2DV517

Project Report

Group 2

Alex Karlsson, Jesper Eriksson, Marcus Cvjeticanin, Olof Haga

Table of Contents

1. Introduction

This is a report of the final group assignment in the course 2dv517: Deployment Infrastructures. The task was to set up an infrastructure using the IaC and DevOps principles learned from previous tasks.

In this report we will present automation tools used in the process, how we automate different tasks such as setting up, removing, editing servers, what problems we encountered, choices we made that had impact on the project and why we made them.

The report is limited to 10 pages which implies that we can't go very deep on technical parts of the automation solutions presented (such as what the components of a database are, how a systemctl service is created etc.). For any code related questions or thought we will refer to our GitHub repository to be found here:
https://github.com/mjovanc/deployment-infrastructures-2DV517

2. Automation Tools

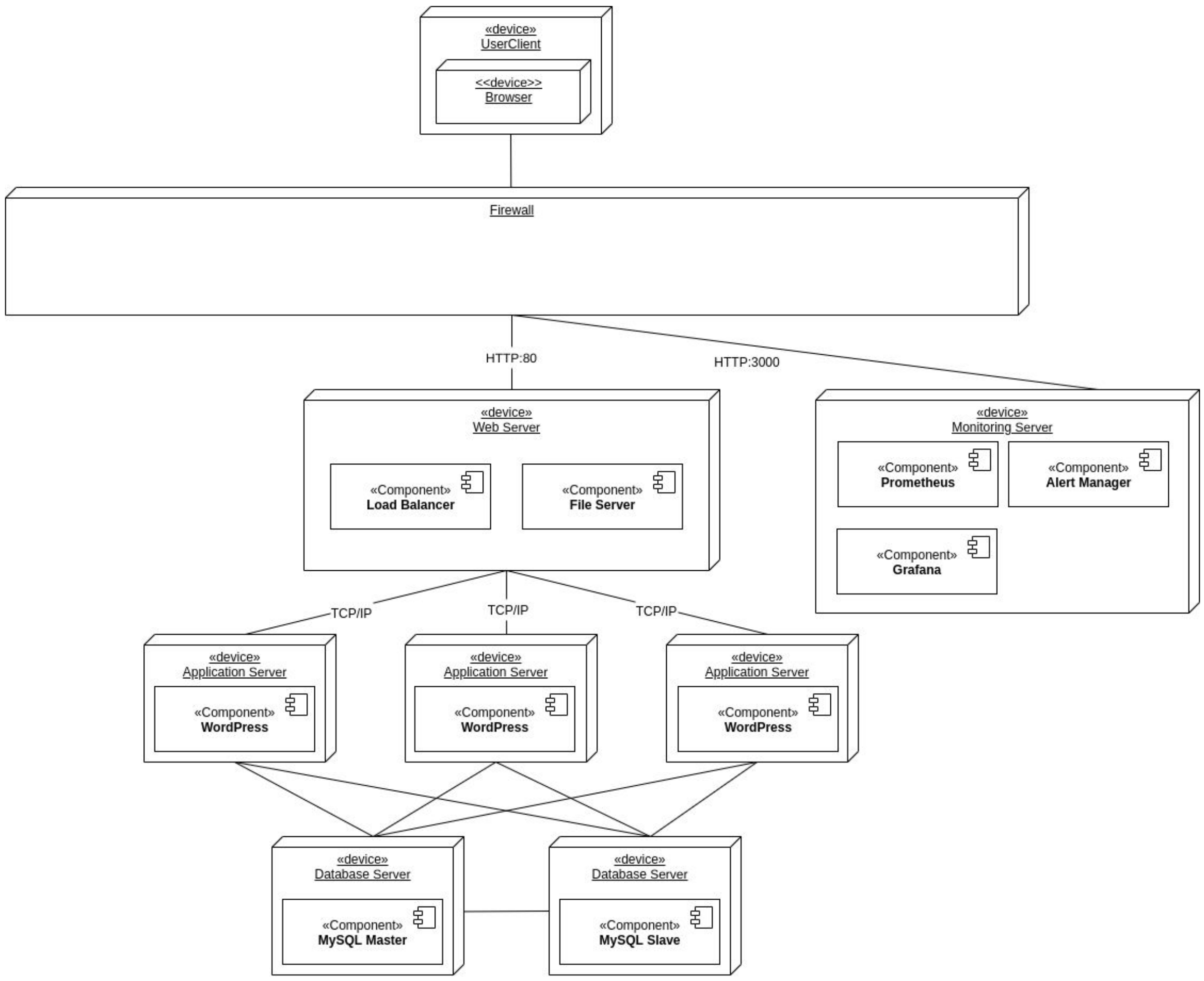
We have used Ansible which enables infrastructure as code to automate our infrastructure. There are multiple reasons behind this choice. The main reason is that Ansible supports all of our requirements and use cases for the infrastructure which are mainly provisioning and configuration management. Another reason is that Ansible has proven itself as it is popular in the industry and among DevOps practitioners.

