastion host working with OpenStack, but it sadly did not work in the end.

Bibliography

[1] https://github.com/DennisDemir24/2DV517-assignment2/tree/main/bash_scripts

[2] https://github.com/DennisDemir24/2DV517-assignment2/tree/main/Terraform

[3] https://github.com/DennisDemir24/2DV517-assignment2/tree/main/Ansible

[4] https://github.com/DennisDemir24/2DV517-assignment2/tree/main/Ansible/templates

[5] https://github.com/DennisDemir24/2DV517-assignment2/blob/main/Ansible/upload.sh

[6] https://github.com/DennisDemir24/2DV517-assignment2/tree/main/Ansible/roles/prometheus

[7] https://github.com/DennisDemir24/2DV517-assignment2/tree/main/Ansible/roles/grafana

[8] https://github.com/DennisDemir24/2DV517-assignment2/tree/main/Ansible/roles/node-exporter

[9] K. Morris, *Infrastructure as Code*, 2nd ed. O'Reilly Media, Inc. 2020.