We also discussed using Terraform for provisioning. The main use case for Terraform is provisioning and compared to Ansible it is more powerful and suited for this task. However, as stated, Ansible supports all of our requirements and this would result in us having to spend more time learning another tool. It can be valuable to use the same tool as much as possible as it lowers the level of expertise of working with the infrastructure.

Another reason is the fact that the infrastructure is relatively simple. The infrastructure is not multi-cloud and is limited to a few services centered around WordPress and monitoring. If the requirements for the infrastructure were more complex and demanding it could be a better option to use different tools for different tasks.

3. The infrastructure

3.1 Overview



The diagram above shows that we use as the first entry point a NGINX instance that serves as a load balancer and also a file server to be able to serve the content of the WordPress instances.

Then we use WordPress with three application servers that are configured to when doing only a read operation to use the MySQL Slave instance and when doing a write operation to use the MySQL Master instance.

Note that the diagram does not show Node Exporter to reduce clutter, but it is running on every instance in our infrastructure. Node Exporter communicates with Prometheus by sending metrics to it frequently, letting Prometheus know the status of the instances.

3.2 Persistent Data

Persistent data is a must since the infrastructure is disposable and instances can be created and deleted. There are two volumes in place for persistency. One volume is attached to the MySQL master as the volume data replicates to the slaves. The other volume is attached to the NGINX load balancer and contains the WordPress *wp-content* folder. All the WordPress instances request this content to reduce redundancy and to keep in sync with each other.

3.3 Load Balancer

To be able to load balance our WordPress application servers we chose to use NGINX for this as it is a very sophisticated tool that is easy to configure and one of the best tools to use today.

The load balancer is balancing the traffic through the WordPress instances that are used with the technique called, *least connected*. It will control the load on our instances more fairly in a situation when some of the requests take longer time to complete on a specific instance. As every instance will be added to a host group on creation it's possible to dynamically add all instances to the load balancer without any manual work.

3.4 WordPress

We have chosen to configure WordPress on the LAMP stack (Linux, Apache, MySQL and PHP) since it is a proven and popular choice that can be seen as the standard for WordPress applications.

In order to get WordPress working we had to make changes to the existing configuration and add some new configuration as well. The changes to the existing configuration we had to make were related to the database and included updating the URL for the website and outdated links. Both of these are now inserted with a variable using templating (Jinja2) to prevent it from becoming a problem in the future.

The new configuration includes standard Apache configuration such as the sites-available configuration (Virtual Host) for the website. It also includes configuration for rewriting requests for the folder *wp-content* to the NGINX load balancer that serves this static content.

We have chosen to keep the entire *wp-content* folder on the volume which contains themes, plugins and uploads among other things. The main reason behind this is to sync changes done in the admin dashboard, e.g., new content or plugins, to the other instances. The reasoning behind this choice is that WordPress is made to be simple to use and update. Other solutions like doing a configuration sync could become technical depending on the nature of the changes. It would also require the customer to contact the system administrators which would be infeasible for small changes.

3.5 MySQL

We used replication for our MySQL database to be able to use a master and a slave in our architecture. The responsibility for the MySQL master is to be able to replicate all the data to all of the MySQL slave instances and to be able to write data.

The MySQL slave instances will only allow you to read data from the database. This makes it more optimized rather than all read/write operations go through one MySQL server, making it work under higher load. It basically helps to stabilize our system.

3.6 Monitoring

The monitoring infrastructure included one head monitor server which initiated monitoring and contained configuration files. The main responsibility of the head monitor server is to install the Node Exporter package on the servers that we want to monitor. Node Exporter is constantly sending out data about the server it runs on (data such as CPU workload, memory, uptime etc.). This was done via an Ansible Role called *monitor_node_exporter_setup*. This Role downloads, unpackages, creates a systemd service for the package, starts the server and enables it on reboot.

When the nodes were up and running we could monitor them using the Prometheus service which read and presented the data gathered from the different Node Exporters that were constantly spitting out data. However, Prometheus presents data in a way that is pretty hard to read and you need to manually search for some of the data using variables. We chose to install Grafana which is a tool that presents data in easy to read graphs. You can customize how you want the data to be presented or import already created customizations made by other Grafana users. These tasks were not automated. The installation of Grafana and Prometheus, however, is.

We also created Ansible Playbooks to install Prometheus and Grafana. They had the same functionalities as the Node Exporter Playbook with the exception of creating configuration files for the Prometheus service. In these configuration files we state which servers are to be monitored. The configuration files are constantly updated when creating new servers in the infrastructure (that we want to monitor). This pretty much sums up the monitoring infrastructure. The infrastructure is automated and also dynamic, meaning that we can add and remove servers to monitor automatically. However, some set up (such as creating graphs in Grafana) is not automated.

3.7 Alerting

On the head monitor server we also have AlertManager, a plug-in for Prometheus, installed. It is installed by an Ansible Role called *monitor_setup_alertmanager*. It downloads, extracts, creates a systemd service and runs the service. This is all automated. However, when AlertManager is running you still need to manually set up some things.

We didn't manage to get the alerts working. We tested by having alerts sent to Slack, Discord and Email without any success. We didn't have the time to look in on how these tools are compatible with Ansible and if there are any modules that support them.

4. Infrastructure Automation

We have a *site.yml* that is the root configuration file (Playbook) for our Ansible project that is used for setting up the initial infrastructure. This file utilizes roles to create the infrastructure in a specific order. We also include a file *delete_site.yml* that does the exact opposite to our project. These files create and delete the network on OpenStack, including local network, local subnet, the router that connects it to the external network and security groups.

The *site.yml* also creates and configures the control node on OpenStack that is used to control the hosts. The control node is then used to provision the rest of the infrastructure with the Playbook *wordpress.yml*.

We used global variables on this project to store variables that were used on different roles such as MySQL database name, user, password etc.

Every role has its specific Playbook that is located in the directory called *tasks*, it specifies what software that is needed to be installed on the specific instance. There are some roles that also include a *files* directory that contains configuration files that will be copied to the instance. There is also a directory called *handlers* that we can use to do multiple repeated tasks from.

We also make use of Jinja2 templating to create dynamic configuration files used in the Playbook. The advantage of using templates is that we can make use of variables and statements such as if, for, foreach etc. This is used in multiple Playbooks.

5. Version Control

For Version Control we solely used Git. We made use of Git branches since we were a team of four that worked on different parts of the system. Our version control strategy was the Git feature branch workflow where each feature or addition to our project was developed using a branch. For example we could have a branch (derived from the master branch) where we tested new functionality such as automating the creation of MySQL servers or automation of a WordPress set up.

When the developer deemed the feature to be ready for production we merged the feature branch with the master branch which integrates the feature into the master branch. To this date the project has hundreds of commits and about 40 different feature branches.

6. Reasoning about our choices

6.1 Containerization

The infrastructure does not make use of containers. The software required for our various services are installed directly on the virtual machines in our infrastructure. Containerization is something that we looked into and discussed a lot during our first week.

We came to the conclusion to avoid containerization as we were not sure that it would be beneficial or reduce the complexity to make the infrastructure easier to manage. This is something that can be hard to assess without spending time and effort. Another reason is that we have limited experience with container management tools which is the norm to use for containers in production.

6.2 Monitoring software

When choosing which software to use for monitoring there was actually a lot to consider. We use three different kinds of software when monitoring. We use software to gather data, produce data, and present data. For the gathering of data which can be seen as the essence of monitoring, we chose between Prometheus and Naigos which seemed to be the most promising ones.

Naigos was actually a bit more promising at the beginning since it was agentless however, the documentation and support for Naigos was really inconsistent and in some areas non existent. Since this was the first time many of us worked with monitoring systems a good documentation and support was needed.

We eventually went with Prometheus. The other choices weren't as hard to make. Node Exporter for example, which we used to create the data to monitor, is really a sort of “plug-in” for Prometheus. Also for visualising the data in graphs, Grafana was the obvious choice since it is closely integrated with Prometheus.

7. Problems we encountered

Creating systemd services using Playbooks was harder than anticipated. Running services on a local test system was a lot different then when we used it in production. For example, a lot of our Playbooks needed to be rewritten since we hadn't correctly configured security and permission details in our Playbooks which led to Playbooks crashing.

Another problem we encountered was that we didn't configure the master-slave replication for the MySQL database to use it in the correct way. We got stuck for a while trying to troubleshoot for several hours without getting nowhere. As we saw in the end, a very small but critical error in the configuration of the MySQL master instance occured so that the slave could not connect to the MySQL master instance.

Related to this we had another problem concerning volumes. The data contained in the mounted volume was not mounted to the slave database. We spent a lot of time on troubleshooting this and in the end the solution was to dump the existing data from the master to the slave in order for them to sync.

As we worked individually at first there were some issues related to integrating the work that we had done separately. We also had some differences in how we structured our files, Ansible configurations and what naming convention to use in general. We should have decided on a Style Guide beforehand but as soon as we noticed the problem we rectified it.

OpenStack could also be problematic at times. Rebuilding instances was faster than deleting and creating new ones. However, when rebuilding instances there were several occasions that the instances were not rebuilt and still had the old software and files on them. There was no indication that a rebuild failed which could falsely lead us to believe that there were problems in our infrastructure code. After a rebuild we had to SSH into the instance to check whether or not it had been rebuilt.

8. Next Steps

Alerting:
We created ansible Playbooks to install and run the AlertManager service on remote servers, however we didn't actually get it to send any Slack notifications which would be prefered. We didn't get to the bottom of why this doesn't work since the configurations are correct as the documentation states.

Storage:
Storing monitored data is something that is currently not available. For example if a server goes down we are able to see and be noticed about, logs of it are currently
just temporarily saved on Prometheus. For this we could use a database to store crash log files to go back and overview them in case of similar crash events in the future and then see how we solved the problem back then.

Playbooks/Roles:
Some of the roles are not able to run again without manually skipping some tasks. This is done by adding tags to the task and then skip these tags when running the Playbook. This could be improved with conditional statements, letting Ansible figure out if the task is needed to run again, to increase automation.

Modules:
For some tasks we haven't had time to find proper modules, we have been using the *shell* module instead to write commands while a more specific module for that task could have been a better choice. Some modules are lacking functionality though, and the documentation for some modules are not the best.

Ansible Vault:
The project does not store any sensitive information such as passwords for our services in an encrypted way. It would be ideal to encrypt these variables so that the passwords are not in plaintext and could get compromised if the Ansible project code would be leaked out from where we store the Ansible project, in this case a Git repository.

HTTPS:
The website does not include sensitive data such as card information. But, in our opinion, all websites should use HTTPS for securing the traffic. We did not have time for this but it is something that should be implemented in the future.

Hardened Snapshots:
To be able to make a slimmed Snapshot image so we only use the packages that are needed and nothing else.